Note: This notebooks is written on VsCode, the layout on Pycharm and VsCode may be different.

Analyze Training Results

During this phase, I trained four models (yolo8s-seg.pt, yolov8m-seg.pt, yolov8s-p2.pt and yolov8m-p2.pt), yolov8l and yolov8x series are too large to train (training time is longer than 15 hours). Here are the results of these model.

Table 1 - model: yolov8s-seg, epochs: 155, running time: 9.176h

Class	Images	Instances	Box(P)	Box(R)	mAP50(B)	mAP50-95(B)	Mask(P)	Mask(R)
all	28	389	0.431	0.269	0.259	0.131	0.365	0.237
scratch	18	79	0.42	0.294	0.26	0.155	0.34	0.318
stain	22	310	0.443	0.245	0.259	0.107	0.387	0.159

Table 2 - model: yolov8m-seg, epochs:284, running time: 15.774

Class	Images	Instances	Box(P)	Box(R)	mAP50(B)	mAP50-95(B)	Mask(P)	Mask(R)
all	28	389	0.4	0.271	0.271	0.138	0.46	0.187
scratch	18	79	0.382	0.329	0.29	0.174	0.433	0.215
stain	22	310	0.419	0.213	0.252	0.103	0.486	0.159

Table 3 - model: yolov8s-p2, epochs:146, running time: 9.147

Class	Images	Instances	Box(P)	Box(R)	mAP50(B)	mAP50-95(B)
all	28	389	0.38	0.262	0.249	0.12
scratch	18	79	0.341	0.291	0.251	0.15
stain	22	310	0.419	0.232	0.247	0.0896

Table 4 - model: yolov8m-p2, epochs:218, running time: 8.251

Class	Images	Instances	Box(P)	Box(R)	mAP50(B)	mAP50-95(B)
all	28	389	0.378	0.323	0.271	0.135

Class	Images	Instances	Box(P)	Box(R)	mAP50(B)	mAP50-95(B)
scratch	18	79	0.287	0.405	0.302	0.176
stain	22	310	0.47	0.242	0.241	0.0943

Note that two computers were used to train models (one is from Viraj the other is from the uni laboratory) so the training time may vary. I also trained those model with preprocessing (tile), but the results were not very good (training result with 'tile' string, i.e., 'model: yolov8s-seg, epochs:209, running time: 10.091, tile: 3*3') P2 model (like yolov8m-p2) does not have segmentation mode, so there is no segmentation parameters in the result (i.e., mAP50(M), 'M' for mask).

There are two dataset version, the results were generated by modified version, (with only 'strain' and 'scratch' classes in dataset). Before that, there were 'chip', 'dent', 'missing', 'scratch' and 'stain'. Dataset was modified since 'missing' and 'dent' lack of instances, 'chip' and 'scratch' quite the same, resulting in bad result.

Use Cocoeval to analyze models

Sahi does not support preprocessing, which means we cannot use YOLO's traning result to measure preformance. Instead, we can analyze models using *cocoeval*, a tool to analyze coco dataset (JSON format). By doing so, we need to transfer our dataset from yolo format to coco format, simply select coco format on *RoboFlow*.

select coco dataset

Usually we need to modify annotation file after downloading since the class number of object in coco dataset starts on 1 rather than 0 on YOLO. Can simply modify categories in *_annotation.json* file to match class numbers.

```
"categories": [
    {
        "id": 0,
        "name": "scratch",
        "supercategory": "defects-pbH2"
    },
    {
        "id": 1,
        "name": "stain",
        "supercategory": "defects-pbH2"
    }
],
```

Install Dependencies

Then we can start to analyze small objects on dataset. First install dependencies.

In [ ]:
```
!pip install sahi
!pip install pycocotools
```

In [ ]:
```
import os
import time

from pycocotools.cocoeval import COCOeval
from pycocotools.coco import COCO

from sahi import AutoDetectionModel
from sahi.predict import predict
```

The dataset is splited into three (train, val and test), we can only use test set to analyze.

