

111-1 DIP Final Project q56105014 阮智軒

問題：

近年來，金屬激光熔化製造（MAM）在醫療行業、製造業、航空航天和精品行業得到廣泛應用。然而，選擇性激光熔化（SLM）製造過程中的瑕疵可能由選擇性激光熔化（SLM）製造過程中的熱應力或硬件故障引起。為了提高產品的質量，在製造過程中使用瑕疵檢測是必要的。

目前，最常用的檢測技術包括：

影像分析：利用攝像頭或其他影像設備拍攝製造過程中的產品，然後對影像進行分析，以發現瑕疵。

光學檢測：利用光學元件，如顯微鏡或光學檢測儀，對產品進行檢測，以發現瑕疵。

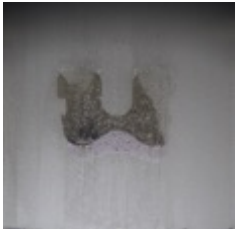
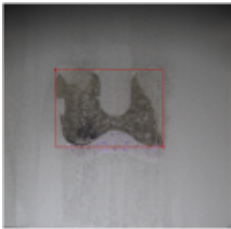
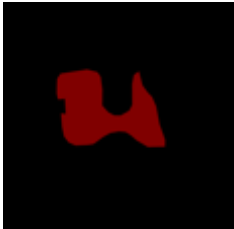

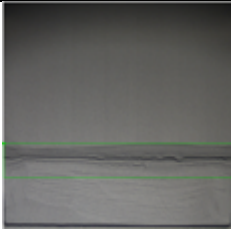
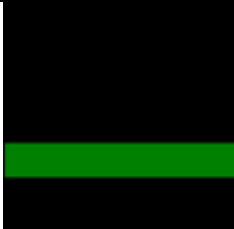



力學測試：利用力學測試儀器，如拉伸試驗機或硬度計，對產品進行檢測，以發現瑕疵。

掃描電子顯微鏡（SEM）：利用掃描電子顯微鏡對產品進行高分辨率成像，以發現瑕疵。

目前有研究使用利用人工智能或深度學習等方法進行影像分析，以提高檢測的準確度和速度。此外，也可以尋求新的方法來減少檢測對產品的影響，如使用更輕的測試儀器或減少檢測次數。

另外，在選擇檢測設備時，也可以考慮選擇能夠支持自動化檢測的設備，這樣可以大大提高生產效率。同時，也可以考慮將檢測納入整個生產流程中，以便能夠及時發現和解決問題。

這次我們會針對 3 種類別來進行偵測和分割，分別是：

Classes	Original	Bounding Box	Mask
Powder uncover			
Powder uneven			
Scratch			

材料與方法：

1. 訓練環境

- OS and programming language
 - Ubuntu 22.04
 - Python 3.7.4
 - Pytorch 1.12
- GPU
 - NVIDIA RTX 2080Ti 12GB
- CPU
 - Intel(R) Core(TM) i7-8700 CPU @ 3.20GHz (12Cores)

2. 資料集

- Powder Uncover、Powder Uneven、Scratch 各 150 張 = 450 張
- Train: Val = 300 : 150

3. 物件偵測 / 影像分割模型

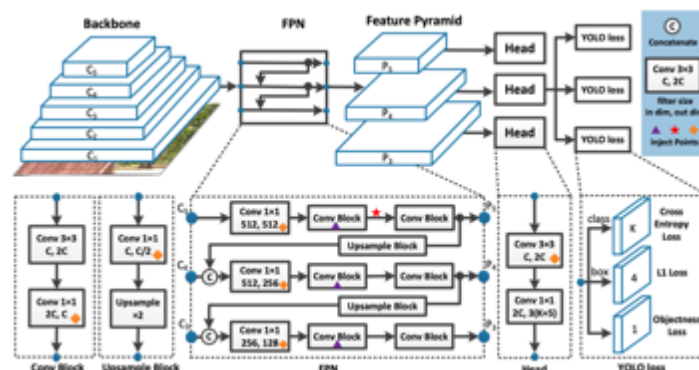
目前我打算利用深度學習的方法來進行檢測，用到的模型分別是：YOLOv7 和 U-Net，以下將分別介紹他們的特色和應用案例。

YOLOv7 (You Only Look Once version 7) 是一種用於目標檢測的深度學習模型。它的特色在於能夠在一次推理中檢測多個目標，並且速度快。YOLOv7 是在 YOLOv4 的基礎上進行改進得到的，主要改進包括增加了許多新的層以及改變了網絡結構。

YOLOv7 利用卷積神經網絡 (CNN) 對輸入的圖像進行特徵提取，然後將提取的特徵輸入到全連接層 (Fully Connected Layer) 中，再通過多個不同的輸出層進行目標檢測。

YOLOv7 常用於車輛檢測、人臉辨識和行人檢測等任務。例如，在車輛檢測中，可以利

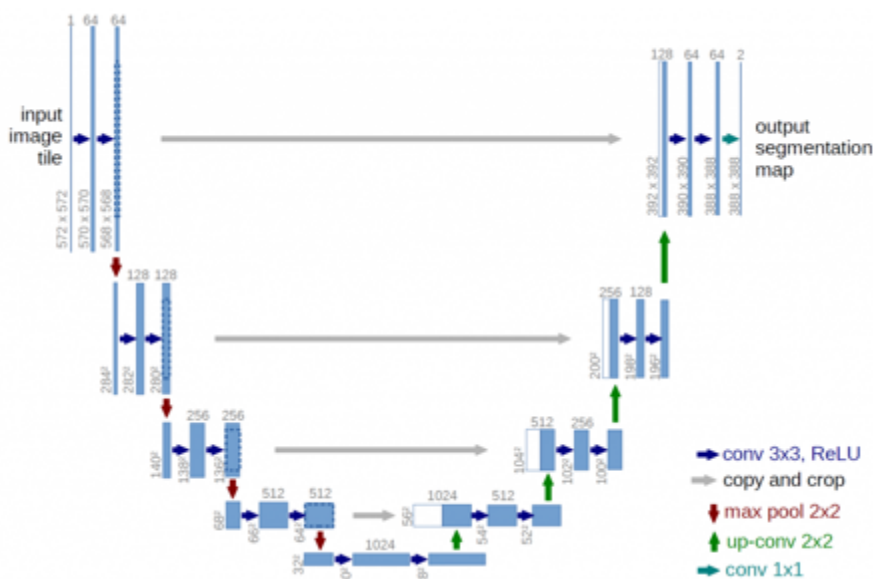
YOLOv7 檢測出圖像中的車輛，並且還能夠檢測出車輛的類型 (例如小車、貨車等)。



U-Net 是一種用於圖像分割的深度學習模型。它的特色在於能夠對圖像中的不同區域進行分割，並且分割效果較好。U-Net 的網絡結構由兩部分組成，即 Encoder 部分和 Decoder 部分；因為形狀貌似 U 型而得名。

U-Net 首先對輸入的圖像進行特徵提取，然後在 Encoder 部分中通過多層卷積和最大池化層進行特徵萃取和縮減，最後在 Decoder 部分中通過上采樣和卷積層進行特徵膨脹（feature expansion）和分割。

U-Net 常用於醫學圖像分割、行人檢測和遙感影像分割等任務。例如，在醫學圖像分割中，可以利用 U-Net 對腫瘤、肝臟或腎臟等區域進行分割。



總之，YOLOv7 和 U-Net 都是常用的深度學習模型，分別適用於目標檢測和圖像分割任務。它們都有著良好的表現，並且在不同的應用領域中得到廣泛應用。

4. Rename

目前影像的檔案名稱由於出現重覆、所以需要利用 Rename Function 來進行處理。

將每一個類別的名稱加到檔案名稱裡，converted 3180 → scratch converted 3180

```
1 def rename_files(dataset_dir):
2     for dataset in dataset_dir:
3         for class_name in class_names:
4             class_dir = os.path.join(dataset_dir, class_name)
5             # Create the image, label and mask directories
6             for type_name in types:
7                 type_dir = os.path.join(class_dir, type_name)
8                 for filename in tqdm(os.listdir(type_dir)):
9                     # Rename the image name with class + image name
10                    # Remove file name with space
11                    if rename == True:
12                        # Rename the image name with class + image name
13                        os.rename(os.path.join(type_dir, filename), os.path.join(type_dir, f'{class_name}_{filename.replace(" ", "_").lower()}'))
14                        # Rename the label name with class + label name
15                        os.rename(os.path.join(type_dir, filename.replace('image', 'label')), os.path.join(type_dir, f'{class_name}_{filename.replace("image", "label").replace(" ", "_").lower()}'))
16                        # Rename the mask name with class + mask name
17                        os.rename(os.path.join(type_dir, filename.replace('image', 'mask')), os.path.join(type_dir, f'{class_name}_{filename.replace("image", "mask").replace(" ", "_").lower()}'))
```

5. RGB to Grayscale

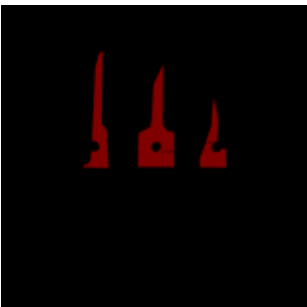
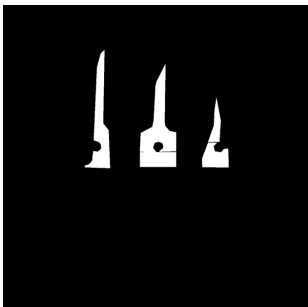
輸入到模型的影像為 Grayscale 的，目前只有 Scratch 的類別是灰階影像，需要進行轉換。

```
1 import cv2
2
3 def rgb2gray(original_img:np.ndarray):
4     gray_img = cv2.cvtColor(original_img, cv2.COLOR_BGR2GRAY)
5     return gray_img
```

