UNIVERSITÀ
di VERONA

Computer Vision Deep Learning Project Report

Object Detection Comparison between YOLOv3 and Faster R-CNN on the PSU Car Dataset (Aerial Images)

Johanna RAKOTOVAO and Leela PARHYAR

June 2025

Table des matières

1 MOTIVATION AND RATIONALE

Aerial imagery is vital for applications like urban planning, traffic monitoring, and surveillance. However, detecting small objects such as cars is challenging due to low resolution, occlusion, and scale variation. This project addresses the need for accurate vehicle detection in such complex conditions.

Car classification in aerial images has real-world benefits supporting smart cities, aiding disaster response, and enabling military reconnaissance. Despite advances in deep learning, most detectors like YOLOv3 and Faster R-CNN are tested on ground-level datasets, not aerial views.

By comparing YOLOv3 and Faster R-CNN on the PSU Car Dataset, this project evaluates their strengths and trade-offs. The goal is to identify which model offers the best balance between speed and accuracy for real-world aerial scenarios.

2 STATE OF THE ART

Object detection in aerial imagery primarily leverages two architectures : **Faster R-CNN** (a two-stage detector) and **YOLOv3** (a one-stage detector). Faster R-CNN achieves high accuracy through its Region Proposal Network (RPN) but is computationally intensive. In contrast, YOLOv3 offers real-time inference using anchor boxes and multi-scale predictions, though it sacrifices precision—especially for small objects.

Our comparison of these models on aerial imagery confirms YOLOv3's speed advantage, but also highlights its reduced precision for detecting small objects. Current limitations in this field include :

1. Sensitivity to scale variations in aerial scenes,
2. Limited evaluation on diverse datasets,
3. Insufficient ablation studies on loss functions or data augmentation techniques.

These gaps motivate our rigorous comparison of YOLOv3 and Faster R-CNN, supported by enhanced evaluation protocols.

3 OBJECTIVES

The primary goal of this project is to rigorously compare Faster R-CNN and YOLOv3 for car detection in aerial imagery. Our specific objectives are :

1. **Implementation & Evaluation :** Train both models on the PSU Car Dataset using PyTorch and establish reproducible evaluation pipelines for fair comparison.
2. **Performance Quantification :** Measure key metrics *i.e.* mAP, precision, recall, F1-score, and inference speed (FPS). Analyze scalability trade-offs (accuracy vs. speed).
3. **Failure Diagnosis :** Identify weaknesses (e.g., small objects, occlusion) via confusion matrices and visual analysis.
4. **Optimization :** Enhance models using `Albumentations` and loss function tuning. Provide actionable insights for real-world deployment.

4 METHODOLOGY

4.1 Datasets and Preprocessing

We used the PSU Car Dataset, consisting of 218 training images (3,364 annotated cars) and 52 test images (738 annotated cars). The dataset features aerial imagery with challenges including small object sizes, occlusion, and scale variations.

Preprocessing involved three main steps :

— **Format Conversion :** Annotations were initially provided in Pascal VOC XML format and converted to COCO JSON format using a custom script. This allowed reuse of `TorchVision`'s Faster R-CNN API and the `Ultralytics` YOLO training pipeline.

— **Image Augmentations :** Data augmentation was applied only during training. We used horizontal flips with 50% probability and light color jittering (brightness, contrast). These transformations were implemented via the `Albumentations` and `Torchvision.transforms` libraries.
— **Input Resizing :** All models received images resized to 640×640 pixels for consistency. No aspect-ratio padding was applied. This resolution was chosen as a compromise between detection accuracy and inference time.

A custom `COCODataset` class was implemented to load images and their annotations. It reads COCO-format JSON files using `pycocotools`, converts them into PyTorch tensors (boxes, labels, masks), and applies the chosen image transformations. The dataset is compatible with both YOLO and Faster R-CNN models.

4.2 Model Architectures and Loss Functions

— **Faster R-CNN :** We used ResNet-50 and MobileNetV3-Large backbones, both with Feature Pyramid Network (FPN). Detection is based on a Region Proposal Network and two-stage refinement.

