Deep Learning-Based Parking Detection System Using CCTV Footage

Introduction

Finding parking in congested areas often leads to increased traffic, fuel wastage, and frustration among drivers. Traditional parking management systems frequently depend on manual monitoring or costly sensors, which can be difficult to maintain. This project introduces a novel deep learning-based solution that analyzes top-view CCTV footage in real-time to accurately detect vacant and occupied parking spaces. By using convolutional neural networks (CNNs) and advanced object detection techniques, the system dynamically updates parking availability status, which can significantly enhance convenience for drivers. This approach is not only cost-effective and efficient but it can also handle varying lighting, weather conditions, and occlusions, which offers practical improvements over existing parking detection methods.

Objective

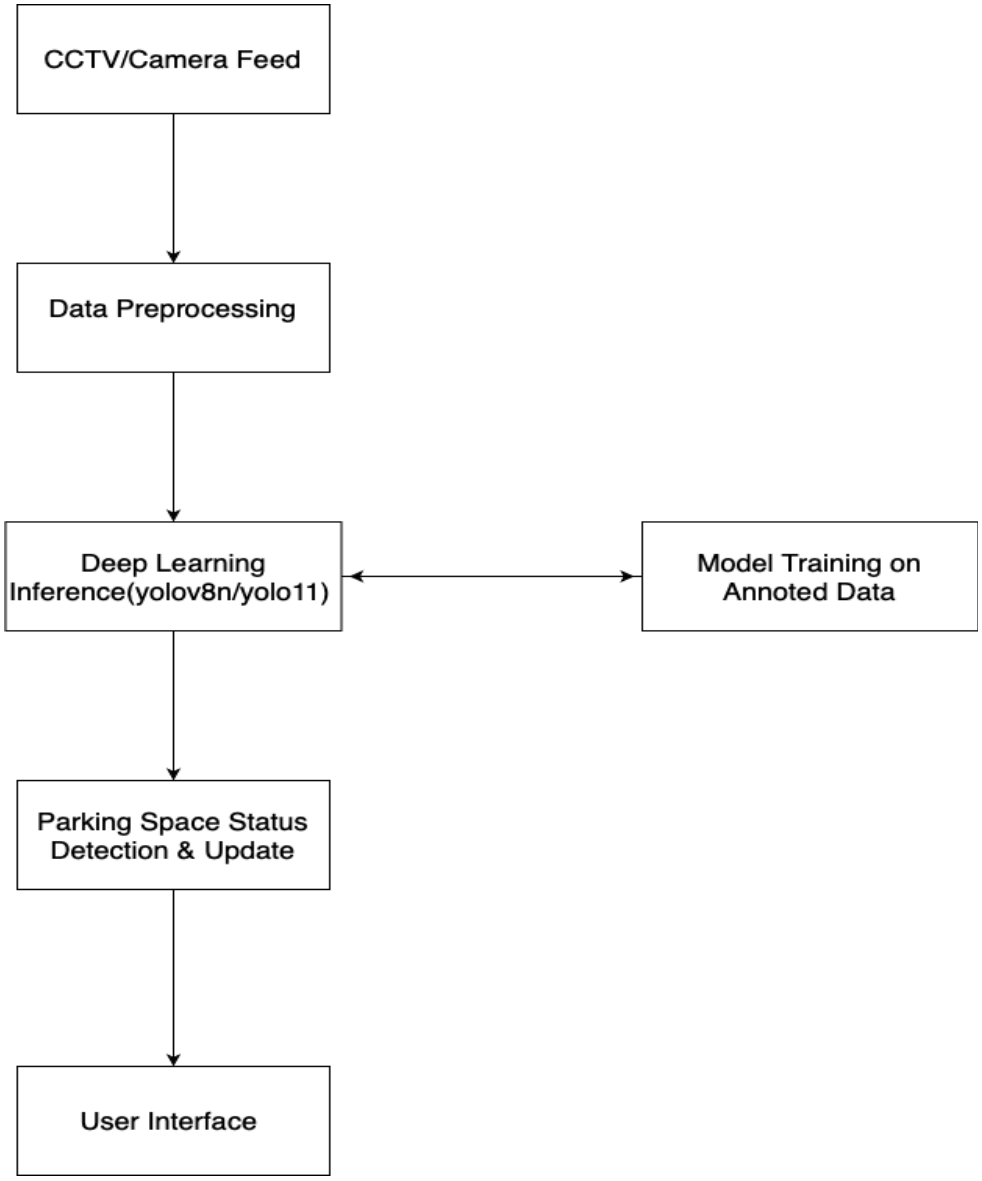
Develop Deep Learning Model: Create a model that accurately identifies vacant and occupied parking spaces from top-view CCTV footage.

Real Time Inference: This project can be used for real time inference by using CCTV recording to update the availability of the spaces dynamically.

User Interface: For now, we are not creating any interface but for the final submission we will deploy this model. This can be further extended to an app where users can check the availability in real time.

Reliability: We are trying to ensure that model can perform reliably in any weather or lighting condition.

Architecture



The architecture begins with capturing CCTV footage that is then preprocessed to standardize input conditions. The deep learning model is trained using labeled data and then deployed for real-time inference. The inference results can be used to update a user-friendly interface that displays the current parking status.

YoLO - Theory and evolution

YOLO is a real-time object detection algorithm that processes images in a single forward pass through a neural network, making it significantly faster than earlier approaches. The algorithm divides an input image into a grid (typically S×S cells), and each grid cell predicts B bounding boxes along with confidence scores and class probabilities. This approach treats object detection as a regression problem rather than using sliding windows or region proposal methods. The network architecture consists of an initial feature extraction backbone (often a CNN like DarkNet), followed by several convolutional layers that simultaneously predict bounding box coordinates, objectness scores, and class probabilities.

What makes YOLO particularly efficient is its unified architecture that directly maps image pixels to bounding box coordinates and class probabilities in one evaluation. During training, YOLO optimizes a multi-part loss function that considers localization error (bounding box coordinates), classification error (class predictions), and confidence error (objectness scores). The algorithm has evolved through several versions (YOLOv1 through YOLOv8+), with each iteration improving accuracy, speed, or both through architectural innovations like anchor boxes, feature pyramid networks, and more sophisticated backbones.

The Intersection over Union (IoU) is a critical metric in YOLO algorithms that measures how well predicted bounding boxes align with ground truth boxes. Here's how it works and why it's important:

How IoU Works in YoLO

IoU calculates the ratio between the intersection and union of two bounding boxes: the predicted box and the ground truth box. The formula is:

IoU = Area of Intersection / Area of Union

The result is a value between 0 and 1, where:

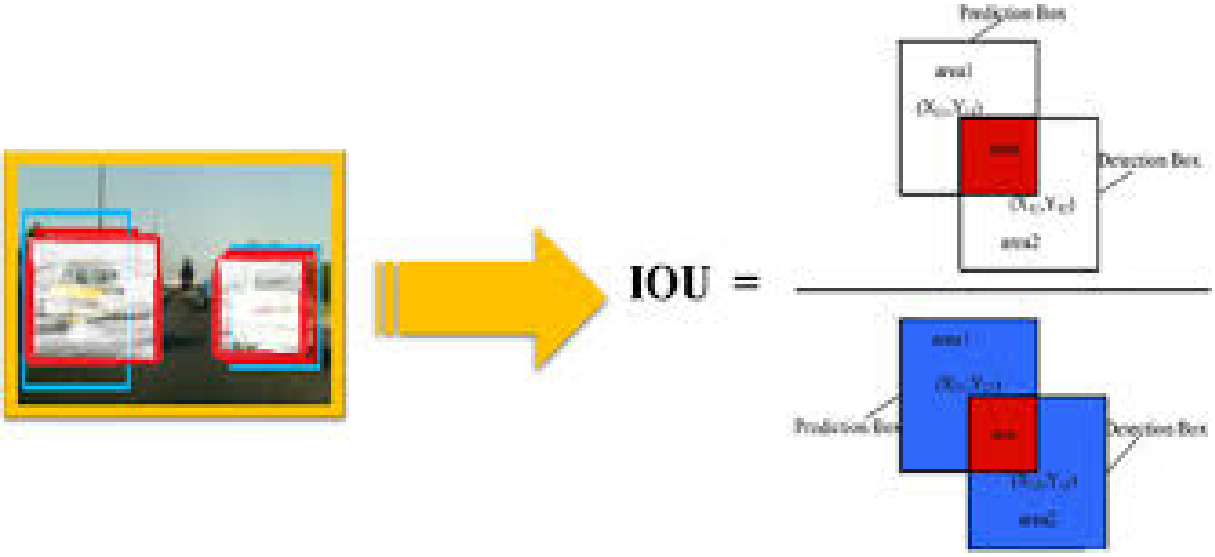
- IoU = 1 means perfect overlap (prediction exactly matches ground truth)
- IoU = 0 means no overlap at all
- Values between 0 and 1 indicate partial overlap

In YOLO algorithms, IoU serves multiple crucial functions:

1. **During training**: IoU helps determine which predicted boxes are responsible for detecting which ground truth objects. YOLO assigns a predicted box to be responsible for detecting an object if its IoU with the ground truth is highest among all predictions.
2. **In loss calculation**: Modern YOLO variants (starting with YOLOv3) incorporate IoU directly into their loss functions. For example, CIoU (Complete IoU) loss considers not just overlap but also center point distance, aspect ratio, and box size.
3. **For Non-Maximum Suppression (NMS)**: After detection, YOLO algorithms use IoU to eliminate redundant predictions through NMS. If multiple predicted boxes have high IoU with each other, the algorithm keeps only the one with the highest confidence score.
4. **In model evaluation**: IoU thresholds (commonly 0.5 or 0.75) determine whether a detection is considered correct when calculating precision, recall, and mAP (mean Average Precision).