6. Mask thresholding

在本次 Image Segmentation 中，要將 Defect 和 Background 進分割、要將 Pixel 進行 Thresholding、做法為將 Mask 中的 Pixel Value 大於某一閾值的像素設為 1，其餘設為 0。

目前因為 Mask 紅色和其他彩色、所以 threshold = 10

Original	After thresholding	Code
		<pre>1 2 def thresholding(img:np.ndarray, threshold:int): 3 4 # convert to grayscale 5 mask_img = img.convert('L') 6 7 img_table = [] 8 9 # thresholding 10 for i in range(256): 11 if i < threshold: 12 img_table.append(0) 13 else: 14 img_table.append(1) 15 16 return mask_img.point(table, '1')</pre>

7. Resize

U-Net 的 Image 和 Mask 影像輸入需要一致、而目前不同類別的 Image 和 Mask 都不一樣，所以要進行 Resize。

```
1 import cv2
2
3 def img_resize(original_image:np.ndarray):
4
5     if original_image.shape != (1244, 1254, 1):
6         resize_image = original_image.resize((1254, 1244))
7     else:
8         continus
9
10    return resize_image
11
```

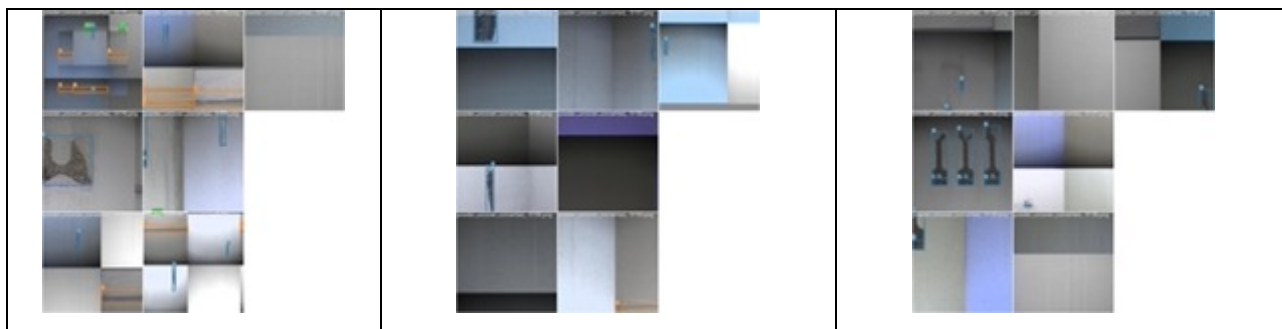
8. Augmentation

針對目前的資料在輸入到模型時有做一系列的資料擴增、如 HSV 色調、飽和度、曝光值、旋轉、平移、縮放、裁剪、透視、翻轉、鏡射、影像混合等處理；資料擴增是在訓練模型時用來增加訓練資料量的技術，可以幫助模型從多種不同的角度去看待資料，從而增加模型的泛化能力，降低過擬合的風險。

對於圖像辨識任務而言，資料擴增也可以幫助模型克服資料不均衡的問題，例如在某個類別中有較少的圖像，透過資料擴增可以增加該類別的圖像數量，避免模型偏向較多圖像的類別。

此外，資料擴增也可以模擬出真實世界中的各種變化，例如不同的角度、亮度、尺寸等，讓模型在接受訓練時能夠更好地應對不確定性。

以下是目前訓練時的影像：



9. Training Parameter

Segmentation

- Batch Size: 16
- Learning Rate: 0.00001
- Epoch: 300
- Optimizer: RMSprop
- Image Size: 1152 * 1134

Object Detection

- Batch Size: 7
- Learning Rate: 0.0001
- Epoch: 300
- Optimizer: Adam
- Image Size: 640 * 640

10. Loss Function

Dice Loss

Dice loss 是一種常用於圖像分割任務中的損失函數，它可以用來檢測模型對於每個像素的預測精度。Dice loss 通常用於二分類任務，其目的是計算模型預測和實際標記之間的相似度。

Dice loss 的值通常在 $[0, 1]$ 之間，值越小表示模型預測和實際標記之間的差距越小，模型的預測精度越高。因此，在訓練模型時，可以通過最小化 Dice loss 來提高模型的預測精度。

```
1 import numpy as np
2
3 def dice_loss(y_true:np.ndarray, y_pred:np.ndarray):
4     smooth = 1
5     y_true_f = y_true.flatten()
6     y_pred_f = y_pred.flatten()
7     intersection = np.sum(y_true_f * y_pred_f)
8
9     return 1 - round((2. * intersection + smooth) / (np.sum(y_true_f) + np.sum(y_pred_f) + smooth),3)
```

YOLO v7

$$Loss_{yolo} = \lambda_{size} \cdot Loss_{size} + \lambda_{center} \cdot Loss_{center} + \lambda_{class} \cdot Loss_{class} + \lambda_{position} \cdot Loss_{position}$$

大小損失（size loss）：計算預測的目標的長度和寬度與實際的目標的長度和寬度之間的差距。

中心損失（center loss）：計算預測的目標的中心和實際的目標的中心之間的差距。

類別損失（class loss）：計算預測的目標的類別和實際的目標的類別之間的差距。

位置損失（position loss）：計算預測的目標在圖像中的位置和實際的目標在圖像中的位置之間的差距。

λ 表示各個損失函數的權重。通過調整這些權重，可以對各個損失函數的貢獻進行優化。

11. Evaluation

Dice Coefficient

Dice Coefficient 是一種衡量兩個集合相似度的指標。它常用於評估圖像分割算法的精度；目前的做法是利用 `numpy.flatten()` 找出 Ground Truth Mask 的面積、再算出 Predict Mask 進利計算。

```
1 import numpy as np
2
3 def dice_coef(y_true:np.ndarray, y_pred:np.ndarray):
4     smooth = 1
5     y_true_f = y_true.flatten()
6     y_pred_f = y_pred.flatten()
7     intersection = np.sum(y_true_f * y_pred_f)
8
9     return round((2. * intersection + smooth) / (np.sum(y_true_f) + np.sum(y_pred_f) + smooth),3)
10
```

IoU (Intersection over Union)

IoU 全名為 Intersection over Union，是一種衡量兩個集合相似度的指標，常用於圖像分割目的是評估 預測的物件區域與真實物件區域的交集大小；本次研究是用來評估 Object Detection Ground Truth 的 Bounding Box 和 Predict 的 Bounding Box 交集差異。

又因為目前輸入 YOLO 模型時需要將 label 的格式轉換成 YOLO Format；在計算 IoU 時算要先逆轉換為原來的格式進行計算。

```
1 def xywh2xyxy(data:list, img_w:int, img_h:int) -> list:
2
3     """Converts a bounding box from xyxy format to xywh format.
4
5     Args:
6
7         xyxy (list): A bounding box in xyxy format.
8
9     Returns:
10
11         list: A bounding box in xywh format.
12
13     """
14
15     bbox_width = float(data[2]) * img_w
16     bbox_height = float(data[3]) * img_h
17     center_x = float(data[0]) * img_w
18     center_y = float(data[1]) * img_h
19
20
21     min_x, min_y = center_x - (bbox_width / 2), center_y - (bbox_height / 2)
22     max_x, max_y = center_x + (bbox_width / 2), center_y + (bbox_height / 2)
23     print(min_x,min_y,max_x,max_y)
24
25     return [min_x, min_y, max_x, max_y]
26
27
28 def IoU(predict_bbox_yolo:list, ground_truth_bbox_yolo:list, w:int, h:int) -> float:
29
30     bbox2 = xywh2xyxy(data=predict_bbox_yolo, img_w=w, img_h=h)
31     bbox1 = xywh2xyxy(data=ground_truth_bbox_yolo, img_w=w, img_h=h)
32
33     x1l = max(bbox1[0], bbox2[0])
34     y1l = max(bbox1[1], bbox2[1])
35     x1r = min(bbox1[2], bbox2[2])
36     y1r = min(bbox1[3], bbox2[3])
37     inter_area = (y1r - y1l) * (x1r - x1l)
38
39     box1_area = (bbox1[2] - bbox1[0]) * (bbox1[3] - bbox1[1])
40     box2_area = (bbox2[2] - bbox2[0]) * (bbox2[3] - bbox2[1])
41
42     union_area = box1_area + box2_area - inter_area
43
44     iou = inter_area / union_area
45
46     return iou
```


