Real Time Object Detection for Autonomous Vehicles

Team Members:

- Salma Ayman Khamis
- Shahd Medhat Tawfeek
- Mohamed Ahmed Ibrahim
- Mohamed Ashraf Mohamed
- Moaz Ahmed Sayed Ahmed

Supervisor:

- Eng. Heba Mohamed

Agenda

Agenda

1. Overview & Workflow
2. Data Collection
3. Exploratory Data Analysis (EDA)
4. Preprocessing
5. Model Development
6. ML Flow Monitoring
7. Model Evaluation
8. Database & Dashboard
9. Gui Demo

1. Overview & Workflow

- There are many accidents that occur because of human error and injuries that can lead sometimes to death. If these errors are minimized, road safety for pedestrians and vehicles is enhanced while also improving the accuracy of cameras on the traffic lights to detect jaywalking, cars not respecting traffic lights circulation, hit and run attempts, etc....

1. Overview & Workflow (Cont.)

- This project aims to develop a Deep Learning (DL)-based real-time object detection system tailored for autonomous vehicles. The system will identify and classify objects such as pedestrians, vehicles, traffic signs, and obstacles from live video feeds. The goal is to enhance road safety, situational awareness, and decision-making for self-driving cars, with potential integration with IoT (e.g., self-aware robotic cars).

1. Overview & Workflow (Cont.)

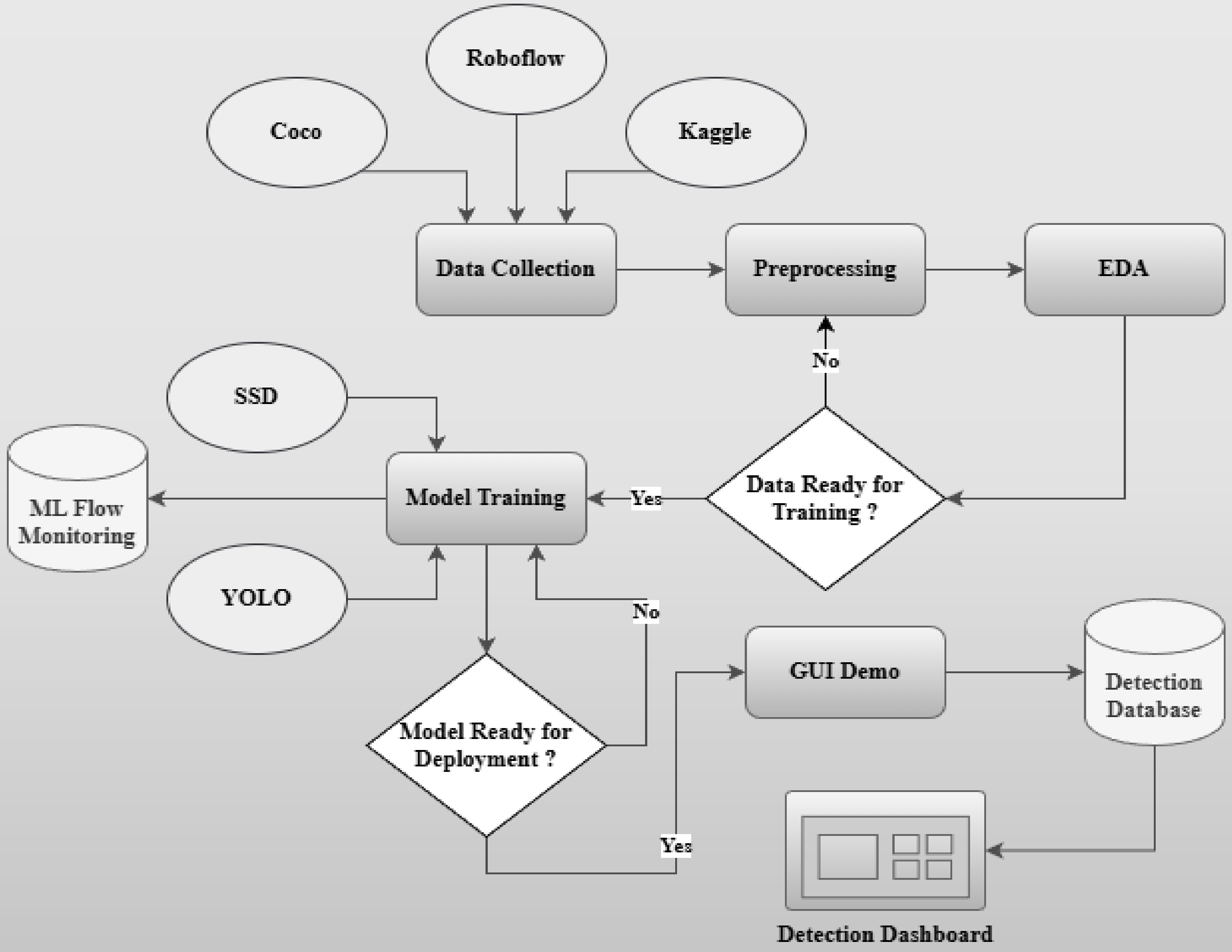


Fig 1. Project Workflow

Agenda

2. Data Collection

2. Data Collection (Cont.)

2.1. KITTI Dataset:

Aspect	Details
Domain	Autonomous driving (urban/suburban roads)
Image Count	~15,000 images (object detection subset)
Classes	Car, pedestrian, cyclist, van, truck, tram, etc. (limited categories)
Annotations	2D/3D bounding boxes, depth maps, LiDAR point clouds
Environment	Single city (Karlsruhe), daytime only, consistent camera angles
Strengths	Highly relevant to self-driving cars; Includes 3D + LiDAR; Realistic driving scenarios
Limitations	Small size; Limited classes; Low diversity; Not suitable for training large models

Table 1. KITTI Dataset Overview

2. Data Collection (Cont.)

2.2. Open Images Dataset:

Aspect	Details
Image Count	9M+ images
Classes	600+ classes
Annotations	Bounding boxes, relationships, image-level labels
Strengths	Massive scale; Excellent for large pretraining
Limitations	Inconsistent quality; Too large for student or small-scale projects

Table 2. Open Images Dataset Overview

2. Data Collection (Cont.)

~~2.1. KITTI Dataset~~

~~2.2. Open Images Dataset~~

2.3. Coco Dataset

2.4. Why Use Coco ?

2.5. Collection Process

2.3. Coco Dataset:

Aspect	Details
Domain	General-purpose, real-life scenes
Image Count	118,000 training + 5,000 validation (COCO 2017)
Classes	80 classes including car, truck, bus, motorcycle, bicycle, person, traffic light, stop sign
Annotation	Bounding boxes, segmentation masks, key points, stuff segmentation
Strengths	Large and diverse; High-quality labels; Industry-standard benchmark
Limitations	Not driving-specific; No LiDAR; Varied camera angles

Table 3. Coco Dataset Overview

2. Data Collection (Cont.)

~~2.1. KITTI Dataset~~

~~2.2. Open Images Dataset~~

~~2.3. Coco Dataset~~

2.4. Why Use Coco ?

2.5. Collection Process

2.4. Why Using Coco is More Appropriate ?

- Contains all key autonomous driving classes with high quantity and variation.
- Much larger and more diverse than KITTI, manageable unlike Open Images.
- All modern detectors (YOLOv3–v10, DETR, etc.) are benchmarked on COCO.
- Strong academic and industry justification.

2. Data Collection (Cont.)

~~2.1. KITTI Dataset~~

~~2.2. Open Images Dataset~~

~~2.3. Coco Dataset~~

~~2.4. Why Use Coco ?~~

2.5. Collection Process

2.5. Collection Process

- **RoboFlow**: this website has many labeled images (15,000 images) containing 11 classes and all these classes are related to road (pedestrian, car, truck, traffic light, etc...).
- **Kaggle Road Signs**: a data set designed for traffic signs, speed limit signs detection and contains 15 classes. This data set is used to reduce class bias (pedestrian, car) by combining it with other data sets like roboflow & coco. This would also enable autonomous vehicles to follow traffic rules and regulations, analyzing every sign whether it’s about speed limit or stop-and-go indications to navigate the roads safely [2].

2.5. Collection Process (Cont.)

- **Coco train 2017**: this is the data set of coco 2017 that's designed for training yolo and contains 118K images for training object detection [1].
- **Coco Val 2017**: this is the data set of coco 2017 that's designed for validating yolo and contains 5K images [1].
- **Coco Annotation**: the labels used for coco train and coco valid in order to train the models properly.

