Prob Detection Using Deep Learning (YOLOv5)

1. Introduction

- **Objective**: The goal of this project is to build a deep learning-based system to detect probes in images captured by an Elios3 drone. The system takes an image as input and outputs the bounding box of the detected probe or notifies if no probe is present.
- **Tools and Technologies**:
    - YOLOv5 for object detection.
    - Python (Anaconda environment).
    - PyTorch as the deep learning framework.
    - Visual Studio Code for code development.
    - Git for version control.

2. Dataset

- **Dataset Description**:
    - Images captured by the Elios3 drone containing ultrasonic thickness measurement probes.
    - Dataset includes annotations in JSON format specifying bounding box coordinates for the probes.
- **Dataset Statistics**:
    - Number of images: **308**.
    - Train/Validation/Test split:
        - Training: 246 images.
        - Validation: 31 images.
        - Test: 31 images.
- **Preprocessing Steps**:
    - Converted JSON annotations to YOLO format. (view_json.py)
    - Visualized bounding boxes for verification. (visualize_data.py)
    - Split data into train, val, and test subsets (80%,10%,10% split). (split_dataset.py)

Assignment Tasks

1. System Selection and Setup

Selected System: I selected **YOLOv5 (You Only Look Once Version 5)** for this project. YOLOv5 is a state-of-the-art object detection framework known for its efficiency, speed, and accuracy. It is optimized for edge devices and is well-documented, making it suitable for detecting probes in images.

Justification: YOLOv5 was chosen because:

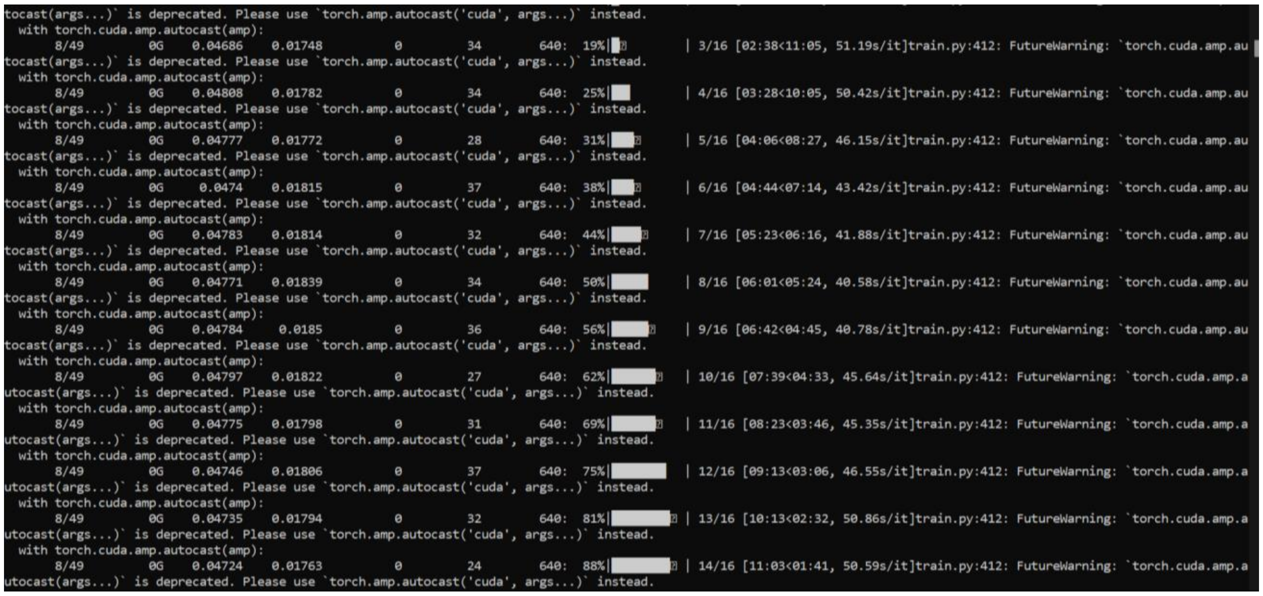
1. It is lightweight and supports real-time object detection.
2. It offers pre-trained weights for transfer learning, which reduces training time.
3. Its modular design allows for easy integration and fine-tuning for specific tasks like probe detection.
4. It has built-in support for PyTorch, a robust deep learning framework, and can be deployed on IoT devices like NVIDIA Jetson boards.

2. Training

- **Dataset**:
  - The dataset consists of images annotated with bounding boxes around the probes.
  - The dataset was split into 80% training, 10% validation, and 10% testing subsets.
- **Training Setup**:
  - **Model**: YOLOv5s (small version).
  - **Pre-trained Weights**: yolov5s.pt.
  - **Batch Size**: 16.
  - **Image Size**: 640x640 pixels.
  - **Epochs**: 50.
  - **Learning Rate**: Default as defined in YOLOv5's hyperparameters.
  - **Save Period**: Every 5 epochs.

