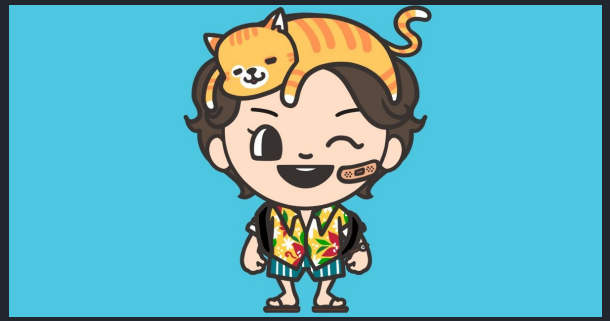
불량 반도체 탐지 프로그램

이당백 팀 : 김진 | 박민아

cakd3

이당백 팀을 소개합니다



김진

팀장, 프로젝트 총괄, 기획 총괄,
이미지 데이터 전처리,어노테이션,
모델링, 파이썬 스크립트 파일 생성, QA

git:https://github.com/rumcrush
e-mail : camuscheer333@gmail.com

박민아

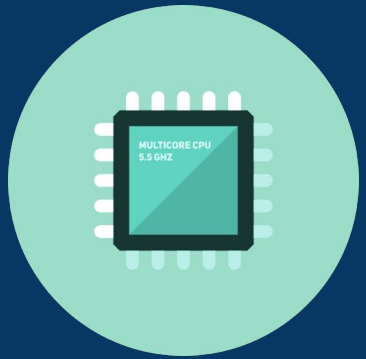
개발 총괄, 데이터 전처리,
이미지 자료 DB 구축(pandas 활용)
OpenCV 활용 전처리, 어노테이션
모델링, 파이썬 스크립트 파일 생성, QA

git : https://github.com/parkmina365
e-mail : parkmina365@gmail.com

목차

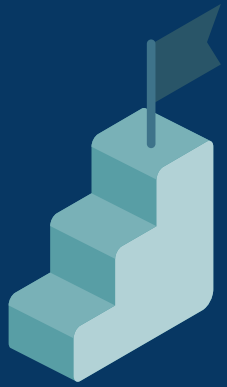
목차

1. 프로젝트 목적 - 프로젝트 개요



FAB 과정
반도체에 전자회로를
그리는 과정

EDS 과정
양품과 불량품
반도체를 검수 하는 과정



패키지과정
최종 제품 형태를
갖추는 과정

1. 프로젝트 목적 - 프로젝트 개요

기존 반도체 품질 검수 작업 단계

1) FDC 시스템 : 시스템으로 반도체 모니터링, 분석 후 불량감지

2) 엔지니어가 수작업으로 불량품 분류

제안 사항

반도체 수작업 품질 검수 단계에
딥러닝 기반 AI 솔루션 도입

1. 프로젝트 목적 - 프로젝트 개요

“AI 반도체 장비 불량품 탐지 프로그램을 통한

반도체 불량품 자동 예측,

*제작 공정상 **자원 절약**”*

1. 프로젝트 목적 - 개발 환경 및 도구

어노테이션 툴

실행 환경

사용 모델

모델 명
YOLOv5
YOLO R
Detectron2
MobileNetSSDv2

사용 라이브러리

라이브러리 명	버전	라이브러리 명	버전	라이브러리 명	버전
Pytorch	-	scipy	1.4.1	PyYAML	5.3.1
tensorboard	2.4.1	matplotlib	3.2.2	seaborn	0.11.0
tensorflow	2.4.1	Pillow	7.1.2		
torchvision	-	open cv	4.1.2		

