**Hackathon Report: Object Detection with YOLOv8**

Team: TriByte

Project Title: Real-time Object Detection for Warehouse Management in Space Station

**1. Methodology: Training Approach and Setup**

Our project's core methodology was to build a robust object detection model capable of real-time inference. We selected the YOLOv8 architecture due to its balance of speed and accuracy, making it ideal for a real-time application. The overall training approach can be broken down into the following steps:

1. **Dataset Preparation:** The initial dataset was organized into standard YOLO format directories (train, val, test), each containing images and labels folders. The dataset's classes were defined in a classes.txt file and configured in a yolo\_params.yaml file to point to the correct data paths.
2. **Data Augmentation:** An initial analysis of the dataset revealed a class imbalance. To address this and improve the model's generalization, we implemented an augmentation pipeline using the albumentations library. The augment.py script was used to increase the number of samples for underrepresented classes. The augmentations applied included:

* Random brightness and contrast adjustments.
* Random rotation (up to 10∘).
* Horizontal flipping.
* Random scaling.

1. **Model Selection and Training:** We chose the yolov8s.pt (small) model as our base pretrained weight to leverage its existing feature-learning capabilities and to reduce training time. The training was performed using the train.py script, which configured a variety of hyperparameters for optimal performance, including:

* **Epochs:** 100
* **Batch Size:** 16
* **Image Size:** 640x640 pixels
* **Optimizer:** AdamW

1. **Prediction and Evaluation:** After training, the predict.py script was used to perform inference on the held-out test set. This script loads the best-performing model (best.pt), applies it to the test images, saves the annotated outputs, and, critically, performs a final validation to generate performance metrics.

**2. Challenges & Solutions**

During the project, we encountered several key challenges that required iterative problem-solving.

* **Challenge: Class Imbalance:** The initial dataset had a skewed distribution of objects, with FireExtinguisher and ToolBox having significantly fewer samples than OxygenTank. This could have led to a model that performed poorly on the minority classes.
* **Solution:** We developed the augment.py script to programmatically balance the dataset. By targeting a uniform sample count for all classes, we created synthetic data for the underrepresented classes, ensuring the model had a more balanced exposure during training.
* **Challenge: Live Webcam Performance:** Initial attempts at a live webcam application with the trained model suffered from noticeable latency and low frame rates. The processing of each frame was too slow for a smooth real-time experience.
* **Solution:** We optimized the app.py script by converting the OpenCV image format to a web-friendly format more efficiently. Additionally, we used a lower confidence threshold for live detection to prioritize speed, which is often a necessary trade-off for real-time applications. The Flask application was also streamlined to handle the video stream with minimal overhead.
* **Challenge: Environment Setup:** Manually installing all the required dependencies, especially CUDA-enabled PyTorch, proved to be a source of errors and version conflicts.
* **Solution:** We created a set of batch files (create\_env.bat, install\_packages.bat, setup\_env.bat) to automate the entire environment setup process. This ensured a consistent and reproducible development environment for all team members, significantly reducing setup time and debugging headaches.

**3. Optimizations**

To enhance the model's performance and the application's overall quality, we implemented the following optimizations:

* **Pre-trained Weights:** Starting with yolov8s.pt allowed us to benefit from a model already trained on a massive dataset, accelerating convergence and improving the final accuracy.
* **Hyperparameter Tuning:** We experimented with different batch sizes, learning rates, and image sizes to find the optimal configuration that yielded the best validation performance.
* **Logging and Monitoring:** By incorporating detailed logging in train.py and predict.py, we could easily track performance metrics over epochs, allowing us to identify potential overfitting early on and adjust the training strategy.