Agenda

3. Exploratory Data Analysis (EDA)

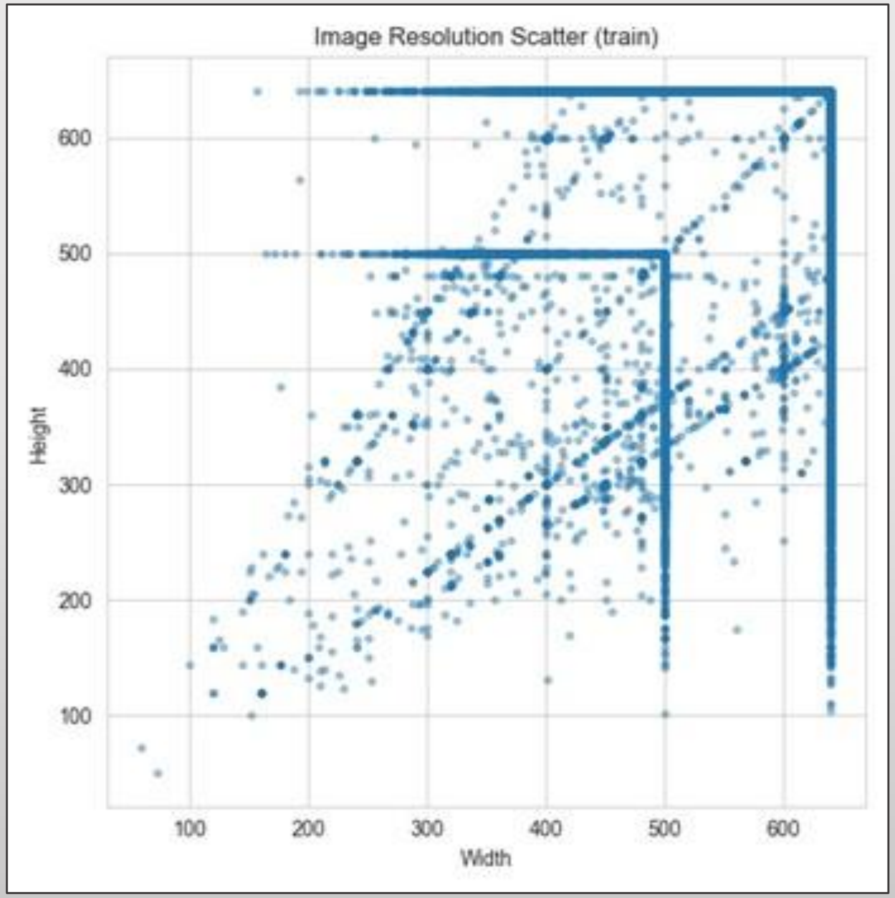


Fig 2. Training set resolution before resizing

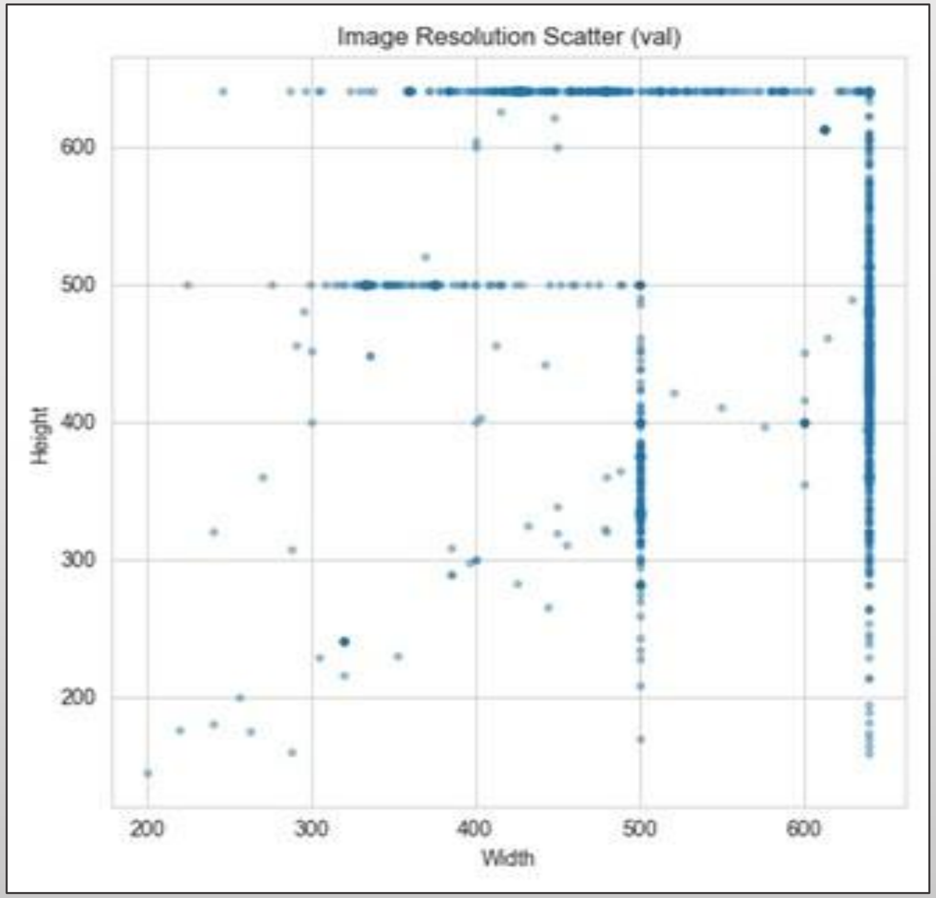


Fig 3. Validation set resolution before resizing

3. Exploratory Data Analysis (EDA) (Cont.)

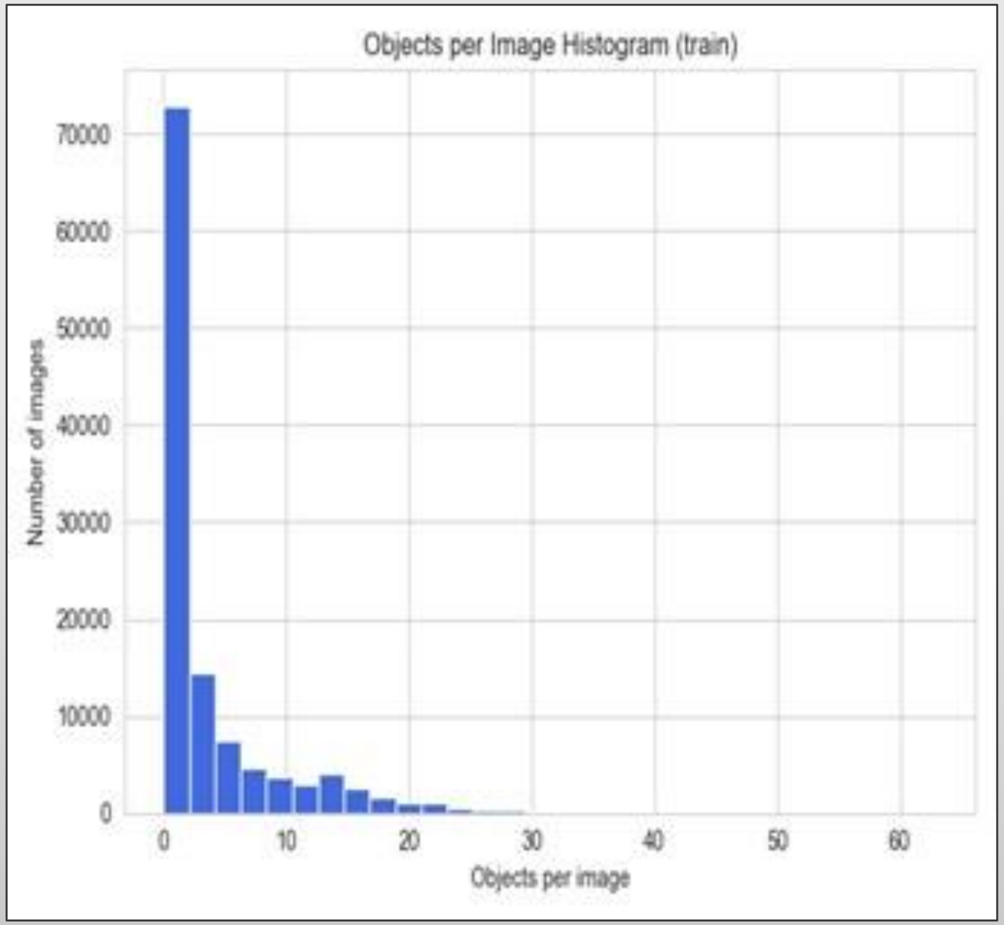


Fig 4. Objects per image histogram (train set)

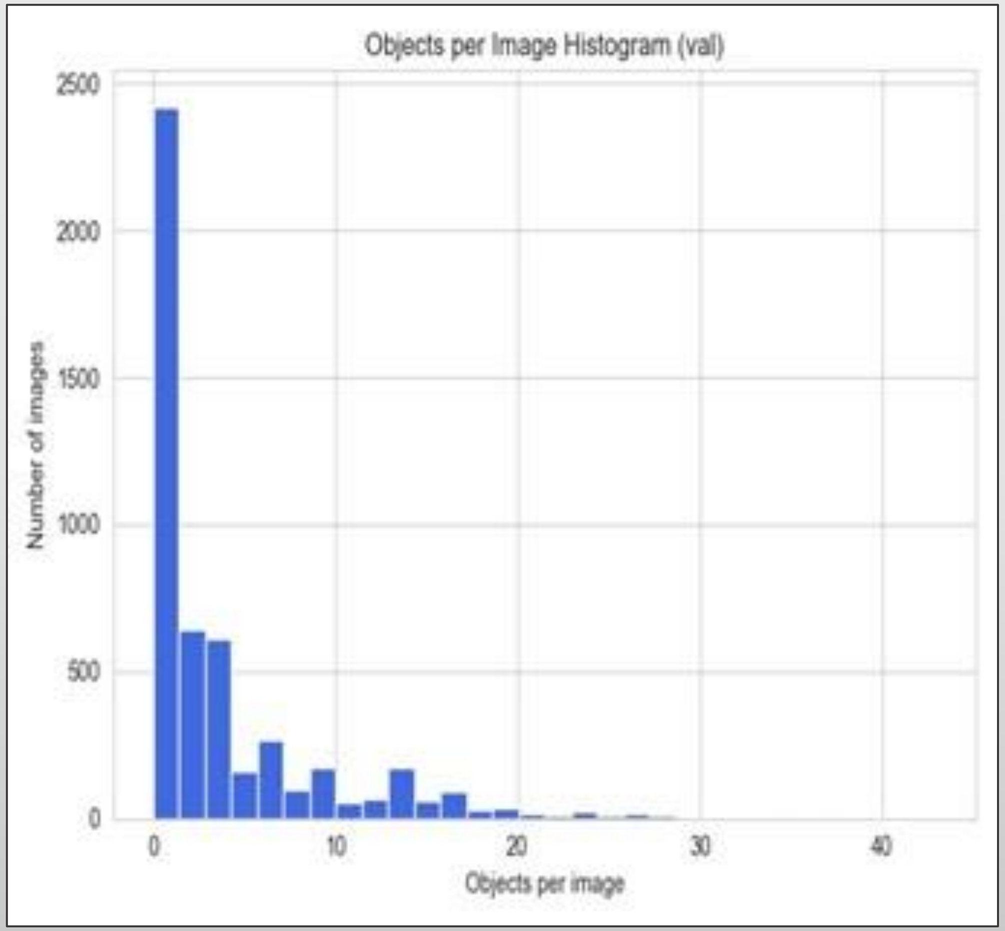


Fig 5. Objects per image histogram (valid set)

