Assignment 3 Report: Object Detection and Counting from Drone-based Images

Yusuf Demir
2210356074

Introduction

In this assignment, we aim to perform object detection and counting on drone-based parking lot images using the YOLOv8 model. The goal is to accurately detect and count the number of cars in each image and compare the predictions with the ground truth annotations.

We work with a subset of the CARPK dataset, which provides bounding box labels for each car in the images. The dataset is divided into training, validation, and test sets.

YOLOv8 is a state-of-the-art object detection model that performs predictions in a single pass, making it both fast and efficient. In our work, we fine-tune the YOLOv8n (nano) variant using different training strategies, including freezing specific parts of the network and experimenting with various hyperparameters. Finally, we evaluate the models using Exact Match Accuracy, Mean Squared Error (MSE), Precision, and Recall.

Dataset Preparation

Dataset Structure

We used a subset of the CARPK dataset, which contains drone-captured images of parking lots with labeled car instances. Each annotation is provided in YOLO format with a single class: `car`. The dataset was split into 1000 training images, 200 validation images, and 200 test images. The labels consist of bounding box coordinates normalized between 0 and 1, stored in text files corresponding to each image.

Preprocessing and Formatting for YOLOv8

To make the dataset compatible with the YOLOv8 training pipeline, we structured it into the following folder hierarchy:

```
/cars_dataset/
    images/
        train/
        val/
        test/
    labels/
        train/
        val/
        test/
```

We ensured that each label file contained lines in the standard YOLOv8 format:

```
class_id x_center y_center width height
```

Additionally, we created a `data.yaml` file that describes the dataset structure for YOLOv8. This file includes the paths to the training, validation, and test image folders, and defines the number of classes (`nc: 1`) and their names (`names: ['car']`).

1 YOLOv8 and Experimental Setup

1.1 YOLOv8 Architecture Overview

YOLOv8 is a single-stage object detector designed for real-time tasks. It follows the common YOLO structure with a Backbone, Neck, and Head. The backbone extracts features, the neck refines multi-scale representations, and the

head makes final predictions. YOLOv8 uses C2f (Cross-Stage Partial Fusion) modules and SPPF (Spatial Pyramid Pooling Fast) for efficient feature extraction. The loss function includes both box regression loss and classification loss.

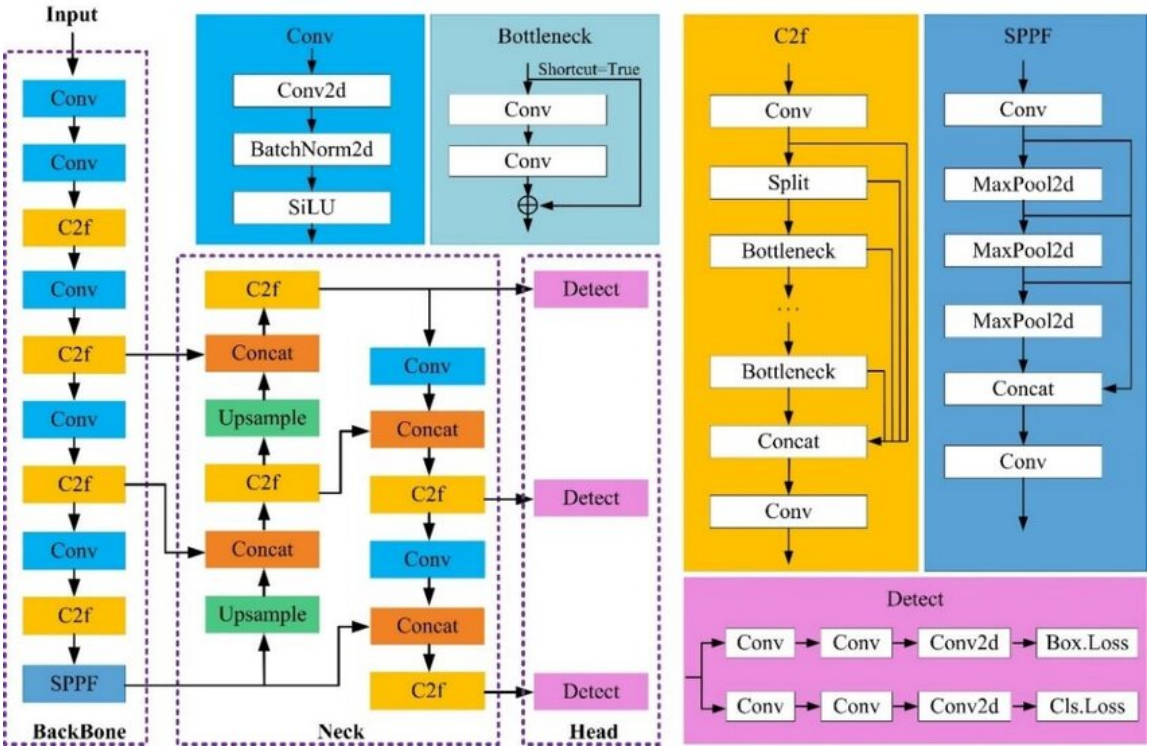

Figure 1: YOLOv8n architecture used in the experiments.

1.2 Freezing Strategies and Training Settings

To investigate the effect of transfer learning, we applied different freezing strategies on the YOLOv8n architecture. Based on the model structure, we defined the following freeze levels:

- **Freeze First 5 Blocks:** The stem (initial convolution) and the first two backbone blocks (C2f) are frozen. This preserves low-level and early mid-level features while allowing deeper layers to adapt to the new dataset.
- **Freeze First 10 Blocks:** The entire backbone and the SPPF (Spatial Pyramid Pooling - Fast) module are frozen. Only the neck and detection head are trained.
- **Freeze First 21 Blocks:** All layers except the detection head are frozen. This results in very fast training, with only the final classification and localization layers being updated.
- **Full Training:** No layers are frozen; the entire network is fine-tuned end-to-end, allowing maximum adaptation to the new domain.

Each of these strategies was evaluated with various hyperparameter settings to compare their performance in terms of detection accuracy and training loss.