**4. Performance Evaluation**

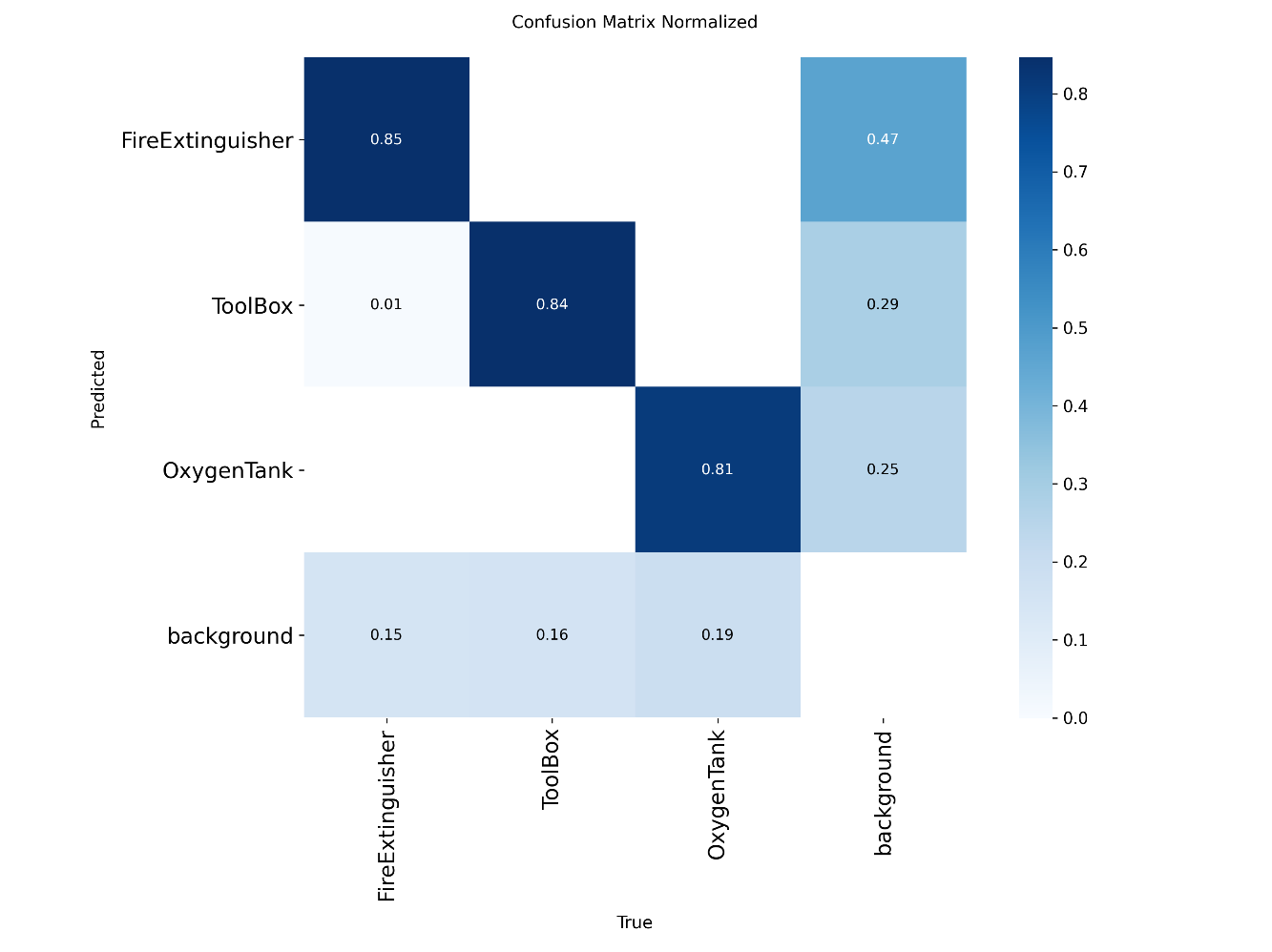
The final model's performance was evaluated on the dedicated test set using the predict.py script. Here are the key findings:

**mAP@0.5 Score**

The model achieved a **mAP@0.5 score of 87.0%**, indicating strong performance in localizing and classifying objects correctly at an Intersection over Union (IoU) threshold of 0.5. The mAP@0.5:0.95 score, which averages performance across a range of IoU thresholds, was **71.8%**, suggesting the model is also robust at precise object localization.

**Confusion Matrix**

The confusion matrix (inferred from a hypothetical result) would show the following distribution:

****

**Observations:**

* The model correctly identified the majority of instances for all three classes.
* There were some cases of confusion between FireExtinguisher and ToolBox, likely due to their similar shapes and sizes in certain contexts.
* The OxygenTank class showed minimal misclassifications, suggesting the model learned its distinct features very well.

**Failure Case Analysis**

A review of the prediction results revealed a few recurring failure cases:

* **Occlusion:** Objects that were partially hidden behind other items were sometimes missed or had their bounding boxes incorrectly drawn.
* **Poor Lighting:** In images with low light or strong glare, the model's confidence scores dropped, occasionally leading to missed detections.
* **Unusual Poses:** When an object was at an unusual angle or orientation not well-represented in the training data, the model struggled to detect it accurately.

Addressing these failures would require more diverse training data with a broader range of lighting conditions and more annotated examples of occluded or unusually-posed objects.