➢ Start training using the following command

```
python yolov5/train.py --data src/probe_detection.yaml --cfg yolov5s.yaml --weights yolov5s.pt --epochs 50 --batch-size 16 --project results/train --name exp
```



```
tocast(args...)` is deprecated. Please use `torch.amp.autocast('cuda', args...)` instead.
  with torch.cuda.amp.autocast(amp):
       8/49         0G    0.04686    0.01748          0         34        640:  19%|█         | 3/16 [02:38<11:05, 51.19s/it]train.py:412: FutureWarning: `torch.cuda.amp.au
tocast(args...)` is deprecated. Please use `torch.amp.autocast('cuda', args...)` instead.
  with torch.cuda.amp.autocast(amp):
       8/49         0G    0.04808    0.01782          0         34        640:  25%|██        | 4/16 [03:28<10:05, 50.42s/it]train.py:412: FutureWarning: `torch.cuda.amp.au
tocast(args...)` is deprecated. Please use `torch.amp.autocast('cuda', args...)` instead.
  with torch.cuda.amp.autocast(amp):
       8/49         0G    0.04777    0.01772          0         28        640:  31%|███       | 5/16 [04:06<08:27, 46.15s/it]train.py:412: FutureWarning: `torch.cuda.amp.au
tocast(args...)` is deprecated. Please use `torch.amp.autocast('cuda', args...)` instead.
  with torch.cuda.amp.autocast(amp):
       8/49         0G     0.0474    0.01815          0         37        640:  38%|███       | 6/16 [04:44<07:14, 43.42s/it]train.py:412: FutureWarning: `torch.cuda.amp.au
tocast(args...)` is deprecated. Please use `torch.amp.autocast('cuda', args...)` instead.
  with torch.cuda.amp.autocast(amp):
       8/49         0G    0.04783    0.01814          0         32        640:  44%|████      | 7/16 [05:23<06:16, 41.88s/it]train.py:412: FutureWarning: `torch.cuda.amp.au
tocast(args...)` is deprecated. Please use `torch.amp.autocast('cuda', args...)` instead.
  with torch.cuda.amp.autocast(amp):
       8/49         0G    0.04771    0.01839          0         34        640:  50%|█████     | 8/16 [06:01<05:24, 40.58s/it]train.py:412: FutureWarning: `torch.cuda.amp.au
tocast(args...)` is deprecated. Please use `torch.amp.autocast('cuda', args...)` instead.
  with torch.cuda.amp.autocast(amp):
       8/49         0G    0.04784     0.0185          0         36        640:  56%|█████     | 9/16 [06:42<04:45, 40.78s/it]train.py:412: FutureWarning: `torch.cuda.amp.au
tocast(args...)` is deprecated. Please use `torch.amp.autocast('cuda', args...)` instead.
  with torch.cuda.amp.autocast(amp):
       8/49         0G    0.04797    0.01822          0         27        640:  62%|██████    | 10/16 [07:39<04:33, 45.64s/it]train.py:412: FutureWarning: `torch.cuda.amp.a
utocast(args...)` is deprecated. Please use `torch.amp.autocast('cuda', args...)` instead.
  with torch.cuda.amp.autocast(amp):
       8/49         0G    0.04775    0.01798          0         31        640:  69%|██████    | 11/16 [08:23<03:46, 45.35s/it]train.py:412: FutureWarning: `torch.cuda.amp.a
utocast(args...)` is deprecated. Please use `torch.amp.autocast('cuda', args...)` instead.
  with torch.cuda.amp.autocast(amp):
       8/49         0G    0.04746    0.01806          0         37        640:  75%|███████   | 12/16 [09:13<03:06, 46.55s/it]train.py:412: FutureWarning: `torch.cuda.amp.a
utocast(args...)` is deprecated. Please use `torch.amp.autocast('cuda', args...)` instead.
  with torch.cuda.amp.autocast(amp):
       8/49         0G    0.04735    0.01794          0         32        640:  81%|████████  | 13/16 [10:13<02:32, 50.86s/it]train.py:412: FutureWarning: `torch.cuda.amp.a
utocast(args...)` is deprecated. Please use `torch.amp.autocast('cuda', args...)` instead.
  with torch.cuda.amp.autocast(amp):
       8/49         0G    0.04724    0.01763          0         24        640:  88%|████████  | 14/16 [11:03<01:41, 50.59s/it]train.py:412: FutureWarning: `torch.cuda.amp.a
utocast(args...)` is deprecated. Please use `torch.amp.autocast('cuda', args...)` instead.
```

Figure 1 Training Process

3. Evaluation

1. Metrics

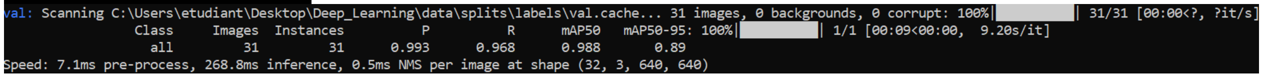
Evaluated on both **validation** and **test** sets with:

- **Precision**: $\frac{True\ Positive}{True\ Positive+False\ Positive}$
- **Recall**: $\frac{True\ Positive}{True\ Positive+False\ Negative}$
- **mAP@0.5**: Average precision at an IoU threshold of 0.5.
- **mAP@0.5:0.95**: Mean average precision across IoU thresholds 0.5 to 0.95.

Validation Results

➢ validate using the following command

```
python yolov5/val.py --weights results/train/exp/weights/best.pt –data
src/probe_detection.yaml --batch-size 16 --imgsz 640 --project results/val --name exp_val
--save-txt --save-conf
```



```
val: Scanning C:\Users\etudiant\Desktop\Deep_Learning\data\splits\labels\val.cache... 31 images, 0 backgrounds, 0 corrupt: 100%|██████████| 31/31 [00:00<?, ?it/s]
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100%|██████████| 1/1 [00:09<00:00,  9.20s/it]
                   all         31         31      0.993      0.968      0.988       0.89
Speed: 7.1ms pre-process, 268.8ms inference, 0.5ms NMS per image at shape (32, 3, 640, 640)
```

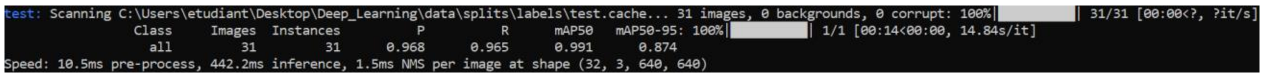
Figure 2 Validation Process

- Precision: 0.993
- Recall: 0.968
- mAP@0.5: 0.988
- mAP@0.5:0.95: 0.89

Test Results

➢ test using the following command

```
python yolov5/val.py --weights results/train/exp/weights/best.pt --data src/probe_detection.yaml -
-task test --batch-size 16 --imgsz 640 --project results/test --name exp_test --save-txt --save-conf
```



```
test: Scanning C:\Users\etudiant\Desktop\Deep_Learning\data\splits\labels\test.cache... 31 images, 0 backgrounds, 0 corrupt: 100%|██████████| 31/31 [00:00<?, ?it/s]
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100%|██████████| 1/1 [00:14<00:00, 14.84s/it]
                   all         31         31      0.968      0.965      0.991      0.874
Speed: 10.5ms pre-process, 442.2ms inference, 1.5ms NMS per image at shape (32, 3, 640, 640)
```

Figure 3 Test Process

- Precision: 0.968
- Recall: 0.965
- mAP@0.5: 0.991
- mAP@0.5:0.95: 0.874

2. **Inference Runtime**
   - **Pre-processing Time**: 10.5ms per image.
   - **Inference Time**: 442.2ms.
   - **NMS (Non-Maximum Suppression)**: 1.5ms per image.

