Probe Detection Using Fine-Tuned Faster R-CNN

Axel Viselthier

January 5, 2025

Contents

1 Introduction and Context

Probes are essential tools used for ultrasonic thickness measurement in various industrial applications. Detecting these probes accurately in images taken by drones like the Elios3 is critical for automating inspection tasks.

This project focuses on developing a deep learning-based system to detect probes in images. The system takes an image as input and must output the bounding box of the probe/probes or notify if no probe is detected. This report outlines the steps taken to achieve this objective, including system selection, data preprocessing, training, evaluation, and future recommendations.

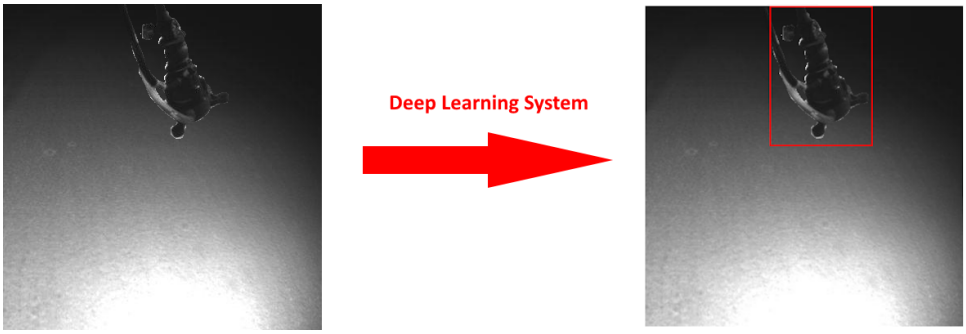

Figure 1: Illustration of the project goal: An unannotated image is processed by the model to detect probes in it.

2 System Selection : Faster R-CNN

I selected **Faster R-CNN with a ResNet-50 backbone** for probe detection. Faster R-CNN, introduced by Ren et al. in [1], is a robust and widely-used object detection model **capable of handling small objects in complex backgrounds**.

A key advantage of Faster R-CNN lies in its **region-based approach**. Even though the dataset contains exactly one probe per image, the model is inherently capable of detecting multiple or no probes. This is because Faster R-CNN evaluates multiple region proposals independently, making it **robust to variations in the number of probes per image, despite being trained on a dataset where each image consistently contains exactly one probe** .

I also considered alternative models such as YOLO (You Only Look Once), which is known for its speed and suitability for real-time applications. However, YOLO tends to struggle with small object detection in cluttered environments, as highlighted in comparative studies. Faster R-CNN, on the other hand, offers higher accuracy for such tasks at the cost of slightly increased inference time.

ResNet-50 was chosen as the backbone for its balance between computational efficiency and feature extraction capabilities. Its skip connections help mitigate the vanishing gradient problem, ensuring effective training with limited data. Additionally, PyTorch provides a pretrained Faster R-CNN implementation, with weights initialized from the COCO dataset, allowing for **fine-tuning on the provided dataset**.

An illustration of this architecture is shown in Figure 2.

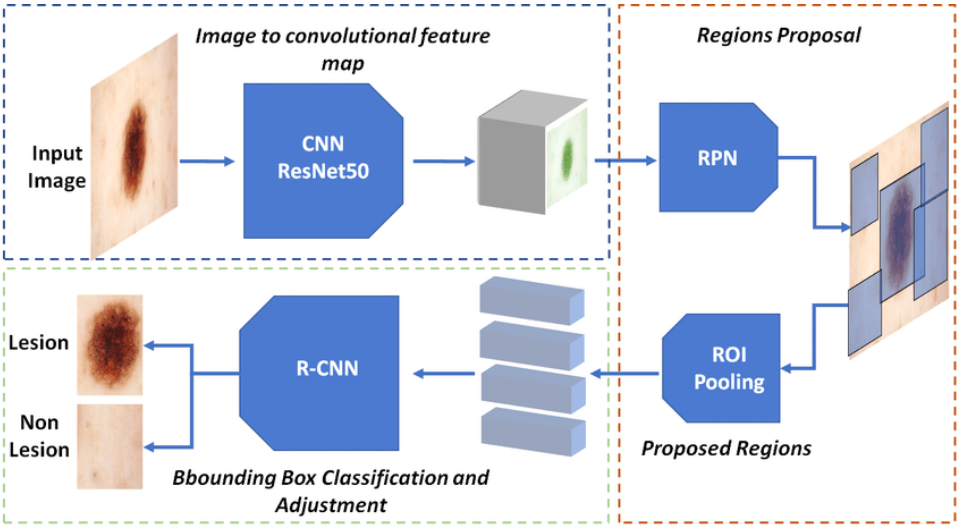

Figure 2: Faster R-CNN architecture: Top left box represents the base network, box on the right represents the region proposal network (RPN) and the bottom left box represents the R-CNN. Here, Faster R-CNN is used for lesion detection, the same principle will be applied for probe detection.

3 Data Preprocessing

The dataset already includes images and annotations specifying bounding boxes for probes. The preprocessing steps included:

- Resizing all images to 640x640 pixels.
- Normalizing pixel values.
- Splitting the dataset into 80% training and 20% validation sets.