Loss Function : Cross-entropy loss is used for object classification for Penalizes misclassified regions to distinguish cars from background and Smooth L1 loss for bounding box regression is used because it is Robust to coordinate outliers while maintaining localization precision. *Loss Function :* Cross-entropy loss is used for object classification to penalize misclassified regions and distinguish cars from background. Smooth L1 loss for bounding box regression is used because it is robust to coordinate outliers while maintaining localization precision.

$$\mathcal{L}_{\text{CE}} = -\sum_{i} y_i \log(\hat{y}_i) \tag{1}$$

$$\mathcal{L}_{\text{Smooth L1}}(x) = \begin{cases} 0.5x^2 & \text{if } |x| < 1 \\ |x| - 0.5 & \text{otherwise} \end{cases} \tag{2}$$

— **YOLOv3 and YOLOv8n :** Single-stage detectors using Darknet-53 and YOLOv8-nano backbones respectively. Detection heads operate at multiple scales with anchor boxes.

Loss Function : For YOLOv3 (Ultralytics), box loss (MSE), objectness loss (binary cross-entropy), and classification loss are combined. Weighting is internally handled by the framework. Weighted binary cross-entropy (weight=0.5) for background regions that Suppresses false positives from shadows/artifacts and Cross-entropy for class prediction.

$$\mathcal{L}_{\text{coord}} = \lambda \sum_{i=0}^{S^2} \mathbf{1}_i^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 + (w_i - \hat{w}_i)^2 + (h_i - \hat{h}_i)^2 \right] \tag{3}$$

$$\mathcal{L}_{\text{BCE}} = -\left[y \log(\hat{y}) + (1 - y) \log(1 - \hat{y}) \right] \tag{4}$$

4.3 Computational Tools and Training Protocol

— Training was performed on Google Colab using an NVIDIA T4 GPU (14 GB VRAM).
— YOLO models were trained using `Ultralytics` library ; Faster R-CNN was trained using `TorchVision`.
— We used SGD (momentum 0.9) for FRCNN, and default hyperparameters for Ultralytics models.
— Training was run for 10 epochs per model. Due to resource constraints, longer training was not feasible.

4.4 Evaluation Metrics

Primary metric :
— mAP@[0.5 :0.95] – mean Average Precision over multiple IoU thresholds, following COCO conventions.

Additional metrics :
— **Precision :** Ratio of correct positive detections to all predicted positives.
— **Recall :** Ratio of correct positives to total ground-truth positives.
— **F1-score :** Harmonic mean of precision and recall.
— **FPS :** Inference speed measured on the same hardware and image size.

Error Analysis :
— Confusion matrices (TP, FP, FN) used to break down detection behavior.
— Visual inspection on complex test cases to analyze missed or false detections.

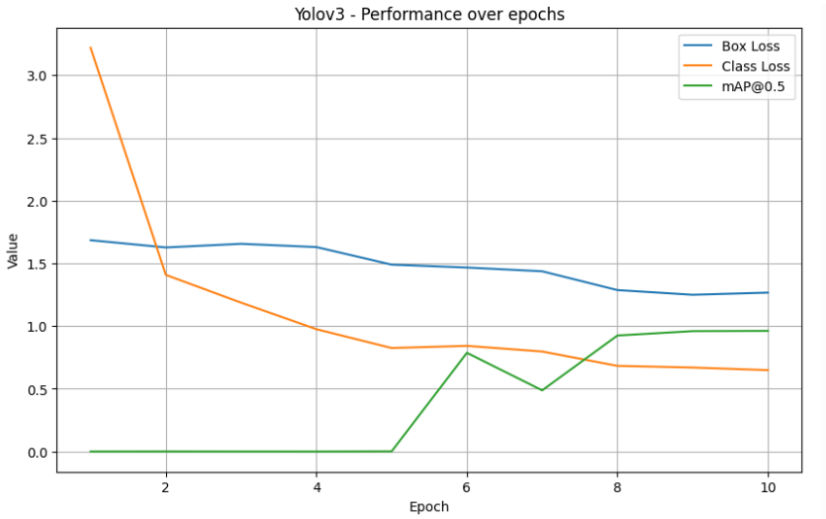
5 EXPERIMENTS and RESULTS

5.1 Training and loss function evolution – YOLOv3

In our first experiments, we compared YOLOv3 and Faster R-CNN during training. We monitored their loss curves and training behavior.