As YOLO evolved, the IoU concept has been enhanced. YOLOv8 uses DIoU (Distance IoU) for NMS, which considers the distance between box centers. YOLOv11 implements a more sophisticated IoU-aware attention mechanism that weights feature importance based on IoU predictions during training.

The evolution of how IoU is used represents a key factor in the improved accuracy of recent YOLO versions, especially for overlapping objects and objects at different scales. The following figure below illustrates how IOU is visually computed.



Methodology

Dataset

We have made use of the PK-Lot dataset(https://gts.ai/dataset-download/pklot-dataset/)

Preprocessing

Frame Extraction: For custom video feeds, individual frames are extracted from continuous CCTV footage.

Normalization and Resizing: Images are resized and normalized to standardize input dimensions and intensity values. These steps can be manually implemented, but for now they are handled by the YOLOv8 framework during training.

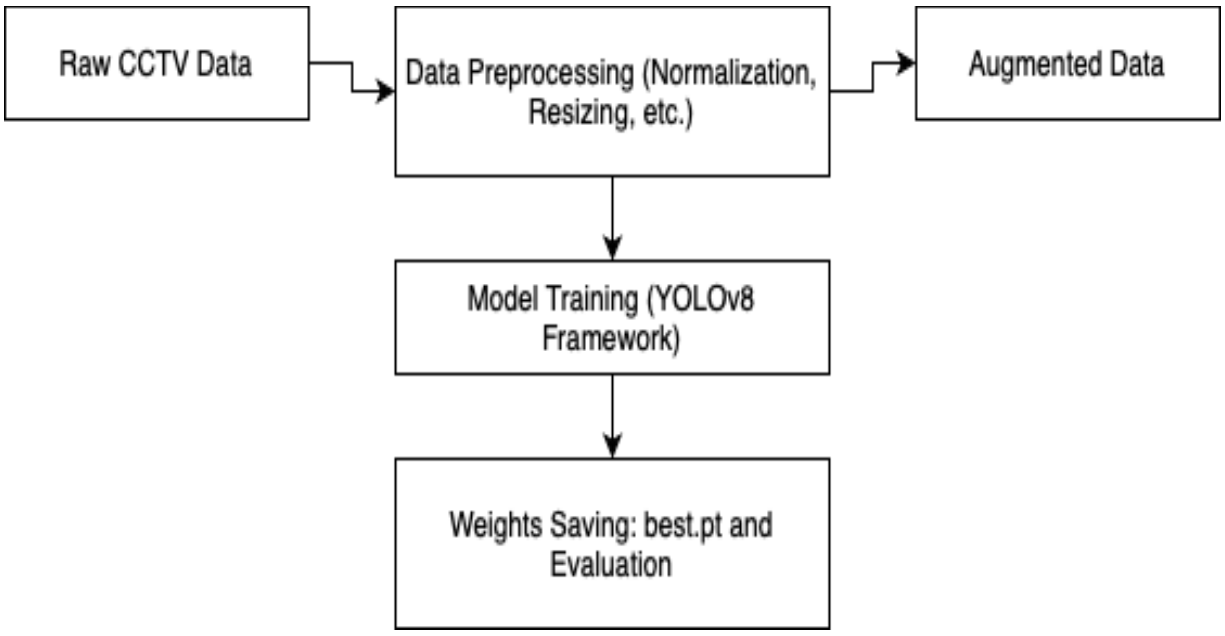
Annotation: Existing datasets come with annotated labels (occupied/vacant). But for custom data we can perform manual annotation.

Model Training

We are using YOLOv8 and YOLO11 as they are excellent for balance between detection accuracy and real-time performance. We can utilize pre-built architecture without needing to write a code of architecture from scratch and fine tune it on desired dataset.

Data Augmentation: Techniques such as random cropping, flipping, and rotation are applied to the training data to increase diversity and improve model's accuracy in different environments.

Training Process: The YOLOv8 framework is used to train the model on our chosen dataset. During training, key preprocessing tasks like normalization and resizing are automatically handled.



Inference & Deployment

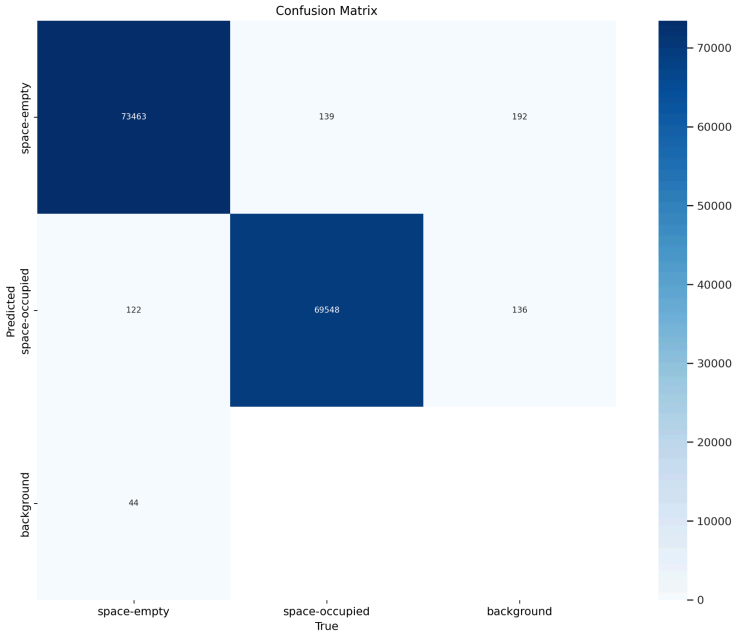
Real-Time Inference: We can use the trained YOLOv8 model to process live frames from CCTV cameras to detect vehicles and determine the occupancy of parking spaces in real time.

Post-Processing: Detected objects are annotated with bounding boxes and confidence scores. We will use these annotations to update and display the current parking availability status on the user interface.

Deployment: This model can be deployed using cloud based platforms so users can access it easily. For our project we will show deployment on the web with minimal functionality, which can be extended to an app with additional functionality.