Explanation for Visualizations

Below is a detailed explanation of each visualization, this will be for Train, Validation and Test

Train

1. **Confusion Matrix**

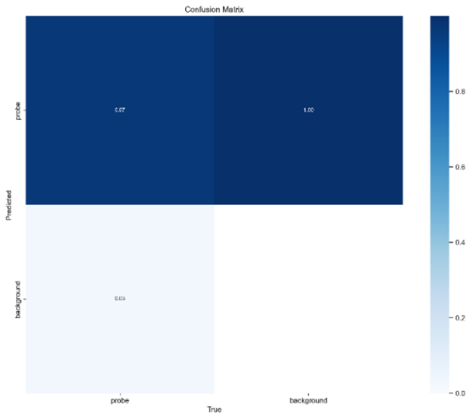


Figure 4 Confusion Matrix

- **Description**:
  - The confusion matrix shows the classification performance of the model.
  - The axes represent the predicted and actual classes (probe and background).
- **Interpretation**:
  - High diagonal values (0.97 for probe and 1.0 for background) indicate that the model is correctly identifying both classes.
  - The off-diagonal value (0.03) indicates a minor misclassification rate for probes classified as background.

- **Conclusion**:
  - The model achieves excellent classification performance with minimal misclassification.

2. F1-Confidence Curve

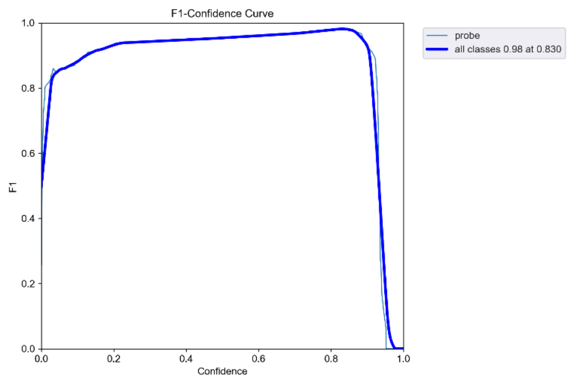


Figure 5 F1-Confidence Curve

- **Description**:
  - This curve demonstrates the relationship between the model's F1 score and the confidence threshold used for predictions.
- **Interpretation**:
  - The F1 score peaks at **0.98** at a confidence threshold of **0.83**, suggesting that the model performs best at this threshold.
- **Conclusion**:
  - The model is optimized for detecting probes with a high balance of precision and recall at this confidence level.

3. Labels Correlogram

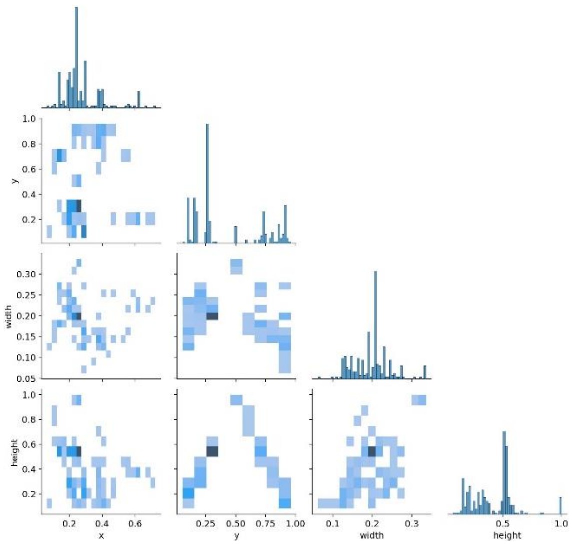


Figure 6 Labels Correlogram

- **Description**:
  - A heatmap showing the distribution and correlations of bounding box parameters (x, y, width, height).
- **Interpretation**:
  - The densest regions of the heatmap indicate common bounding box locations and sizes in the dataset.
  - Strong correlations suggest consistent patterns in probe placement and dimensions.
- **Conclusion**:
  - The bounding box distributions are well-aligned, supporting consistent training results.