4 Training and Hyperparameters

The Faster R-CNN model was fine-tuned using the following hyperparameters:

- **Learning Rate**: 0.005, reduced by a factor of 0.1 after 5 epochs.
- **Batch Size**: 2.
- **Number of Epochs**: 15.
- **Optimizer**: SGD with momentum 0.9 and weight decay 0.0005.

The model was trained to minimize the total loss, which includes both classification loss and bounding box regression loss.
The choice of 15 epochs and the learning rate schedule was driven by the nature of fine-tuning. The model demonstrated **rapid convergence** in early epochs due to its pretrained weights. Reducing the learning rate after 5 epochs allowed for more precise adjustments to the weights, avoiding overshooting and improving final accuracy. This strategy ensures the model adapts effectively to the specific characteristics of the dataset while leveraging prior knowledge from pretraining.

5 Evaluation and Results

5.1 Performance Metrics

The system was evaluated using Intersection over Union (IoU) and its distribution across validation samples. The results are summarized below:

- **Mean IoU on Validation Dataset: 0.9241**.
- **IoU Distribution on Validation Dataset**: Over 95% of validation samples achieved an IoU above 0.8, indicating **highly consistent performance**.

Given these high IoU scores, additional metrics such as precision and recall were deemed less informative for this specific task. **The exceptional IoU results demonstrate the model's accuracy in localizing probes**, validating the effectiveness of the chosen architecture and training strategy.

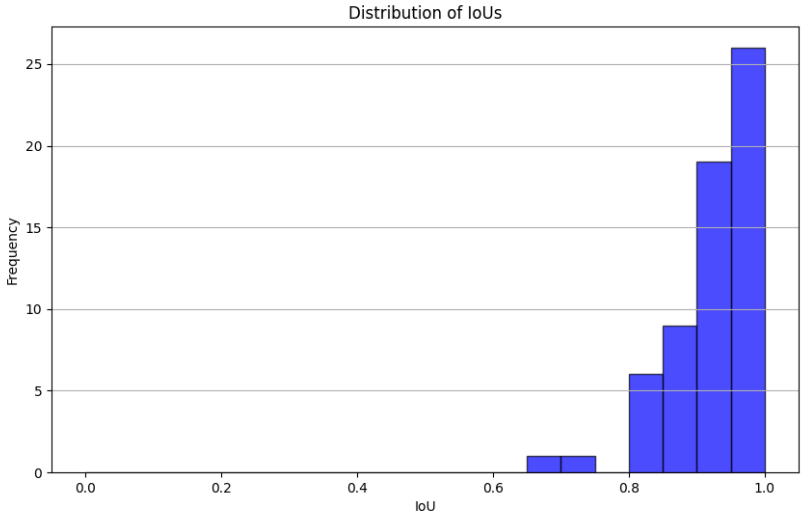

Figure 3: IoU distribution across validation samples: Over 95% of samples achieved IoU above 0.8.

5.2 Visualizations

Sample bounding box predictions are shown in Figure 4. Green boxes indicate predictions, while blue boxes represent ground truth.

Figure 4: Sample bounding box predictions.

5.3 Analysis

Given Flyability's products (Drones for indoor inspection), it is interesting to assess if the model can be used both in real time and on an actual IoT board. Based on the resource usage metrics gathered during evaluation, the feasibility of deploying the model on specific hardware platforms for video is assessed as follows:

- **Deployment on Drones for Real-Time Analysis**: The model's inference speed on a T4 GPU achieves 8.88 images per second, translating to approximately 113 milliseconds per image. This makes real-time analysis viable for drones operating in environments with moderate processing demands. However, memory constraints must be considered for drones with limited onboard computational resources.
- **IoT Boards such as NVIDIA Jetson Family**: The memory usage of 158.05 MB for the model size and peak GPU memory of approximately 986.44 MB suggests that deployment on devices like the Jetson Xavier NX or AGX Xavier is feasible. However, the Jetson Nano, with 2 GB VRAM and 4 GB RAM, would require further optimizations such as model quantization or pruning to ensure compatibility. These further optimizations will be discussed in the next section.

Resource	**Required**	**Jetson Xavier NX**	**Jetson Nano**
Peak GPU Memory	986.44 MB	6 GB	2 GB
RAM Usage	2256.30 MB	8 GB	4 GB

Table 1: Resource requirements compared to Jetson Xavier NX and Nano capabilities.

6 Future Improvements and Enhancements

Although the system has demonstrated excellent performance with high IoU scores and robust resource usage, further enhancements can definitely improve its generalization, robustness, and compatibility with constrained hardware. These optimizations were considered in my workflow but not implemented, as the model already achieved exceptional results.

6.1 Performance Enhancements

- **Data Augmentation**: Applying advanced data augmentation techniques can improve the model's robustness to **variations in probe orientation, size, and lighting conditions**. Libraries such as `Albumentations` provide efficient tools for augmenting data with transformations like rotation, contrast adjustment, and noise addition, which could enhance generalization and ensure consistent performance across diverse environments.