3. Exploratory Data Analysis (EDA) (Cont.)

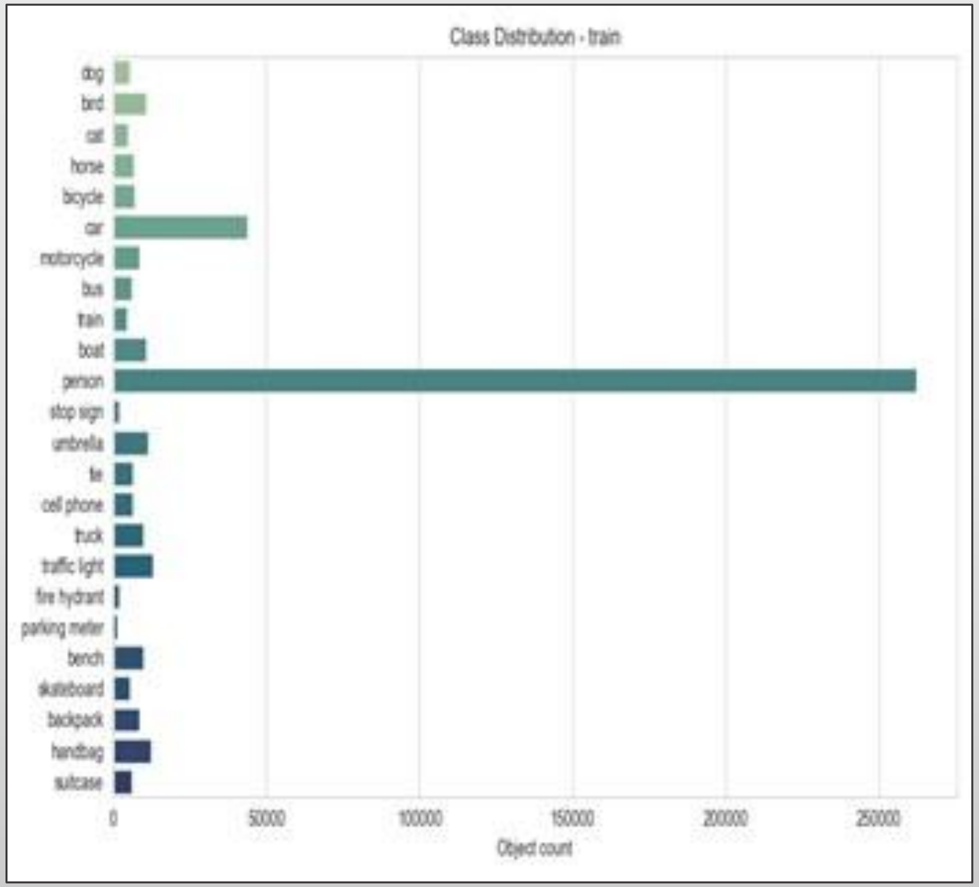


Fig 6. Class Distribution – Train set (before balancing)

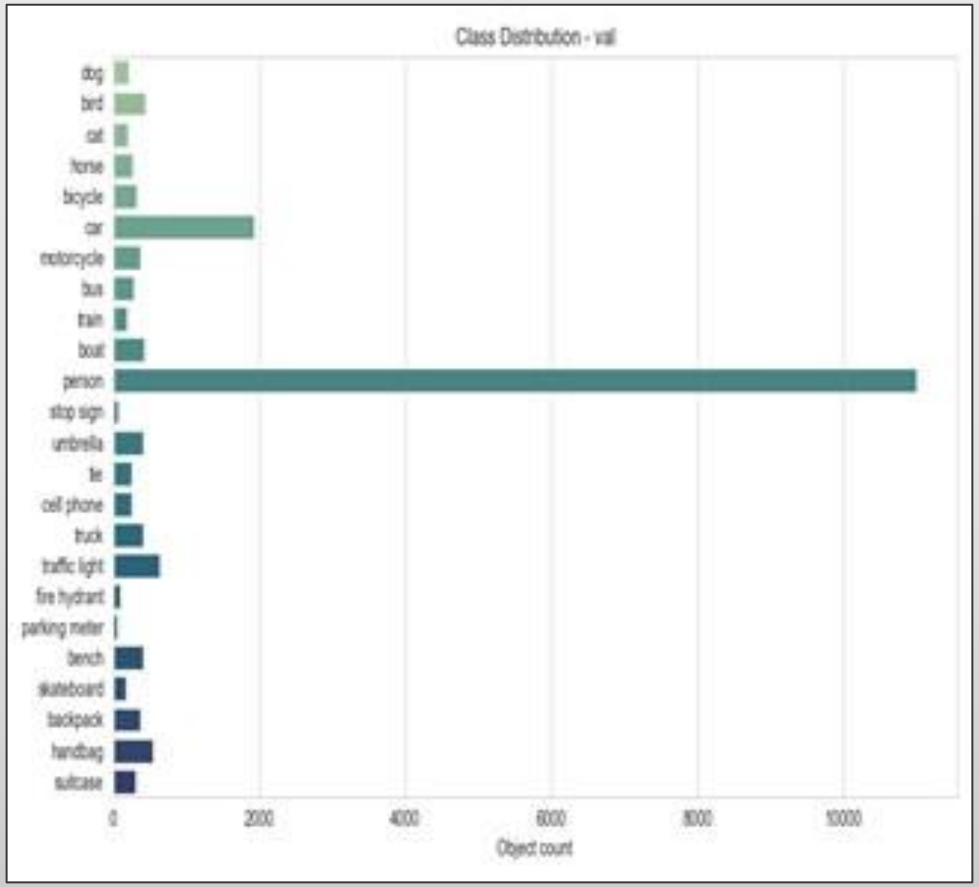


Fig 7. Class Distribution – Validation set (proportional, before balancing)

3. Exploratory Data Analysis (EDA) (Cont.)

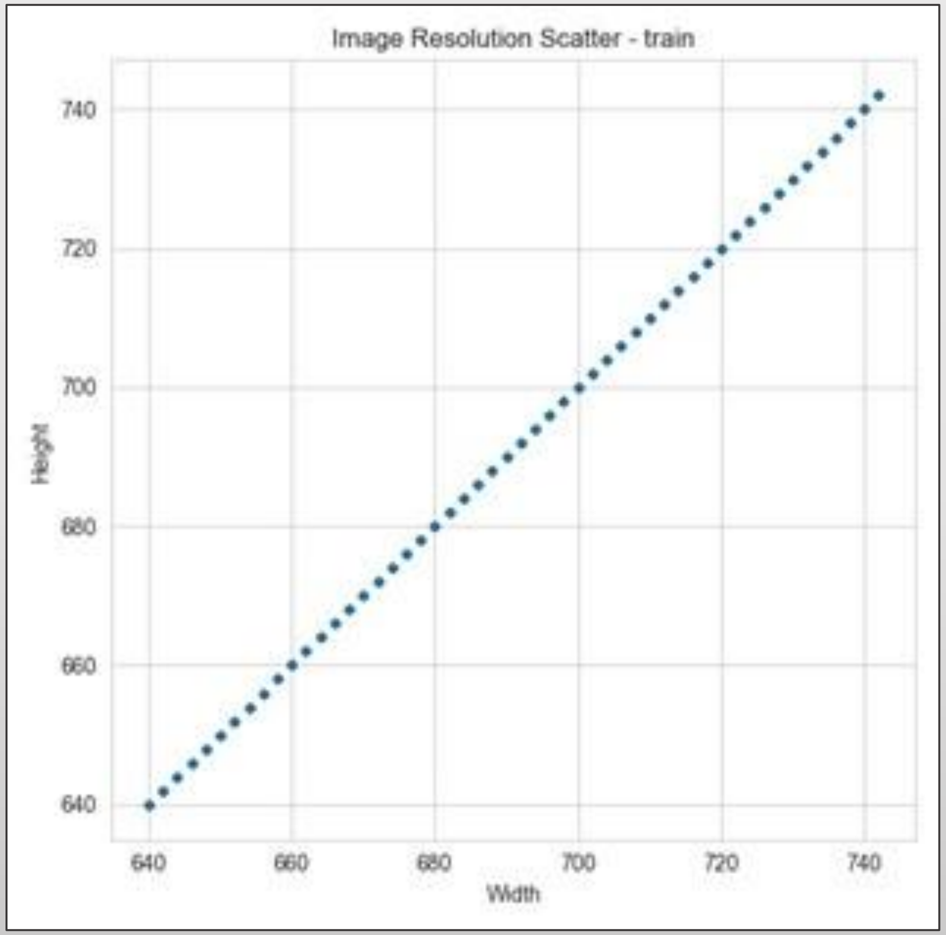


Fig 8. Resolution scatter after resizing (train set)