목차

2. 데이터 전처리

1) 이미지 데이터 탐색

2) 원본 json 파일 탐색

3) 전처리 진행 및 제안

2. 데이터 전처리 - 전체 제작 흐름

1단계
데이터 탐색

1) 이미지 데이터 탐색
2) 전처리 용 이미지 선정

2단계
데이터 전처리

1) 어노테이팅
2) 바운딩박스 만들기

3단계
모델링 학습 및 평가,
프로그램 구축

1) 모델링
2) 파이썬 스크립트 파일 생성
3) QA
4) 완성

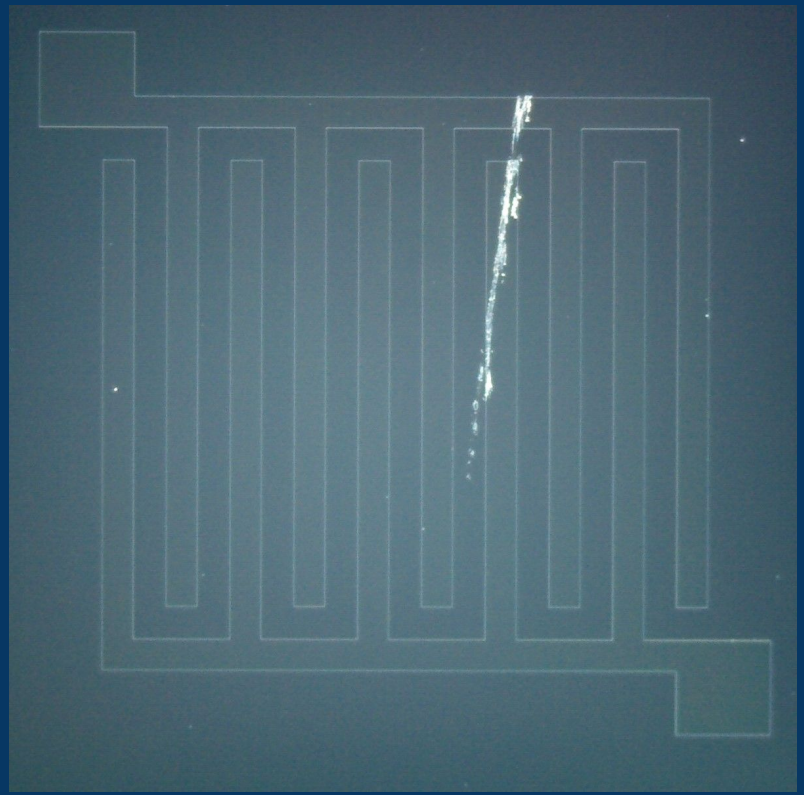
2. 데이터 전처리 - 이미지 데이터 탐색

원본 이미지 데이터 상세 내역

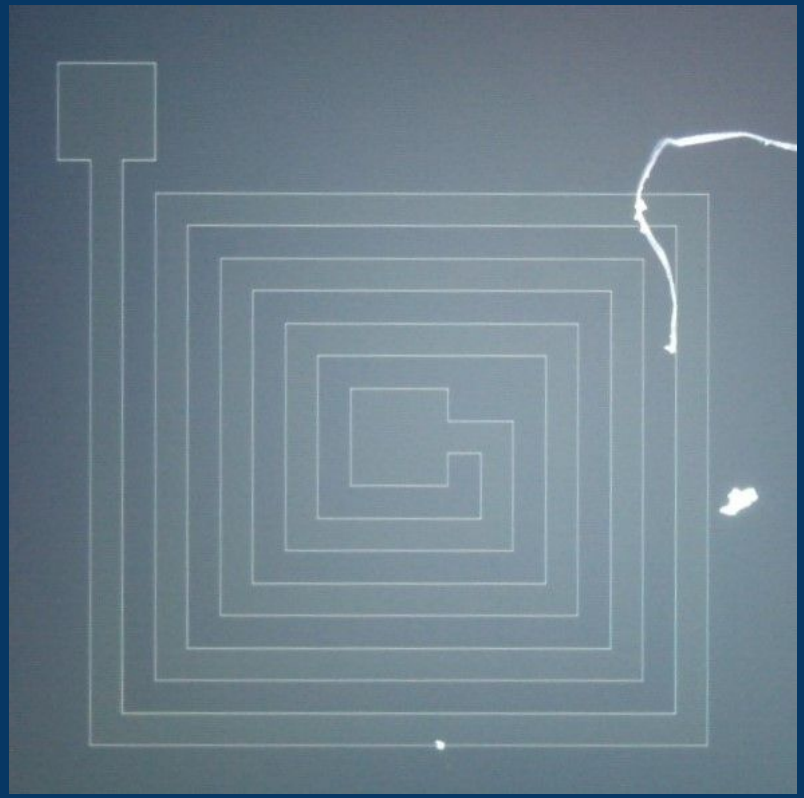
- **총 이미지 데이터 장수 : 337 장**
- 이미지 정보 : 1200 * 1200 px
- 파일 형식 : jpg

패턴 종류	반도체 층 종류	이미지 데이터 개수	어노테이션 클래스 종류
1	9	118	9
2	9	162	9

2. 데이터 전처리 - 이미지 데이터 탐색



Pattern1(총 9종)



Pattern2(총 9종)