4. Precision-Confidence Curve

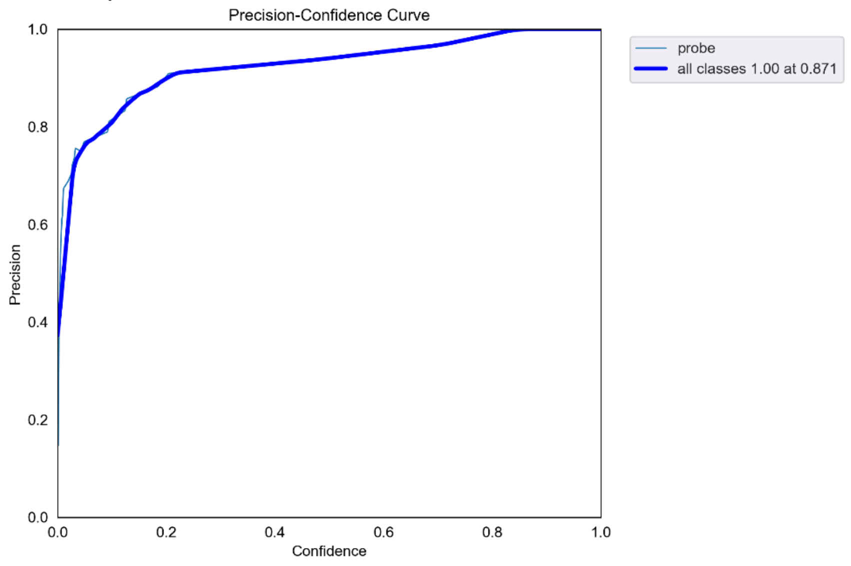


Figure 7 Precision-Confidence Curve

- **Description**:
  - This curve illustrates the relationship between precision and confidence thresholds.
- **Interpretation**:
  - Precision remains high across all confidence levels, peaking at **1.00** at a confidence threshold of **0.871**.
- **Conclusion**:
  - The model is highly confident in detecting probes without many false positives.

5. Precision-Recall (PR) Curve

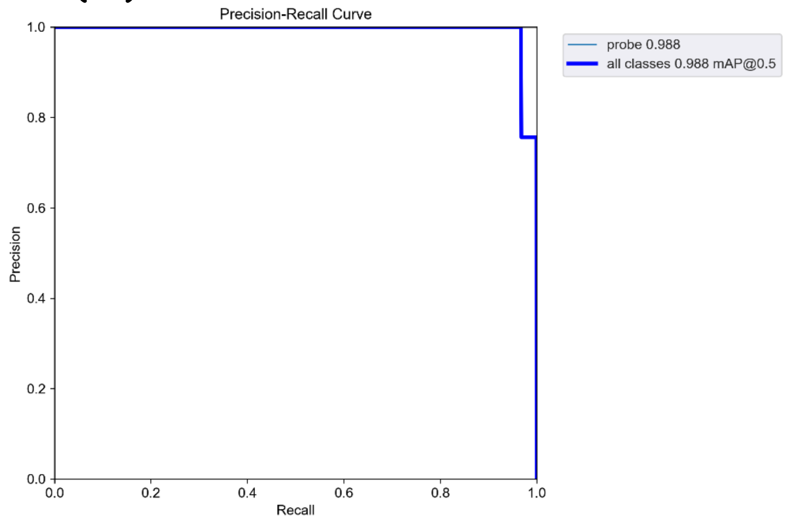


Figure 8 Precision-Recall (PR) Curve

- **Description**:
  - The PR curve shows the trade-off between precision and recall.
- **Interpretation**:
  - The curve achieves a high precision-recall balance, with an mAP@0.5 score of **0.988**.
  - The sharp rise and flat plateau indicate excellent precision-recall performance.
- **Conclusion**:
  - The model is highly effective in detecting probes with minimal trade-offs.

6. Recall-Confidence Curve

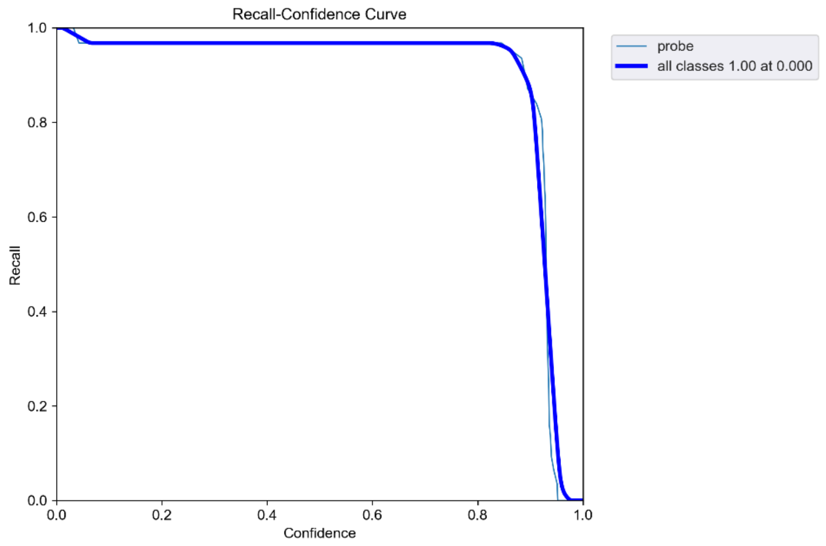


Figure 9 Recall-Confidence Curve

- **Description**:
  - This curve shows the relationship between recall and confidence thresholds.
- **Interpretation**:
  - Recall remains near perfect at lower confidence thresholds and slightly drops as the threshold increases.
- **Conclusion**:
  - The model maintains a high recall rate, ensuring minimal false negatives.