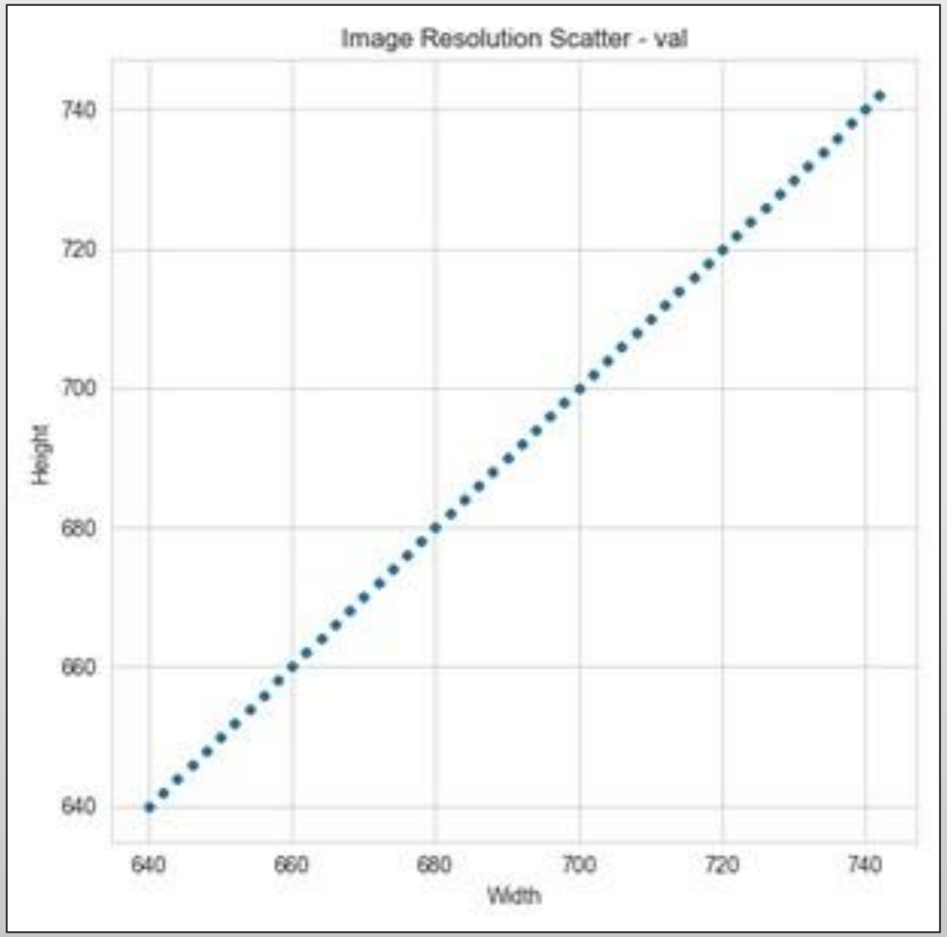


Fig 9. Resolution scatter after resizing (validation set)

Agenda

4. Preprocessing

4.1. Roboflow

4.2. RoboFlow & Kaggle & Sub-part of Coco

4.3. Full Coco Dataset 2017

4. Preprocessing (Cont.)

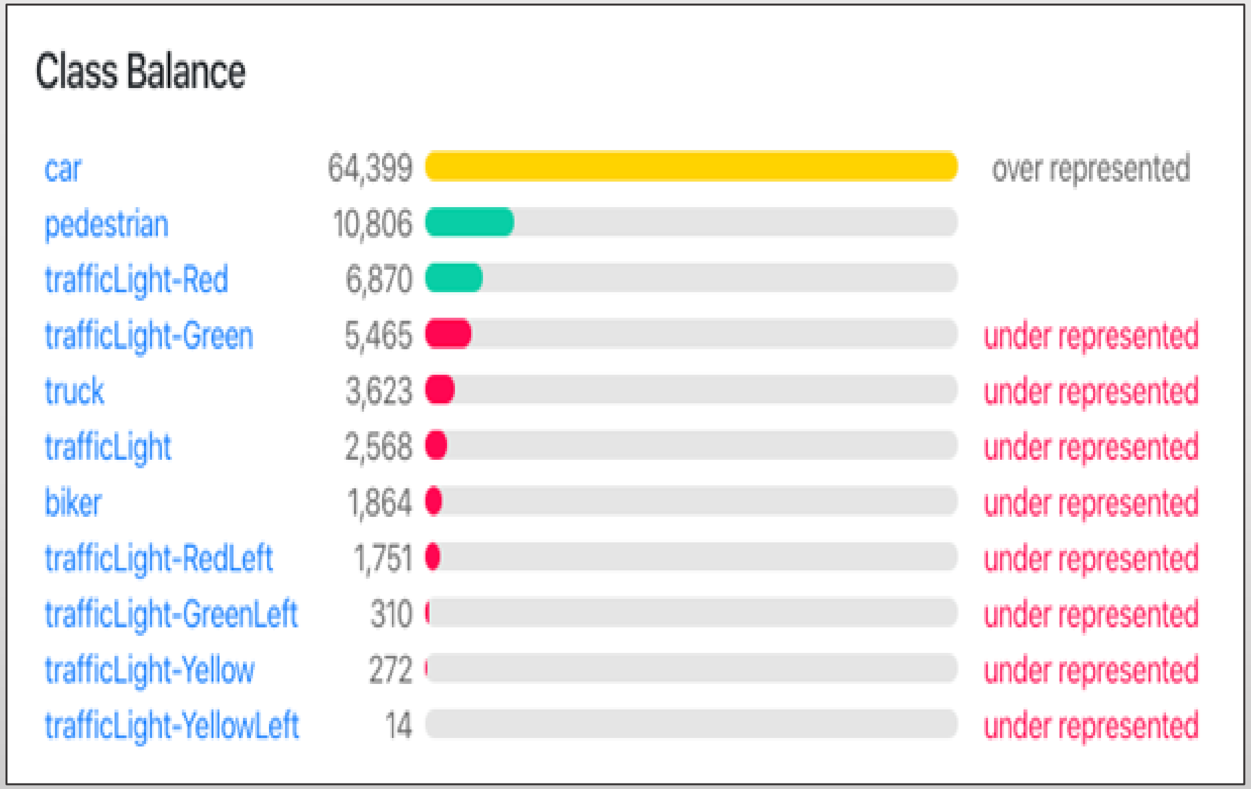
4.1. Roboflow

4.2. RoboFlow & Kaggle & Sub-part of Coco

4.3. Full Coco Dataset 2017

4.1. Roboflow

- This dataset didn't need that much preprocessing as it was already labeled, augmented and so on. But there is a problem where there is a high bias for the class **"car"** which was over-represented and most of the other classes were under-represented as we can see in the **(Fig 10)**. So, under sampling for this class was implemented to reduce the bias of the class



Class Balance

car	64,399	over represented
pedestrian	10,806	
trafficLight-Red	6,870	
trafficLight-Green	5,465	under represented
truck	3,623	under represented
trafficLight	2,568	under represented
biker	1,864	under represented
trafficLight-RedLeft	1,751	under represented
trafficLight-GreenLeft	310	under represented
trafficLight-Yellow	272	under represented
trafficLight-YellowLeft	14	under represented

Fig 10. Roboflow Represented Classes

4. Preprocessing (Cont.)

~~4.1. Roboflow~~

4.2. RoboFlow & Kaggle & Sub-part of Coco

4.3. Full Coco Dataset 2017

4.2. RoboFlow & Kaggle & Sub-part of Coco

- In this trial, combining all of these data sets in order to add more classes related to the road like (road signs, bus, train, etc...), fix the distribution of the classes by increasing the images for all the classes that’s under-represented and under sample some of the high classes to reduce bias and fix the labels across all data sets to unify them in one format to prevent confusion (e.g., index of class **“person”** is 1 in roboflow data set but in coco it’s index is 3).

4. Preprocessing (Cont.)

~~4.1. Roboflow~~

~~4.2. RoboFlow & Kaggle & Sub-part of Coco~~

4.3. Full Coco Dataset 2017

4.3. Full Coco Dataset 2017

- Using the full coco data set then filtering the data based on the problem by selecting specific classes since the data set contained over 80 classes, as selecting these specific classes for the problem will help train the model faster and consume less resources.

4.3. Full Coco Dataset 2017 (Cont.)

- **Key Steps Performed:**
  - Downloaded train 2017 (118k) + val 2017 (5k) + annotations.
  - Filtered to 24 road-relevant classes.
  - Cleaned invalid annotations and corrupted images.
  - Analyzed distributions (see figures above).
  - Applied targeted augmentation only to rare classes.
  - Converted to YOLO format (normalized center-x, center-y, w, h).

4.3. Full Coco Dataset 2017 (Cont.)

- **The Selected 24 Classes:**
  - person, bicycle, car, motorcycle, airplane, bus, train, truck, boat, traffic light, fire hydrant, stop sign, parking meter, bench, bird, cat, dog, backpack, umbrella, handbag, suitcase, skateboard, bottle, cup.

4.3. Full Coco Dataset 2017 (Cont.)

- **Applied only to underrepresented classes using Albumentations:**
  - Horizontal/Vertical flip, Rotation (±15°), RandomScale.
  - Brightness/Contrast/Hue adjustments.
  - CutOut, GridDropout, Gaussian noise.
  - Mosaic (YOLO-style) when applicable.
- All bounding boxes were correctly transformed and clipped.

