Duality AI's Space Station Challenge: Safety Object Detection

By Promptify

Github Repo : https://github.com/nileshh-27/Safety-Object-Detection-Duality-AI

Project Objective

- The primary objective of this project was to develop a robust object detection model capable of identifying 7 critical safety items in a synthetic space station environment.
- Our goal was to exceed the baseline performance benchmark of 40-50% mAP@0.5 by carefully training and evaluating a YOLOv8 model.
- An Synthetic dataset duplicated by Falcon is used to train and test the model in different conditions.
- Additionally as a bonus task an SIMS or **Safety and Inventory Management System**.

1. Dataset Overview

- We utilized the synthetic dataset provided by Duality AI, generated via the Falcon platform.
- The dataset contained images with challenging conditions, including varied lighting, object occlusions, and diverse camera angles, ensuring the model would be trained for robust, real-world performance.
- Additionally, we also augmented the existing dataset to train the model in additional variations of conditions.

2. Model Architecture

 We selected the **YOLOv8m (medium) architecture** from Ultralytics, a state-of-the-art single-stage detector renowned for its excellent balance of speed and accuracy.

 The 'm' variant was a strategic upgrade from our initial 's' model, providing a higher learning capacity to achieve a better score while remaining feasible to train within the hackathon's time constraints.

 We employed **transfer learning** by initializing our model with weights pre-trained on the large-scale COCO dataset, rather than training from scratch.

 This fine-tuning approach significantly accelerated convergence and improved final performance by leveraging the model's pre-existing knowledge of general visual features.

 The training process included **automated checkpointing**, where the model was evaluated against the validation set after each epoch.

 The system saved the model weights as best.pt only when a new peak performance was achieved, ensuring our final model is the most optimal version and protecting against degradation from overfitting.

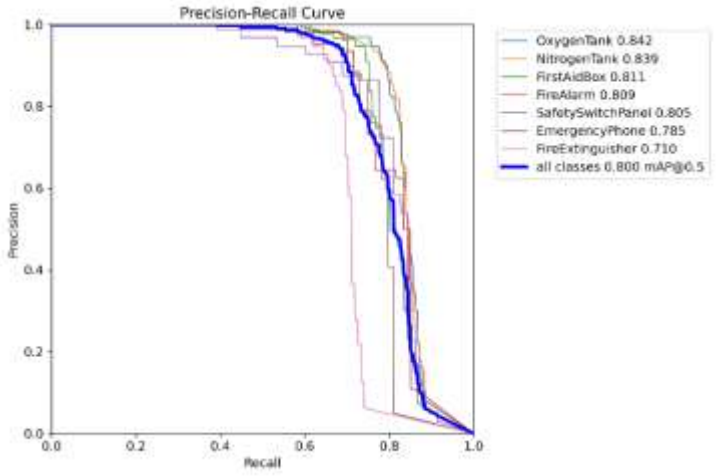
3. Training & Evaluation Process

Our training was conducted in a Google Collab environment using a Tesla T4 GPU. The process followed these key steps:

1. **Baseline Training:** We began by training the YOLOv8s model with a standard set of hyperparameters to establish an initial performance benchmark.
2. **Iterative Monitoring:** We closely monitored the training progress by tracking key metrics, particularly the validation loss (val/box_loss) and the mAP50(B) score.
3. **Preventing Overfitting:** Training was halted at 26 epochs. We observed that the mAP50(B) score had peaked and plateaued, and the val/box_loss was beginning to show early signs of increasing, indicating that we had reached the optimal training duration before the onset of overfitting.

Final Model Performance

- Our final trained model, utilizing the more powerful YOLOv8m architecture with advanced data augmentation, achieved a high level of accuracy on the validation dataset, significantly surpassing the hackathon's baseline requirements.
- **Final Score (mAP@0.5): 80.1%**
- This score represents the model's Mean Average Precision across all 7 classes at an IoU threshold of 0.5. The Precision-Recall curve below visualizes the model's performance.