- **Model Ensemble Strategy**: Instead of relying solely on a single detection model like Faster R-CNN, **combining multiple models** such as SSD (Single Shot Multibox Detector) and YOLO (You Only Look Once) can create a robust ensemble. This approach leverages the strengths of each model: SSD and YOLO are faster and well-suited for real-time applications, while Faster R-CNN provides higher accuracy, especially for detecting small objects in cluttered backgrounds.

  By using a weighted ensemble or decision fusion, the system can adapt dynamically to different scenarios, achieving a balance between speed and accuracy.

6.2 Resource Optimizations for Deployment

To ensure the model is efficiently deployable on resource-constrained platforms such as IoT boards or drones, the following optimizations can be implemented:

- **Quantization**: Quantization reduces the precision of model parameters from 32-bit floating point (FP32) to lower bit-widths like 16-bit or even 8-bit integers. This significantly reduces memory requirements and improves inference speed without substantial loss in accuracy.

- **TensorRT Optimization**: TensorRT, an SDK from NVIDIA, can further accelerate inference by optimizing the model's execution for the target hardware. It also reduces latency, making the system suitable for real-time video analysis on drones.

- **Model Pruning**: Pruning eliminates redundant weights and neurons from the model while preserving its performance. Structured pruning, where entire channels or layers are removed, is particularly effective for deployment on hardware with strict memory and computational limits. This can lead to a lighter model with faster inference speeds, especially beneficial for less powerful devices like Jetson Nano.

7 Conclusion

This project demonstrated the feasibility of using a quantized Faster R-CNN model for probe detection. The system achieved excellent performance with a mean IoU of 0.9241. The system demonstrated feasibility for deployment on industrial platforms with advanced IoT boards. Further optimizations, such as Data Augmentation, Quantization and model pruning, can still enhance its robustness and practicality for industrial applications.

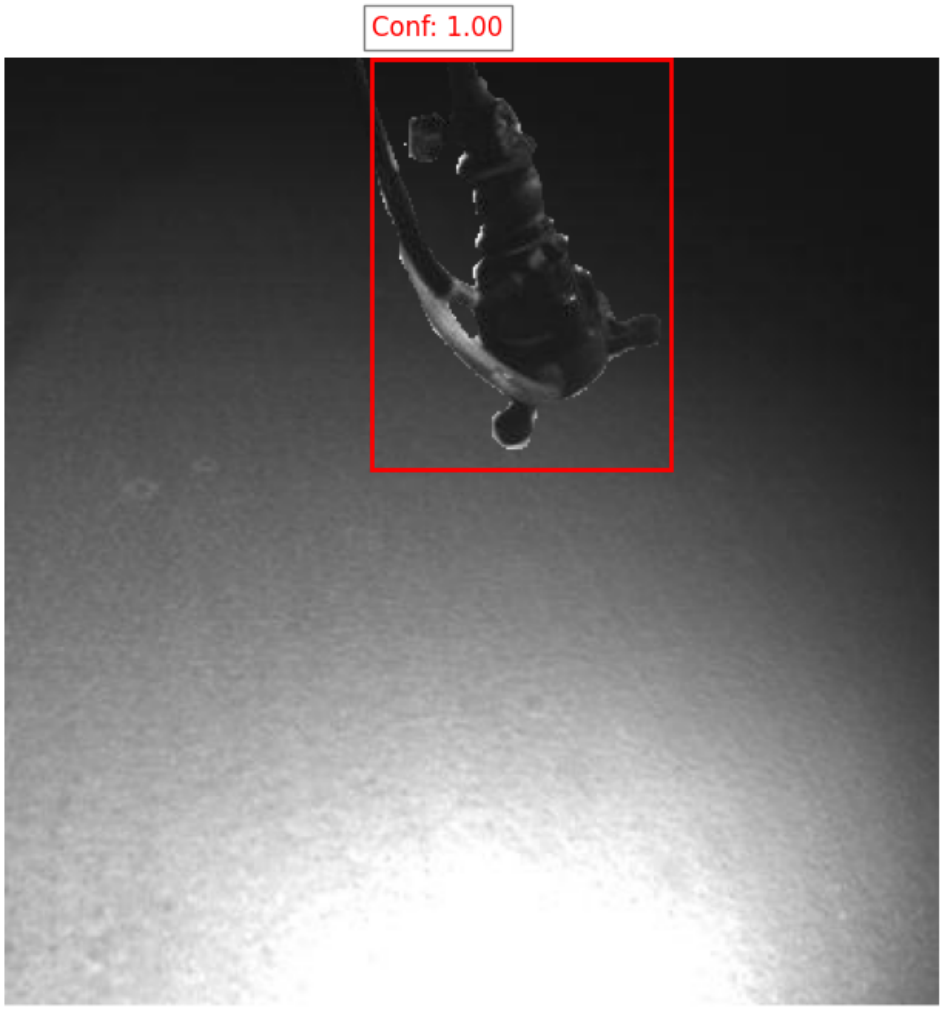

Figure 5: Example of a prediction of the model on a customed image that wasn't in the provided dataset.

8 References

1. Ren, S., He, K., Girshick, R., & Sun, J. (2015). Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence.*