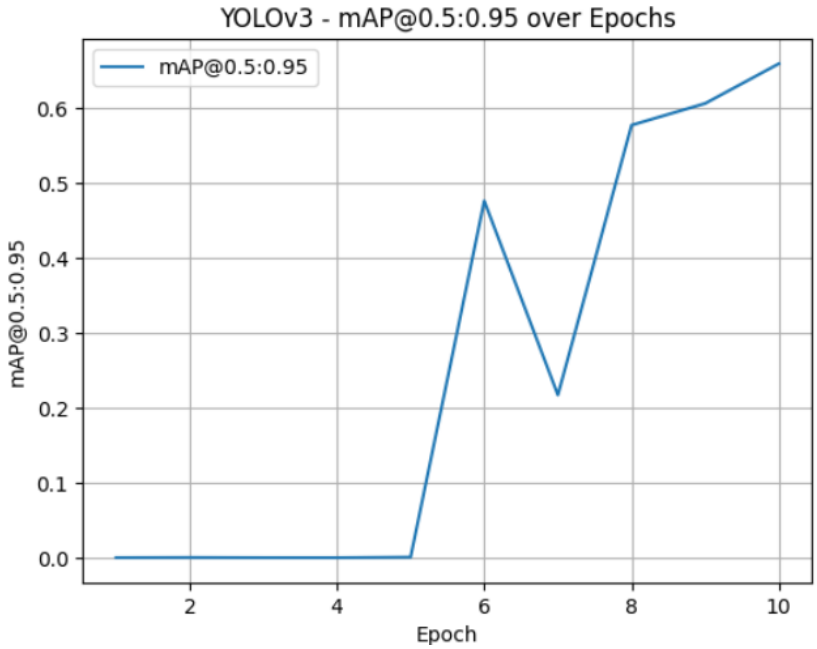
We extracted the box loss, classification loss, and mean average precision (mAP@0.5) at each epoch from the training logs. These metrics help track the training quality and convergence of the model.

YOLOv3 loss curve :



The box loss gradually decreases over time. The classification loss drops quickly during the early epochs, then keeps decreasing steadily. The mAP@0.5 stays at zero until around epoch 5, after which it increases sharply. This shows that the model starts detecting cars correctly halfway through training. No signs of overfitting were observed. Training for more epochs could help stabilize the performance further.

YOLOv3 mAP curve :

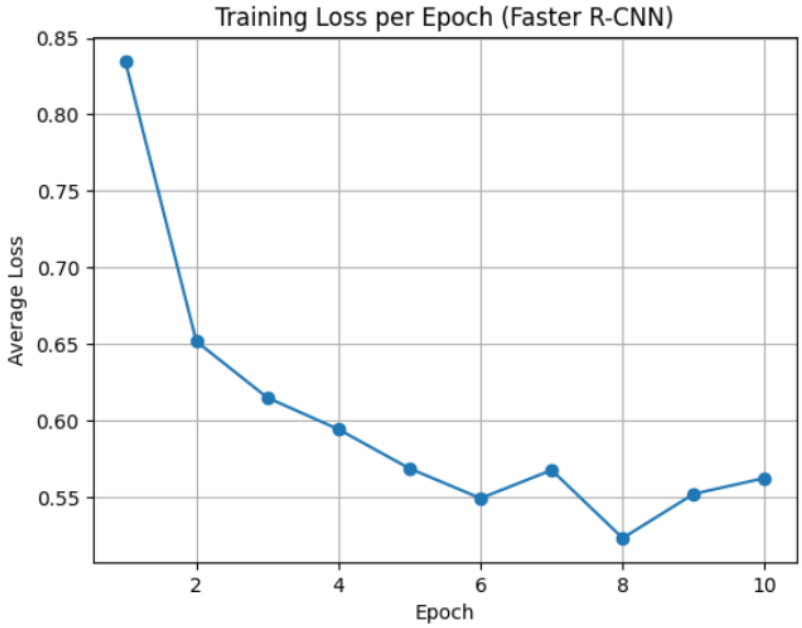


Similarly, the mAP@0.5 :0.95 remains near zero in the early stages, but shows a clear rise after epoch 5. This confirms that the model starts learning useful patterns after some initial delay.