2 Hyperparameter Tuning

2.1 Variants and Parameter Choices

We experimented with the following hyperparameters to observe their effects on training:

- **Optimizers:** Adam, SGD
- **Learning Rates:** 0.001, 0.0005, 0.0001
- **Batch Sizes:** 16, 32

2.2 Effect of Optimizer, Learning Rate, and Batch Size

From our experiments, the Adam optimizer generally led to more stable and accurate training. SGD showed slightly higher loss and lower scores. A learning rate of 0.001 worked best in most cases. Smaller learning rates like 0.0001 caused slower learning. The effect of batch size was minor, but batch size 32 helped with smoother convergence.

2.3 Freeze-5 Training Results

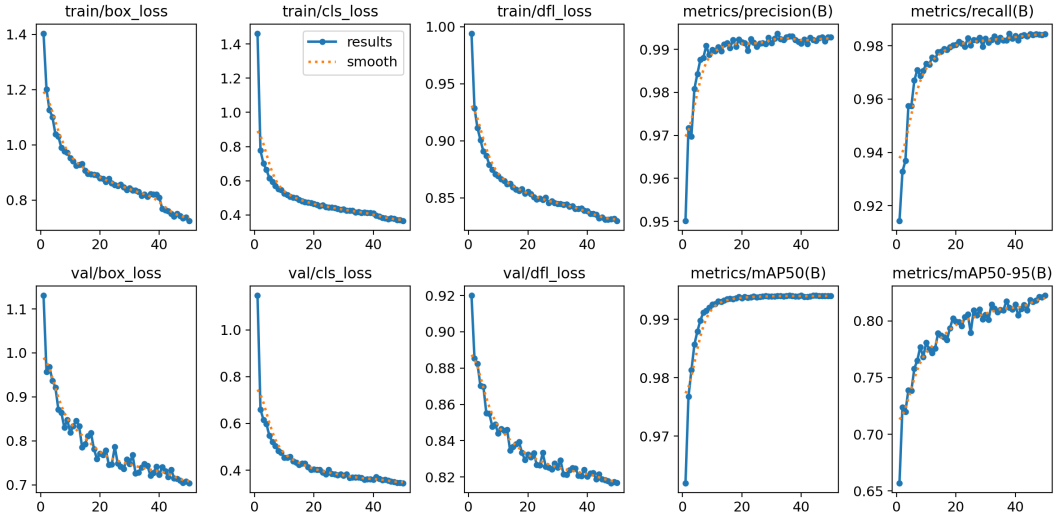

Figure 2: Freeze-5: Adam optimizer, LR = 0.001 (Best Model)

