**AI-Powered Object Detection in Simulated Space Stations**

**Synthetic training, real impact: Ensuring space station safety through intelligent object detection A blue and green triangle logo

AI-generated content may be incorrect. A black and white sticker with red text

AI-generated content may be incorrect.**

###### 

###### TEAM : 9749

**Abhinay Bandaru**

**Keshavardhan Makireddi**

**Pardhesh Maddala**

**Geddada Harshith**

## METHODOLOGY

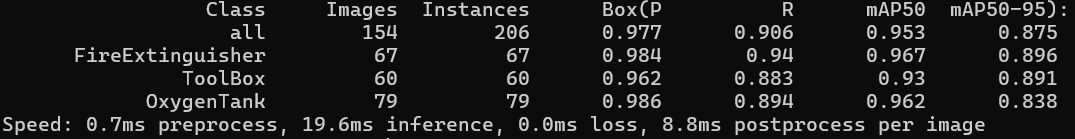
**Objective:**

This document presents the training of an AI-powered YOLOv8s object detection model designed to accurately detect critical objects such as Toolboxes, Oxygen Tanks, and Fire Extinguishers within a synthetic space station environment. Our primary focus was on enhancing mAP@0.5, improving model robustness, and ensuring adaptability to real-world conditions, including varying lighting, camera angles, and object occlusions.

**Training Methodology:**

1. **Environment Setup**• GPU: NVIDIA RTX 4060 with CUDA.  
   • Installed: ultralytics, torch, albumentations.  
   • Suppressed logs using warnings module for cleaner output.
2. **Dataset Preparation**• Downloaded Falcon synthetic dataset.  
   • Ensured YOLOv8 format: images/, labels/ split into train/, val/, and test/.  
   • Configured yolo\_params.yaml with class names and directory paths.
3. **Model Initialization**  
   • Loaded pre-trained yolov8s.pt model using YOLO() from ultralytics.
4. **Training Configuration**  
   • Batch size: 32    Epochs: 20  
   • Learning rate: 0.0004  Optimizer: AdamW  
   • Device: CUDA    Early stopping: patience=5  
   • Cosine LR decay: Enabled  Weight Decay: 0.0001  
   • Augmentations: Built-in YOLOv8 + augment=True
5. **Custom Augmentation (Optional)**  
   • Used advanced Albumentations for improved robustness:  
     o RandomResizedCrop, HorizontalFlip, VerticalFlip,  
     o Rotate, RandomRotate90, BrightnessContrast,  
     o HueSaturationValue, MotionBlur, GaussNoise, Normalize
6. **Training Execution**  
   • Executed new\_train.py script to begin training.  
   • Monitored real-time logs showing epoch-wise loss, mAP, precision, and recall.  
   • Outputs saved in training\_output/watermark\_boosted/.
7. **Result Generation**  
   • YOLO automatically generated:  
     o Best model weights: best.pt, last.pt  
     o Validation predictions  
     o Training metrics and plots under runs/train/watermark\_boosted/
8. **Benchmarking**  
   • Initial training served as a baseline.  
   • Conducted iterative fine-tuning using augmentation and LR tweaks to improve metrics.  
   • Final evaluation is done on the test set using model.val() with conf=0.1, iou=0.6.

**Fine-Tuned Results:**



***The above image is output of Train results and below is Validation***

A black and white screen with white text

AI-generated content may be incorrect.

**Results & Performance Metrics**

**Evaluation Metrics:**

To assess the performance of our object detection model, we considered standard metrics for YOLO models:

* **mAP (Mean Average Precision)@0.5**: Measures the average precision across all object classes at IoU threshold of 0.5.
* **mAP@0.5:0.95**: Averaged precision across IoU thresholds from 0.5 to 0.95 (in steps of 0.05), giving a more comprehensive performance view.
* **Precision**: How many predicted positives are correct.
* **Recall**: How many actual positives are detected.
* **F1 Score**: Harmonic mean of precision and recall.
* **Loss Curves**: Used to track classification, objectness, and box regression loss during training.