12. Model Summary

U-Net

Layer (type)	Output Shape	Param #
=====		
Conv2d-1	[-1, 64, 572, 572]	576
BatchNorm2d-2	[-1, 64, 572, 572]	128
ReLU-3	[-1, 64, 572, 572]	0
Conv2d-4	[-1, 64, 572, 572]	36,864
BatchNorm2d-5	[-1, 64, 572, 572]	128
ReLU-6	[-1, 64, 572, 572]	0
DoubleConv-7	[-1, 64, 572, 572]	0
MaxPool2d-8	[-1, 64, 286, 286]	0
Conv2d-9	[-1, 128, 286, 286]	73,728
BatchNorm2d-10	[-1, 128, 286, 286]	256
ReLU-11	[-1, 128, 286, 286]	0
Conv2d-12	[-1, 128, 286, 286]	147,456
BatchNorm2d-13	[-1, 128, 286, 286]	256
ReLU-14	[-1, 128, 286, 286]	0
DoubleConv-15	[-1, 128, 286, 286]	0
Down-16	[-1, 128, 286, 286]	0
MaxPool2d-17	[-1, 128, 143, 143]	0
Conv2d-18	[-1, 256, 143, 143]	294,912
BatchNorm2d-19	[-1, 256, 143, 143]	512
ReLU-20	[-1, 256, 143, 143]	0
Conv2d-21	[-1, 256, 143, 143]	589,824
BatchNorm2d-22	[-1, 256, 143, 143]	512
ReLU-23	[-1, 256, 143, 143]	0
DoubleConv-24	[-1, 256, 143, 143]	0
Down-25	[-1, 256, 143, 143]	0
MaxPool2d-26	[-1, 256, 71, 71]	0
Conv2d-27	[-1, 512, 71, 71]	1,179,648
BatchNorm2d-28	[-1, 512, 71, 71]	1,024
ReLU-29	[-1, 512, 71, 71]	0
Conv2d-30	[-1, 512, 71, 71]	2,359,296
BatchNorm2d-31	[-1, 512, 71, 71]	1,024
ReLU-32	[-1, 512, 71, 71]	0
DoubleConv-33	[-1, 512, 71, 71]	0
Down-34	[-1, 512, 71, 71]	0
MaxPool2d-35	[-1, 512, 35, 35]	0
Conv2d-36	[-1, 1024, 35, 35]	4,718,592
BatchNorm2d-37	[-1, 1024, 35, 35]	2,048
ReLU-38	[-1, 1024, 35, 35]	0
Conv2d-39	[-1, 1024, 35, 35]	9,437,184
BatchNorm2d-40	[-1, 1024, 35, 35]	2,048
ReLU-41	[-1, 1024, 35, 35]	0
DoubleConv-42	[-1, 1024, 35, 35]	0
Down-43	[-1, 1024, 35, 35]	0
ConvTranspose2d-44	[-1, 512, 70, 70]	2,097,664
Conv2d-45	[-1, 512, 71, 71]	4,718,592
BatchNorm2d-46	[-1, 512, 71, 71]	1,024
ReLU-47	[-1, 512, 71, 71]	0
Conv2d-48	[-1, 512, 71, 71]	2,359,296
BatchNorm2d-49	[-1, 512, 71, 71]	1,024
ReLU-50	[-1, 512, 71, 71]	0
DoubleConv-51	[-1, 512, 71, 71]	0
Up-52	[-1, 512, 71, 71]	0
ConvTranspose2d-53	[-1, 256, 142, 142]	524,544
Conv2d-54	[-1, 256, 143, 143]	1,179,648
BatchNorm2d-55	[-1, 256, 143, 143]	512
ReLU-56	[-1, 256, 143, 143]	0
Conv2d-57	[-1, 256, 143, 143]	589,824
BatchNorm2d-58	[-1, 256, 143, 143]	512
ReLU-59	[-1, 256, 143, 143]	0
DoubleConv-60	[-1, 256, 143, 143]	0
Up-61	[-1, 256, 143, 143]	0
ConvTranspose2d-62	[-1, 128, 286, 286]	131,200
Conv2d-63	[-1, 128, 286, 286]	294,912
BatchNorm2d-64	[-1, 128, 286, 286]	256
ReLU-65	[-1, 128, 286, 286]	0
Conv2d-66	[-1, 128, 286, 286]	147,456
BatchNorm2d-67	[-1, 128, 286, 286]	256
ReLU-68	[-1, 128, 286, 286]	0
DoubleConv-69	[-1, 128, 286, 286]	0
Up-70	[-1, 128, 286, 286]	0
ConvTranspose2d-71	[-1, 64, 572, 572]	32,832
Conv2d-72	[-1, 64, 572, 572]	73,728
BatchNorm2d-73	[-1, 64, 572, 572]	128
ReLU-74	[-1, 64, 572, 572]	0
Conv2d-75	[-1, 64, 572, 572]	36,864
BatchNorm2d-76	[-1, 64, 572, 572]	128
ReLU-77	[-1, 64, 572, 572]	0
DoubleConv-78	[-1, 64, 572, 572]	0
Up-79	[-1, 64, 572, 572]	0
Conv2d-80	[-1, 2, 572, 572]	130
OutConv-81	[-1, 2, 572, 572]	0
=====		
Total params: 31,036,546		
Trainable params: 31,036,546		
Non-trainable params: 0		
=====		
Input size (MB): 1.25		
Forward/backward pass size (MB): 5087.77		
Params size (MB): 118.40		
Estimated Total Size (MB): 5207.41		