4.3. Full Coco Dataset 2017 (Cont.)

- **Final Data Structure:**

```
autonomous_driving_dataset/
images/
train/                  (~140k images incl. augmented)
val/                    (5k original)
labels/
train/                  (YOLO .txt format)
val/
classes.txt
data.yaml
stats_report.json
```

Agenda

5. Model Development

- ➢ There are different types of models to choose from, and the choice will depend on a couple of factors (e.g., accuracy, speed of inference).
- ➢ Choosing Yolo and its versions especially (Yolov8) and SSD over Faster R-CNN seems like the right move since Yolo is known for fast and efficient object detection which makes it suitable for real-time systems. SSD on the other hand is another real-time detection model that is optimized for speed and accuracy. But Faster R-CNN may provide high accuracy but isn’t suitable for real-time detection since it’s slower than both of them. Having a model that predicts accurately without high speed of inference is a useless model.

5. Model Development (Cont.)

- **Yolo's Architecture:**
  - YOLO architecture is similar to **GoogleNet**. As illustrated, it has 24 convolutional layers, four max-pooling layers, and two fully connected layers. **[4]**

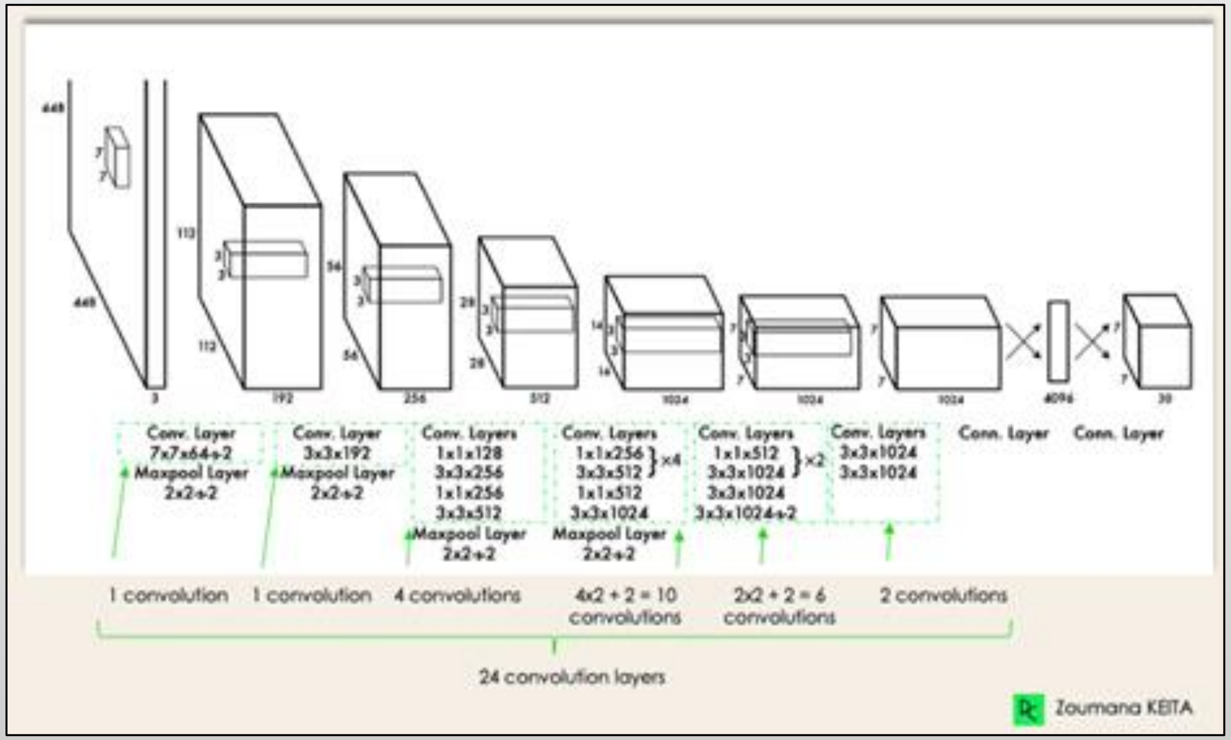

Conv. Layer 7x7x64-s-2 Maxpool Layer 2x2-s-2
Conv. Layer 3x3x192 Maxpool Layer 2x2-s-2
Conv. Layers 1x1x128 3x3x256 1x1x256 3x3x512 Maxpool Layer 2x2-s-2
Conv. Layers 1x1x256 3x3x512 ×4 1x1x512 3x3x1024 Maxpool Layer 2x2-s-2
Conv. Layers 1x1x512 3x3x1024 ×2 3x3x1024 3x3x1024-s-2
Conv. Layers 3x3x1024 3x3x1024
Conn. Layer
Conn. Layer
1 convolution
1 convolution
4 convolutions
4x2 + 2 = 10 convolutions
2x2 + 2 = 6 convolutions
2 convolutions
24 convolution layers
Zoumana KEITA

Fig 11. YOLO architecture from the original paper

5. Model Development (Cont.)

- **SSD Architecture:**
  - Generally, the architecture of an SSD typically consists of a base network, such as VGG or ResNet, that is pre-trained on a large image classification dataset, such as ImageNet. [3]

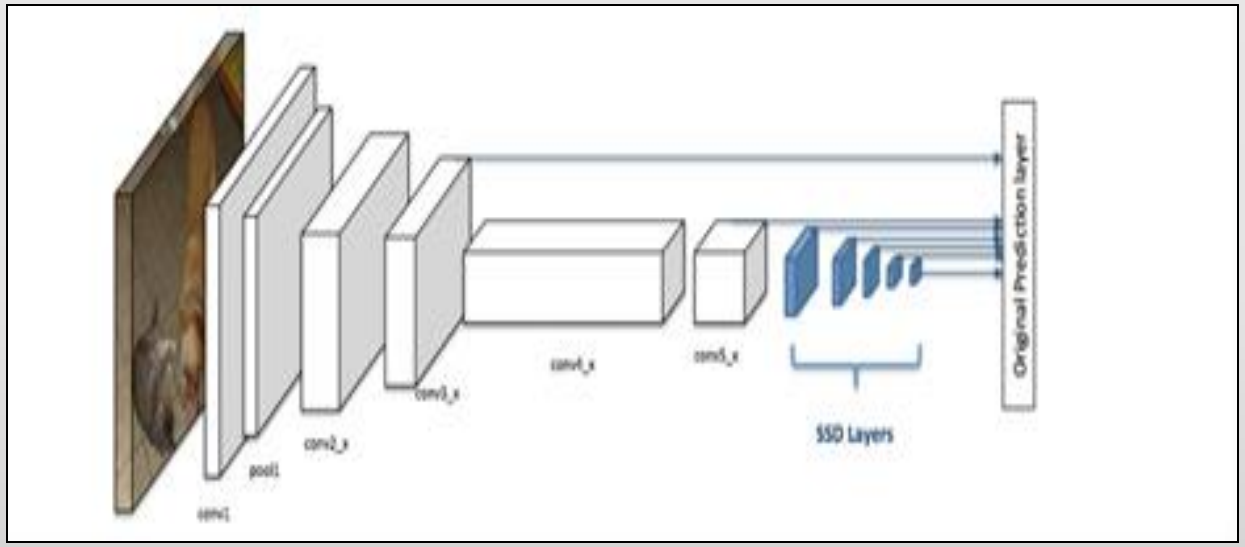


Fig 12. SSD's Architecture

5. Model Development (Cont.)

Summary of Experiments:

Experiment	Model	Data	Epochs	Accuracy (mAP)
First	Yolov8n	Roboflow without under-sampling	50	70%
Second	Yolov8n	Robflow with under-sampling	50	75%
Third	Yolov8s	Roboflow with under-sampling	100	85%
Fourth	Yolov8s	Roboflow with under-sampling & Kaggle & subpart of Coco	100	68%

5. Model Development (Cont.)

Experiment	Model	Data	Epochs	Accuracy (mAP)
Fifth	Yolov8n	Robflow with under-sampling	100	84%
Sixth	SSD300	Full 24 Class Coco dataset	10	33%
Seventh	Yolov8l	Full 24 Class Coco dataset	23	67%

Table 4. Summary of experiments

Agenda

6. ML Flow Monitoring

6. ML Flow Monitoring (Cont.)

6.1. The Challenge of ML Experiment

- **Complexity of ML Lifecycle**
  - Managing numerous runs, parameters, metrics, and artifacts becomes overwhelming without a centralized platform.
- **Reproducibility Issues**
  - Difficulty in recreating past results due to unlogged configurations and missing experiment metadata.

6.1. The Challenge of ML Experiment (Cont.)

- **Model Comparison**
  - Lack of objective evaluation framework for comparing different model variants and configurations.
- **Performance Tracking**
  - No standardized way to track and visualize metrics across multiple experiments and iterations.
- ✓ **The Solution: MLflow Tracking**
  - MLflow provides a robust, standardized platform to systematically log experiments, track metrics, manage artifacts, and enable objective model comparison.

6. ML Flow Monitoring (Cont.)

6.2. ML Flow Setup & Logging

1. **MLflow Setup**
   - ***mlflow***.*set_tracking_uri("file://mlruns")* ***mlflow***.*set_experiment("AutonomousVehicle_ObjectDetection")*
   - Initializes the tracking server to store experiment data locally and groups all runs under a named experiment.

6.2. ML Flow Setup & Logging (Cont.)

2. **Logging Function**
   - ***Def*** *log_model(model_path, run_name, metrics, params):* ***with*** *mlflow.start_run(run_name=run_name):* ***for*** *k, v* ***in*** *params.items(): mlflow.log_param(k, v)* ***for*** *k, v* ***in*** *metrics.items(): mlflow.log_metric(k, v) mlflow.log_artifact(model_path)*
   - Standardized function ensuring consistent logging of parameters, metrics, and model artifacts.