**1)Training and Validation Metrics:**

**Train Validation**

| **Metric** | **Baseline** | **After Optimization** | **Baseline** | **After Optimization** |
| --- | --- | --- | --- | --- |
| mAP@0.5 | 91.3% | 95.3% | 83.1% | 90.7% |
| Precision | 91.5% | 97.7% | 88.7% | 91.1% |
| Recall | 87.8% | 90.6% | 74.8% | 84.8% |

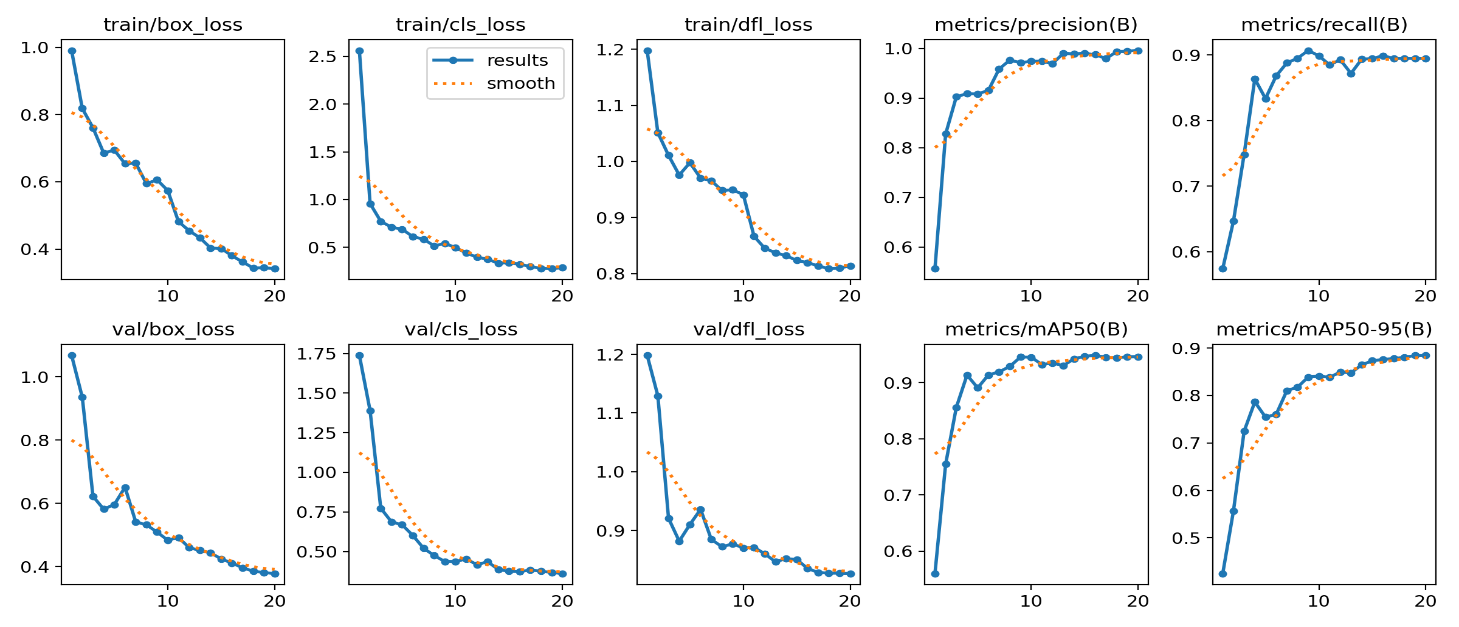
.