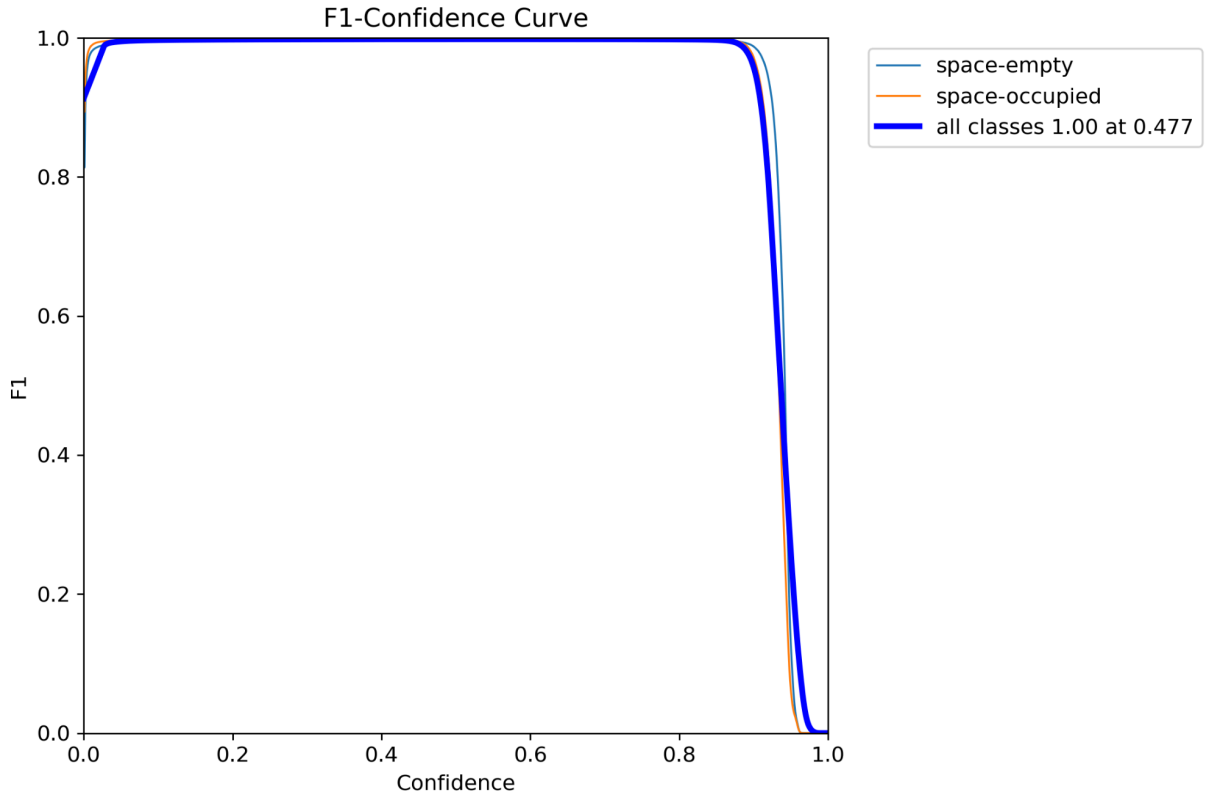
Evaluation of YOLO11

Confusion Matrix



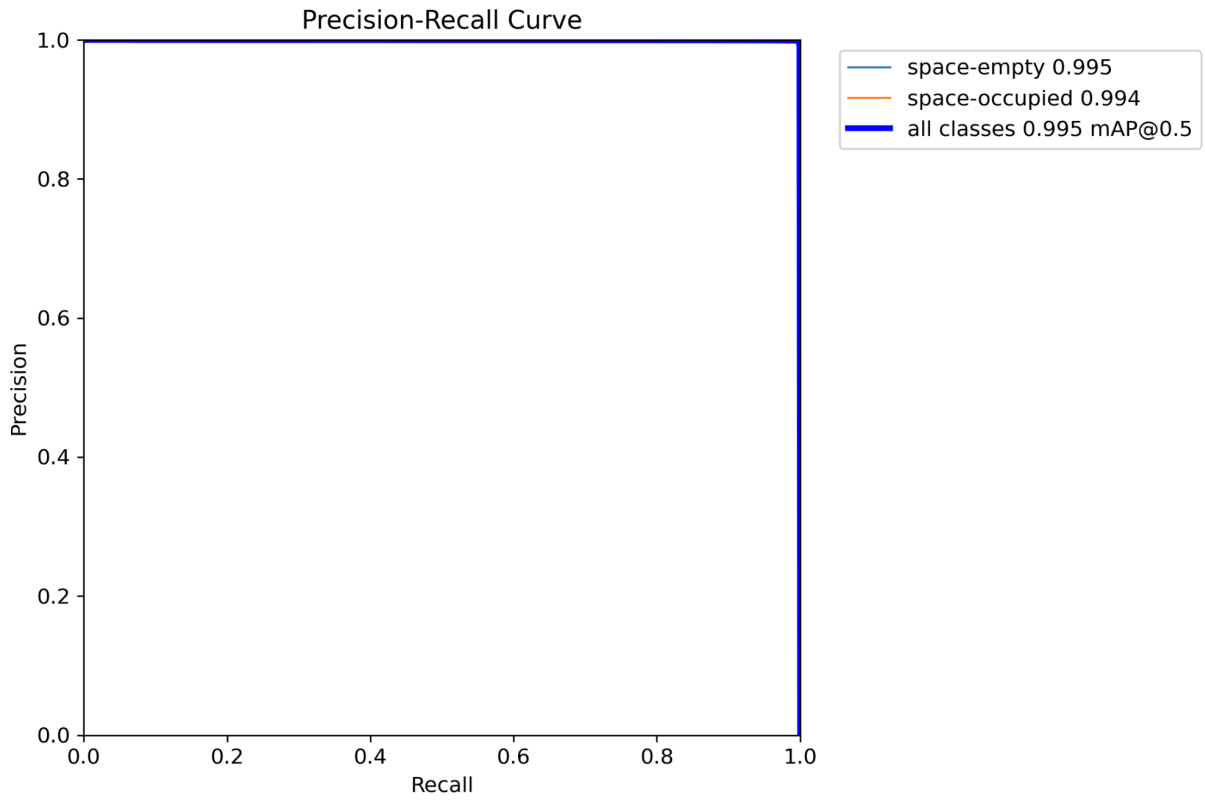
This image represents the standard confusion matrix. Here, each cell contains the raw number of predictions: true positives, false positives, true negatives, and false negatives. Analyzing this matrix helps identify specific error types for example if there are many false positives (vehicles predicted as occupying when they are vacant) or false negatives.

F1 Curve



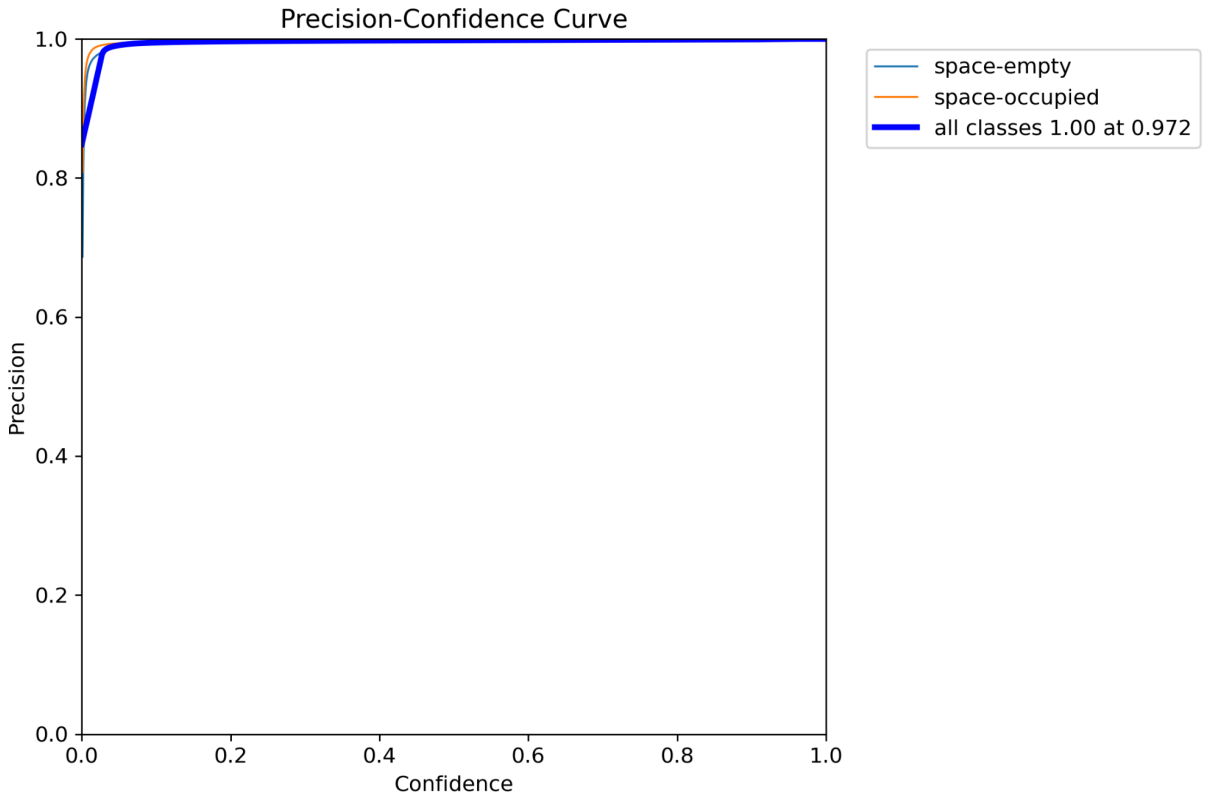
Here, F1 curve plots the F1 score over different training epochs or decision thresholds. The F1 score is a harmonic mean of precision and recall and it offers a balanced measure of the model's performance. We can clearly see a stable or rising F1 curve during training which indicates that the model is learning effectively and balancing precision and recall.

Precision Recall Curve



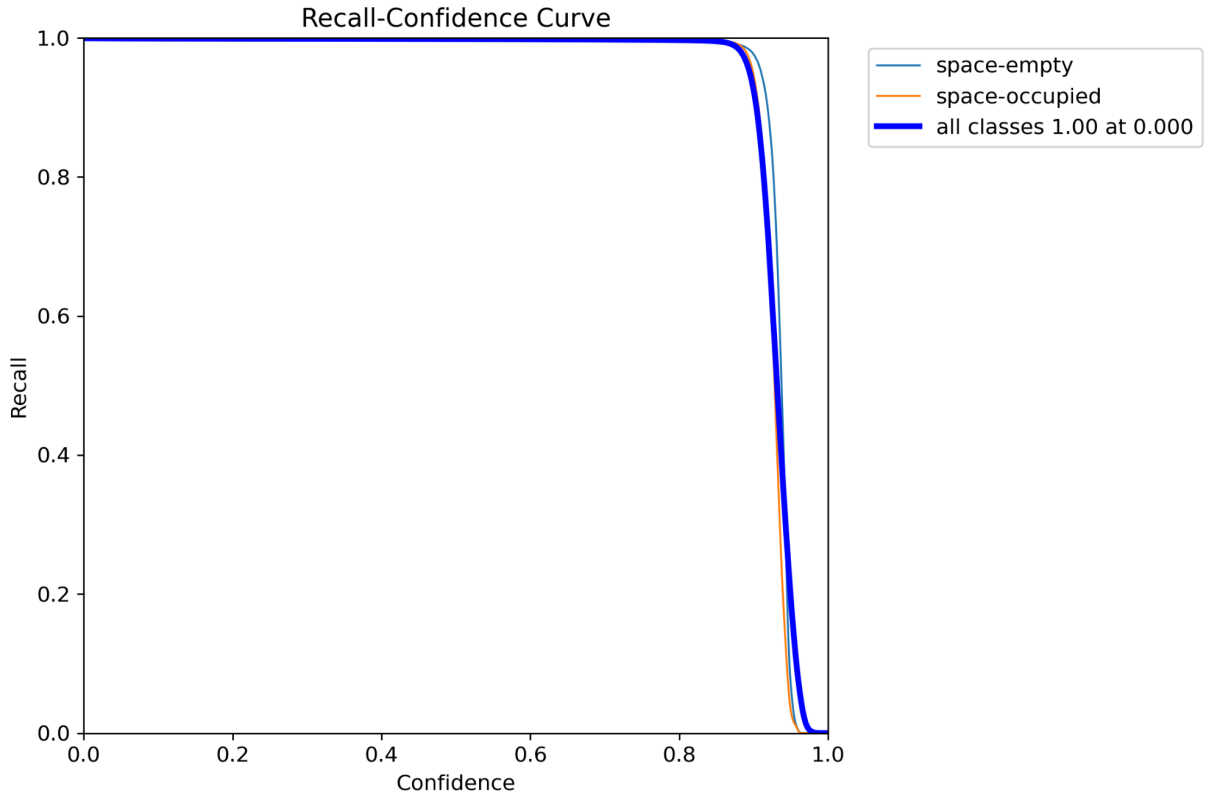
The Precision-Recall (PR) curve plots precision against recall for various thresholds. This curve is very useful when we have an imbalanced dataset as it focuses on the performance of the positive class. The area under the PR curve (AUC-PR) provides a single metric for model performance like a larger area means a better balance between precision and recall.