In [ ]:
```
# Paths
dir_path = os.getcwd()
image_dir = os.path.join(dir_path, 'cocodataset/test')
coco_json_path = os.path.join(image_dir, '_annotations.coco.json')
model_dir_path = os.path.join(dir_path, '/training_results/')
m_p2 = 'm-p2/train2/weights/best.pt'
s_p2 = 'yolov8s-p2/train/weights/best.pt'
m_seg = 'm-seg/train3/weights/best.pt'
s_seg = 's-seg/train2/weights/best.pt'

# export result
no_sahi_path = 'runs/no_sahi'
sahi_2_2_path = 'runs/2_2'
sahi_3_3_path = 'runs/3_3'
```

In [ ]:
```
# define parameters, note that original image size is 1920*1080
# slice_height = 420
# slice_width = 740
overlap_height_ratio = 0.2
overlap_width_ratio = 0.2
h, w = 1080, 1920
prediction_times = []
```

In [ ]:
```
# slice num is the number of silce of each edge, i.e., 2 for 2*2 slices.
def analyze_model(model_path, slice_num, export_json_path):
    detection_model = AutoDetectionModel.from_pretrained(
        model_type='yolov8',
        model_path=model_path,
        confidence_threshold=0.3,
        device="cuda"  # or "cpu"
    )

    W = slice_num - 0.2 * (slice_num - 1)

    start_time = time.time()
    coco_result = predict(
        detection_model=detection_model,
        source=image_dir,
        slice_height=int(h / W),
        slice_width=int(w / W),
```

```
        overlap_height_ratio=overlap_height_ratio,
        overlap_width_ratio=overlap_width_ratio,
        project=sahi_3_3_path,
        name="exp",
        export_coco=True,
        dataset_json_path=coco_json_path
    )

    if os.path.exists(export_json_path):
        print(f"Prediction results exported successfully to {export_json_pat
    else:
        print(f"Error: {export_json_path} not found!")

    coco_gt = COCO(coco_json_path)
    coco_pred = coco_gt.loadRes(export_json_path)

    # TP, FP, FN = calculate_tp_fp_fn(coco_gt, coco_pred, iou_threshold=0.3)
    # print(f"TP: {TP}, FP: {FP}, FN: {FN}")
    coco_eval = COCOeval(coco_gt, coco_pred, iouType="bbox") # can change ic
    coco_eval.evaluate()
    coco_eval.accumulate()
    coco_eval.summarize()

    total_detections = len(coco_pred.getAnnIds())
    print(f"Number of detected objects: {total_detections}")

    tp = coco_eval.eval['precision'][0, :, :, 0, 2]  # IoU=0.5, area=all, ma
    true_positives = int(tp.sum())
    print(f"Number of TP: {true_positives}")

    false_positives = total_detections - true_positives
    false_negatives = len(coco_gt.getAnnIds()) - true_positives
    print(f"Number of FP: {false_positives}")
    print(f"Number of FN: {false_negatives}")
```

In [ ]:

```
# reanalyze if missed up (evaluation results already exists)
def reanalyze(export_json_path):
    coco_gt = COCO(coco_json_path)
    coco_pred = coco_gt.loadRes(export_json_path)

    # TP, FP, FN = calculate_tp_fp_fn(coco_gt, coco_pred, iou_threshold=0.3)
    # print(f"TP: {TP}, FP: {FP}, FN: {FN}")
    coco_eval = COCOeval(coco_gt, coco_pred, iouType="bbox")
    coco_eval.evaluate()
    coco_eval.accumulate()
    coco_eval.summarize()

    total_detections = len(coco_pred.getAnnIds())
    print(f"Number of detected objects: {total_detections}")

    tp = coco_eval.eval['precision'][0, :, :, 0, 2]  # IoU=0.5, area=all, ma
    true_positives = int(tp.sum())
    print(f"Number of TP: {true_positives}")

    false_positives = total_detections - true_positives
    false_negatives = len(coco_gt.getAnnIds()) - true_positives
```

```python
    print(f"Number of FP: {false_positives}")
    print(f"Number of FN: {false_negatives}")
```

Calculate average inference time based on the evaluation results.