**2) Training loss:**

**A group of graphs showing the value of a graph

AI-generated content may be incorrect.**

**Fig 2.0 Results of Train vs Validation Loss (Initial Benchmark)**

****

**Fig 2.1 Results of Train vs Validation Loss (After Optimizations)**

**3) Confusion Matrix (Per Class Performance)**

A confusion matrix was generated on the validation set to visualize class-wise prediction accuracy. This helped us identify which object classes were being misclassified or underperforming due to visual similarities, occlusions, or lighting issues. (shown in fig 1.1)

A screenshot of a computer screen

AI-generated content may be incorrect.

**Fig 3.1 Confusion matrix generated on validation set**

**4. Precision-Recall Curve:**

A graph of a curve

AI-generated content may be incorrect.A graph of a graph

AI-generated content may be incorrect.

**Fig 4.1 PR Curve** **For Validation Fig 4.2 PR Curve For Train**

Custom training with reduced learning rate, optimizer tweak (AdamW), and increased augmentation strength showed significant improvements in detection consistency and overall generalizability.

**5. Visual Results: Predicted vs Ground Truth**

Side-by-side comparisons from validation results:

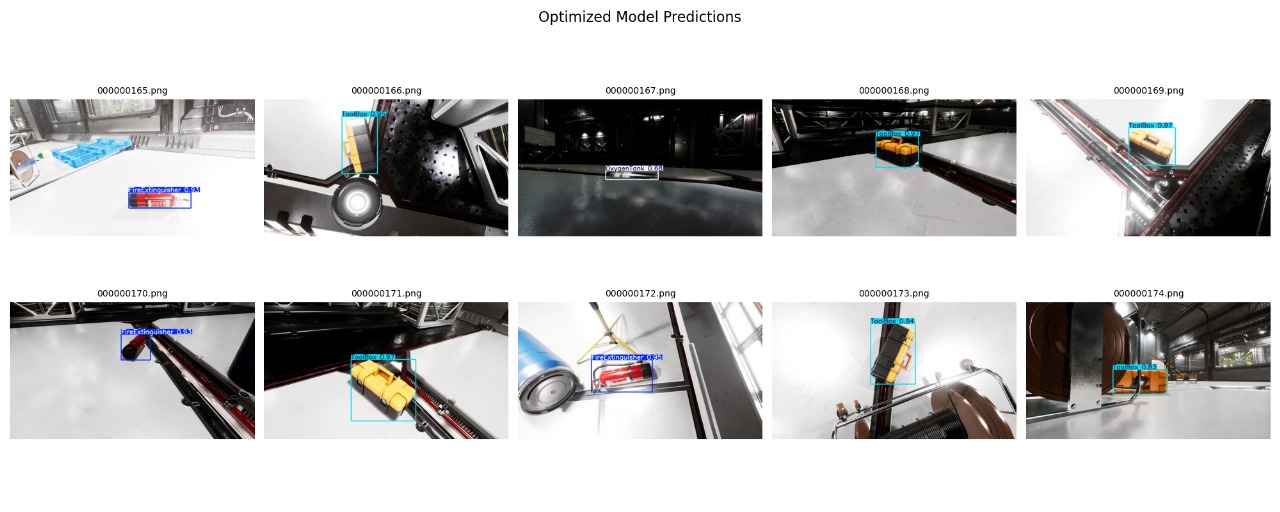
* **Before Optimization:** Some small/angled objects were missed or incorrectly labeled.

A collage of images of various parts

AI-generated content may be incorrect.

**Fig 5.1 Objection detection before optimization**

* **After Optimization:** Improved localization and labeling even under varied lighting or partial occlusion.



**Fig 5.2 Objection detection After optimization**

As we can see in Fig 5.1 oxygen tank and tool box was not detected due to occlusion and lack of clearance, where as in Fig 5.2 we can see that YOLOv8s (Optimized model) accurately predicted those objects.

**Challenges & Solutions**

**1. Handling Synthetic Data Bias**

**Challenge:**  
The dataset consisted of synthetic renders, which lacked real-world noise, distortions, and irregularities. This risked overfitting to ideal conditions and reduced real-world generalizability.