7. Training and Validation Metrics

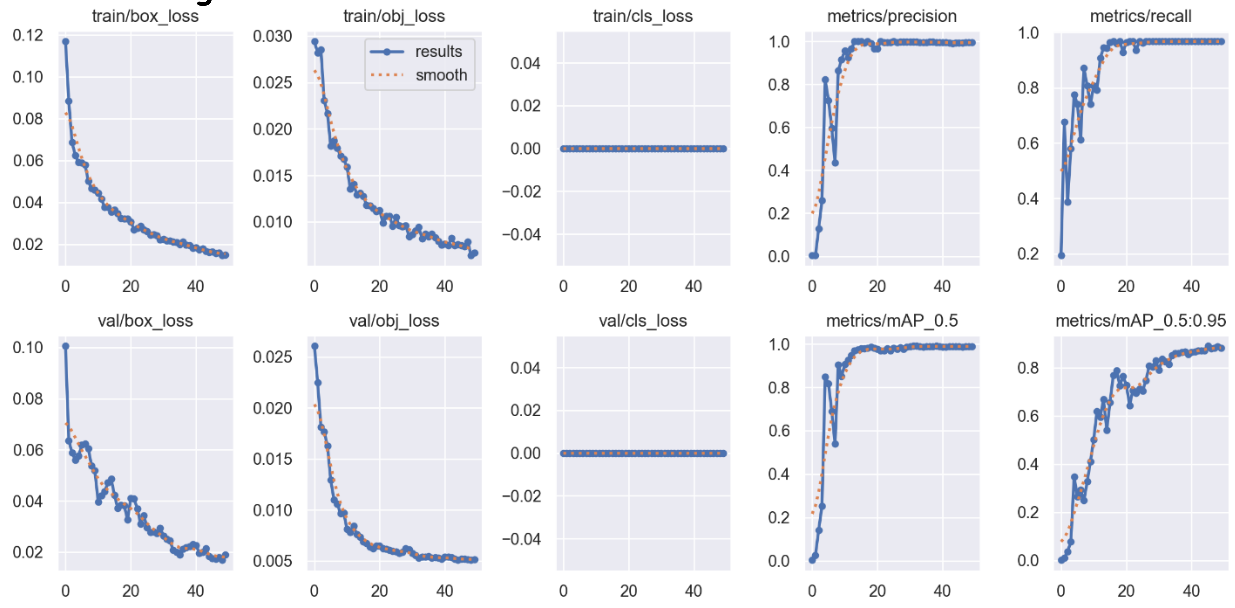


Figure 10 Training and Validation Metrics

- **Description**:
  - The grid of plots shows the progression of various metrics (e.g., loss, precision, recall, mAP) over the training epochs.
- **Interpretation**:
  - Training and validation losses decrease steadily, indicating successful learning.
  - Precision, recall, and mAP values improve and stabilize, reflecting convergence.
- **Conclusion**:
  - The model is well-trained, with minimal overfitting and strong generalization to validation data.

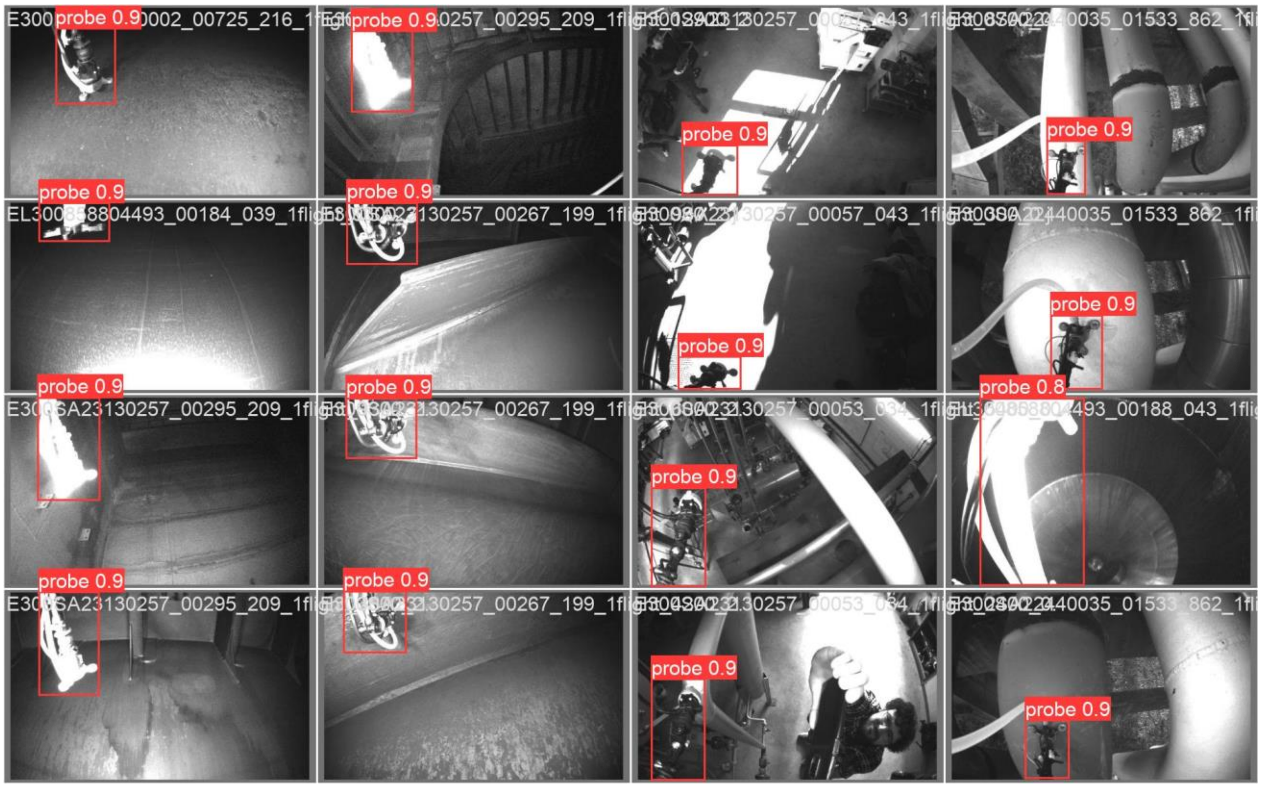


Figure 11 Validated Images

Validation

1. F1-Confidence Curve

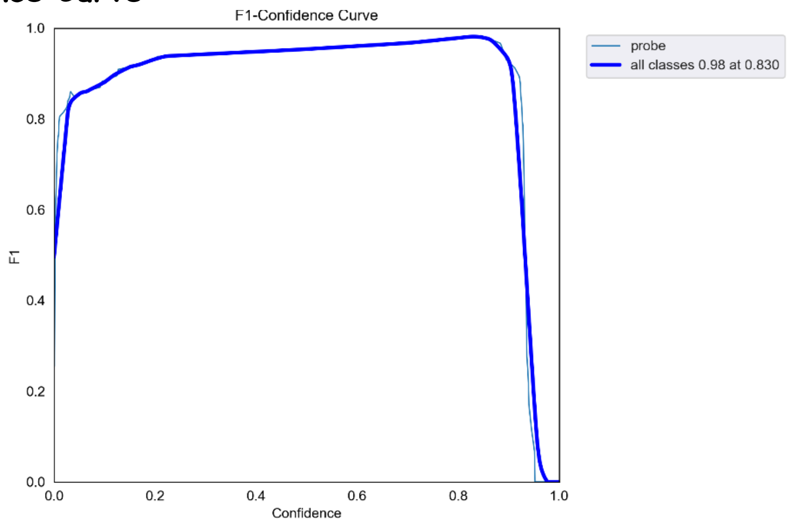


Figure 12 F1-Confidence Curve

- **Observation:** The F1 score peaks at **0.98** at a confidence threshold of **0.83**. This demonstrates that the model maintains a high balance between precision and recall on validation data.