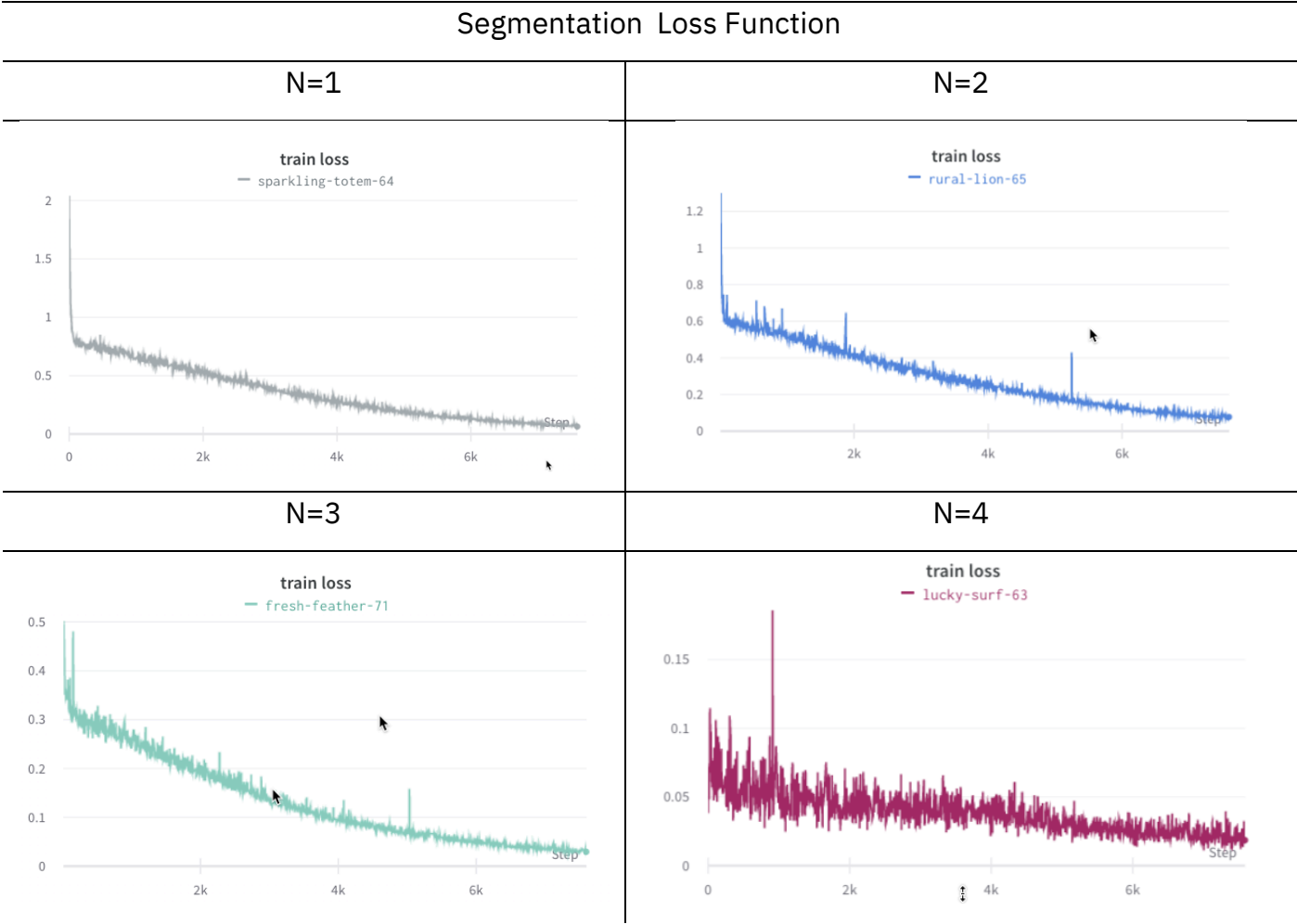
YoloV7

```

from s      params      module      arguments
-1      1      818      models.common.Conv      [32, 32, 3, 1, None, 1, LeakyRelu(negative_slope=0.1)]
-1      1      18560      models.common.Conv      [32, 64, 3, 2, None, 1, LeakyRelu(negative_slope=0.1)]
-1      1      2112      models.common.Conv      [64, 32, 1, 1, None, 1, LeakyRelu(negative_slope=0.1)]
-2      1      2112      models.common.Conv      [64, 32, 1, 1, None, 1, LeakyRelu(negative_slope=0.1)]
-1      1      9280      models.common.Conv      [32, 32, 3, 1, None, 1, LeakyRelu(negative_slope=0.1)]
-1      1      9280      models.common.Conv      [64, 32, 3, 1, None, 1, LeakyRelu(negative_slope=0.1)]
[-1, -2, -3, -4]      1      0      models.common.Concat      [1]
-1      1      8320      models.common.Conv      [128, 64, 1, 1, None, 1, LeakyRelu(negative_slope=0.1)]
-1      1      0      models.common.MP      [1]
-1      1      4224      models.common.Conv      [64, 64, 1, 1, None, 1, LeakyRelu(negative_slope=0.1)]
-2      1      4224      models.common.Conv      [64, 64, 1, 1, None, 1, LeakyRelu(negative_slope=0.1)]
-1      1      36892      models.common.Conv      [64, 64, 3, 1, None, 1, LeakyRelu(negative_slope=0.1)]
-1      1      36892      models.common.Conv      [64, 64, 3, 1, None, 1, LeakyRelu(negative_slope=0.1)]
[-1, -2, -3, -4]      1      0      models.common.Concat      [1]
-1      1      33824      models.common.Conv      [256, 128, 1, 1, None, 1, LeakyRelu(negative_slope=0.1)]
-1      1      0      models.common.MP      [1]
-1      1      16640      models.common.Conv      [128, 128, 1, 1, None, 1, LeakyRelu(negative_slope=0.1)]
-2      1      16640      models.common.Conv      [128, 128, 1, 1, None, 1, LeakyRelu(negative_slope=0.1)]
-1      1      147712      models.common.Conv      [128, 128, 3, 1, None, 1, LeakyRelu(negative_slope=0.1)]
-1      1      147712      models.common.Conv      [128, 128, 3, 1, None, 1, LeakyRelu(negative_slope=0.1)]
[-1, -2, -3, -4]      1      0      models.common.Concat      [1]
-1      1      133184      models.common.Conv      [512, 256, 1, 1, None, 1, LeakyRelu(negative_slope=0.1)]
-1      1      0      models.common.MP      [1]
-1      1      46848      models.common.Conv      [256, 256, 1, 1, None, 1, LeakyRelu(negative_slope=0.1)]
-2      1      46848      models.common.Conv      [256, 256, 1, 1, None, 1, LeakyRelu(negative_slope=0.1)]
-1      1      990336      models.common.Conv      [256, 256, 3, 1, None, 1, LeakyRelu(negative_slope=0.1)]
-1      1      990336      models.common.Conv      [256, 256, 3, 1, None, 1, LeakyRelu(negative_slope=0.1)]
[-1, -2, -3, -4]      1      0      models.common.Concat      [1]
-1      1      105112      models.common.Conv      [1024, 512, 1, 1, None, 1, LeakyRelu(negative_slope=0.1)]
-1      1      133184      models.common.Conv      [512, 256, 1, 1, None, 1, LeakyRelu(negative_slope=0.1)]
-2      1      133184      models.common.Conv      [512, 256, 1, 1, None, 1, LeakyRelu(negative_slope=0.1)]
-1      1      0      models.common.SP      [9]
-1      1      0      models.common.SP      [9]
-1      1      0      models.common.SP      [13]
[-1, -2, -3, -4]      1      0      models.common.Concat      [1]
-1      1      263456      models.common.Conv      [1024, 256, 1, 1, None, 1, LeakyRelu(negative_slope=0.1)]
-1      1      0      models.common.MP      [1]
-1      1      133184      models.common.Conv      [512, 256, 1, 1, None, 1, LeakyRelu(negative_slope=0.1)]
-1      1      33824      models.common.Conv      [256, 128, 1, 1, None, 1, LeakyRelu(negative_slope=0.1)]
-1      1      0      torch.nn.modules.upsampling.Upsample      [None, 2, 'nearest']
-1      1      33824      models.common.Conv      [256, 128, 1, 1, None, 1, LeakyRelu(negative_slope=0.1)]
[-1, -2]      1      0      models.common.Concat      [1]
-1      1      16832      models.common.Conv      [256, 64, 1, 1, None, 1, LeakyRelu(negative_slope=0.1)]
-2      1      16832      models.common.Conv      [256, 64, 1, 1, None, 1, LeakyRelu(negative_slope=0.1)]
-1      1      36892      models.common.Conv      [64, 64, 3, 1, None, 1, LeakyRelu(negative_slope=0.1)]
-1      1      36892      models.common.Conv      [64, 64, 3, 1, None, 1, LeakyRelu(negative_slope=0.1)]
[-1, -2, -3, -4]      1      0      models.common.Concat      [1]
-1      1      33824      models.common.Conv      [256, 128, 1, 1, None, 1, LeakyRelu(negative_slope=0.1)]
-1      1      8320      models.common.Conv      [128, 64, 1, 1, None, 1, LeakyRelu(negative_slope=0.1)]
-1      1      0      torch.nn.modules.upsampling.Upsample      [None, 2, 'nearest']
-1      1      8320      models.common.Conv      [128, 64, 1, 1, None, 1, LeakyRelu(negative_slope=0.1)]
[-1, -2]      1      0      models.common.Concat      [1]
-1      1      4160      models.common.Conv      [128, 32, 1, 1, None, 1, LeakyRelu(negative_slope=0.1)]
-2      1      4160      models.common.Conv      [128, 32, 1, 1, None, 1, LeakyRelu(negative_slope=0.1)]
-1      1      9280      models.common.Conv      [32, 32, 3, 1, None, 1, LeakyRelu(negative_slope=0.1)]
-1      1      9280      models.common.Conv      [64, 32, 3, 1, None, 1, LeakyRelu(negative_slope=0.1)]
[-1, -2, -3, -4]      1      0      models.common.Concat      [1]
-1      1      8320      models.common.Conv      [128, 64, 1, 1, None, 1, LeakyRelu(negative_slope=0.1)]
-1      1      73984      models.common.Conv      [64, 32, 3, 1, None, 1, LeakyRelu(negative_slope=0.1)]
[-1, -2]      1      0      models.common.Concat      [1]
-1      1      16832      models.common.Conv      [256, 64, 1, 1, None, 1, LeakyRelu(negative_slope=0.1)]
-2      1      16832      models.common.Conv      [256, 64, 1, 1, None, 1, LeakyRelu(negative_slope=0.1)]
-1      1      36892      models.common.Conv      [64, 64, 3, 1, None, 1, LeakyRelu(negative_slope=0.1)]
-1      1      36892      models.common.Conv      [64, 64, 3, 1, None, 1, LeakyRelu(negative_slope=0.1)]
[-1, -2, -3, -4]      1      0      models.common.Concat      [1]
-1      1      33824      models.common.Conv      [256, 128, 1, 1, None, 1, LeakyRelu(negative_slope=0.1)]
-1      1      299424      models.common.Conv      [128, 128, 3, 2, None, 1, LeakyRelu(negative_slope=0.1)]
[-1, -2]      1      0      models.common.Concat      [1]
-1      1      40792      models.common.Conv      [512, 128, 1, 1, None, 1, LeakyRelu(negative_slope=0.1)]
-2      1      40792      models.common.Conv      [512, 128, 1, 1, None, 1, LeakyRelu(negative_slope=0.1)]
-1      1      147712      models.common.Conv      [128, 128, 3, 1, None, 1, LeakyRelu(negative_slope=0.1)]
-1      1      147712      models.common.Conv      [128, 128, 3, 1, None, 1, LeakyRelu(negative_slope=0.1)]
[-1, -2, -3, -4]      1      0      models.common.Concat      [1]
-1      1      133184      models.common.Conv      [512, 256, 1, 1, None, 1, LeakyRelu(negative_slope=0.1)]
87      1      73984      models.common.Conv      [64, 32, 3, 1, None, 1, LeakyRelu(negative_slope=0.1)]
88      739424      models.common.Conv      [128, 256, 3, 1, None, 1, LeakyRelu(negative_slope=0.1)]
79      1388472      models.common.Conv      [256, 128, 3, 1, None, 1, LeakyRelu(negative_slope=0.1)]
[34, 78, 76]      22844      models.yolo.Detect

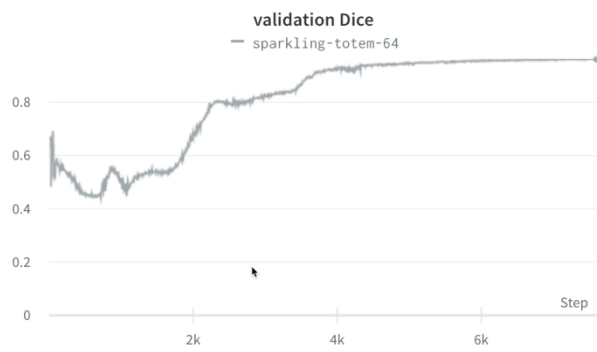
```