6.2. ML Flow Setup & Logging (Cont.)

- **Key Components Logged**
  - **Parameters:** Model variant (vanilla, hypertuned, best)
  - **Metrics:** mAP, IoU, Precision, Recall, F1_score
  - **Artifacts:** Trained model files (.pt)
  - **Metadata:** Run name, timestamp, source information

6. ML Flow Monitoring (Cont.)

~~6.1. The Challenge of ML Experiment~~

~~6.2. ML Flow Setup & Logging~~

6.3. Three Model Variants

6.4. Performance Metrics Comparison

6.5. Conclusion & Future Work

6.3. Three Model Variants

Experiment Name: "AutonomousVehicle_ObjectDetection"
All training runs are grouped under this experiment, facilitating direct comparison of parameters and metrics across all model variants.

YOLOv8n_Vanilla	YOLOv8n_Hypertuned	YOLOv8s_Best
Baseline The baseline model trained with default configurations. Serves as the control point for measuring the impact of optimization and hyperparameter tuning.	**Optimized** A model optimized through systematic hyperparameter tuning to improve performance. Demonstrates the effectiveness of fine-tuning strategies.	**Final Selection** The final, best-performing model variant selected after comprehensive experimentation. Ready for deployment in autonomous vehicle systems.

Table 5. ML Flow Runs

6. ML Flow Monitoring (Cont.)

6.4. Performance Metrics Comparison

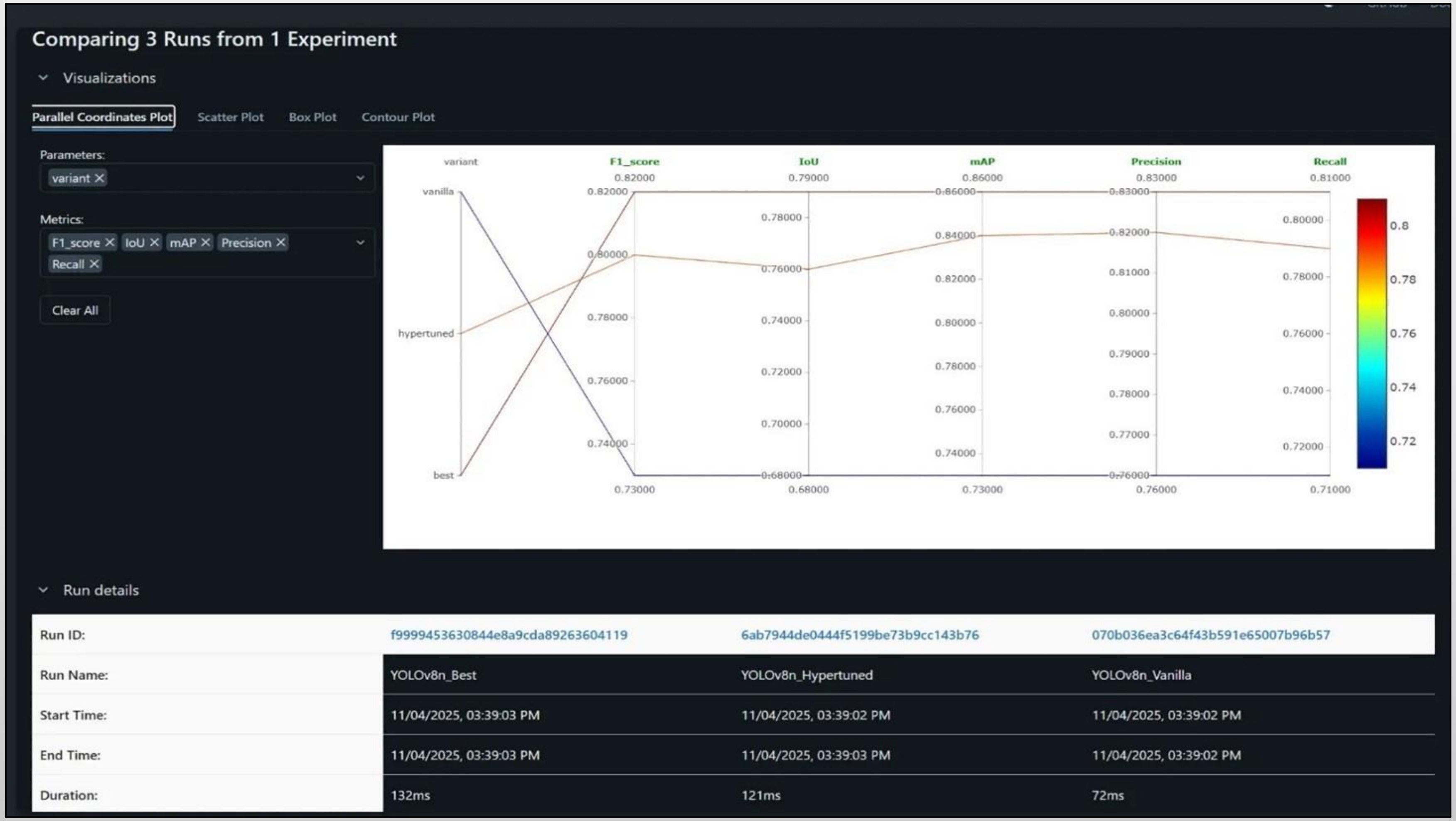
Run Name	Variant	mAP	IoU	Precision	Recall	F1_score
YOLOv8s_ Best	best	0.86	0.79	0.83	0.81	0.82
YOLOv8n_ Hypertuned	hypertuned	0.84	0.76	0.82	0.79	0.80
YOLOv8n_ Vanilla	vanilla	0.73	0.68	0.76	0.71	0.73

Table 6. ML Flow Runs' Metrics

6.4. Performance Metrics Comparison (Cont.)

- **Key Performance Insights:**
  - YOLOv8s_Best demonstrates **+13% mAP improvement** over the vanilla baseline, validating the optimization process.
  - Hyperparameter tuning achieved **+11% mAP gain** from vanilla to hypertuned, with further refinements yielding the best variant.
  - Consistent improvements across all metrics indicate robust optimization without metric trade-offs or overfitting.

6.4. Performance Metrics Comparison (Cont.)



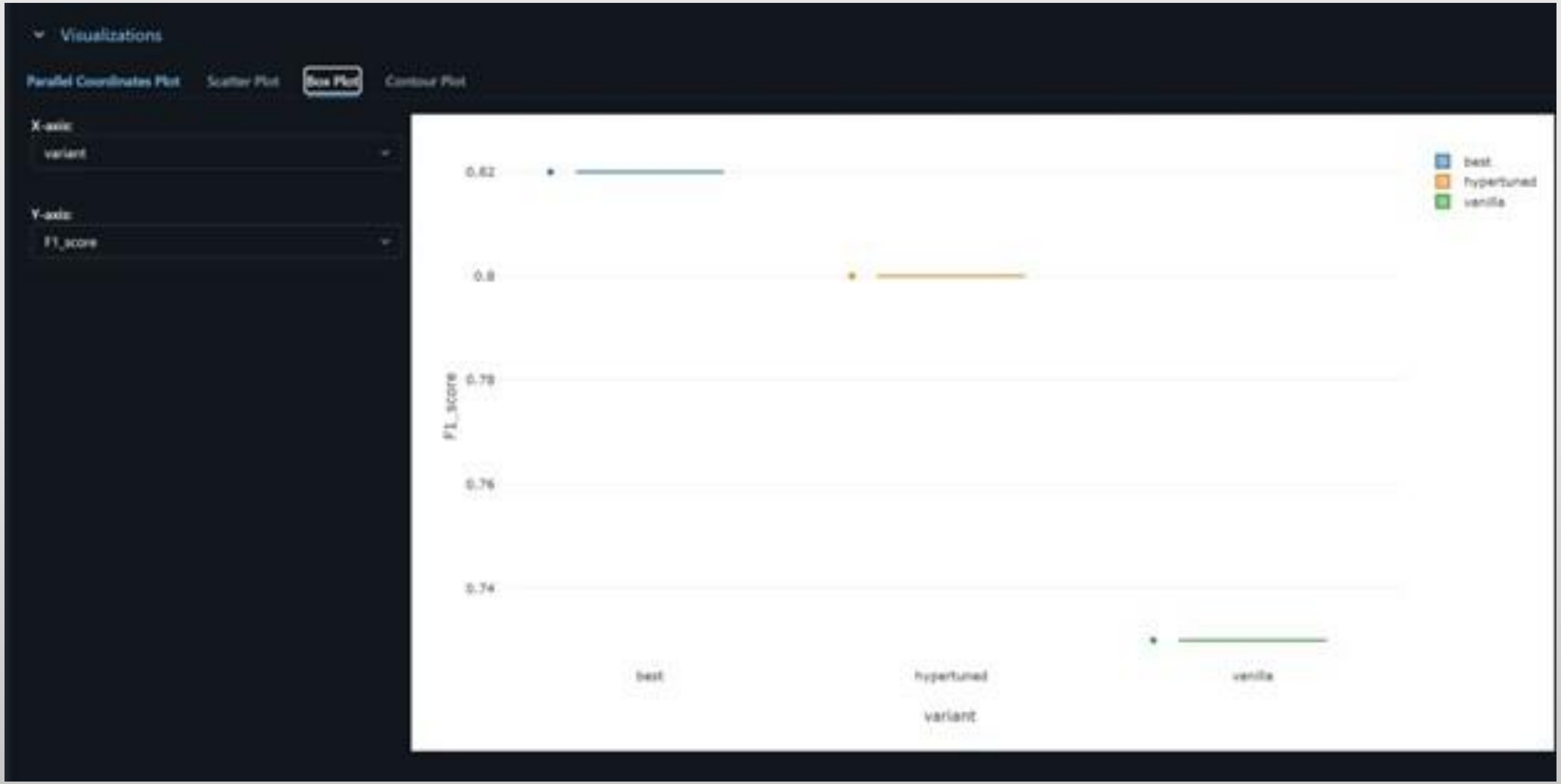
Run ID:	f9999453630844e8a9cda89263604119	6ab7944de0444f5199be73b9cc143b76	070b036ea3c64f43b591e65007b96b57
Run Name:	YOLOv8n_Best	YOLOv8n_Hypertuned	YOLOv8n_Vanilla
Start Time:	11/04/2025, 03:39:03 PM	11/04/2025, 03:39:02 PM	11/04/2025, 03:39:02 PM
End Time:	11/04/2025, 03:39:03 PM	11/04/2025, 03:39:03 PM	11/04/2025, 03:39:02 PM
Duration:	132ms	121ms	72ms