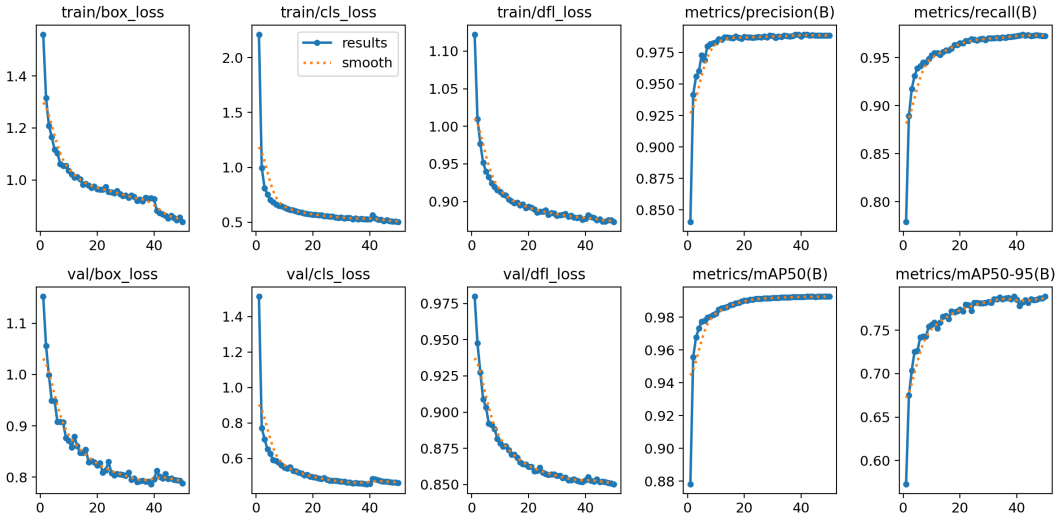

Figure 3: Freeze-5: SGD optimizer, LR = 0.001

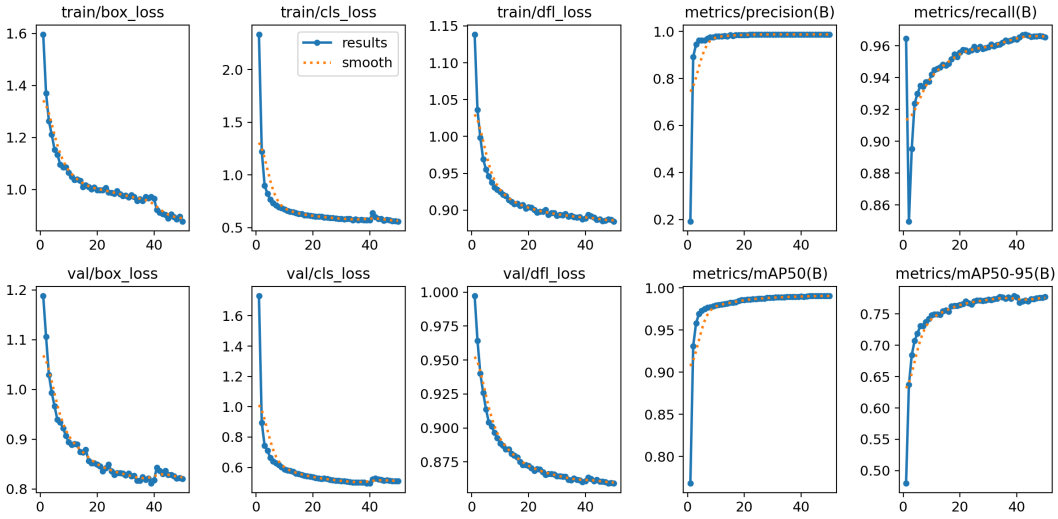

Figure 4: Freeze-5: SGD optimizer, LR = 0.0005

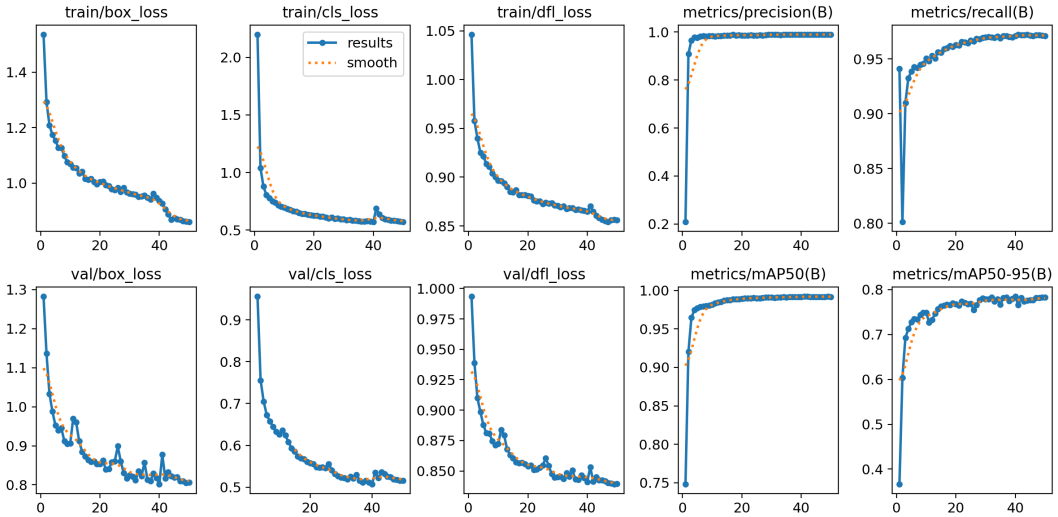

Figure 5: Freeze-5: Adam optimizer, LR = 0.0001

Model	**Precision**	**Recall**	**Box Loss**	**Cls Loss**	**Score**
yolov8n_adam_lr001_freeze5	0.99285	0.98446	0.70407	0.34508	0.6860
yolov8n_sgd_lr001_freeze5	0.98856	0.97259	0.78811	0.46389	0.6592
yolov8n_sgd_lr0005_freeze5	0.98714	0.96540	0.82037	0.50861	0.6481
yolov8n_adam_lr0001_freeze5	0.99023	0.97092	0.80660	0.51590	0.6522

Table 1: Freeze-5: Results from 4 training configurations.

2.4 Freeze-10 Training Results

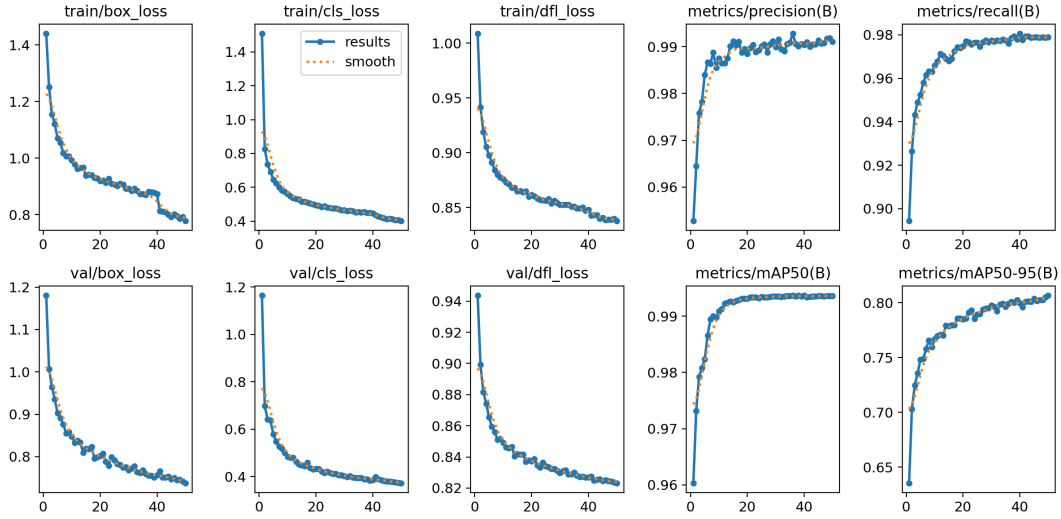

Figure 6: Freeze-10: Adam optimizer, LR = 0.001 (Best Model)