結果：

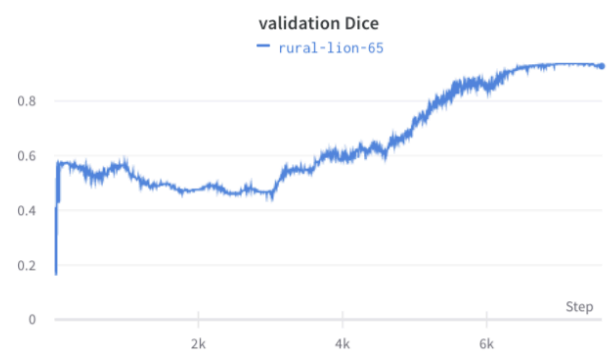


Validation dice coefficient

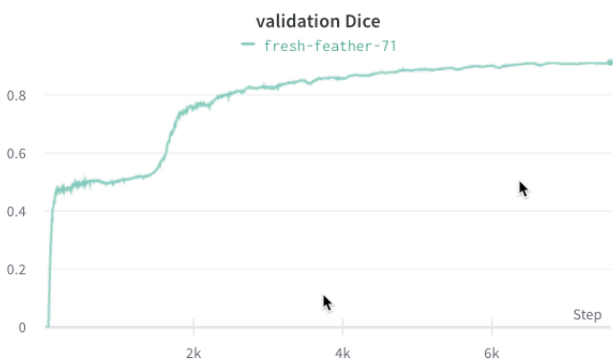
N=1



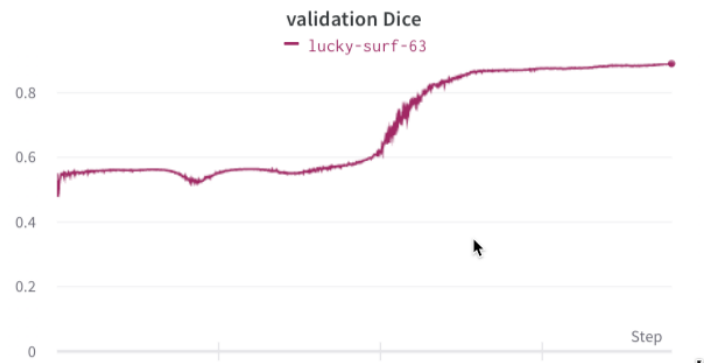
N=2



N=3

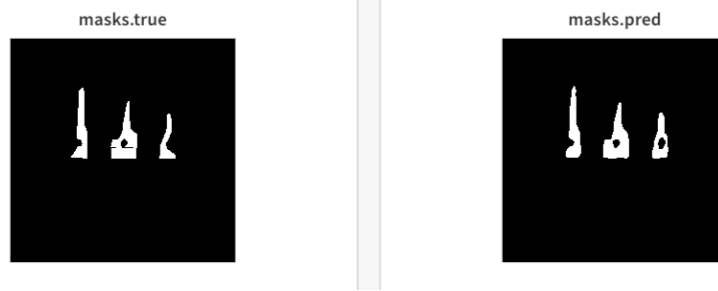


N=4



Predict Case

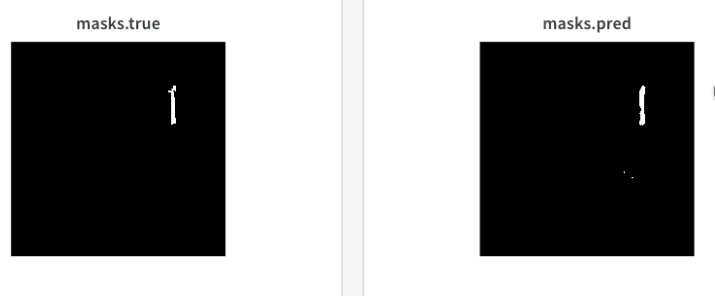
N=1



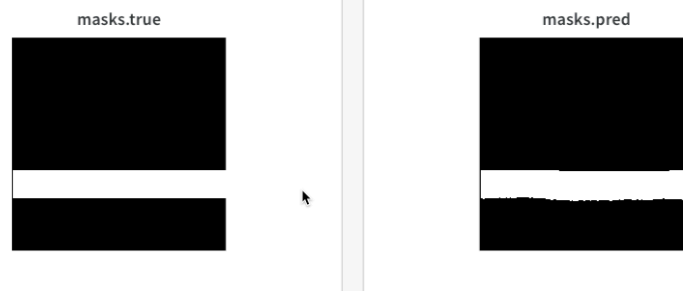
N = 2



N = 3

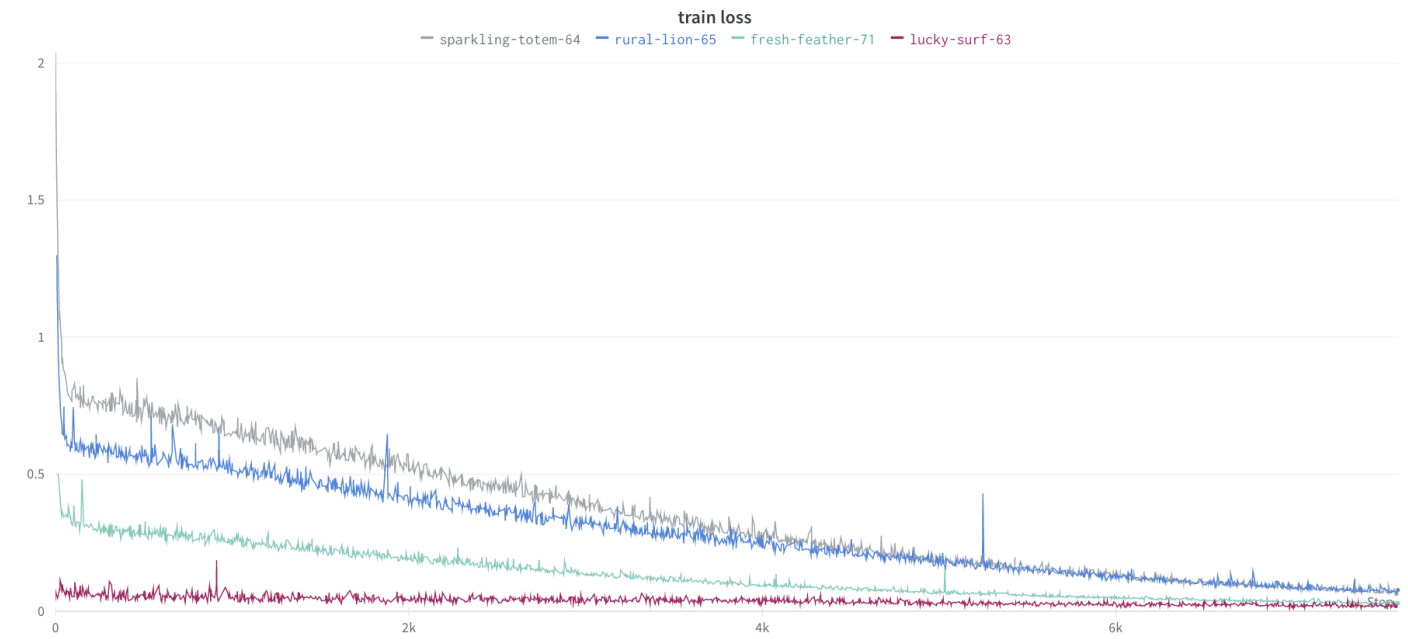


N = 4

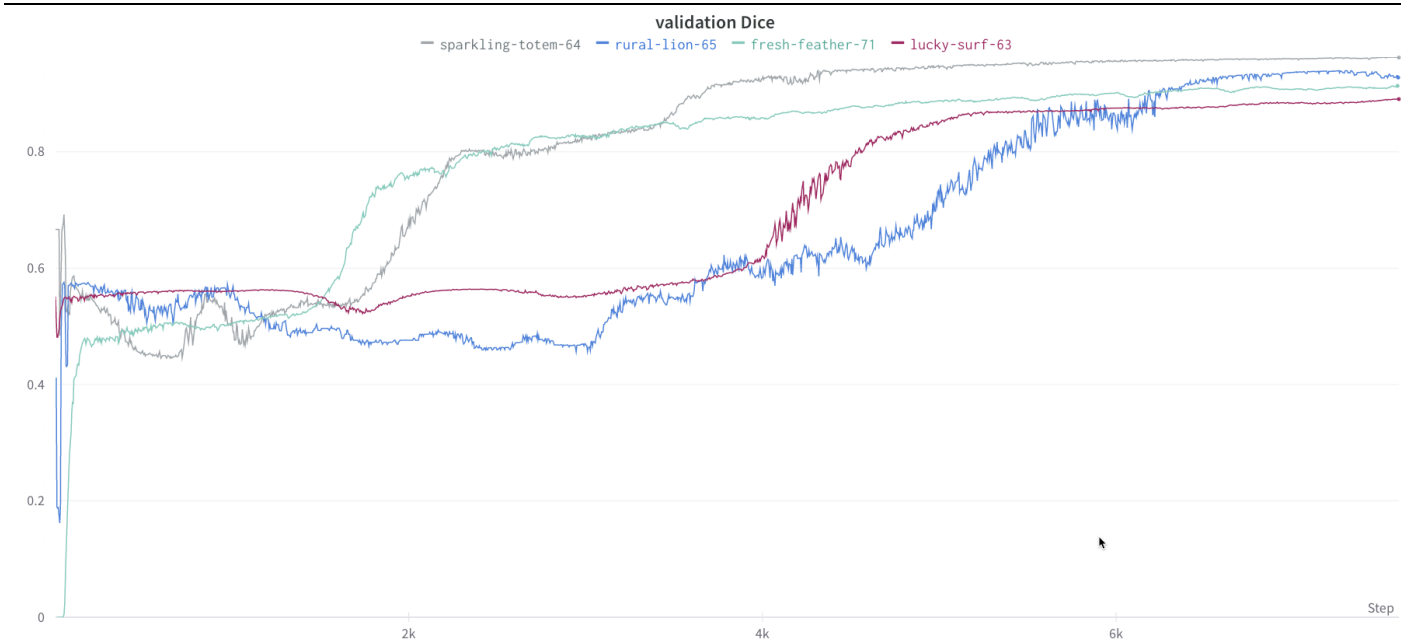


Compare 4-Fold

Loss Function