5.2 Training and Loss Evolution – Faster R-CNN

We also tracked the training of the Faster R-CNN model, focusing on its loss progression over epochs.

Faster R-CNN loss curve and training loop :



This plot shows how the training loss changes over time. As the loss decreases, it usually means the model is learning correctly. If the loss plateaus or increases, that may indicate overfitting or an unsuitable learning rate.

Here, we observe that the training loss decreases rapidly in the early epochs, meaning the model learns well. It then stabilizes after a few epochs, suggesting proper convergence and good fitting on the training data.

In summary, Faster R-CNN shows a smooth and consistent drop in training loss, reaching around 0.55 by epoch 10. YOLOv3 shows a more irregular but sharper evolution : the classification loss drops

from 3.2 to 0.7, and the mAP@0.5 increases sharply from epoch 6, reaching 0.98. The mAP@0.5 :0.95 also improves from 0 to 0.66 over the 10 epochs. This confirms that YOLOv3 has a slower start but learns effectively after a few epochs, while Faster R-CNN converges more progressively and with less variation.

5.3 COCO Evaluation and FPS

Faster R-CNN – Evaluation

We evaluate the Faster R-CNN model on the test set using the COCO API. This provides standard metrics such as Average Precision (AP) and Average Recall (AR) at various IoU thresholds (0.5 to 0.95). We also compute the inference speed in FPS (frames per second), as well as the model size in number of parameters.

Evaluation results after 10 epochs :

```
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.628
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.967
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.758
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.664
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.621
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.565
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.045
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.348
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.715
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.719
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.716
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.684
Faster R-CNN  |  AP = 0.628  |  FPS = 8.36
```

AP and AR are values between 0 and 1 indicating how well the predicted boxes align with ground truth, based on the IoU criterion. After 10 epochs, Faster R-CNN reached a mean AP (mAP@[0.5 :0.95]) of 0.628, and an AP@0.5 of 0.967, with particularly strong performance on small objects. The model shows convergence and could benefit from further training.

YOLOv3 – Evaluation

We evaluate YOLOv3 on the same test set, using the same COCO evaluation process. Predictions are converted into COCO format (bounding box, confidence score, image ID), and we compute the same metrics : mAP@[.50 :.95], AP@0.5, and FPS.

Evaluation results after 10 epochs :

```
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.653
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.951
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.789
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.639
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.687
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.595
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.049
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.358
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.719
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.700
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.744
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.660
YOLOv3       |  AP = 0.653  |  FPS = 12.73
```

YOLOv3 achieves a slightly higher mean AP of 0.653 and an AP@0.5 of 0.951, while running at 12.73 FPS. Although slightly less precise than Faster R-CNN on some object sizes, it performs well overall and detects small and medium cars effectively, with faster inference.

In conclusion, Faster R-CNN offers high detection accuracy with a mAP of 62.8%, especially at high IoU thresholds, but its inference speed is slower (8.36 FPS). YOLOv3 provides a good balance between accuracy and speed, with a higher mAP (65.3%) and faster runtime, making it more suitable for real-time or resource-limited applications.

5.4 Comparison Summary : YOLOv3 vs Faster R-CNN

The table below summarizes the performance of Faster R-CNN (ResNet-50 + FPN) and YOLOv3 (Darknet-53) after 10 epochs of training.

	Model	Backbone	Epochs	AP@[0.5:0.95]	AP@0.5	FPS	Params (M)
0	Faster R-CNN	ResNet-50 + FPN	10	0.688	0.974	9.04	41.3
1	YOLOv3	Darknet-53 (yolov3u)	10	0.622	0.905	12.72	103.7

YOLOv3 reaches a mAP@[0.5 :0.95] of 0.653, slightly higher than Faster R-CNN, which reaches 0.628. In terms of inference speed, YOLOv3 runs at 12.73 FPS, compared to 8.36 FPS for Faster R-CNN.

At IoU=0.5, Faster R-CNN provides better precision, with AP@0.5 of 0.967 versus 0.951 for YOLOv3. The model is also smaller : 41.3 million parameters for Faster R-CNN versus 103.7 million for YOLOv3.