Fig 13. Parallel Coordinates Plot: Multi-Metric Performance

6.4. Performance Metrics Comparison (Cont.)

- **YOLOv8s_Best (Brown Line)**
  - Consistently achieves the highest values across all metrics: F1_score ≈ 0.82, mAP ≈ 0.86, IoU ≈ 0.79
- **YOLOv8s_Hypertuned (Orange Line)**
  - Shows significant improvement over vanilla with mid-range metrics: F1_score ≈ 0.80, mAP ≈ 0.84
- **YOLOv8s_Vanilla (Violet Line)**
  - Serves as the baseline lower bound, demonstrating the necessity of optimization and tuning
- **Key Insight**
  - The plot clearly illustrates the trade-offs and improvements gained from hyperparameter tuning, validating the iterative development approach and confirming YOLOv8s_Best as the optimal choice for autonomous vehicle deployment.

6.4. Performance Metrics Comparison (Cont.)



X-axis: variant

Y-axis: F1_score

Fig 14. F1-Score Analysis: Optimal Balance

6.4. Performance Metrics Comparison (Cont.)

- **Key Findings:**
  - **Best Variant**
    - **0.82**
    - YOLOv8s_Best achieves the highest F1-score, demonstrating superior balanced performance.
  - **Hypertuned Variant**
    - **0.80**
    - YOLOv8s_Hypertuned shows significant improvement over the baseline with strong performance.
  - **Vanilla Variant**
    - **0.73**
    - YOLOv8s_Vanilla serves as the baseline, demonstrating the impact of optimization efforts.

6.4. Performance Metrics Comparison (Cont.)

- **Why F1-Score Matters**
  - The F1-score is the harmonic mean of Precision and Recall, providing a balanced metric that minimizes both false positives and false negatives. This is critical for autonomous vehicle systems where both missing objects and false detections can have safety implications.

6. ML Flow Monitoring (Cont.)

6.5. Conclusion & Future Work

- **MLflow as the Foundation for Data-Driven Decisions**
  - MLflow proved indispensable for managing the complexity of the object detection pipeline. By systematically tracking parameters and metrics, it enabled a data-driven decision to select the optimal model, ensuring that the final model deployed is the most robust and accurate variant developed.

6.5. Conclusion & Future Work (Cont.)

- **Selected Model for Deployment:**
  - **YOLOv8s_Best**

F1_score	mAP	IoU
0.82	0.86	0.79

Table 7. Best Model of ML Flow Runs

6.5. Conclusion & Future Work (Cont.)

Future Work and Enhancements:

- **Model Registry Integration**
  - Implement MLflow Model Registry for versioning, stage transitions (Staging → Production), and collaborative model management.
- **Production Metrics**
  - Log inference performance metrics including latency, throughput, and resource utilization for production readiness assessment.
- **Architecture Expansion**
  - Expand experiments to include alternative architectures (EfficientDet, Faster R-CNN) for comprehensive comparative analysis.

Agenda

7. Model Evaluation

- To ensure a valid evaluation of the experiments conducted we had to test it in the street, to ensure valid object detection, how fast the inference and will the model be affected by different factors (**e.g.,** rain, storms, lighting, etc.…).
- We filmed a video for each model. **You can check it below:**
  - Video Explanation

Agenda

1. ~~Overview & Workflow~~
2. ~~Data Collection~~
3. ~~Exploratory Data Analysis (EDA)~~
4. ~~Preprocessing~~
5. ~~Model Development~~
6. ~~ML Flow Monitoring~~
7. ~~Model Evaluation~~
8. Database & Dashboard
9. Gui Demo

8. Database & Dashboard

- **Why Create a Database?**
  - Creating a database will help in storing the detected objects in an organized fashion. Also, will help to visualize it easily.
- **Why Choosing Relational database ?**
  - Relational databases have a fixed schema. Primary Keys to ensure no duplications and data consistency.

8. Database & Dashboard (Cont.)

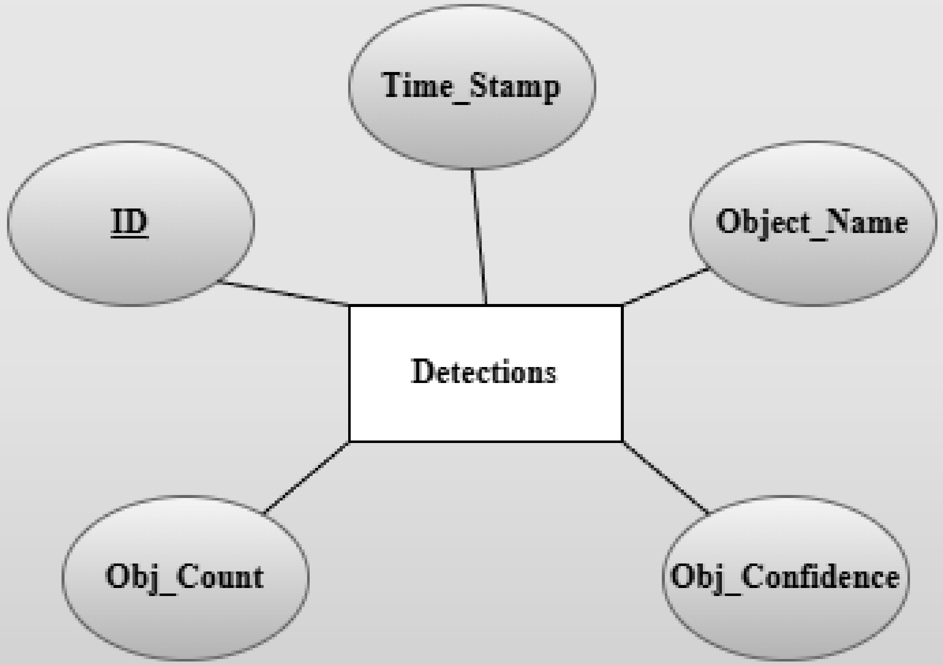
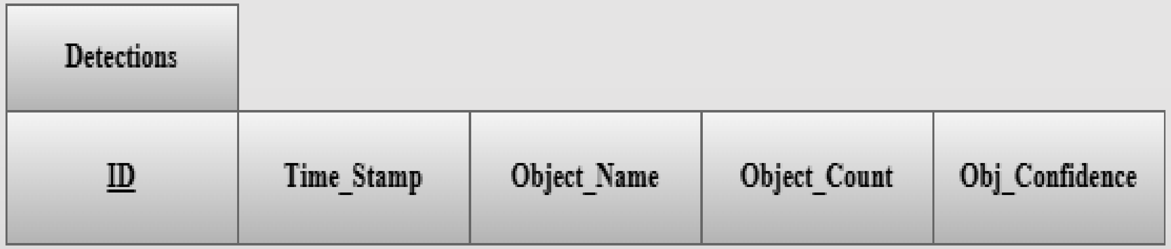


Fig 15. ERD of the database



Detections				
ID	Time_Stamp	Object_Name	Object_Count	Obj_Confidence

Fig 16. Shema of the database

8. Database & Dashboard (Cont.0

- Reading data from tables, databases or excel sheets can be tiresome and annoying. That’s why dashboards make it easier to read a lot of data in a visualized matter and extract information and insights from it with ease.