2. 데이터 전처리 - 원본 json파일 탐색

```
import json
with open('pattern_1/pattern_1.json') as data_file:
  pattern_1_json = json.load(data_file)
pattern_1_json['images'][60]
```

```
{'dataset_name': 'pattern_type_1',
 'height': 1199,
 'image_id': '6b431058-8652-4fd7-93e4-36a039e4d034',
 'image_mode': 'YCbCr',
 'image_name': 'pattern_1-1-4-12-D.jpg',
 'image_status': 'DONE',
 'labels': [{'attributes': {},
   'bbox': [585, 523, 750, 979],
   'class_name': 'defect_type_4',
   'id': '6c202233-71d3-426e-9008-afcd1c4bbccb',
   'mask': None,
   'polygon': None,
   'z_index': 1},
  {'attributes': {},
   'bbox': [176, 608, 189, 621],
   'class_name': 'defect_type_1',
   'id': 'f7fc0281-c94c-43a0-9848-be22c9b8955b',
   'mask': None,
   'polygon': None,
   'z_index': 0}],
 'tags': [],
 'width': 1200}
```

- json 데이터에서 이미지 정보가 있는 labels 값을 활용. 이미지 별 바운딩박스 좌표, 어노테이트 클래스 파악

2. 데이터 전처리 - 원본 json파일 탐색

	image_name	label_nums	class_name1	0	1	2	3	class_name2	4	5	6	7
0	pattern_1-1-1-12-D.jpg	1	defect_type_1	139.0	1058.0	162.0	1077.0	NaN	NaN	NaN	NaN	NaN
1	pattern_1-1-1-19-D.jpg	1	defect_type_1	702.0	1009.0	712.0	1018.0	NaN	NaN	NaN	NaN	NaN
2	pattern_1-1-1-27-D.jpg	1	defect_type_1	978.0	712.0	1009.0	749.0	NaN	NaN	NaN	NaN	NaN
3	pattern_1-1-1-28-D.jpg	1	defect_type_1	416.0	1037.0	561.0	1122.0	NaN	NaN	NaN	NaN	NaN
4	pattern_1-1-1-29-D.jpg	1	defect_type_1	1111.0	942.0	1172.0	1195.0	NaN	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...
104	pattern_1-1-6-10-D.jpg	6	defect_type_6	1009.0	310.0	1086.0	427.0	defect_type_1	1058.0	936.0	1073.0	962.0
105	pattern_1-1-7-04-D.jpg	7	defect_type_7	245.0	131.0	311.0	194.0	defect_type_1	13.0	421.0	21.0	436.0
106	pattern_1-1-2-04-D.jpg	8	defect_type_1	501.0	724.0	585.0	1016.0	defect_type_1	751.0	136.0	775.0	180.0
107	pattern_1-1-6-09-D.jpg	8	defect_type_1	399.0	164.0	423.0	221.0	defect_type_6	1012.0	310.0	1078.0	427.0
108	pattern_1-1-8-07-D.jpg	8	defect_type_1	882.0	670.0	894.0	686.0	defect_type_1	102.0	1151.0	157.0	1178.0

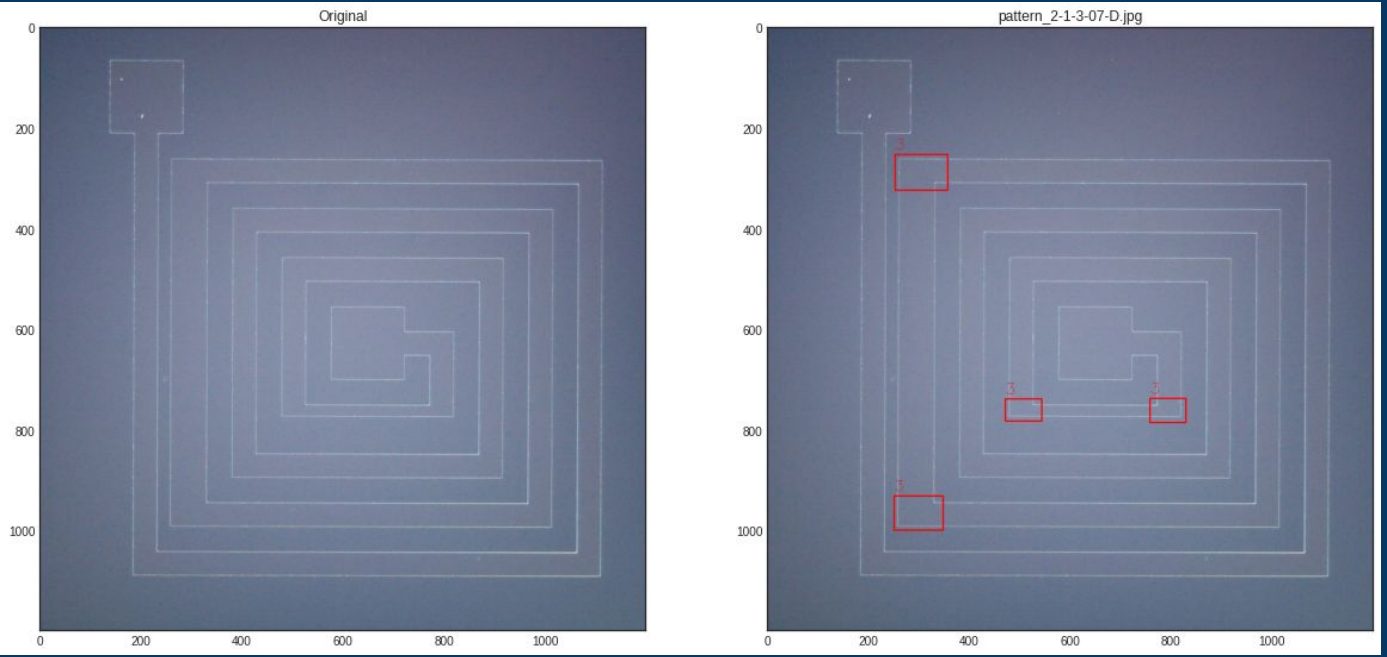
- 각 이미지 당 최대 결함 개수: 8개
- 어노테이트 클래스 종류: 9개