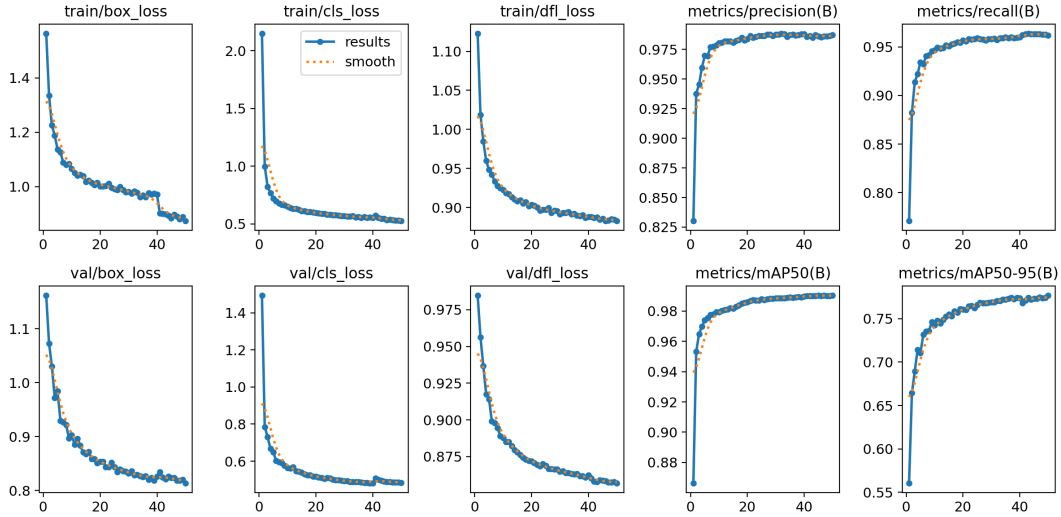

Figure 7: Freeze-10: SGD optimizer, LR = 0.001

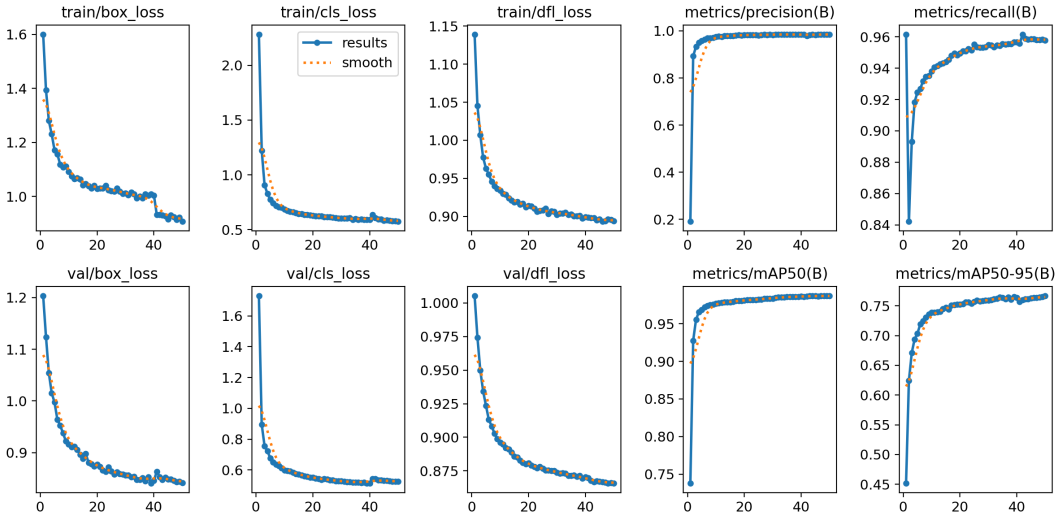

Figure 8: Freeze-10: SGD optimizer, LR = 0.0005

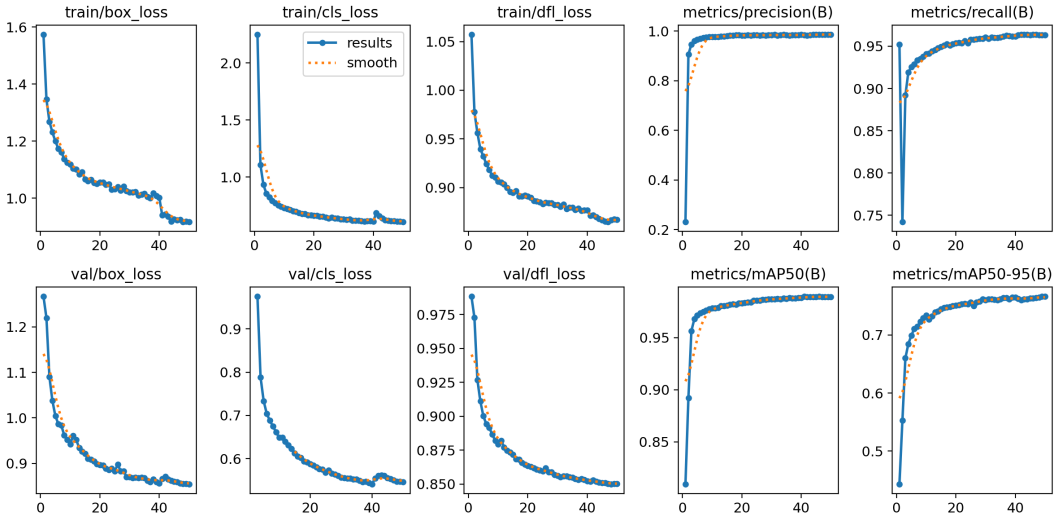

Figure 9: Freeze-10: Adam optimizer, LR = 0.0001

Model	**Precision**	**Recall**	**Box Loss**	**Cls Loss**	**Score**
yolov8n_adam_lr001_freeze10	0.99107	0.97886	0.73772	0.37254	0.6769
yolov8n_sgd_lr001_freeze10	0.98716	0.96184	0.81367	0.48319	0.6499
yolov8n_sgd_lr0005_freeze10	0.98537	0.95775	0.84131	0.52438	0.6406
yolov8n_adam_lr0001_freeze10	0.98681	0.96338	0.85438	0.54607	0.6400

Table 2: Freeze-10: Results from 4 training configurations.

2.5 Freeze-21 Training Results

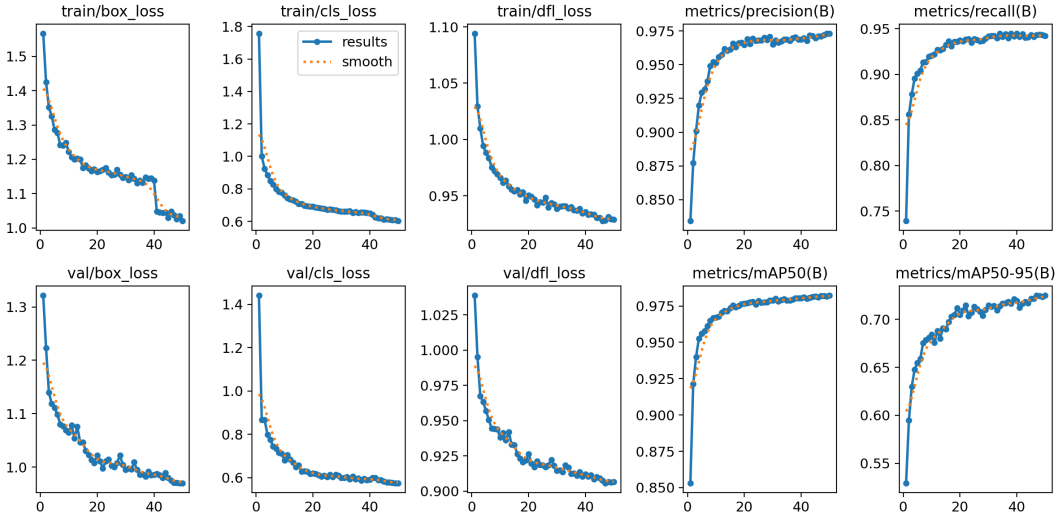

Figure 10: Freeze-21: Adam optimizer, LR = 0.001 (Best Model)