Precision Curve



This image shows the Precision curve that how precision varies over epochs or across different confidence thresholds. Precision measures the accuracy of positive predictions like the proportion of true positive detections among all predicted positives. A high precision value means that when the model predicts a parking space is occupied, it is usually correct.

Recall Curve

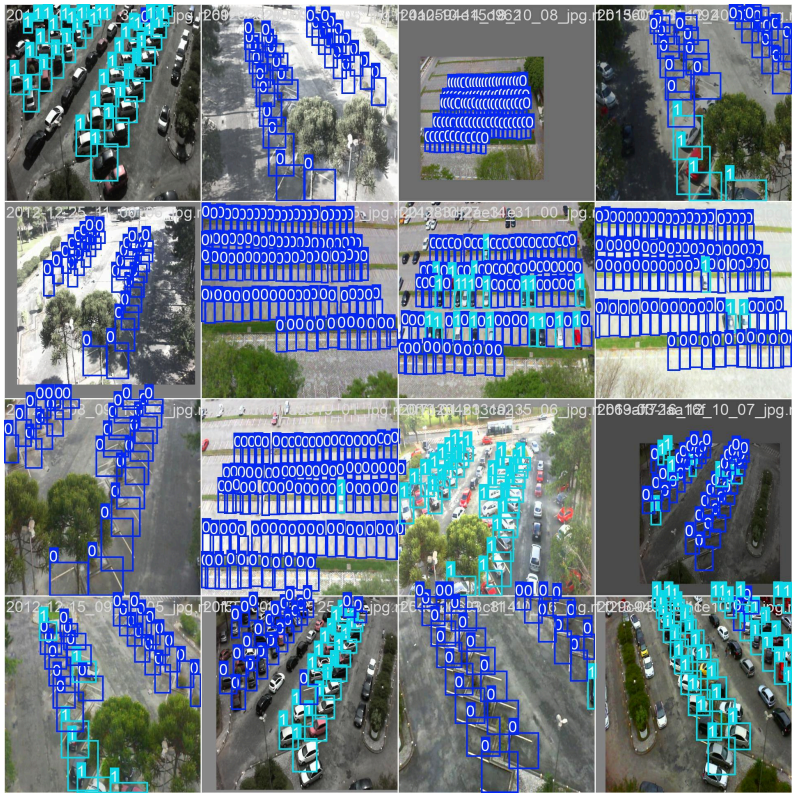


This image displays how recall changes over training epochs or thresholds. Recall (or sensitivity) measures the model's ability to correctly detect all actual positive cases. For example, all truly occupied parking spots. A high recall is important in applications where missing an occupied space would lead to incorrect decisions.

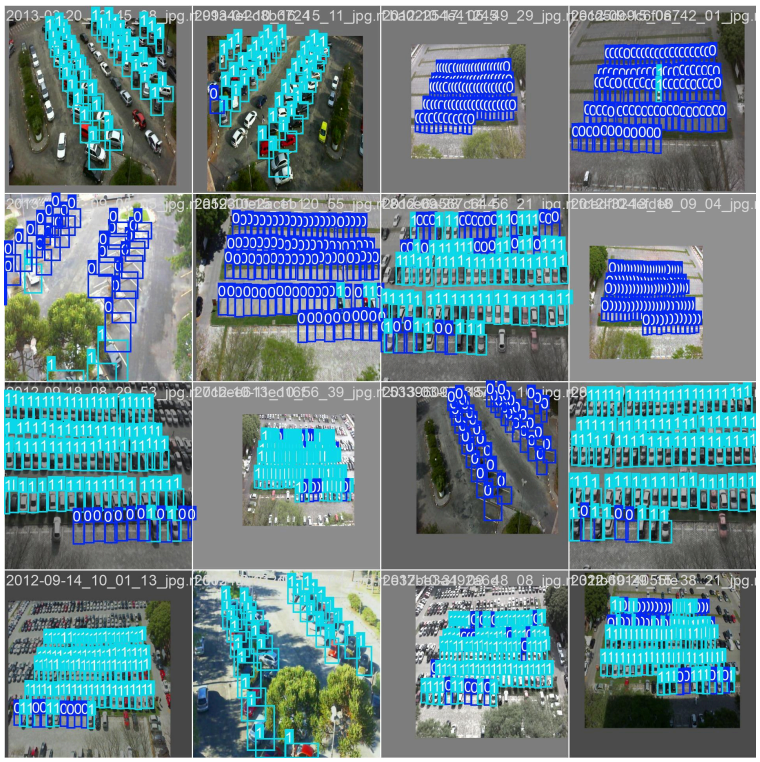
Train Batch

Here bounding boxes and class labels are shown after completion of training of each of 3 batches.