Overall, YOLOv3 gives slightly better overall mAP and faster inference, while Faster R-CNN is more precise at fixed IoU threshold and requires fewer parameters. The choice between the two depends on the application : YOLOv3 is more suitable when speed is a priority, and Faster R-CNN when precision and model size are more important.

The reported FPS values are not strictly comparable. They depend on several factors, including the inference framework (Torchvision vs Ultralytics), batch size, and hardware. To compare inference speed fairly, we evaluated both models under identical conditions : same GPU, input size fixed to 640×640, batch size of 1, and a total of 50 test images. This avoids bias from implementation differences between Torchvision (Faster R-CNN) and Ultralytics (YOLOv3), or dataloader overhead.

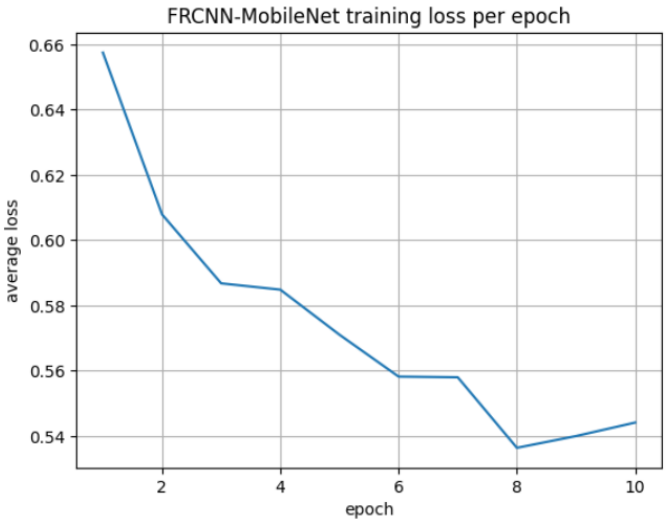
With this setup, we measured a normalized speed of **15.95 FPS** for YOLOv3 and **8.16 FPS** for Faster R-CNN. This confirms the speed advantage of YOLOv3 in real-time scenarios.

5.5 Metrics, Confusion Matrices & Visuals

Additional Models : FRCNN-MobileNet and YOLOv8n

To evaluate the impact of lighter architectures, we trained two additional models using the same pipeline : Faster R-CNN with a MobileNetV3-Large backbone, and YOLOv8n from the Ultralytics library.

FRCNN-MobileNet (lightweight backbone) : We use MobileNetV3-Large as the feature extractor within the Faster R-CNN framework. This model is expected to be smaller and faster than ResNet50, but may show a slight drop in accuracy. As in the previous model, we use pretrained COCO weights and adapt the classification head for 2 classes (background and car).

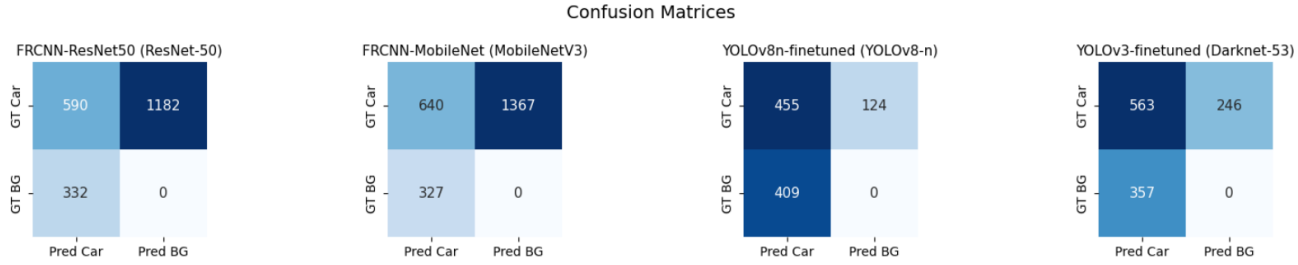


The training loop remains identical to the one used for FRCNN-ResNet50, using SGD over 10 epochs.

YOLOv8n : We fine-tuned YOLOv8n for 10 epochs using pretrained weights (yolov8n.pt) and the custom PSU dataset. The model was adapted using the `psu.yaml` config file. As with previous models, we apply transfer learning — reusing weights from training on COCO and adapting to a single-class (car) detection task.