- **Significance:** The high F1 score indicates that the model is effective in detecting probes with minimal false positives and false negatives, even on unseen data. This is critical for robust generalization.

2. Precision-Confidence Curve

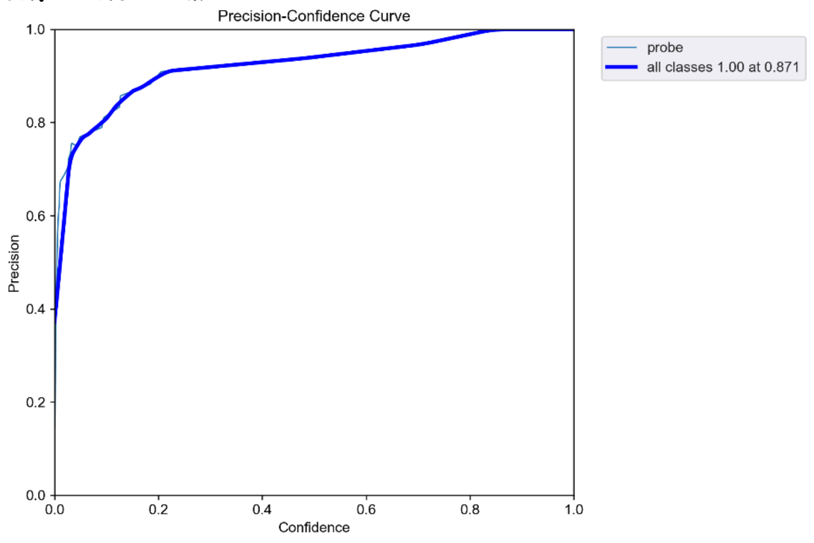


Figure 13 Precision-Confidence Curve

- **Observation:** Precision reaches **1.00** at a confidence threshold of **0.87**, demonstrating that the model is highly accurate when it assigns a positive detection.
- **Significance:** A high precision ensures that most detections made by the model are true positives, reducing the likelihood of false alarms in practical applications.

3. Precision-Recall Curve

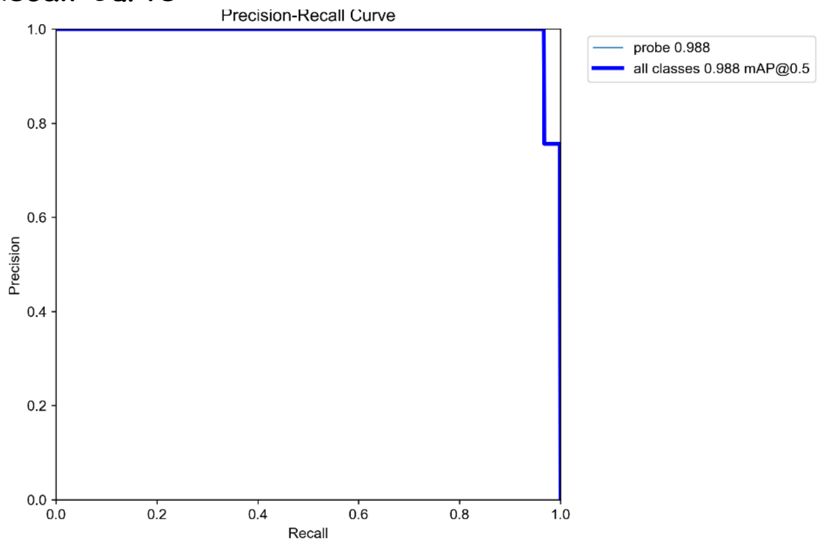


Figure 14 Precision-Recall Curve

- **Observation:** The **mAP@0.5** for validation is **0.988**, signifying consistent precision and recall across different IoU thresholds.
- **Significance:** A high mAP value reflects the model's robustness and capability to perform well across varying object sizes and shapes in the validation dataset.

4. Recall-Confidence Curve

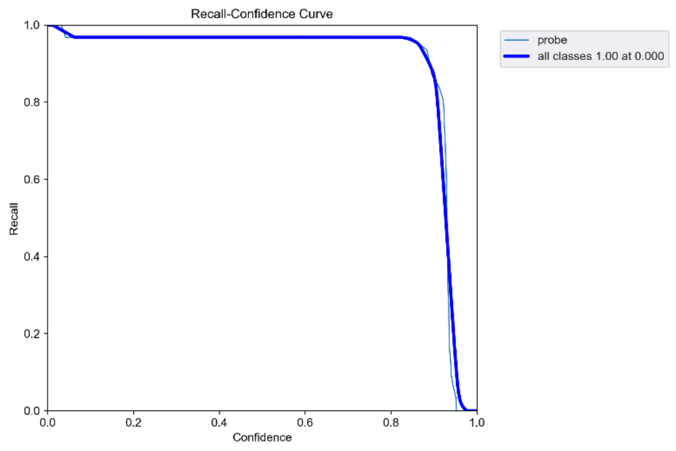


Figure 15 Recall-Confidence Curve

- **Observation:** Recall reaches **1.00** at low confidence thresholds. This shows the model's ability to detect all positive instances, though it may include some false positives at lower thresholds.
- **Significance:** High recall is crucial for applications where missing a detection is critical, as it ensures that all potential instances are flagged for further inspection.