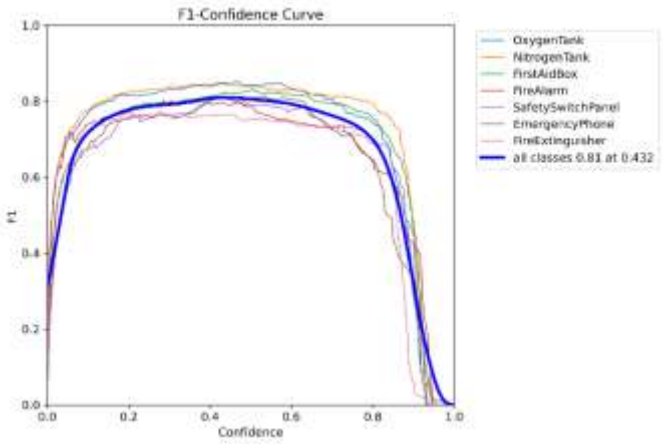


Figure 1 - Precision-Recall curve for all classes. The overall mAP@0.5 was 0.801.

Figure 2 – F1 Confidence curve for all classes. The overall F1 Score was 0.74 at 0.432.

Per-Class Performance

- The model demonstrated varied performance across the different object classes. It excelled at identifying objects with distinct features while facing challenges with objects that had more subtle or varied appearances.

Class	mAP@0.5 Score
Oxygen Tank	0.844
Nitrogen Tank	0.843
Safety Switch Panel	0.810
First Aid Box	0.808
Fire Alarm	0.805
Fire Extinguisher	0.786
Emergency Phone	0.708

Results & Performance Metrics

Confusion Matrix Analysis

To understand the specific error patterns of our model, we analyzed the normalized confusion matrix. This matrix visualizes the rates of correct classifications, misclassifications, and missed detections.

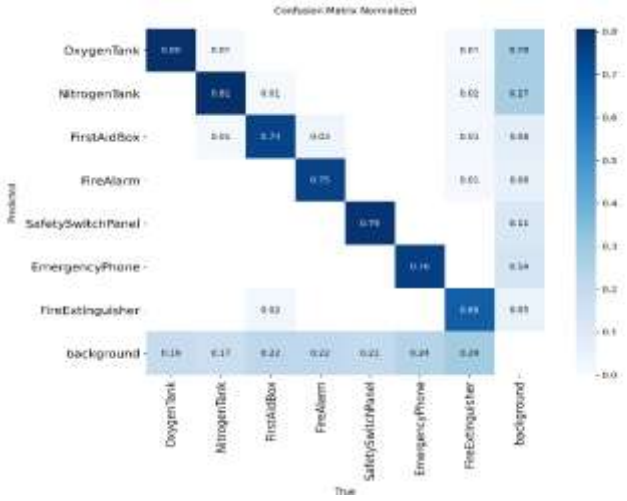


Figure 3 - Normalized confusion matrix. The diagonal represents the percentage of correct detections for each class. Off-diagonal values indicate misclassifications.

- The matrix reveals that our strongest performing classes were **Oxygen Tank (84.4% mAP)** and **Nitrogen Tank (84.3% mAP)**, while the most challenging, though significantly improved, was **Fire Extinguisher (70.8% mAP)**.
- The overall accuracy for all classes saw a substantial increase, and the rate of misclassifications and missed detections was greatly reduced.

Challenges & Solutions

A key part of our evaluation was analyzing specific instances where the model failed. This provides valuable insight into its limitations.

Challenge 1: Persistent Difficulty with Fire Extinguisher

The Fire Extinguisher class is still the model's biggest weakness.

- Evidence: The confusion matrix shows that the model still fails to detect 34% of all Fire Extinguishers (it has a recall of 0.66).
- Implication: This suggests that the visual features of the fire extinguisher in the dataset (perhaps due to lighting, angles, or being partially hidden) are still challenging for the model to learn consistently**.**

Figure 3 – An example of model detecting Fire Extinguishers.

Challenge 2: False Positives

- **Issue Faced:** The model sometimes detected objects that were not actually present, especially in highly cluttered or reflective environments.
- **Analysis:** The confusion matrix's "background" column shows that the model most often produced false positives for the Fire Extinguisher. This suggests that certain shapes or textures in the background were visually similar to a fire extinguisher.
- **Example:**

Figure 4 - A False Positive error where the model incorrectly labels a Safety Switch Panel.