Validation dice coefficient



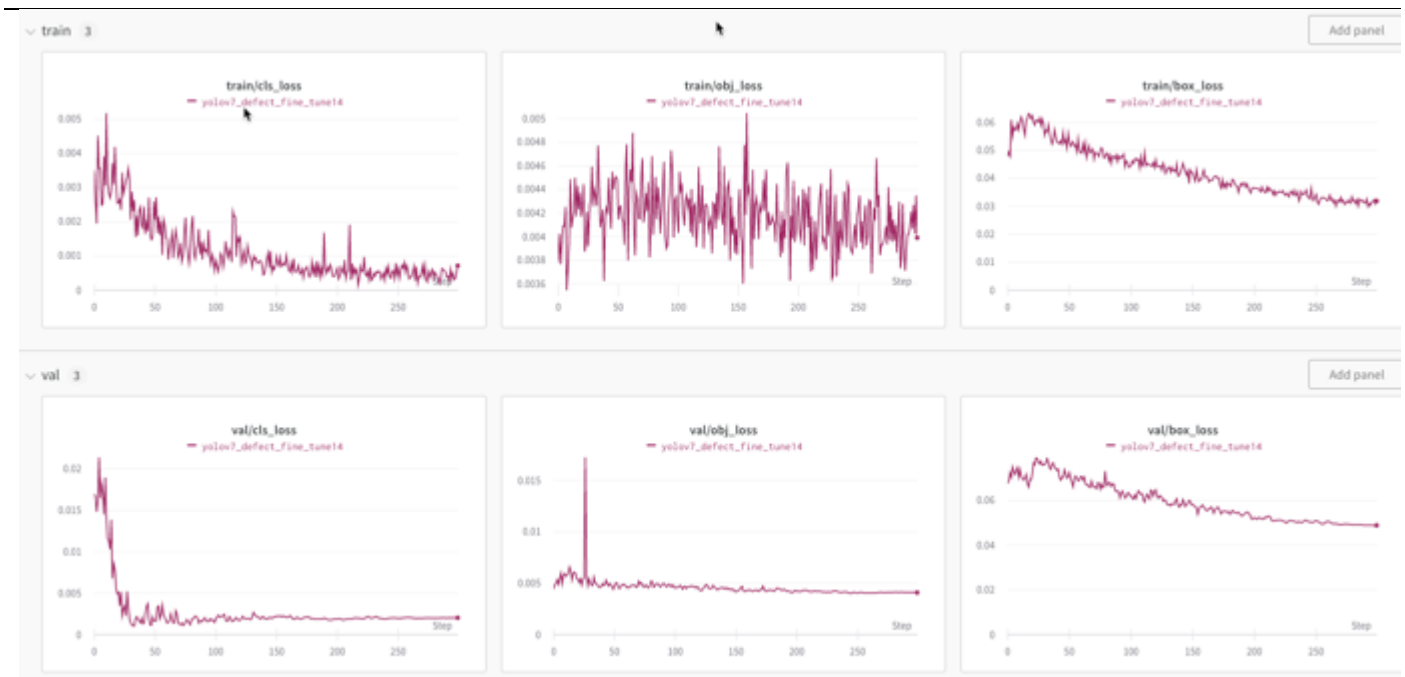
Objection Detection

Loss Function

N=1



N=2



N=3



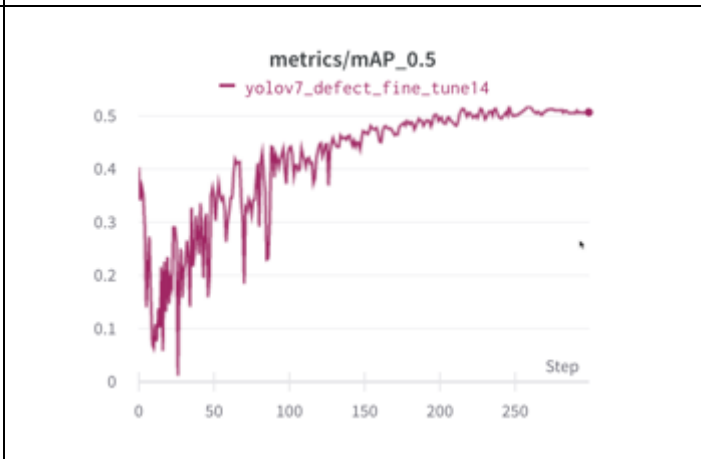
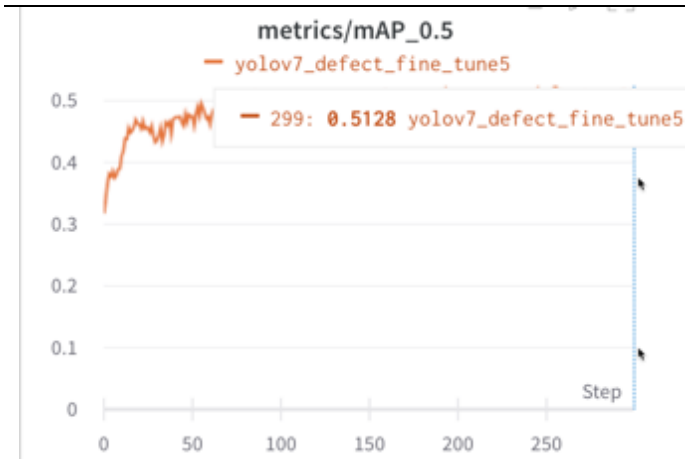
N=4



AP₅₀

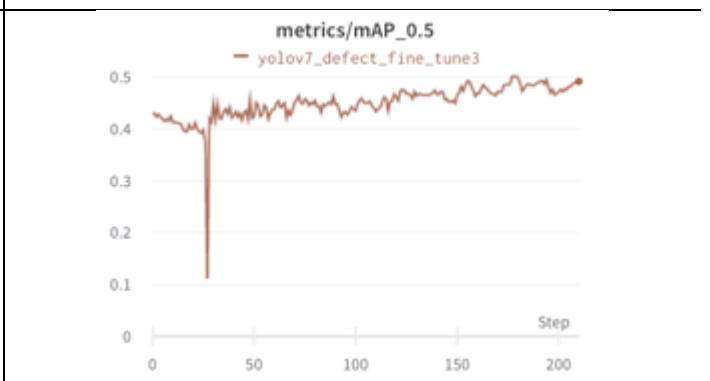
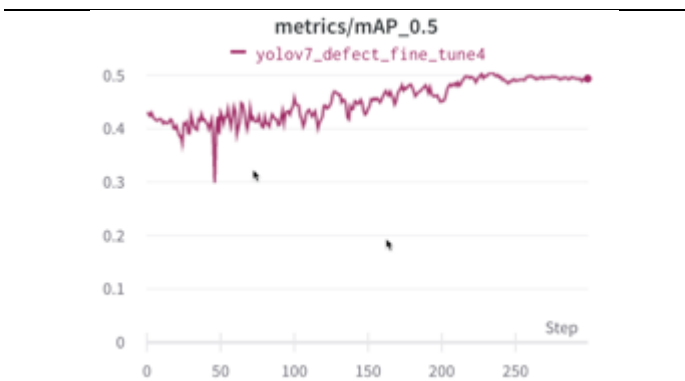
N=1, AP₅₀ = 0.51

N=2, AP₅₀ = 0.50



N=3, AP₅₀ = 0.49

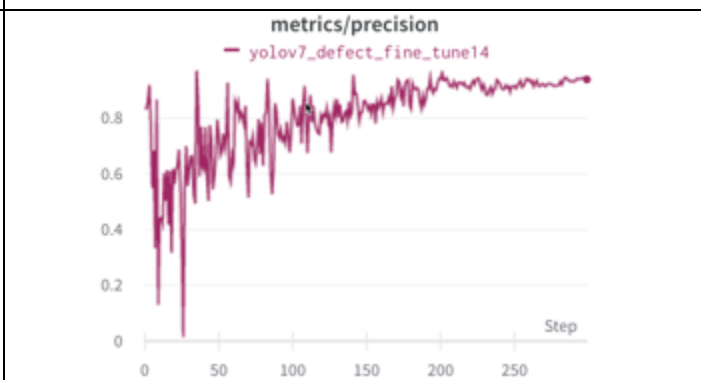
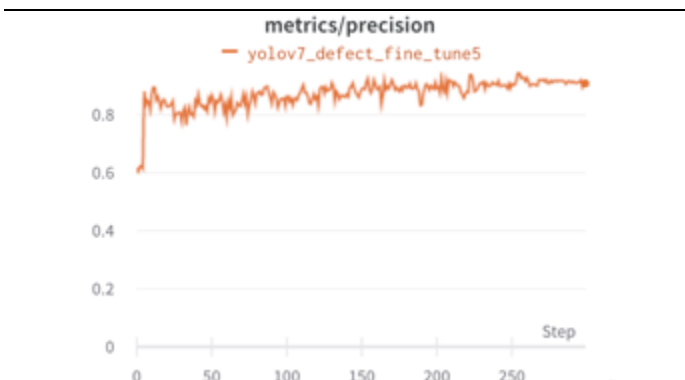
N=4, AP₅₀ = 0.49



