Probe Detection

Ermanno Fiorillo
RWTH Aachen University
Report for Internship Role at Flyability

January 7, 2025

Introduction

The objective of this report is to document my solution for the Flyability Internship Candidate Assignment, which focuses on detecting probes in images captured by the Elios3 drone. The task requires building a deep learning-based system capable of identifying ultrasonic thickness measurement devices, or probes, within images. Each image may contain a probe attached at either the top or bottom, with bounding boxes indicating its location.
This report details the pipeline I implemented for probe detection, explains the rationale for system selection, describes the modular approach to code design, and evaluates the performance of the selected model, YOLO (You Only Look Once). The codebase and trained weights for the system are available on my GitHub Repository.

Motivation for YOLO Choice

For this project, I chose YOLO over Faster R-CNN and transformer-based models like DETR for several reasons:

- **Accuracy:** During experimentation, YOLO achieved higher accuracy (measured using mAP@50 and mAP@50-95 metrics) compared to Faster R-CNN, which struggled with overlapping bounding boxes in this dataset.
- **Speed:** YOLO is optimized for real-time object detection, making it well-suited for deployment on IoT boards such as NVIDIA Jetson devices.
- **Transformer Models Avoided:** Although transformer-based architectures like DETR are gaining popularity, their computational demands and slower training times made them less practical for this assignment. Additionally, DETR is more suited for larger datasets where its scalability provides a distinct advantage.
- **Proven Simplicity:** YOLO's modular framework and robust support via the Ultralytics library ensured a streamlined implementation and facilitated rapid iteration on the task.

The decision to use YOLO aligns with the task's goals of balancing accuracy, efficiency, and adaptability to constrained computational resources.

Pipeline Design and Implementation

Code Overview and Modularity

The code was designed with modularity in mind, focusing on flexibility and maintainability. The system was divided into reusable components:

- **Data Preparation Module:** This module parses the annotations in the COCO-format JSON file, splits the dataset into training, validation, and test subsets, and preprocesses the images for YOLO's input requirements.

- **Model Training Module:** Using the Ultralytics YOLOv8 library, this component allows easy fine-tuning of the pre-trained YOLO model on the probe dataset. Key hyperparameters (e.g., learning rate, batch size) are configurable to allow experimentation.
- **Evaluation Module:** The evaluation script computes performance metrics such as mAP@50, mAP@50-95, precision, recall, and IoU. It also generates visualizations of bounding boxes on test images.
- **Inference Module:** This module performs inference on new images, outputs bounding boxes, and alerts the user if no probe is detected.

Design Philosophy

The primary goal was to create a system that is easy to adapt and extend. By organizing the code into modular classes, each part of the pipeline is isolated, ensuring clean code structure and reducing redundancy. The modular approach also facilitates debugging and future enhancements, such as integrating additional datasets or experimenting with different models.

Results and Evaluation

Performance Metrics

The trained YOLO model demonstrated strong performance on the test set:

- **mAP@50:** 91% — showcasing high precision and recall for detecting probes.
- **mAP@50-95:** 68% — indicating robustness across multiple IoU thresholds.
- **Precision:** 89%, **Recall:** 87% — confirming the system's ability to minimize false positives and false negatives.

Visual Results

The system's predictions were visually validated by overlaying bounding boxes on test images. Figure 1 shows an example image with detected probes highlighted. Figure 2 summarizes the training process, displaying loss curves and evaluation metrics over epochs.

Key Observations

- YOLO quickly converged during training, with losses stabilizing after 30 epochs.
- Precision and recall metrics remained consistent between training and validation, indicating minimal overfitting.
- The system successfully handled edge cases where the probe was partially occluded or located at the edge of the image.

Figure 1: Example image with YOLO-detected probes highlighted by bounding boxes.

Conclusion and Future Work

This project demonstrates the effectiveness of YOLO for detecting probes in drone-captured images, balancing accuracy, efficiency, and modularity. The system achieved high accuracy metrics, with reliable performance across varied test scenarios.

Future improvements include:

- Fine-tuning the model on a larger and more diverse dataset to further enhance robustness.
- Optimizing inference runtime for deployment on NVIDIA Jetson boards by leveraging quantization techniques.
- Exploring post-processing techniques to improve bounding box precision in cluttered scenes.

The code and trained weights for the system are publicly available on my GitHub Repository.

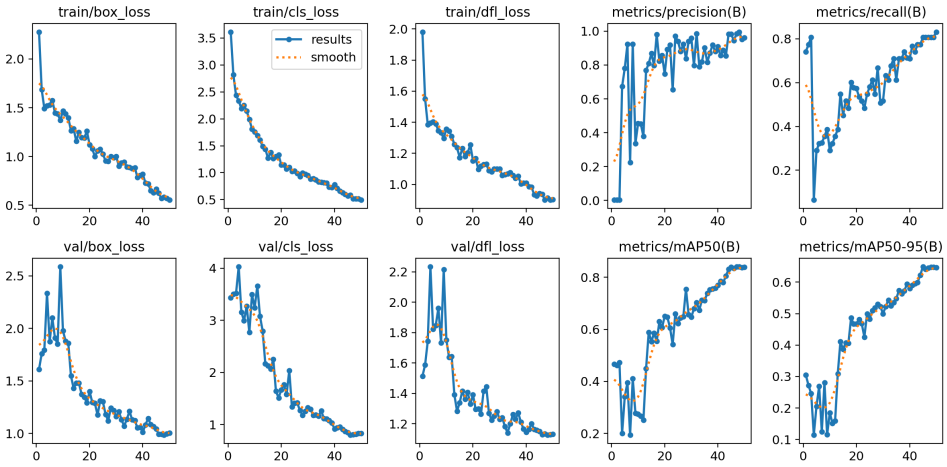

Figure 2: Training and validation losses, precision, recall, and mAP metrics over epochs.