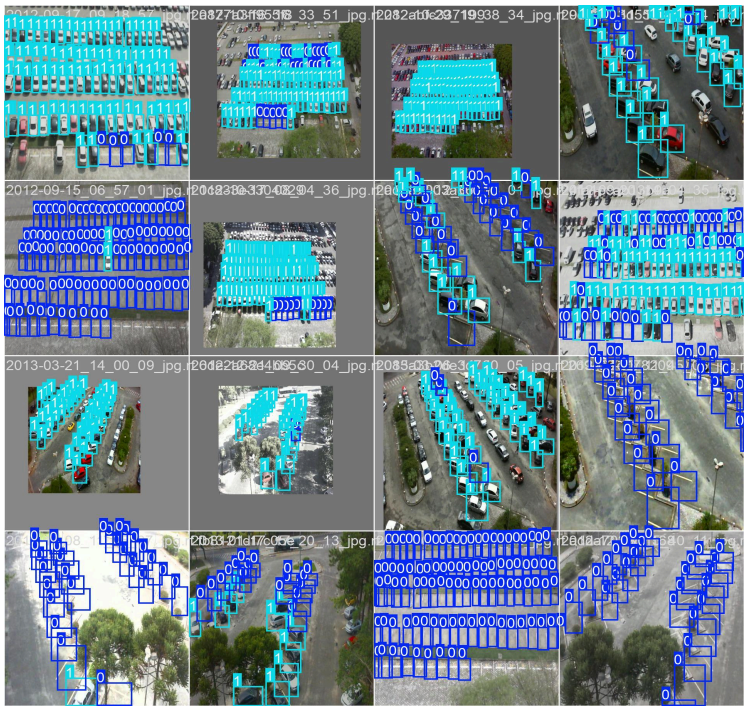
Batch 0



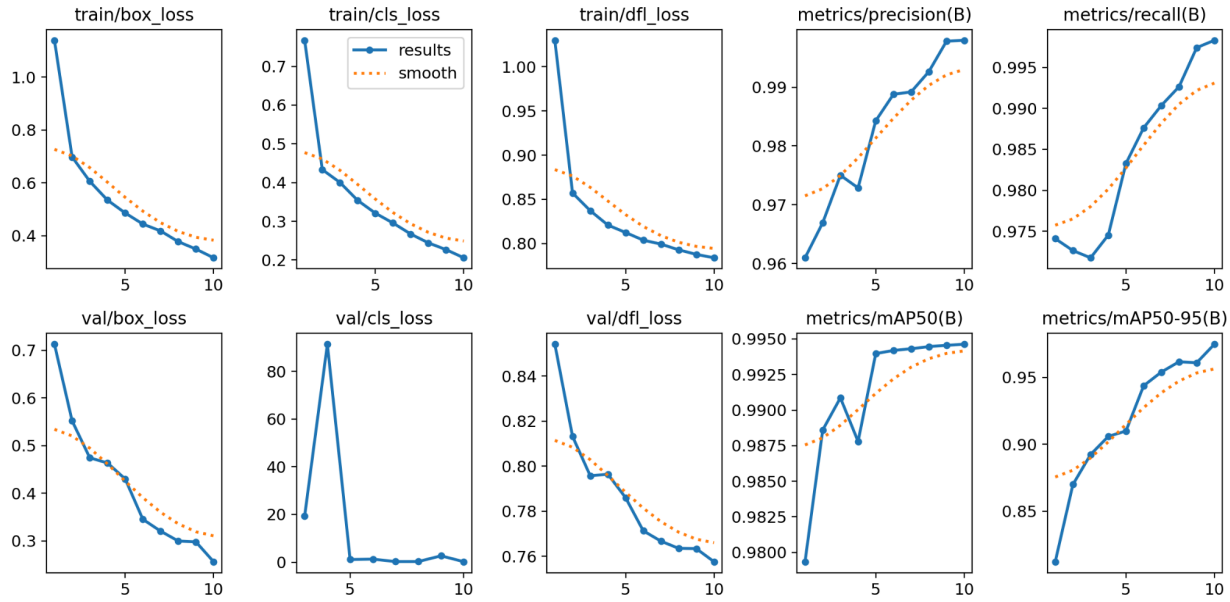
Batch 1



Batch 2



Results

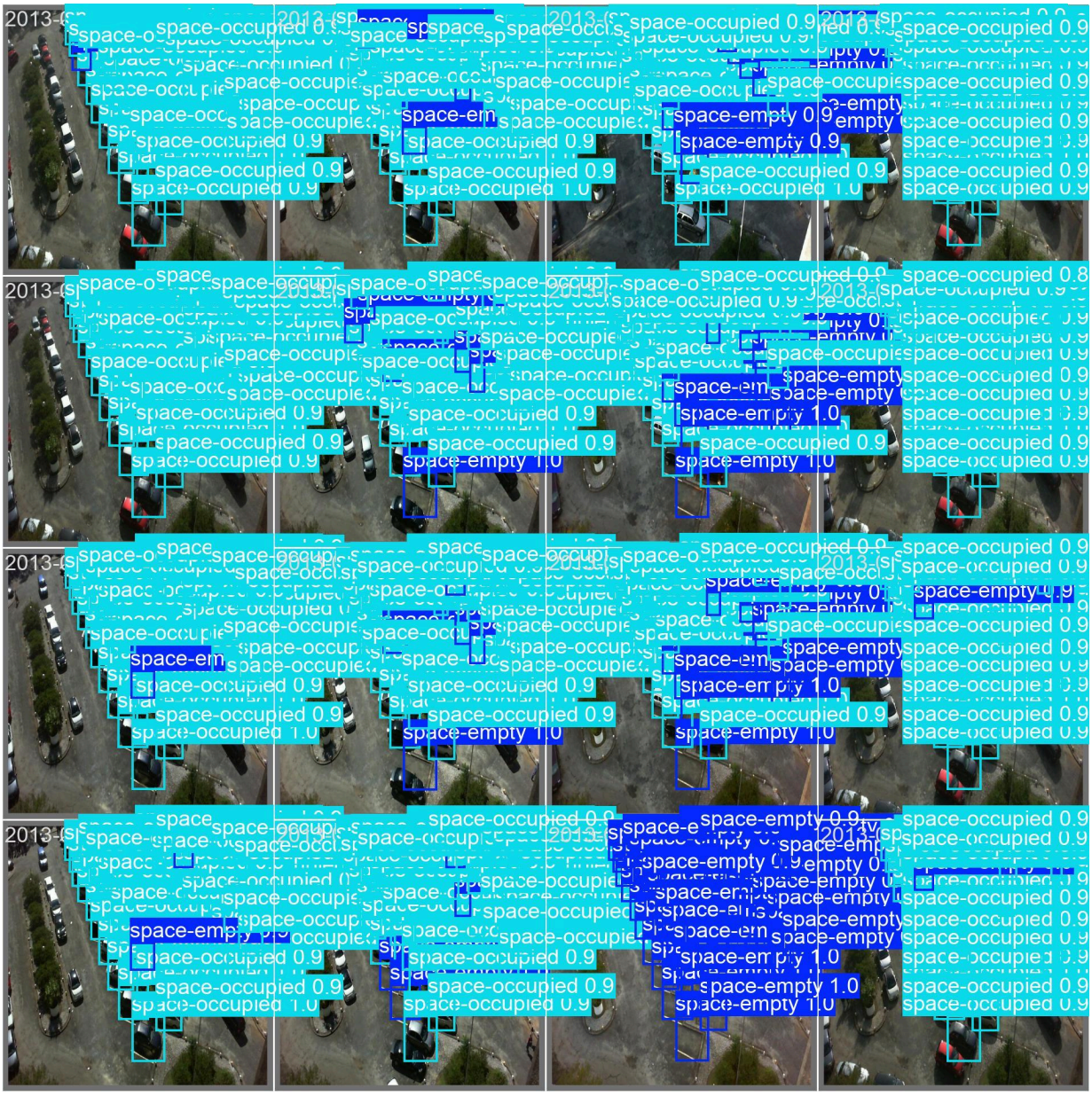


The YOLOv8 training graph shows that training and validation losses (box, cls, dfl) consistently decrease, while precision, recall, and mAP metrics steadily increase, this indicates that the model is effectively learning and generalizing; a temporary spike in val/cls_loss at epoch 2 means possible data noise or augmentation effects. But the training is successful and performance improves with each epoch.

Validation

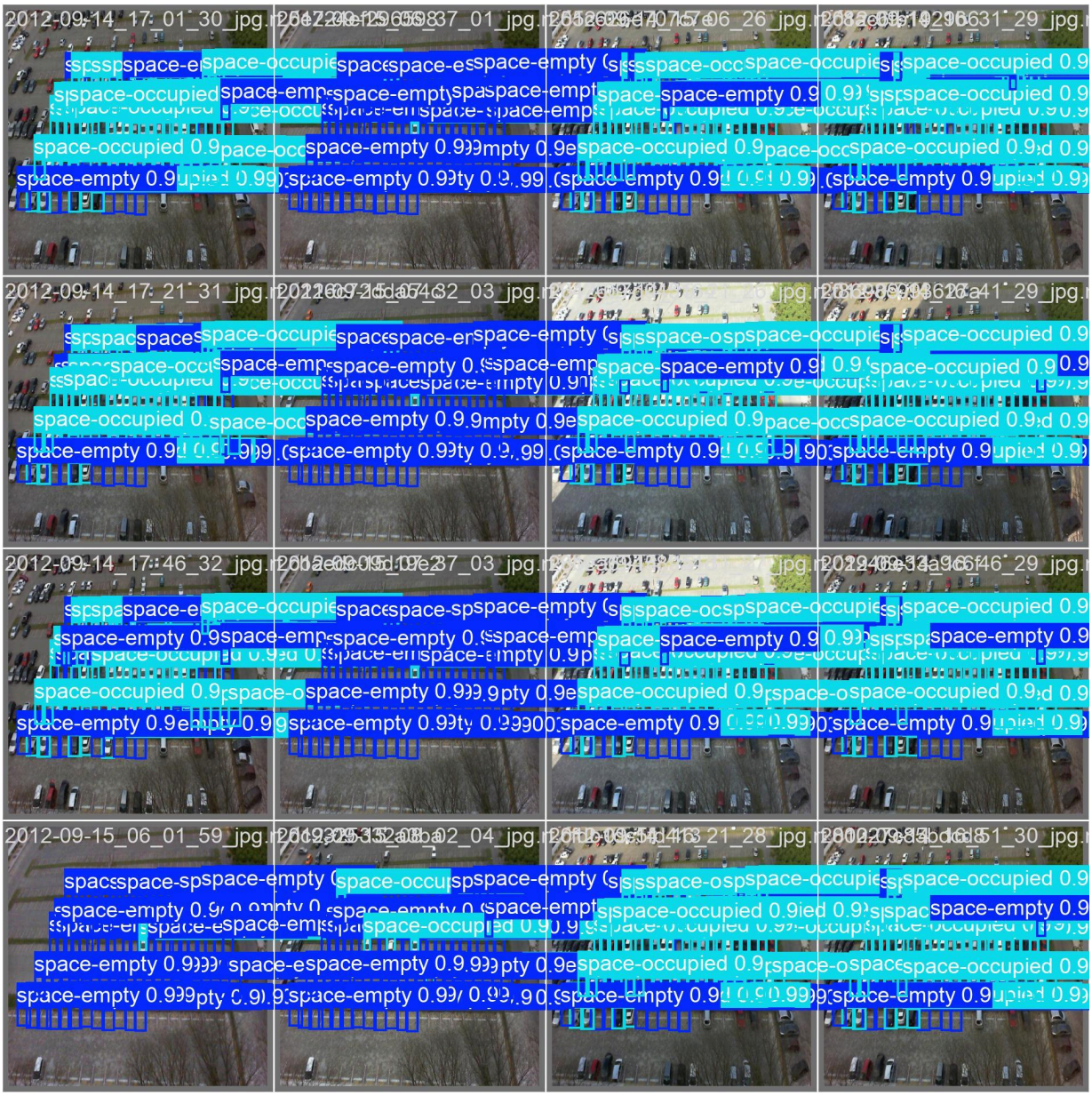
Here all batches shows that model is able to correctly detect occupied and empty spaces

Batch 0



Batch 1

Batch 2



ENHANCED MODELLING - RF DETR

RF-DETR (Roboflow Detection Transformer) is a real-time, transformer-based object detection model developed by Roboflow. It is designed for high speed and accuracy, even on devices with limited computing power, and is open-source under the Apache 2.0 license