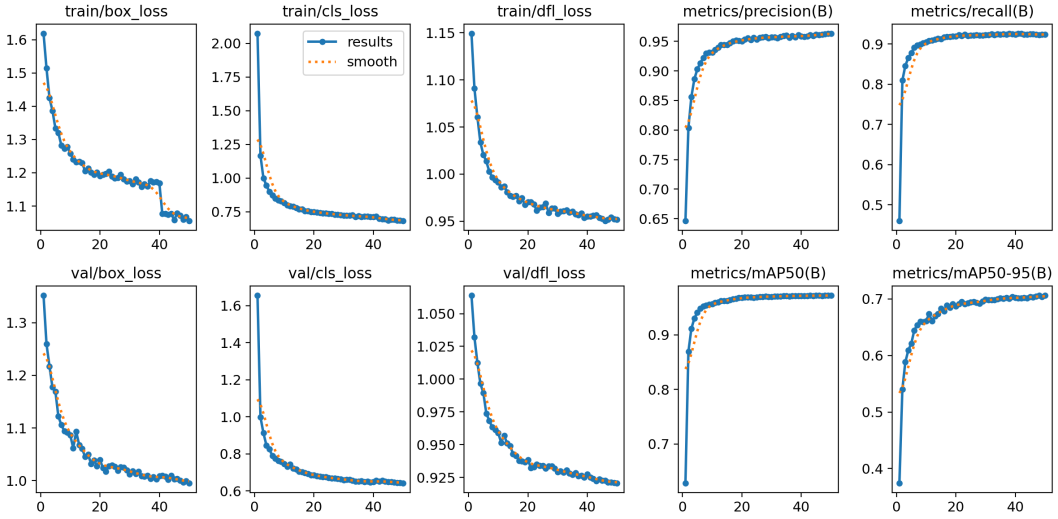

Figure 11: Freeze-21: SGD optimizer, LR = 0.001

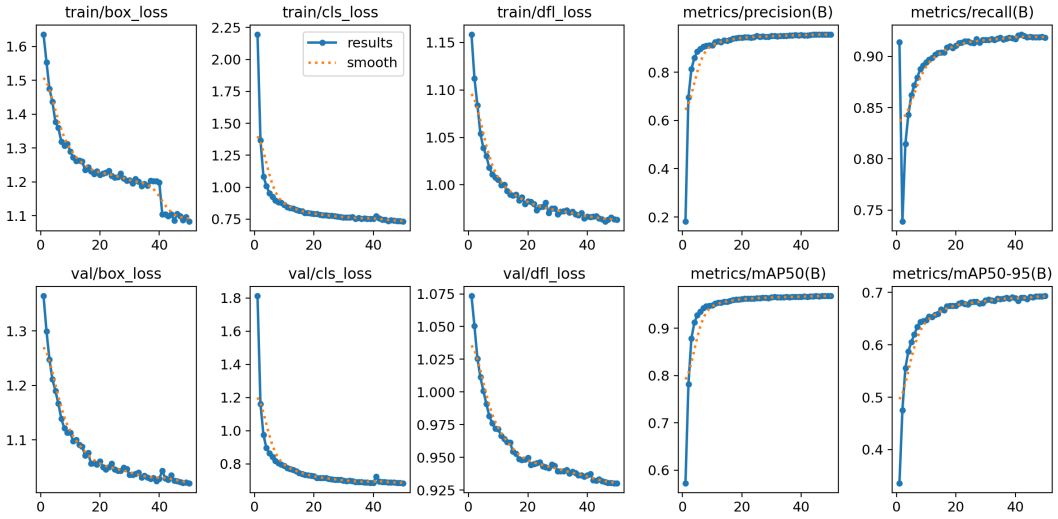

Figure 12: Freeze-21: SGD optimizer, LR = 0.0005

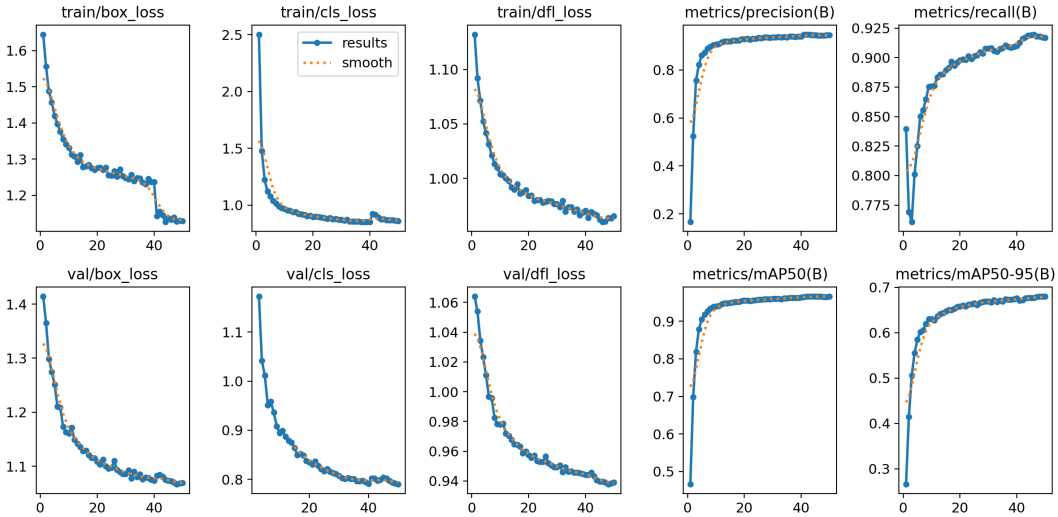

Figure 13: Freeze-21: Adam optimizer, LR = 0.0001

Model	**Precision**	**Recall**	**Box Loss**	**Cls Loss**	**Score**
yolov8n_adam_lr001_freeze21	0.97310	0.94205	0.97033	0.57483	0.6115
yolov8n_sgd_lr001_freeze21	0.96337	0.92341	0.99538	0.64271	0.5909
yolov8n_sgd_lr0005_freeze21	0.95652	0.91840	1.02073	0.68383	0.5795
yolov8n_adam_lr0001_freeze21	0.94560	0.91684	1.06930	0.79049	0.5589

Table 3: Freeze-21: Results from 4 training configurations.

2.6 Full Training Results

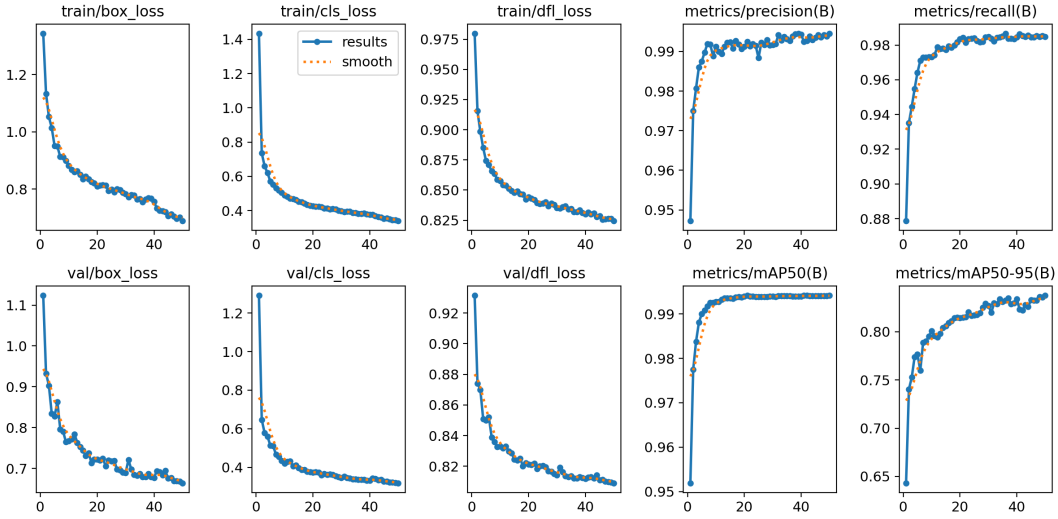

Figure 14: Full training: Adam optimizer, LR = 0.001 (Best Model)