8. Database & Dashboard (Cont.)

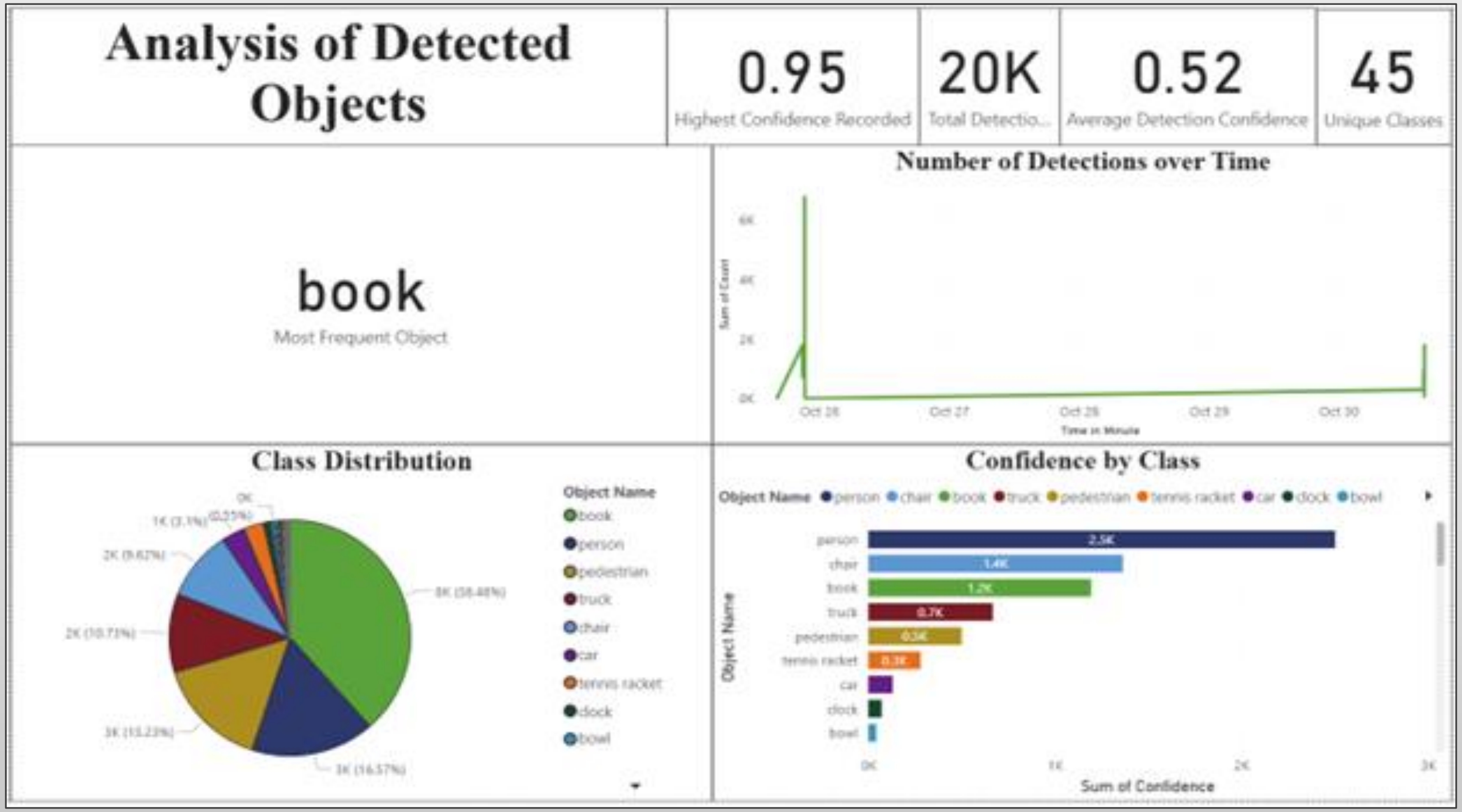


Fig 17. Detections Dashboard

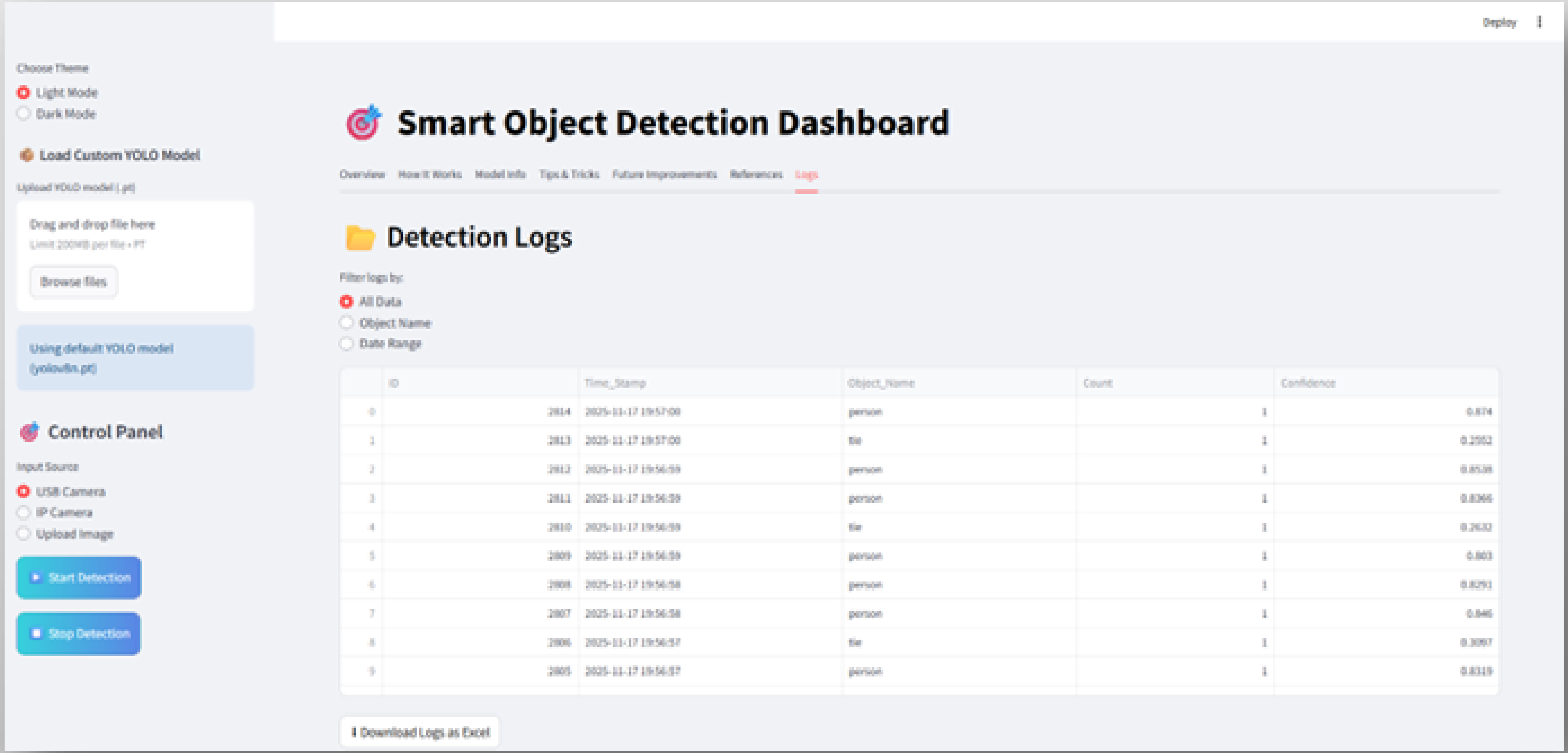
Agenda

1. ~~Overview & Workflow~~
2. ~~Data Collection~~
3. ~~Exploratory Data Analysis (EDA)~~
4. ~~Preprocessing~~
5. ~~Model Development~~
6. ~~ML Flow Monitoring~~
7. ~~Model Evaluation~~
8. ~~Database & Dashboard~~
9. Gui Demo

9. Gui Demo

- This application is a **real-time object detection dashboard** built using **Streamlit** as the frontend framework and **Ultralytics YOLOv8** as the core deep learning model for object detection.
- It supports live video streams (USB/IP cameras), static image analysis, customizable models, persistent logging via SQLite, live statistics visualization, theme switching (Light/Dark), animated visual effects, and data export capabilities.
- The system combines **computer vision**, **web development**, **data persistence**, and **interactive visualization** into a modern, user-friendly monitoring dashboard suitable for security, industrial monitoring, smart cities, or research applications.

9. Gui Demo (Cont.)



Deploy

Choose Theme

[✓ Light Mode]

Dark Mode

Load Custom YOLO Model

Upload YOLO model (.pt)

Drag and drop file here

Limit 200MB per file • PT

Browse files

Using default YOLO model (yolov8n.pt)

Control Panel

Input Source

[✓ USB Camera]

IP Camera

Upload Image

Start Detection

Stop Detection

Smart Object Detection Dashboard

Overview | How It Works | Model Info | Tips & Tricks | Future Improvements | References | Logs

Detection Logs

Filter logs by:

[✓ All Data]

Object Name

Date Range

	ID	Time_Stamp	Object_Name	Count	Confidence
0	2814	2025-11-17 19:57:00	person	1	0.874
1	2813	2025-11-17 19:57:00	tie	1	0.2552
2	2812	2025-11-17 19:56:59	person	1	0.8538
3	2811	2025-11-17 19:56:59	person	1	0.8366
4	2810	2025-11-17 19:56:59	tie	1	0.2632
5	2809	2025-11-17 19:56:59	person	1	0.803
6	2808	2025-11-17 19:56:58	person	1	0.8291
7	2807	2025-11-17 19:56:58	person	1	0.846
8	2806	2025-11-17 19:56:57	tie	1	0.3097
9	2805	2025-11-17 19:56:57	person	1	0.8319

Download Logs as Excel

Fig 18. GUI Window

9. Gui Demo (Cont.)

- **Key Components:**
  1. Dynamic model selection.
  2. Dynamic source selection (e.g., IP camera, USB camera, picture).
  3. Theme selection to appeal to all users (e.g., light or dark theme).
  4. Starting/ending the live feed with a press of button for IP/USB cameras only.
  5. Logging of detected objects in a readable format and exporting them as a csv/xlsx.

References

[1] **Ultralytics**, “COCO Dataset Structure,” *Ultralytics Documentation*. Available: https://docs.ultralytics.com/datasets/detect/coco/

[2] **P. K. Darabi**, “Car Detection Dataset,” *Kaggle*. Available: https://www.kaggle.com/datasets/pkdarabi/cardetection

[3] **Baeldung**, “SSD: Single Shot Detector,” *Baeldung on Computer Science*. Available: https://www.baeldung.com/cs/ssd

[4] **DataCamp**, “YOLO Object Detection Explained,” *DataCamp Blog*. Available: https://www.datacamp.com/blog/yolo-object-detection-explained

Go Back