Architectural Novelties

RF-DETR belongs to the DETR (Detection Transformer) family of models, which leverage transformer architectures. Key novel aspects of its architecture include:

- Hybrid Foundation: It builds upon the foundations of Deformable DETR and LW-DETR, integrating a pre-trained DINOv2 vision transformer as its backbone. This combination enhances its ability to adapt to new domains due to the knowledge stored in the DINOv2 backbone
- Single-Scale Features: Unlike the original Deformable DETR which uses multi-scale features, RF-DETR extracts image feature maps from a single-scale backbone to improve efficiency and reduce computational overhead.
- Elimination of NMS: Unlike YOLO models which require Non-Maximum Suppression (NMS) as a post-processing step to filter bounding boxes, RF-DETR's architecture eliminates the need for NMS. This supports end-to-end training and simplifies the inference pipeline
- Global Context Modeling: The transformer architecture excels at modeling global context, which can lead to higher accuracy in complex scenes and better identification of partially occluded or ambiguous objects
- Multi-Resolution Training: The model is trained at multiple resolutions, allowing users to select different input resolutions at runtime to balance accuracy and latency without needing to retrain the model

Performance Enhancements Over YOLO Models

RF-DETR has demonstrated significant performance advantages compared to Convolutional Neural Network (CNN)-based YOLO models and other real-time detectors:

- Accuracy: RF-DETR was the first real-time model architecture to achieve over 60 mean Average Precision (mAP) on the standard COCO dataset benchmark Specifically, RF-DETR-Large reached 60.5 mAP, while the smaller RF-DETR-Base achieves 53.3 mAP, outperforming comparable YOLOv8m (50.6 mAP) and YOLO11m (51.5 mAP) models
- Domain Adaptability: It achieves state-of-the-art results on the RF100-VL benchmark, which measures a model's ability to adapt to diverse, real-world domains across various categories like aerial, industrial, and biological imagery. RF-DETR-Base scored 60.3 mAP@50:95 on RF100-VL, surpassing YOLOv8m (59.8) and YOLO11m (59.7)
- Speed: When considering "Total Latency," which includes the NMS time required for YOLO models but not for DETR models, RF-DETR shows comparable speed to leading YOLO models. For example, RF-DETR-Base has a total latency of 6.0 ms on an NVIDIA T4 GPU, similar to LW-DETR-M (6.0 ms) and YOLOv8m (6.3 ms), and slightly slower than YOLO11m (5.7 ms)
- Handling Occlusion: Studies comparing RF-DETR with YOLOv12 in complex orchard environments found RF-DETR superior at identifying partially occluded or ambiguous greenfruits, likely due to its strength in global context modeling.
- Training Convergence: In the greenfruit detection study, RF-DETR demonstrated faster convergence during training compared to YOLOv12 configurations, reaching stable performance in fewer epochs, especially in single-class detection

RF-DETR

Data & Annotation Strategy

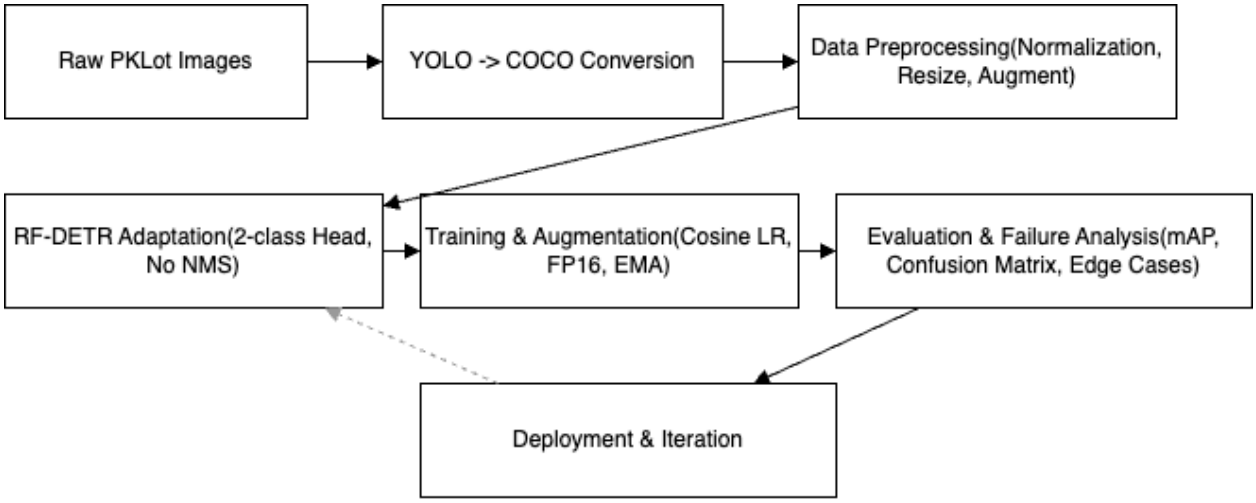
We have begun by using the PKLot dataset's well-curated parking-lot images, but our focus is on transforming its native YOLO labels into the COCO-style format that RF-DETR requires. Rather than manual conversions, we have architected a single pipeline that reads each normalized YOLO box, converts it into pixel-accurate COCO annotations, and validates bounds and label consistency in one pass. Based on our training, this yielded perfectly synchronized 80/20 splits—no mismatches or out-of-range coordinates and drove training loss from 4.37→3.20→2.83 while validation loss fell from 4.62→3.18→2.58 over three epochs.

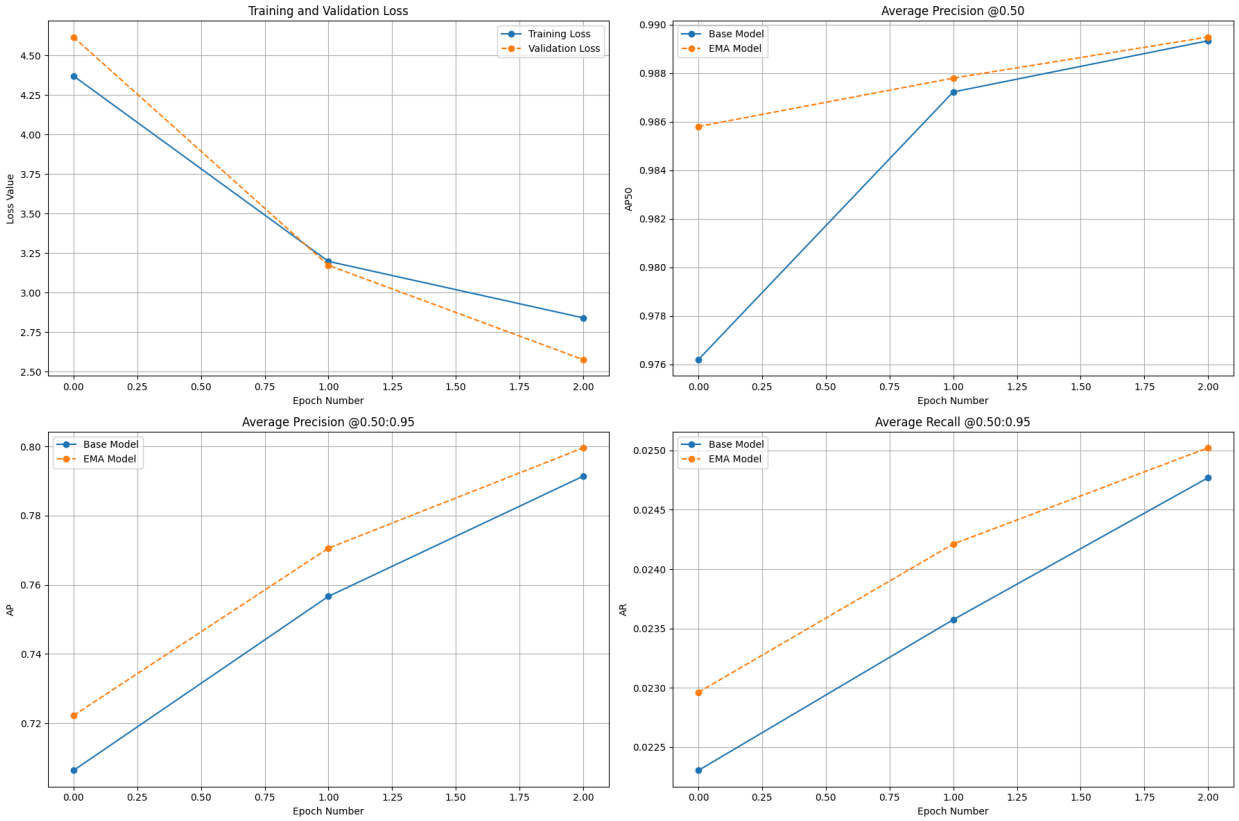
Model Adaptation Strategy

At the core, we have adopted the RFDETRBase pretrained on COCO but immediately reconfigured its detection head for our binary (empty vs. occupied) task. We have removed the original multi-class head for a Xavier-initialized 2-class head to preserve the rich, global features of the DINOv2 backbone as well as to cut redundant complexity. Because the transformer is end-to-end, we have eliminated NMS entirely yet EMA smoothing still give AP@0.50 from 0.987→0.989 and AP@[.50:.95] from 0.772→0.800 by epoch 2, which confirms that global context alone handles overlaps and partial blockages.