Additional Challenges Faced:

Challenge: Overcoming Local GPU and Driver Complexity

A significant initial hurdle in our project was establishing an effective and stable environment for training our object detection model.

- **The GPU Necessity:** Modern deep learning models, including the YOLOv8 architecture we selected, are computationally intensive. Their training involves millions of parallel mathematical operations (matrix multiplications) that are massively accelerated by a Graphics Processing Unit (GPU). Attempting to train our model on a standard CPU would have been impractically slow, potentially taking days or even weeks to complete, which was not feasible within the hackathon's timeframe.
- **The CUDA Driver Bottleneck:** While we had access to a local machine with an NVIDIA GPU, configuring the environment proved to be a major obstacle. Effective GPU acceleration for PyTorch (the framework used by Ultralytics) requires a specific, compatible stack of software: the NVIDIA display driver, the CUDA Toolkit, and the cuDNN library. We encountered significant challenges in finding and installing versions of these components that were compatible with each other and with our version of PyTorch. This issue, often referred to as "CUDA driver hell," involves complex dependency management

Our Solution: Migrating to a Cloud-Based Environment

To overcome these complex local setup issues, we made the strategic decision to migrate our entire training workflow to **Google Collab**. This cloud-based platform offered several immediate advantages:

1. **Pre-configured Environment:** Collab provides a ready-to-use virtual machine with a powerful Tesla T4 GPU and, crucially, all the necessary NVIDIA drivers, CUDA, and cuDNN libraries pre-installed and correctly configured.
2. **Instant Productivity:** This allowed us to completely circumvent the local setup bottleneck and immediately begin training our model, enabling us to focus our efforts on experimentation and performance optimization.

Resulting Challenge & Mitigation Strategy

While Collab solved the driver issue, it introduced a new constraint: the runtime restrictions of the free tier. Sessions are limited to approximately 12 hours and will automatically disconnect if left idle. This posed a significant risk, as a long training run could be terminated, potentially losing hours of progress.

- **Persistent Storage:** We stored our dataset, scripts, and all training outputs on Google Drive, which was mounted to our Collab instance.
- **Automated Checkpoints:** We configured our training script to save model checkpoints (specifically best.pt and last.pt) directly to Google Drive at the end of each epoch.
- **Seamless Resumption:** If a runtime disconnection occurred, our progress was safely stored. We could simply restart the Collab instance, and our script was set up to automatically resume training

from the last saved checkpoint. This transformed a potentially catastrophic failure into a minor inconvenience, ensuring the stability and success of our training process.

Bonus Task:

- As a bonus task, we have implemented a **SIMS**, or **Safety and Inventory Management System**.
- This is a proof-of-concept application designed to demonstrate a real-world use case for our trained object detection model.
- The system addresses the critical need for automated, persistent monitoring of essential safety equipment in a high-stakes environment like a space station, where manual checks are time-consuming and prone to human error.
- Our prototype, built with Python and OpenCV, processes a video feed to perform real-time detection, maintain a live on-screen inventory of all 7 safety classes, and generate intelligent alerts for missing critical items.