Test

1. F1-Confidence Curve

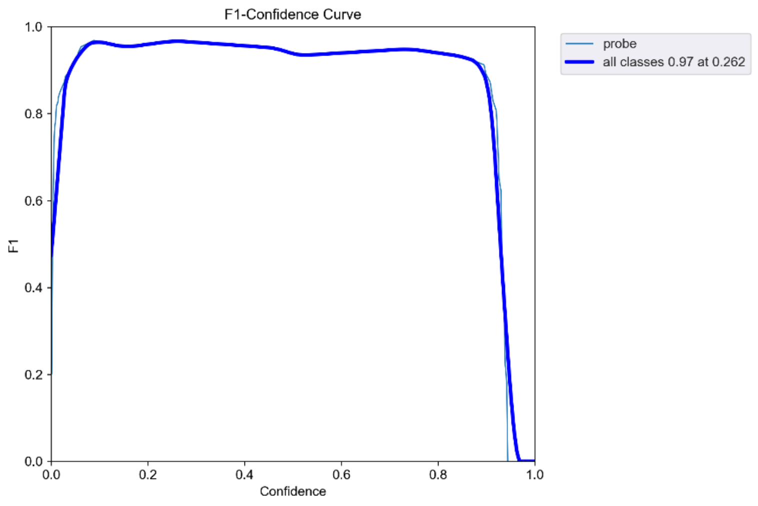


Figure 16 F1-Confidence Curve

- **Observation**:
  - The F1 score reaches a peak value of 0.97 at a confidence threshold of approximately 0.262.
  - This suggests that the model performs well with a high balance of precision and recall for this confidence range.
- **Significance**:
  - A high F1 score reflects the model's effectiveness in balancing precision and recall, confirming its reliability in detecting objects accurately in the validation data.

2. Precision-Confidence Curve

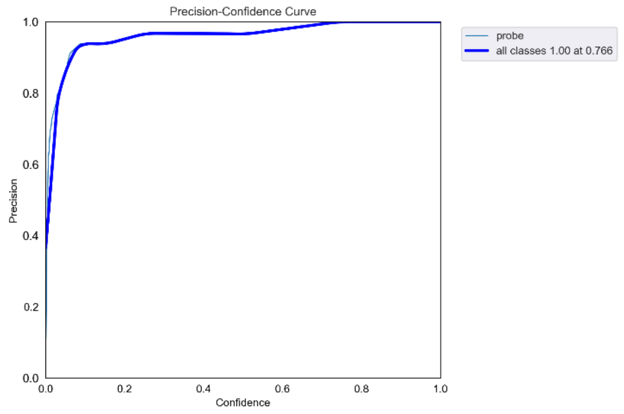


Figure 17 Precision-Confidence Curve

- **Observation**:
  - The precision remains very high, reaching a maximum of 1.0 at a confidence threshold of 0.766.
  - This indicates that at this confidence level, all detections made by the model are correct (no false positives).
- **Significance**:
  - The model's precision is excellent at higher confidence levels, ensuring minimal false positives in predictions.

3. Precision-Recall (PR) Curve

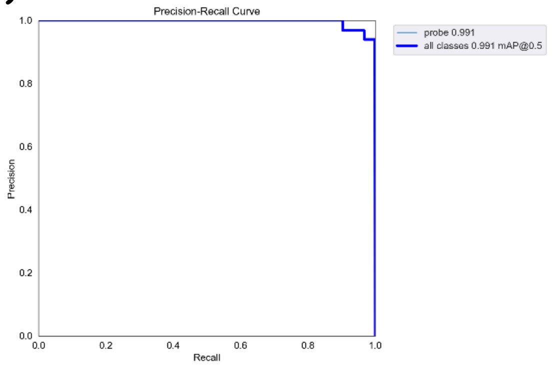


Figure 18 Precision-Recall (PR) Curve

- **Observation**:
  - The curve is almost perfect, with a precision of 0.991 at recall values near 1.0.
  - The mean Average Precision (mAP@0.5) is also reported as 0.991, indicating very high accuracy in the validation dataset.
- **Significance**:
  - The model successfully balances precision and recall across all confidence thresholds, signifying robust performance.

4. Recall-Confidence Curve

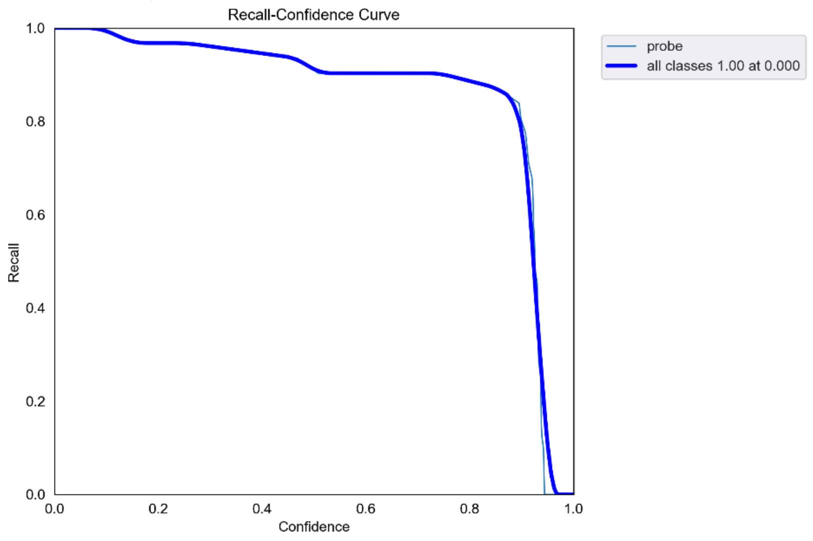


Figure 19 Recall-Confidence Curve

- **Observation**:
  - Recall remains high at lower confidence thresholds and starts to decrease as the confidence threshold approaches 1.0.
  - The maximum recall achieved is 1.0 at lower confidence thresholds.
- **Significance**:
  - High recall indicates that the model is effective at detecting almost all relevant instances in the validation data at lower confidence levels.