2. 데이터 전처리 - 원본 json파일 탐색

```
# df_json_1_label_1
import cv2
import matplotlib.pyplot as plt

for i in range(len(df_json_1_label_1)):
  img = cv2.imread(df_json_1_label_1['image_name'].iloc[i])
  img_draw = cv2.imread(df_json_1_label_1['image_name'].iloc[i])
  start1 = (df_json_1_label_1[0].iloc[i].astype(int), df_json_1_label_1[1].iloc[i].astype(int))
  start1_1 = (df_json_1_label_1[0].iloc[i].astype(int), df_json_1_label_1[1].iloc[i].astype(int)-10)
  end1 = (df_json_1_label_1[2].iloc[i].astype(int), df_json_1_label_1[3].iloc[i].astype(int))

  img_draw = cv2.rectangle(img_draw, start1, end1, (0,0,255), 2)
  img_draw = cv2.putText(img_draw, df_json_1_label_1['class_name1'].iloc[i][-1:], start1_1, cv2.FONT_ITALIC, 1, (0,0,255))

  plt.figure(figsize=(20,20))
  plt.subplot(121), plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB)), plt.title('Original')
  plt.subplot(122), plt.imshow(cv2.cvtColor(img_draw, cv2.COLOR_BGR2RGB)), plt.title(df_json_1_label_1['image_name'].iloc[i])
  plt.show()
```

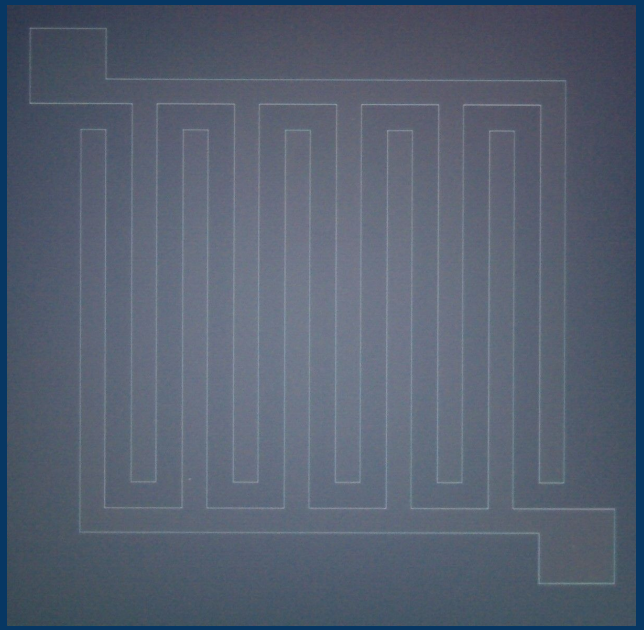
- json 파일 상의 바운딩 박스 좌표값으로 원본 이미지에 바운딩 박스 출력(OpenCV 활용)

2. 데이터 전처리 - 원본 json파일 탐색(결함 이슈 - 결함에 바운딩 박스 없음)