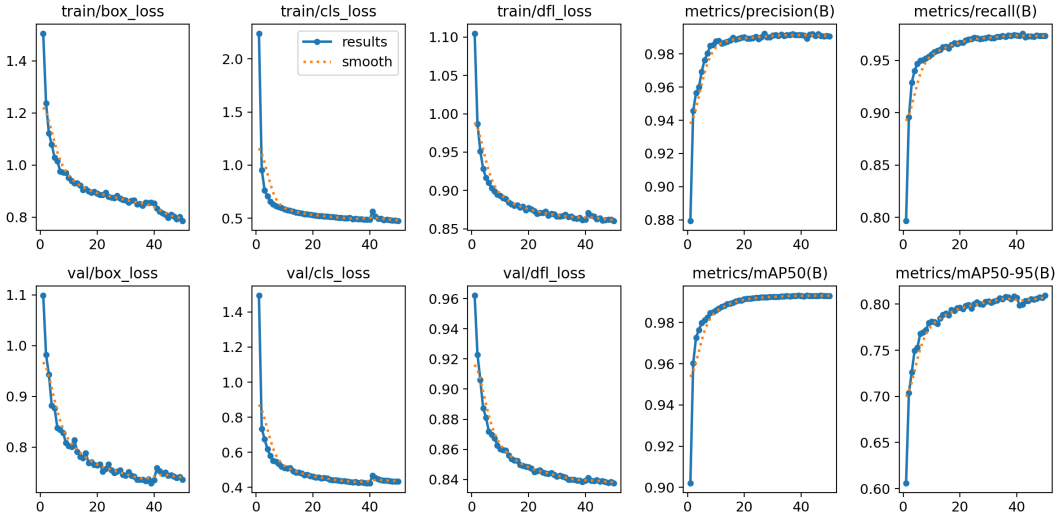

Figure 15: Full training: SGD optimizer, LR = 0.001

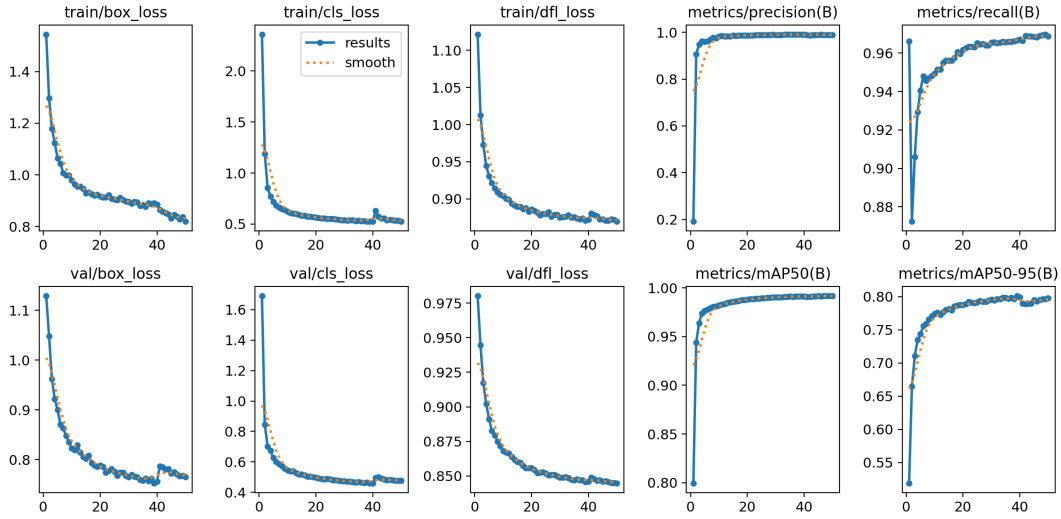

Figure 16: Full training: SGD optimizer, LR = 0.0005

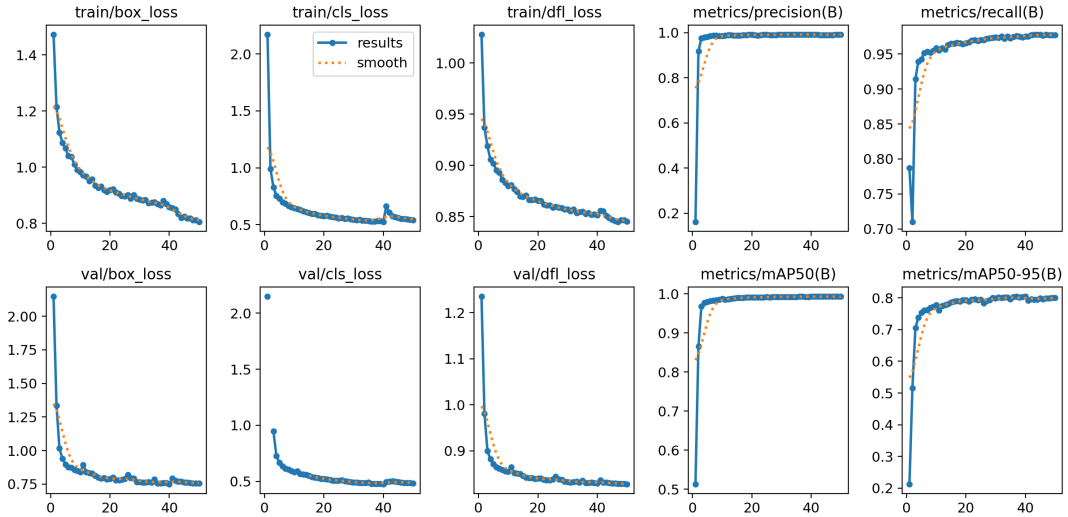

Figure 17: Full training: Adam optimizer, LR = 0.0001

Model	Precision	Recall	Box Loss	Cls Loss	Score
yolov8n_adam_lr001_fulltrain	0.99457	0.98478	0.66406	0.31819	0.6935
yolov8n_sgd_lr001_fulltrain	0.99068	0.97355	0.73563	0.43332	0.6688
yolov8n_sgd_lr0005_fulltrain	0.98949	0.96876	0.76497	0.47591	0.6592
yolov8n_adam_lr0001_fulltrain	0.99124	0.97690	0.75566	0.48494	0.6632

Table 4: Full training: Results from 4 training configurations.

2.7 Score Calculation Formula

To objectively compare models trained with different hyperparameters, we defined a custom scoring formula that balances detection quality and loss values:

$$\text{Score} = 0.4 \cdot \text{Precision} + 0.4 \cdot \text{Recall} - 0.1 \cdot \text{BoxLoss} - 0.1 \cdot \text{ClsLoss}$$

This score rewards higher precision and recall, while penalizing large loss values. The model with the highest score was selected as the best for each freezing strategy.

2.8 Hyperparameter Tuning Summary and Observations

The custom score formula enabled a fair and consistent comparison of models trained under different hyperparameter configurations. By balancing precision, recall, and loss terms, it emphasized models that were both accurate and stable during training.

Across all freezing strategies, models trained with the Adam optimizer and a learning rate of 0.001 consistently achieved the best scores. These configurations produced high precision and recall while maintaining relatively low box and classification losses. In contrast, models trained with SGD or lower learning rates generally showed slower convergence and higher loss values, which negatively impacted their overall score.