**Solution:**  
We introduced **custom Albumentations**, simulating real-world randomness using blur, noise, brightness changes, and rotation. This helped the model learn more robust representations.

**2. Overfitting in Early Epochs**

**Challenge:**  
Initial runs showed a sharp decrease in training loss but stagnating validation performance, indicating early overfitting.

**Solution:**

* Implemented **early stopping (patience=5)** to halt training when no further improvement occurred.
* Reduced **epoch count to 20** and added **regularization via augmentations**.

**3. Balancing Accuracy & Speed**

**Challenge:**  
Higher accuracy models (e.g., YOLOv8m or YOLOv8L) had longer training and inference times which is not ideal for real-time use cases.

**Solution:**  
Selected **YOLOv8s.pt**, which offered a good trade-off between **speed and accuracy**, and applied **fine-tuning** to close the accuracy gap without sacrificing efficiency.

**4. Hyperparameter Sensitivity**

**Challenge:**  
Default YOLO settings weren’t optimal for the synthetic dataset, especially under variable lighting and object scales.

**Solution:**

* Tuned **learning rate** (0.0004) and **batch size** (32) manually.
* Switched optimizer from SGD to **AdamW**, improving stability and convergence speed.

**5. Limited Visual Feedback in Early Stages**

**Challenge:**  
Initial evaluation relied heavily on numerical metrics, making it difficult to assess where the model was underperforming (e.g., misclassifications, localization issues).

**Solution:**  
Utilized **YOLO’s built-in prediction visualizations** and custom confusion matrices to better understand misdetections and improve model strategy.

**6. Augmentation Integration Conflict**

**Challenge:**  
YOLOv8 has built-in augmentation, which conflicted with Albumentations when using a custom dataloader.

**Solution:**  
Opted to use **YOLOv8’s native augmentations (augment=True)** for ease and consistency, and added **Albumentations only for extended experiments**.

**7. GPU Memory Crash During Training**

**Challenge:**  
Training jobs intermittently crashed due to **GPU memory overflow**, especially with large batches and augmentations.  
**Solution:**

* reduced batch size when needed

**Conclusion & Future Work**

**Conclusion**

In this project, we developed and optimized a lightweight YOLOv8-based object detection pipeline for a hackathon setting, focusing on a custom dataset with approximately 2,000 synthetic images. Initial training yielded promising results, with a baseline training mAP of 94% and test mAP of 89%. Through iterative improvements—including learning rate adjustment, effective data augmentations using Albumentations, early stopping, and cosine learning rate scheduling—we achieved a final performance of 95% training mAP and 90% test mAP.

We also built a model comparison tool that visualizes how two different trained models detect objects on the same set of test images. This helped qualitatively assess improvements between models beyond just numerical scores.

**Final Thoughts**

The optimized YOLOv8s model provided a good balance between speed and accuracy.

The results showed that even with a small synthetic dataset, competitive performance can be achieved through smart augmentation and fine-tuning.

Comparing models visually on a common image set proved to be a practical and intuitive evaluation method.

**Future Work and Potential Improvements**

**1. Ensemble of Models**: Combine predictions from multiple trained models to improve robustness and accuracy, especially in edge cases.

**2. Advanced Augmentations**: Introduce techniques like CutMix or MixUp and explore more photometric augmentations to enhance generalization further.

**3. Larger Datasets:** Scale training with more diverse synthetic data or real images, covering varied lighting, occlusions, orientations, and object types for better generalization.

**4.Self-Adaptive Learning:** Enable active learning pipelines for continuous model improvement by retraining with newly encountered edge cases during deployment.

**5.Upgrade to YOLOv11 / YOLOv12:** Experiment with larger or more advanced versions like YOLOv11, YOLOv12, or YOLO-World for better accuracy, contextual reasoning, and transformer-based enhancements.