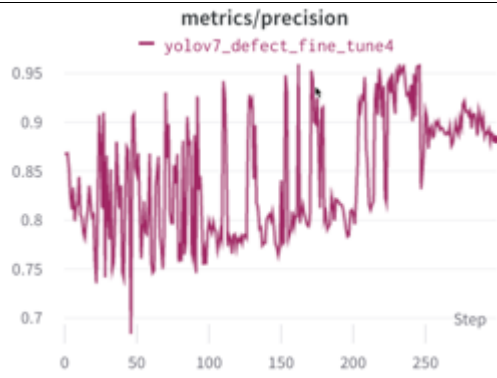
Precision

N=1, P = 0.90

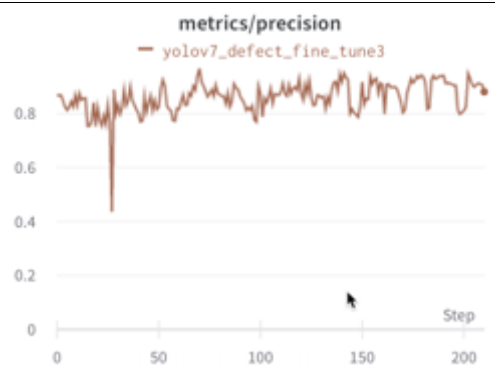
N=2, P = 0.93



N=3, P = 0.88



N=4, P = 0.88

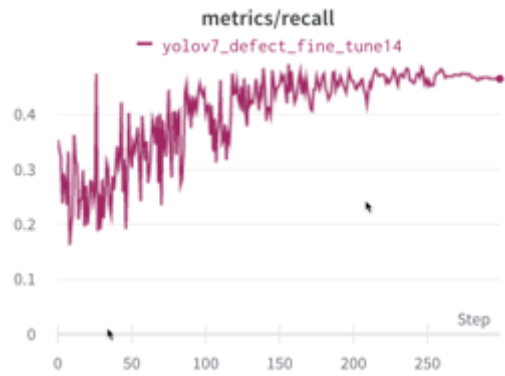


Recall

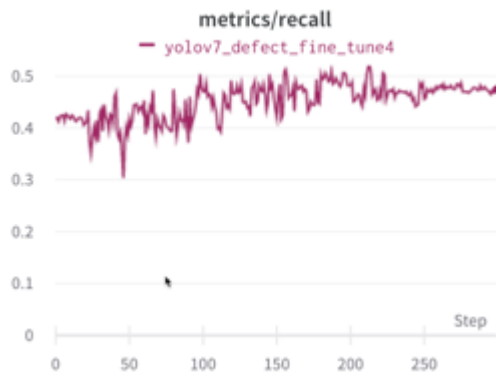
N=1, R = 0.47



N=2, R = 0.46



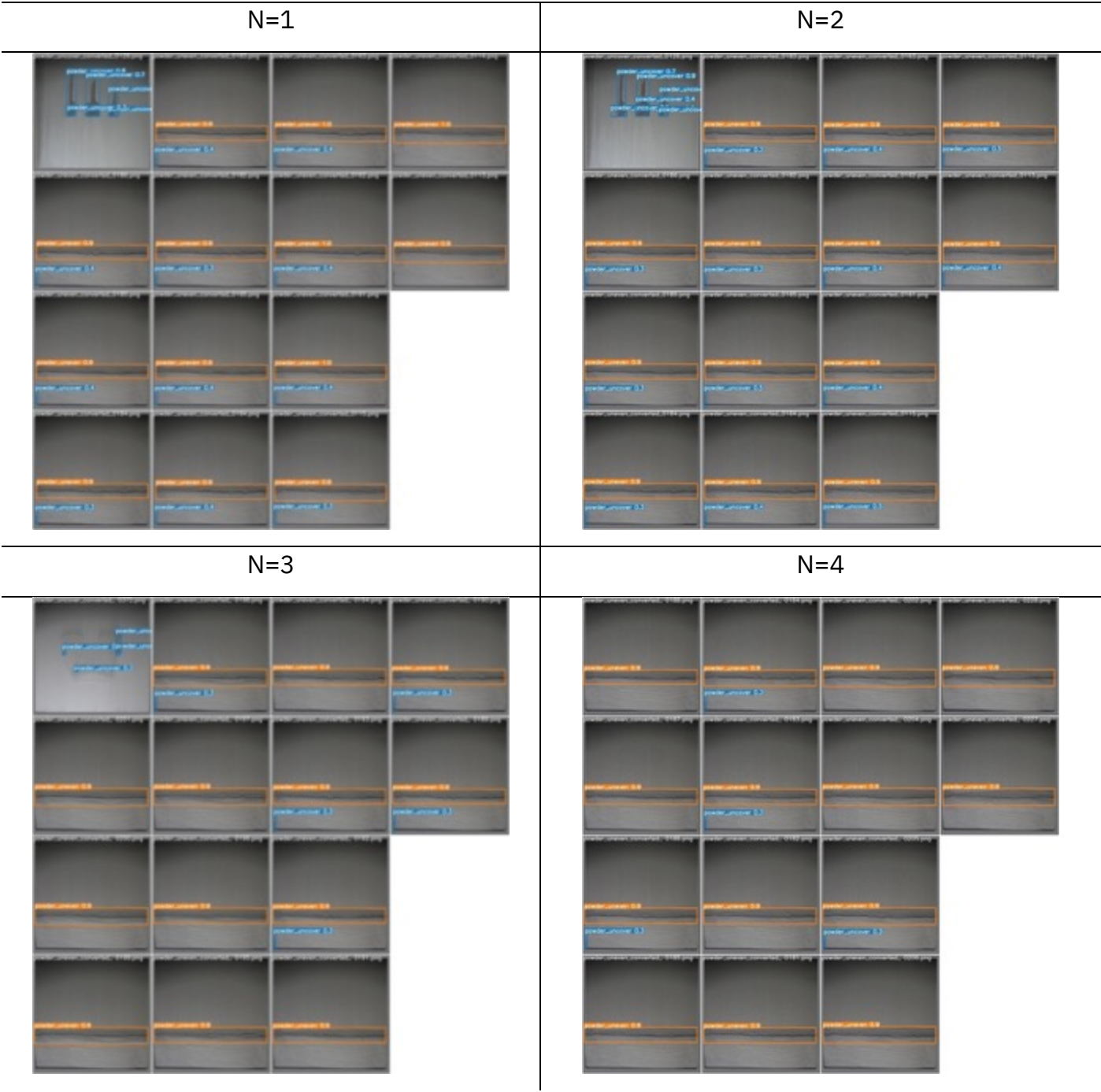
N=3, R = 0.47



N=4, R = 0.47



Predict Case

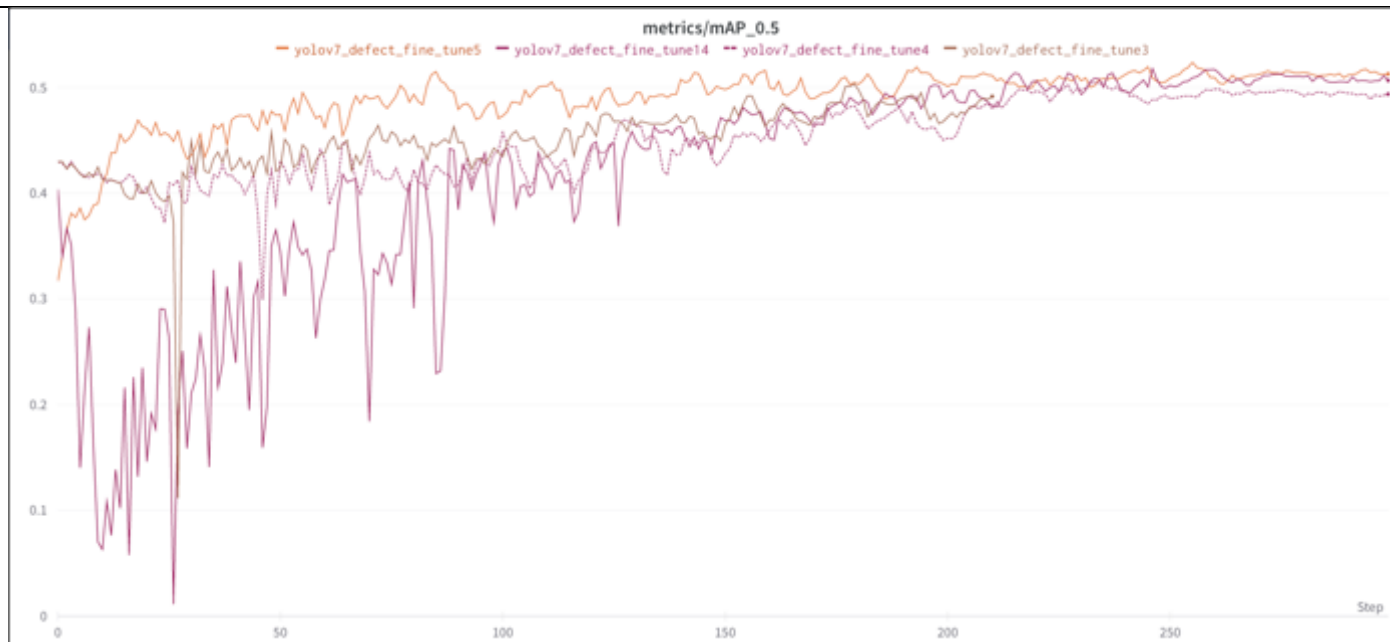


Compare 4-Fold

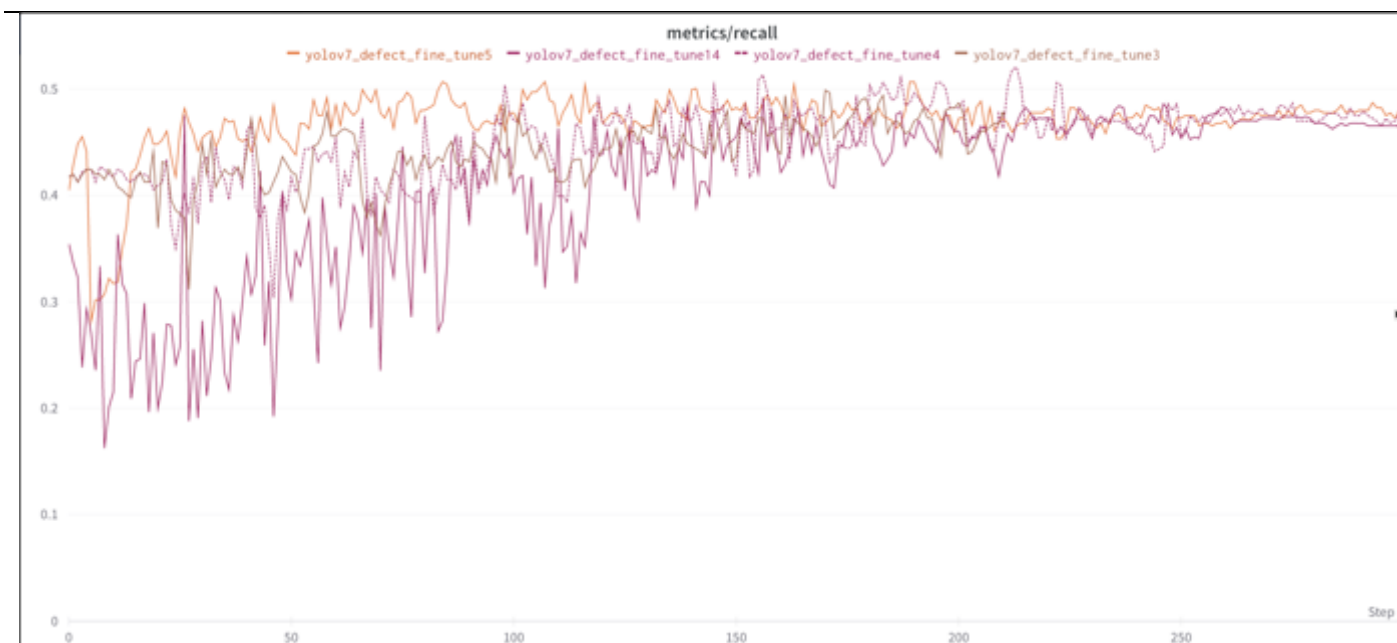
Loss Function



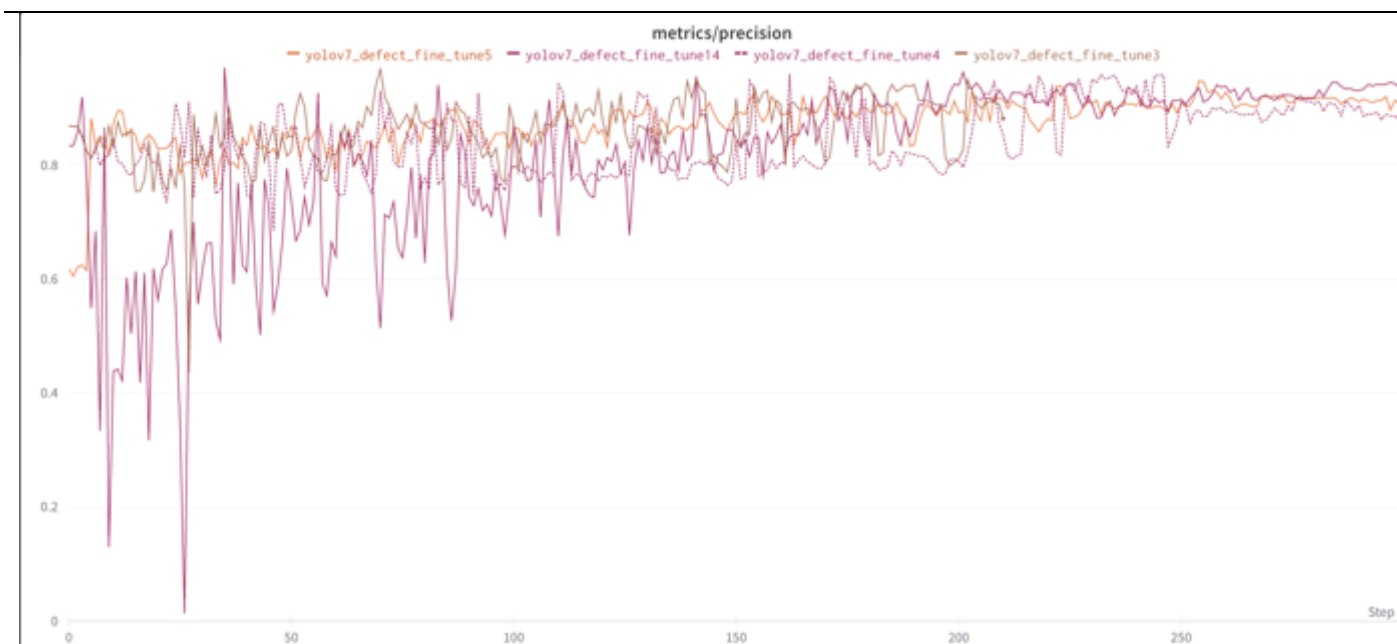
AP50



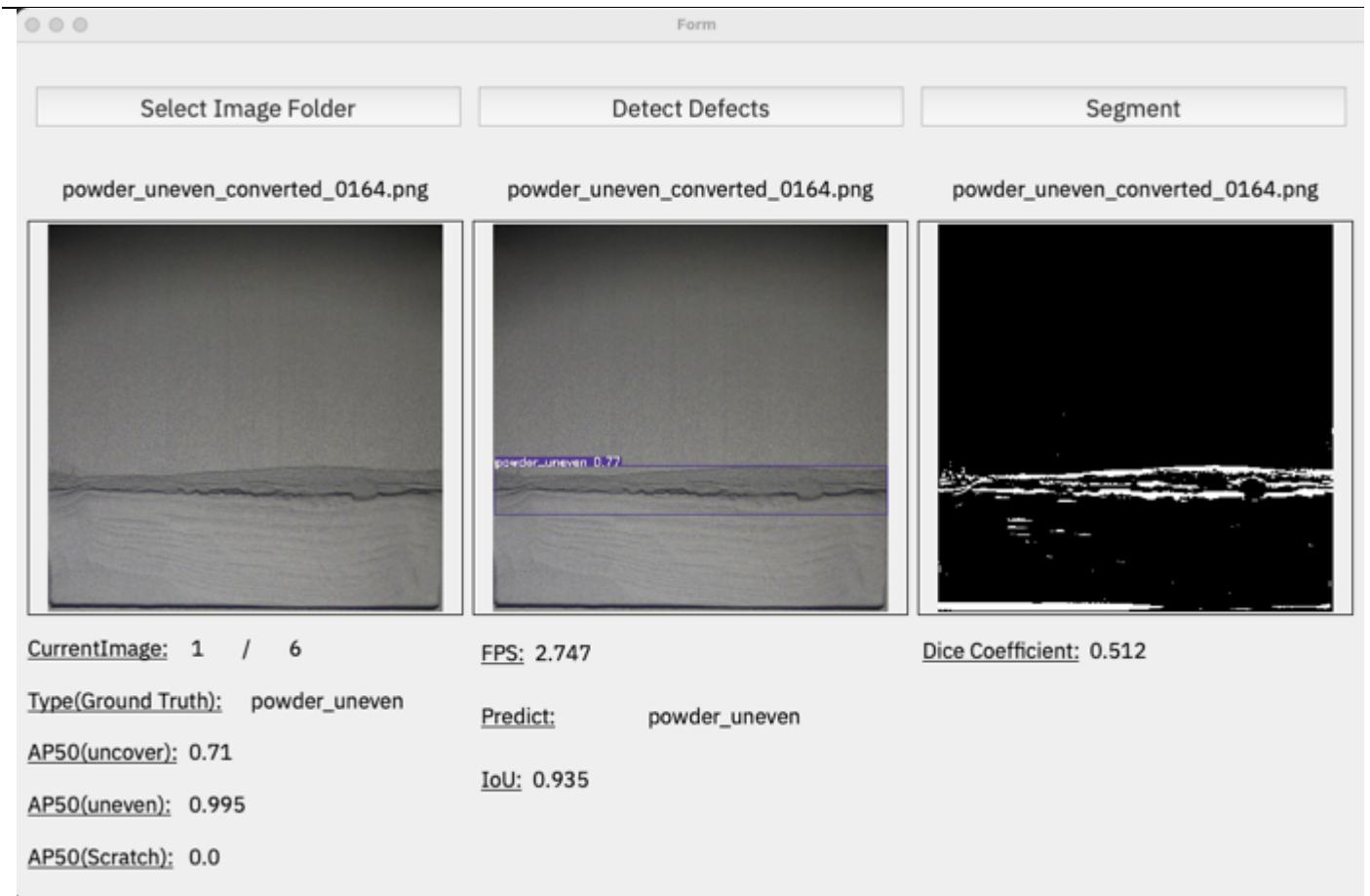
Recall



Precision



GUI



討論：

YOLOv7 在 COCO 資料集上取得了非常優秀的成績；但在目前的瑕疵偵測目標上可以看學習的成果欠佳，縱使將設為 Epoch=300、但模型依舊無法有效收斂至穩定的狀況；反之的 U-Net 在約 150 -200 Epoch 間就會穩定下來保持 0.8-0.9 的表 Dice Coefficient。