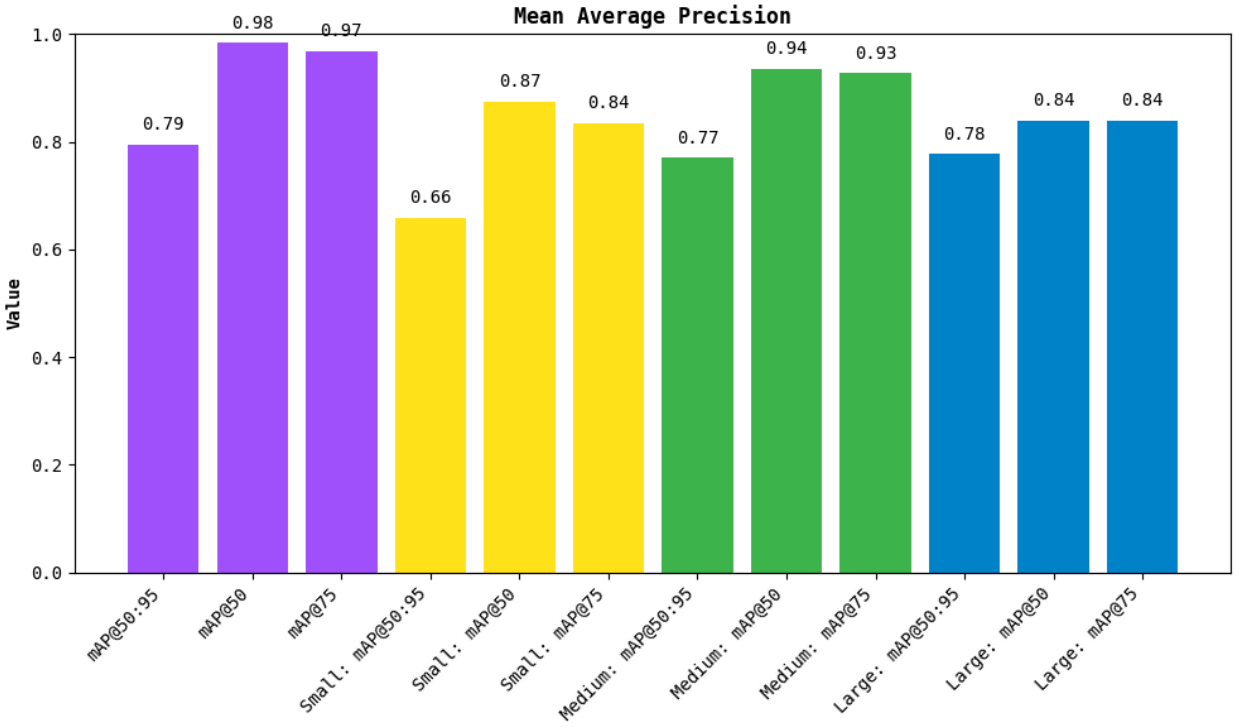
Training & Augmentation Strategy

To maximize real-world precision, we have trained for three epochs with a conservative 1e-4 base learning rate (cosine warmup), mixed-precision (fp16) and gradient accumulation to emulate large batches. Targeted augmentations like +- 5° rotations, horizontal flips, minor color jitters means that it simulates camera tilt, variable lighting and weather without overfitting. Over the run, cross-entropy loss decreased from 0.3466→0.3086 and GIoU loss from 0.2036→0.1790, while class error on the test set dropped from 2.6% to 0%. We retained only the top checkpoints by validation mAP, ensuring we never revert to an under- or over-trained model.

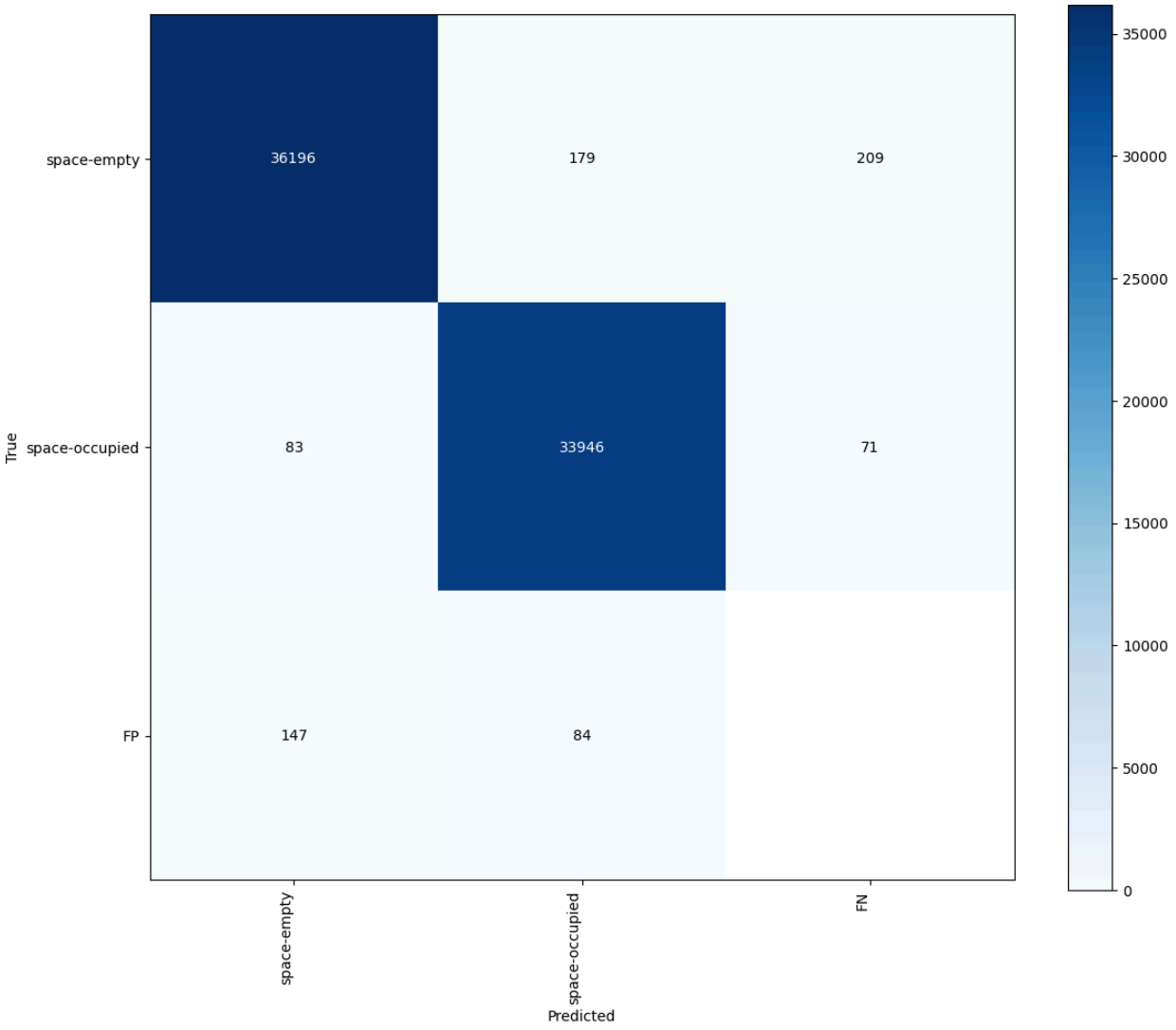




Over three epochs, training and validation loss dropped in lockstep from ≈4.5 to ≈2.6 with no overfitting, AP@.50 climbed from .976 to .989 (EMA always 0.002 higher), AP@[.50:.95] rose from .706 to .800 (EMA up by 0.01–0.02), and recall similarly improved—by epoch 2 the curves begin to flatten, which indicates solid, stable gains but diminishing returns without further tuning. Due to resource limitation we were unable to do training for more epochs.