For each freeze level (5, 10, 21, and full training), four variants were compared, resulting in the selection of the following best-performing configurations:

- **Freeze 5:** `yolov8n_adam_lr001_freeze5`
- **Freeze 10:** `yolov8n_adam_lr001_freeze10`
- **Freeze 21:** `yolov8n_adam_lr001_freeze21`
- **Full Training:** `yolov8n_adam_lr001_fulltrain`

Interestingly, while freezing large portions of the network (such as in Freeze 21) reduces training time and computational requirements, it also limits the model's ability to adapt to new domains, leading to a noticeable performance drop. On the other hand, Freeze 5 demonstrated performance that was very close to full training, suggesting that freezing the early layers can still yield competitive results if the deeper layers are allowed to adapt.

Overall, the score metric proved effective in holistically quantifying performance. These results indicate that **appropriate hyperparameter tuning—particularly optimizer and learning rate—has a greater impact on model quality than the choice of freeze level alone**.

3 Evaluation Metrics and Implementation

3.1 Counting Evaluation (Exact Match Accuracy and MSE)

To evaluate the counting task, we compared the predicted number of cars with the ground-truth count in each image. Two metrics were used:

- **Exact Match Accuracy**: Percentage of images where the predicted count exactly matches the ground truth.
- **Mean Squared Error (MSE)**: Measures how far off predictions are from true values on average.

All predictions were filtered using a confidence threshold of 0.25 to remove low-confidence detections before counting.

3.2 Precision and Recall

Precision and recall were extracted from YOLO's `results.csv` logs. Precision measures the correctness of detections, while recall measures how many cars were successfully detected. These metrics were used as part of the custom scoring function.