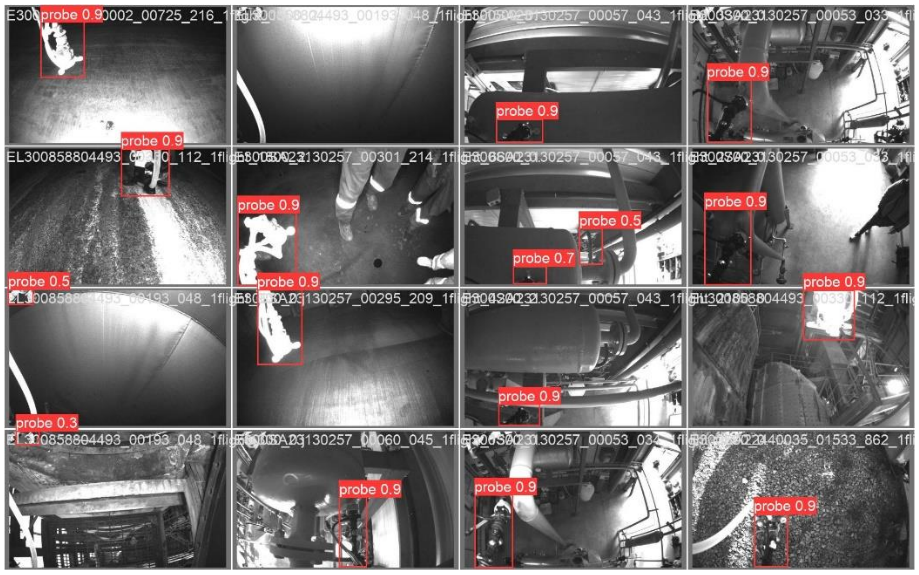


Figure 20 Tested Images

- Using the following command, the detection of prob is tested

```
python yolov5/detect.py --weights results/train/exp/weights/best.pt –source
data/splits/images/test --save-crop --project results/test --name exp_detect
```

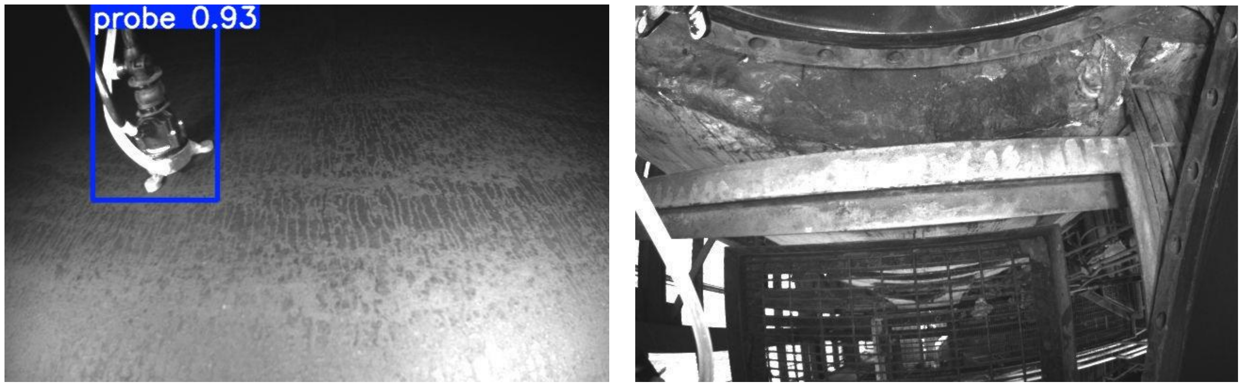


Figure 21 Detected Prob and no detections

Comparison Between Training, Validation and Test Results

Metric	Training	Validation	Test
F1 Score	0.98 at 0.83	0.98 at 0.83	0.97 at 0.262
Precision (Confidence Curve)	1.00 at 0.871	1.00 at 0.87	1.00 at 0.766

Recall (Confidence Curve)	1.00 at 0.0	1.00 at 0.00	1.00 at 0.0
Precision-Recall Curve	Smooth, consistent high precision	Smooth, consistent with training	Slightly higher precision at higher recall

Overall Observations

1. **Training and Validation Similarity**:
   - Metrics for training and validation are nearly identical, suggesting good generalization without overfitting.
   - For both datasets, F1 Score peaks at the same confidence threshold (0.83), and precision maintains consistency.
2. **Test Dataset Performance**:
   - The test dataset shows a slightly **higher mAP@0.5** (0.991 compared to 0.988 for training/validation), which indicates the model's robustness on unseen data.
   - The F1 Score on the test set is slightly lower at a lower confidence (0.97 at 0.26), indicating that the test dataset might have marginally more challenging examples.
3. **Precision-Recall Curves**:
   - All datasets demonstrate excellent precision and recall, with slight differences in performance for higher recall values in the test set.

4. Future Improvements

1. **Performance / Robustness**
   - ✓ **Data Augmentation**: Incorporate more synthetic transformations (lighting changes, rotations, perspective shifts).
   - ✓ **Larger Model**: If hardware allows, we can consider yolov5m or yolov5l for potentially higher accuracy.
   - ✓ Collect and include additional images with varying probe positions, orientations, and environments to ensure the model performs well in real-world scenarios.
2. **Inference Runtime**
   - ✓ **Model Quantization**: We can use techniques like **FP16** or **INT8** quantization on NVIDIA Jetson for faster inference.
   - ✓ We can use NVIDIA TensorRT for quantization and optimization, as it is highly efficient for Jetson boards