目前推測是因為資料的同質性太高，Objection Detection 的演算法在學習類別的不同樣態時遇到困難，導致 YOLOv7 無法有效地學習並收斂到穩定的狀態。

提高模型的表現有許多方法可以嘗試，包括：

1. 增加資料量：通常來說，模型的表現會隨著資料量的增加而提高。因此，可以嘗試增加資料量來提升模型的表現。
2. 調整超參數：可以試著調整超參數，如學習率、batch size 等，以達到最佳的表現。
3. 對模型進行更多的微調：可以試著對模型進行更多的微調，如更換不同的損失函數、更換不同的優化器等，以達到最佳的表現。
4. 嘗試使用不同的模型：如果 YOLOv7 在這個目標上表現不佳，可以嘗試使用不同的模型，如：RCNN 系列。

總結：

在金屬激光熔化製造（MAM）中，瑕疵檢測和分割是非常重要的步驟。近年來，研究人員已經取得了不錯的結果，這些結果對於提高 MAM 技術的效率和質量具有重要意義。

在本次的實驗當中我們使用了 YOLOv7 來偵測瑕疵的位置、並使用 U-Net 來進行分割，分別取得了 $\text{dice coefficient} = 0.92$ 和 $\text{AP50} = 0.50$ 的成績；雖然 Object Detection 未如理想、但 Segmentation 仍舊有不錯的成績。

然而，還有許多挑戰需要解決，例如如何在高速生產過程中實時偵測瑕疵，如何更準確地分割出瑕疵區域，以及如何提高瑕疵偵測和分割的準確度。通過繼續探索這些問題，研究人員可以為 MAM 技術的發展做出貢獻。