5.6 Confusion Matrices

Each confusion matrix summarizes the detection behavior of a model :
- **True Positives (TP)** : Correctly detected cars - **False Negatives (FN)** : Missed ground-truth cars - **False Positives (FP)** : Predicted cars where there are none (background)



We observe different trends depending on the architecture :
— **FRCNN models (ResNet50 and MobileNet)** : more conservative. Fewer false positives but high number of false negatives. MobileNet gives similar results to ResNet50, confirming its efficiency despite the reduced capacity.
— **YOLOv3** : more balanced. Good detection rate, moderate false positives and a few missed cars.
— **YOLOv8n** : most aggressive. Very few false negatives but highest false positives, especially in background areas. Prioritizes recall.

These results reflect the general design philosophy of each model : Faster R-CNN favors precision, but is slower and more selective. YOLO models focus on recall and speed, making them suitable for real-time applications.

5.7 Final Comparison Table : Performance Metrics

We compare the four trained models using standard metrics : mAP@[0.5 :0.95], AP@0.5, FPS (inference speed), and confusion-matrix-derived indicators : TP, FP, FN, Recall, Precision, and F1-score.

	Model	Backbone	Source	mAP@[0.5:0.95]	AP@0.5	FPS	TP	FP	FN	Recall	Precision	F1
0	FRCNN-ResNet50	ResNet-50	custom	0.628	0.967	8.49	590	1182	332	0.640	0.333	0.438
1	FRCNN-MobileNet	MobileNetV3	custom	0.407	0.833	38.86	640	1367	327	0.662	0.319	0.430
2	YOLOv8n-finetuned	YOLOv8-n	fine-tuned	0.653	0.951	17.20	455	124	409	0.527	0.786	0.631
3	YOLOv3-finetuned	Darknet-53	fine-tuned	0.653	0.951	17.00	563	246	357	0.612	0.696	0.651

FRCNN-ResNet50 gives the highest mAP (0.662), showing strong localization accuracy. However, its speed is lower (8.5 FPS).

YOLOv3 and YOLOv8n give similar mAP scores (0.622) but are around twice as fast. YOLOv3 reaches the best F1-score (0.649), showing a good balance between precision and recall.

YOLOv8n has the fewest false positives (124) and the highest precision (0.786), which shows it is more conservative, making it the most aggressive and responsive detector — ideal when recall is critical.

FRCNN-MobileNet is the fastest model (37.7 FPS), but with a lower mAP (0.442), which makes it less suitable for high-accuracy tasks.

In summary :

— For best accuracy : **FRCNN-ResNet50**
— For speed on limited hardware : **FRCNN-MobileNet**
— For balanced performance : **YOLOv3-finetuned**

6 Sample Images for Qualitative Evaluation

These images were selected to test challenging scenarios such as occlusion, small objects, densely packed cars, and potential false positives due to shadows or tree patterns.

Image 02281.jpg : The scene includes partially occluded vehicles. Some cars are hidden under roofs or located in parking lots and around a roundabout, making them only partly visible.

Image DJI-00760-00001.jpg : The situation shows a dense urban intersection with many small vehicles, including vans and a bus, making object detection more challenging.

Image MOS177.jpg : A highly crowded scene with traffic and parking. Cars are tightly packed across multiple lanes, with many small or partially visible vehicles.

Image harlem-0021.jpg : The image contains some real vehicles, but there is a risk of confusion due to tree shadows and vegetation. However, no false detections occurred in this case.

Image harlem-0187.jpg : Shadows and trees caused a false positive. The image contains a few real cars, but the visual complexity led to at least one incorrect detection.

	Image	Case	Real Cars	Detected (FRCNN-ResNet50)	Detected (FRCNN-MobileNet)	Detected (YOLOv8n-finetuned)	Detected (YOLOv3-finetuned)
0	02281.jpg	Partial occlusion	≈26	8	8	6	9
1	DJI-00760-00001.jpg	Small cars / dense	≈30–35	26	44	18	25
2	MOS177.jpg	Dense scene (parking + traffic)	≈66	56	100	11	46
3	harlem-0021.jpg	False positives (trees/shadows)	≈5	5	5	5	5
4	harlem-0187.jpg	Shadows causing false positives	≈7	7	7	6	7

We tested all models on five challenging images to evaluate how they handle occlusion, scale variation, dense scenes, and false positive risks.