3.3 Evaluation Code Details

Custom Python scripts were used to:

- Parse YOLO result files and extract relevant predictions
- Count predicted and ground-truth cars from `.txt` label files
- Calculate accuracy and MSE for each image and aggregate them
- Visualize predictions by overlaying bounding boxes on selected test images

The evaluation pipeline was implemented in a modular way to support batch evaluation of multiple trained models.

4 Results and Discussion

4.1 Comparison of Best Models Across Freezing Strategies

The table below compares the best model from each freezing strategy. These were selected using the custom scoring formula defined earlier.

Model	Precision	Recall	Box Loss	Cls Loss	Score
yolov8n_adam_lr001_fulltrain	0.99457	0.98478	0.66406	0.31819	0.6935
yolov8n_adam_lr001_freeze5	0.99285	0.98446	0.70407	0.34508	0.6860
yolov8n_adam_lr001_freeze10	0.99107	0.97886	0.73772	0.37254	0.6769
yolov8n_adam_lr001_freeze21	0.97310	0.94205	0.97033	0.57483	0.6115

Table 5: Comparison of best models across freezing strategies.

As shown in Table 5, the fully trained model achieved the highest score. However, the Freeze-5 variant performed very closely, suggesting that freezing only the earliest layers still allows the network to adapt effectively while potentially saving training time and resources. Performance steadily decreased as more layers were frozen (Freeze-10 and Freeze-21), indicating reduced learning capacity and adaptation to the target dataset.

4.2 Bounding Box Visualizations

Qualitative results are also important. Below are three sample outputs from the final best model:

Figure 18: Example 1: Correct bounding boxes and count.

Figure 19: Example 2: Another accurate prediction result.

Figure 20: Example 3: Missed detection due to partial visibility at the image edge.

The first two samples show clear and well-aligned predictions. In the third case, a single car was missed; however, it is important to note that this car is partially visible, with more than half of it outside the image frame. Such borderline cases are extremely difficult even for state-of-the-art models, and the fact that all fully visible cars were correctly detected shows that the model performs robustly. This example, therefore, does not indicate a major weakness, but rather highlights the strength of the model in handling dense and partially occluded scenes effectively.

4.3 Final Best Model and Its Accuracy

The final selected model was:

yolov8n_adam_lr001_fulltrain

Its performance on the validation set was:

- **Exact Match Accuracy:** 49.00%
- **Mean Squared Error (MSE):** 3.61

What does 49% exact match mean? This metric measures the percentage of images for which the predicted number of cars exactly matches the ground truth. Although 49% may appear low, it is actually reasonable for this task, as even a ± 1 error causes failure under this metric. Given the density and complexity of aerial car images, achieving nearly half exact matches demonstrates strong detection performance.
Why is MSE more informative? A lower MSE (3.61) means the model typically deviates from the correct count by only about ± 2 cars per image on average. This shows consistency and practical reliability, even when exact counts are slightly off.

4.4 Observations and Possible Improvements

Several key observations emerged from this assignment:

- Full training achieved the best overall performance, though Freeze-5 was a close runner-up.
- Adam optimizer with learning rate 0.001 was consistently the best choice.
- Exact count matching is a strict metric, and small differences in predictions can strongly impact accuracy.
- Results may improve further with:
  - Longer training durations
  - Better annotated labels or refined datasets
  - Post-processing steps (e.g., Non-Max Suppression tuning, ensemble models)

Overall, the results are strong given the dataset difficulty and model size (YOLOv8n), and the methodology used is scalable to more complex detection tasks.

5 Alternative Methods for Object Counting

Besides object detection-based approaches like YOLOv8, there are other effective techniques for object counting tasks. These methods vary in complexity, interpretability, and performance depending on the specific application.

Density Map Estimation

Instead of detecting each object, this method uses convolutional neural networks to predict a density map over the image. The total sum of the density map represents the object count. It is especially effective in crowded or occluded scenes, and is widely used in crowd counting [3].

Regression-Based Counting

In this approach, a CNN is trained to directly predict the number of objects in an image. While it doesn't provide localization, it can be computationally efficient and useful when object locations are not necessary [4].

Segmentation with Connected Component Analysis

Here, the image is segmented to separate foreground objects from the background, followed by counting the number of connected regions. This method is suitable for applications with clear and separated objects [5].

Transformer-Based Counting

Newer methods utilize transformers, such as DETR (Detection Transformer), for end-to-end detection and counting. These models are capable of handling complex scenes with variable object sizes and occlusions [6].

Each technique has trade-offs in accuracy, interpretability, and performance, and the choice depends on the specific problem domain and dataset characteristics.

6 Conclusion

In this assignment, we explored the object detection-based approach to count cars in aerial images using YOLOv8. We prepared and formatted the dataset for YOLOv8 training, experimented with various freezing strategies and hyperparameters, and evaluated models using multiple metrics.

We compared the impact of optimizer choice, learning rate, and freezing strategies on detection accuracy and counting performance. A custom scoring function allowed us to systematically select the best model across all configurations. The final best model, `yolov8n_adam_lr001_fulltrain`, achieved strong precision and recall, with a 49% exact match accuracy and a low mean squared error of 3.61.

Through this process, I gained practical experience in deep learning model evaluation, training strategy design, and performance analysis on real-world image data. This assignment also helped me better understand the strengths and limitations of object detection models in high-density scenarios.

7 References

References

[1] Ultralytics YOLOv8 Documentation, `https://docs.ultralytics.com`

[2] YOLOv8 GitHub Repository, `https://github.com/ultralytics/ultralytics`

[3] Medium Blog on Density Maps, `https://medium.com/towards-data-science/objects-counting-by-estimating-a-density-map-with-convolutional-neural-networks-c01086f3b3ec`

[4] Crowd Counting using Deep Learning - Analytics Vidhya, `https://www.analyticsvidhya.com/blog/2021/06/crowd-counting-using-deep-learning/`

[5] Connected Component Analysis - Scaler Topics, `https://www.scaler.com/topics/connected-component-analysis-in-image-processing/`

[6] Vision Transformers for Object Detection - Hugging Face, `https://huggingface.co/learn/computer-vision-course/en/unit3/vision-transformers/vision-transformer-for-objection-detection`