In [ ]:
```python
log_text = """
Performing prediction on 9 slices.
Prediction time is: 1604.85 ms
Performing prediction on 9 slices.
Prediction time is: 613.62 ms
Performing prediction on 9 slices.
Prediction time is: 467.03 ms
Performing prediction on 9 slices.
Prediction time is: 492.12 ms
Performing prediction on 9 slices.
Prediction time is: 540.01 ms
Performing prediction on 9 slices.
Prediction time is: 474.16 ms
Performing prediction on 12 slices.
Prediction time is: 617.48 ms
Performing prediction on 9 slices.
Prediction time is: 584.09 ms
Performing prediction on 12 slices.
Prediction time is: 612.03 ms
Performing prediction on 9 slices.
Prediction time is: 504.72 ms
Performing prediction on 9 slices.
Prediction time is: 502.49 ms
Performing prediction on 12 slices.
Prediction time is: 687.79 ms
Performing prediction on 9 slices.
Prediction time is: 518.58 ms
Performing prediction on 9 slices.
Prediction time is: 512.17 ms
Performing prediction on 9 slices.
Prediction time is: 457.43 ms
Performing prediction on 9 slices.
Prediction time is: 648.55 ms
Performing prediction on 9 slices.
Prediction time is: 667.46 ms
Performing prediction on 9 slices.
Prediction time is: 482.98 ms
Performing prediction on 9 slices.
Prediction time is: 476.35 ms
Performing prediction on 9 slices.
Prediction time is: 512.53 ms
Performing prediction on 9 slices.
Prediction time is: 606.90 ms
Performing prediction on 9 slices.
Prediction time is: 737.83 ms
"""
```

In [ ]:
```python
import re

def extract_times_and_calculate_average(log_text):
```

```
    time_pattern = r"Prediction time is: (\d+\.\d+) ms"
    times = re.findall(time_pattern, log_text)

    prediction_times = [float(time) for time in times]

    if prediction_times:
        average_time = sum(prediction_times) / len(prediction_times)
        return average_time
    else:
        return None

average_time = extract_times_and_calculate_average(log_text)
print(f'Average time: {average_time:.2f} ms')
```

We can start with yolov8m-seg model.

In [ ]:
```
model_path = os.path.join(model_dir_path, m_seg)

# assume using 2*2 slices
export_dir_path = os.path.join(dir_path, sahi_2_2_path)
export_json_path = os.path.join(export_dir_path, "exp/result.json")

analyze_model(model_path=model_path, slice_num=2, export_json_path=export_js
```

For example, the result:

COCO Evaluation Metrics

Metric	IoU Threshold	Area	Max Detections	Value
Average Precision (AP)	0.50:0.95	all	100	0.065
Average Precision (AP)	0.50	all	100	0.185
Average Precision (AP)	0.75	all	100	0.046
Average Precision (AP)	0.50:0.95	small	100	0.012
Average Precision (AP)	0.50:0.95	medium	100	0.300
Average Precision (AP)	0.50:0.95	large	100	0.430
Average Recall (AR)	0.50:0.95	all	1	0.027
Average Recall (AR)	0.50:0.95	all	10	0.091
Average Recall (AR)	0.50:0.95	all	100	0.103
Average Recall (AR)	0.50:0.95	small	100	0.038
Average Recall (AR)	0.50:0.95	medium	100	0.402
Average Recall (AR)	0.50:0.95	large	100	0.477

Confusion Matrix and Inference Time

Number of detected objects	Number of TP	Number of FP	Number of FN	Average time
379	37	342	338	446.76ms