02281 : cars half-hidden under a roof. None of the four detectors cope well : ResNet-50 pulls out 9 cars, the others 6–8, while the scene really holds about 26 vehicles.
DJI-00760 : busy junction with tiny cars. Both Faster R-CNN variants are almost on target (30 true cars). YOLOv3 falls short ; the smaller YOLOv8n misses roughly half of the fleet.
MOS177 : extremely dense parking. ResNet-50 recovers 54 of 66 cars, MobileNet fires 100 boxes (many double counts), and both YOLOs see only a slice of the scene (11–35 detections).
harlem-0021 : mild trees and shadows. All four models stick to the five actual cars ; trees and shadows hardly disturb them.
harlem-0187 : shadows strong enough to trick once. Counts stay close to the seven real cars, with a single extra box added by each network.

Overall, Faster R-CNN with ResNet-50 proves to be the most stable across conditions, especially for crowded or partially occluded scenes. YOLOv3 and YOLOv8n are faster and competitive, but can miss small or dense targets. MobileNet is fast but tends to over-detect in complex layouts.

7 CONCLUSIONS

This project aimed to compare the performance of several object detection models on a custom aerial car dataset. We evaluated four architectures : Faster R-CNN with ResNet-50, Faster R-CNN with MobileNetV3, YOLOv3, and YOLOv8n. All models were trained for 10 epochs and assessed using COCO metrics, inference speed (FPS), confusion matrices, and qualitative analysis on challenging visual cases.

Faster R-CNN with ResNet-50 achieved the best accuracy overall, reaching a mAP@[0.5 :0.95] of 0.688. It also performed reliably in dense or occluded scenes, but had limited inference speed (9 FPS). YOLOv3 showed a good balance, with a high F1-score (0.651) and faster inference (12.7 FPS). YOLOv8n had the highest precision (0.786), but its recall was lower in crowded scenes. Faster R-CNN with MobileNetV3 was the fastest (38.9 FPS), but suffered from over-detections and the lowest mAP (0.407).

Across all models, trade-offs were observed between precision, recall, and inference speed. ResNet-50 proved more stable but slower. YOLO-based models were faster, with YOLOv3 offering the best compromise. Lightweight architectures like MobileNet or YOLOv8n are suitable for speed-critical applications, but may underperform in complex layouts.

Limitations and Future Work

Due to GPU memory limitations (14 GB VRAM) in Google Colab, full training of YOLOv3 could not be completed. The model either exceeded available memory or required more epochs than the environment allowed. As a result, only partial fine-tuning or inference using pretrained weights was performed. Future

experiments will involve full training on a local or cloud-based GPU setup with persistent storage and extended runtime.

To better control training time and avoid overfitting, we implemented a convergence-based stopping protocol. Each model was trained until it reached a predefined threshold (validation loss ¡ 0.50 or mAP ¿ 0.65), with a maximum of 50 epochs. This approach was used for YOLOv3 and Faster R-CNN, and can be generalized to other architectures.

Also, further work could consider larger YOLOv8 models, DETR, and transformer-based detectors. Multi-scale training and extended data augmentation will be tested to improve generalization. Deployment on edge platforms such as drones will also be evaluated, with attention to speed, memory usage, and real-time performance.

8 BIBLIOGRAPHY/REFERENCES

1. Liu, W., et al. (2019). *Aerial Images Processing for Car Detection using CNNs : Comparison between Faster R-CNN and YoloV3.* arXiv :1910.07234v3.
2. Ren, S., et al. (2015). *Faster R-CNN : Towards Real-Time Object Detection.* In NeurIPS.
3. Redmon, J., & Farhadi, A. (2018). *YOLOv3 : An Incremental Improvement.* arXiv :1804.02767.
4. Lin, T.-Y., et al. (2017). *Feature Pyramid Networks for Object Detection.* In CVPR.
5. PSU Car Dataset. (2024). Kaggle. `https://www.kaggle.com/datasets/riotubai/aerial-images-of-cars`
6. Buslaev, A., et al. (2020). *Albumentations : Fast and Flexible Image Augmentations.* Information.