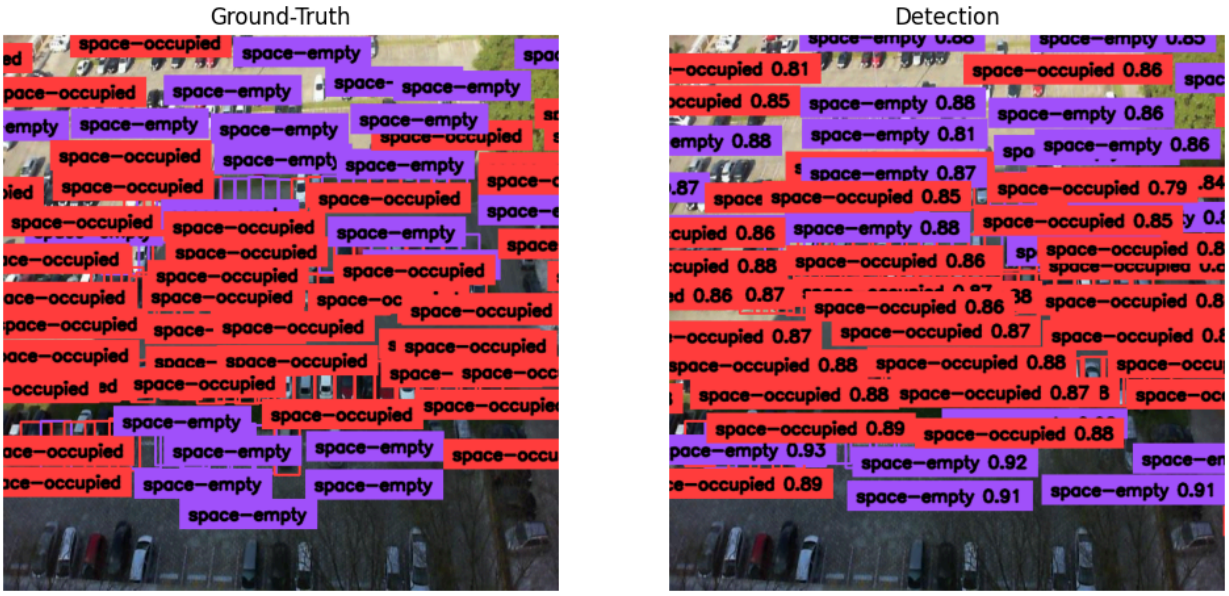


Our RF-DETR model is achieving a 0.79 mAP at the standard 50–95% IoU range and jumps to 0.98 AP at IoU 0.50 and 0.97 at IoU 0.75. When we split by space size, it scores 0.66 mAP for small spots (with AP@.50 = 0.87 and AP@.75 = 0.84), 0.77 mAP for medium spots (AP@.50 = 0.94, AP@.75 = 0.93), and 0.78 mAP for large spots (AP@.50 and AP@.75 both 0.84). In short, it's excellent on medium and large spaces but still we need to tune it for tiny spots.

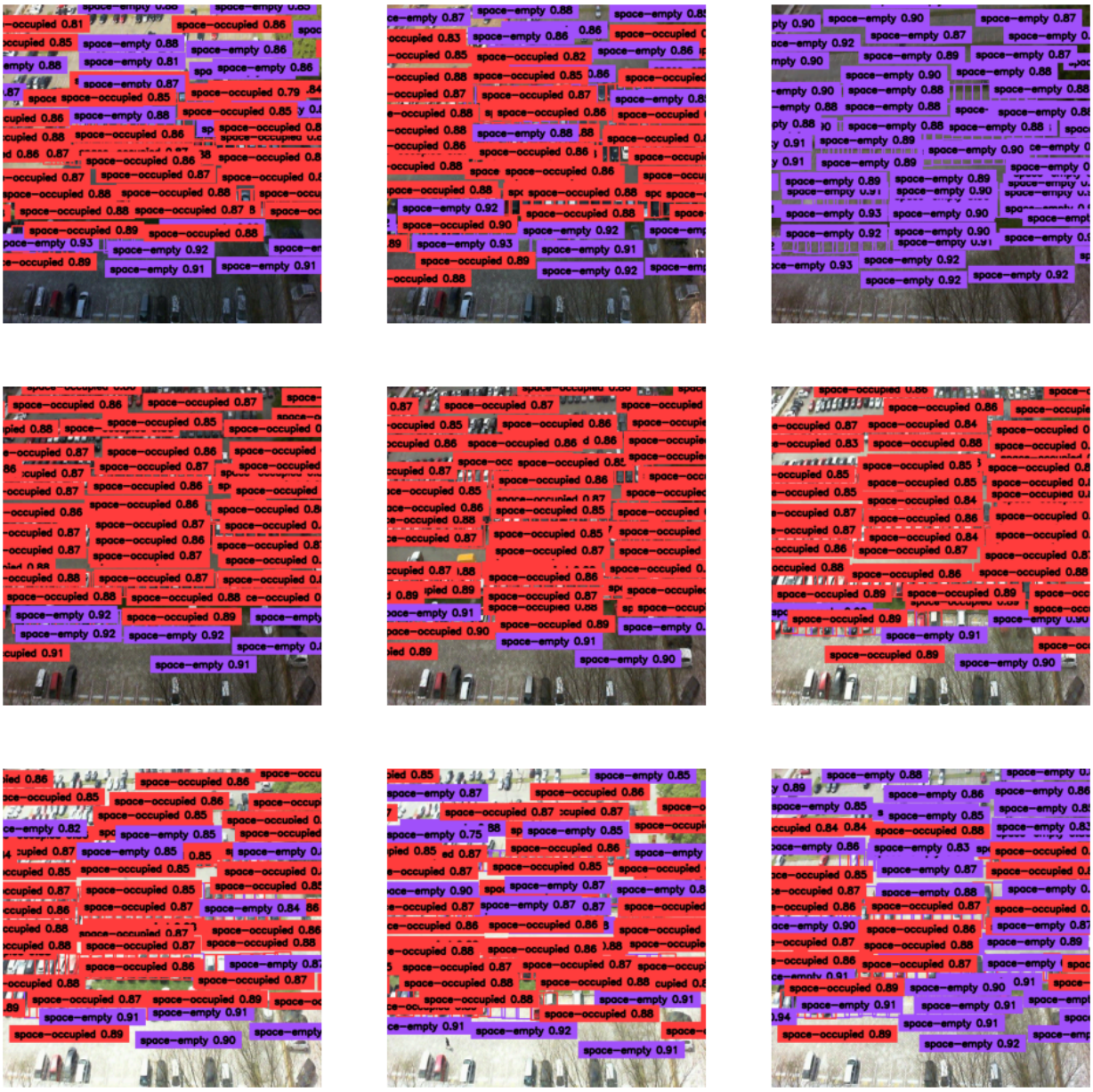


The confusion matrix shows our model getting almost every space right: out of about 36 584 empty spots, 36 196 were correctly labeled empty, only 179 were wrongly flagged as occupied and 209 were missed; similarly, of roughly 34 100 occupied spots, 33 946 were correctly found, just 83 were called empty and 71 slipped through. Overall false positives (147 empty, 84 occupied) and false negatives (209 empty, 71 occupied) are very small compared to the tens of thousands of true detections which means the model is good at achieving both very high precision and recall on each class.

Evaluation of RF-DETR



The side-by-side image shows that the “truth” map of every parking slots—purple boxes for empty spots and red for occupied—laid over the lot image, while the right panel shows our model’s picks with matching red and purple boxes, each tagged with its confidence score (e.g. “space-occupied 0.88”). You can immediately see that nearly every ground-truth box has a corresponding detection at a high confidence level, and only a very small number of spots, mostly at the edges or under shadows, are slightly misaligned or mislabeled. This visual comparison makes it easy to spot exactly where the detector works (tight, correctly colored boxes over most cars and open bays) and where it might need more tuning (blocked or low-contrast areas).



This 3×3 grid shows nine different lot crops with our model's detections—red for occupied and purple for empty—each tagged with its confidence score. You can see that across varied viewpoints, lighting, and blockages (cars partly hidden by shadows or edge framing), the detector consistently labels spaces with high confidence (mostly between ~0.85 and ~0.93). Occupied spots reliably get tight red boxes around cars, and empty bays are correctly outlined in purple. A few edge cases (deep shadows or extreme angles) dip slightly

in confidence, but overall the grid demonstrates our model's precision in real-world parking scenes.

Conclusion

In this project, we have demonstrated that modern deep-learning detectors—both convolutional YOLO variants and transformer-based RF-DETR—can be effectively adapted to the parking-space occupancy task using only top-down CCTV imagery and existing PKLot annotations. By streamlining label conversion, fine-tuning a two-class head on a DINOv2 backbone, and applying targeted augmentations, our RF-DETR model drove validation loss from 4.6 to 2.6 in three epochs and achieved an overall mAP@[.50:.95] of 0.80 (AP@.50 = 0.99, AP@.75 = 0.97), with strong performance on medium and large spaces and minimal false alarms or misses in tens of thousands of samples.

Looking ahead, further gains should be attainable through extended training, synthetic occlusion overlays, and semi-supervised fine-tuning on live feeds—while our NMS-free, end-to-end pipeline is already lightweight enough for real-time edge deployment. By integrating a feedback loop that ingests field misclassifications and exploring adaptive input resolutions, this system offers a scalable, robust foundation for next-generation smart-parking solutions.

References

- Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press. URL: https://www.deeplearningbook.org
- Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You Only Look Once: Unified, Real-Time Object Detection. CVPR 2016. URL: https://arxiv.org/abs/1506.02640
- Bochkovskiy, A., Wang, C.-Y., & Liao, H.-Y. M. (2020). YOLOv4: Optimal Speed and Accuracy of Object Detection. URL: https://arxiv.org/abs/2004.10934
- Jocher, G., et al. (2023). YOLOv8: The Next Generation of Real-Time Object Detection. Ultralytics. URL: https://github.com/ultralytics/ultralytics

- Carion, N., et al. (2020). End-to-End Object Detection with Transformers. ECCV 2020. URL: https://arxiv.org/abs/2005.12872
- Roboflow (2023). RF-DETR: Real-Time Object Detection Transformer. URL: https://github.com/roboflow/rf-detr
- Meta AI (2023). DINOv2: Self-Supervised Pre Training for Vision. URL: https://ai.meta.com/publications/dinov2/
- de Lima, D., & Gonçalves, A. (2015). PKLot: A Robust Dataset for Parking Lot Classification. URL: https://gts.ai/dataset-download/pklot-dataset/