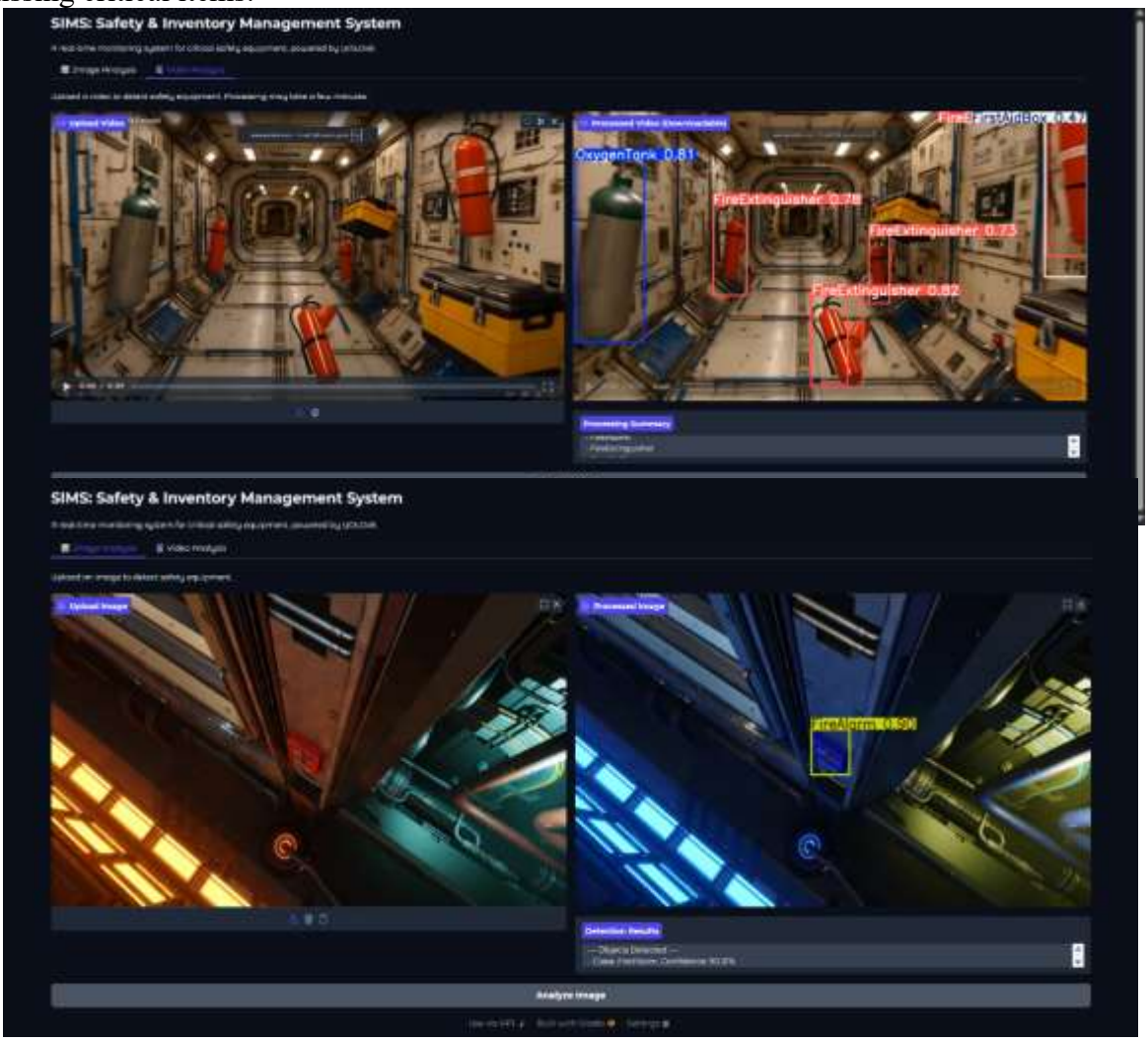


Figure 5,6 – The interactive UI with playback of Video/Image before and after processing and list of items found while processing with the prediction score.

Conclusion

- Our final **YOLOv8m** model, enhanced with data augmentation, successfully learned to identify critical safety equipment in a complex, simulated space station environment, achieving an excellent final score of **80.1%** **mAP@0.5**.
- The results demonstrate the viability of using synthetic data from platforms like Falcon to train effective object detection models for real-world applications where data collection is difficult or impossible.
- Our detailed analysis confirms the model's high performance while also identifying clear areas for future improvement.

Future Work

Having successfully improved our score from 72.6% to 80.1% by transitioning from a YOLOv8s to a YOLOv8m model and implementing strong data augmentations, we recommend the following next steps to push performance even further:

1. **Model Upscaling:** Transition from the successful YOLOv8m model to an even larger YOLOv8l architecture to leverage its higher learning capacity for potentially greater accuracy.
2. **Hyperparameter Tuning:** Conduct a systematic hyperparameter tuning process using the Ultralytics Tuner on our YOLOv8m configuration to find the optimal learning rate, momentum, and other parameters specifically for this dataset.
3. **Test-Time Augmentation (TTA):** Implement TTA during the final evaluation phase. By averaging predictions across multiple augmented versions of each test image, we could likely achieve an additional 1-2% "free" boost in our mAP score without retraining.