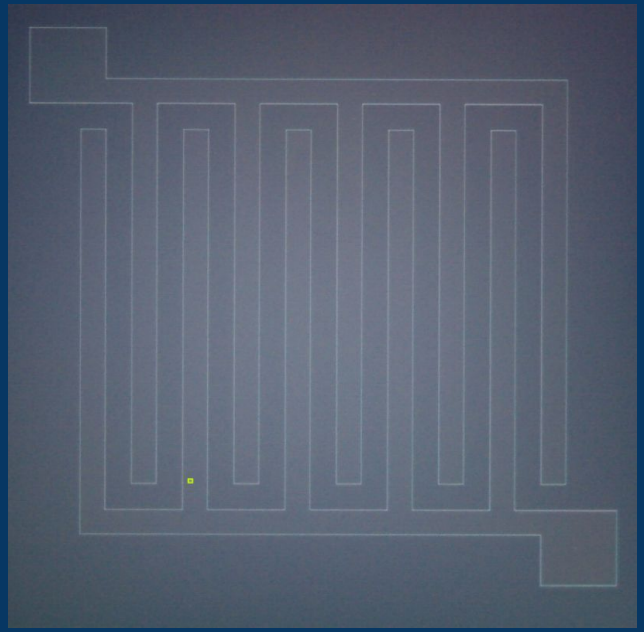


- 결함이 아닌 곳에 회로선 부분에 바운딩 박스가 쳐지고, 결함에는 바운딩 박스가 없는 이슈
- 실제 결함 위치에 바운딩 박스 처리 필요

2. 데이터 전처리 - 원본 json파일 탐색(결함 이슈 - 양품으로 오판)



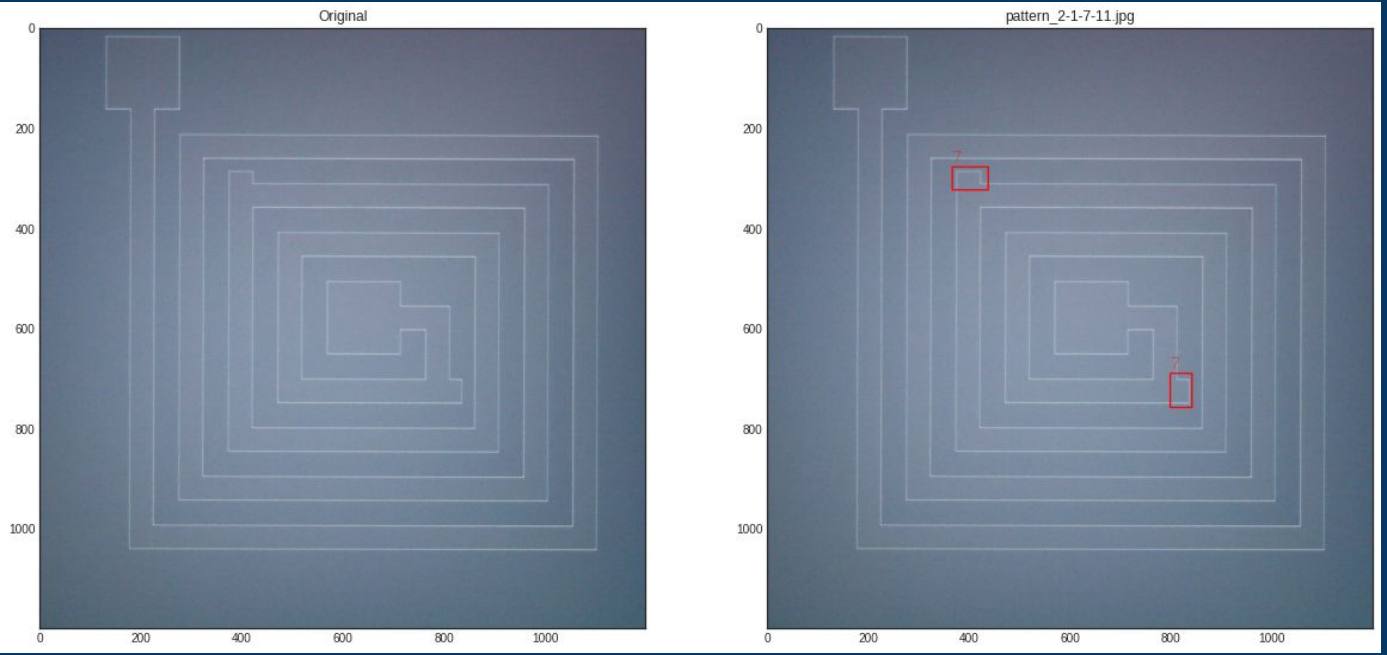
pattern_1-1-1-04.jpg
원본 이미지



pattern_1-1-1-04.jpg
전처리 후 바운딩박스 처리 결과

- 양품으로 판정 되었으나(파일 이름에 -D가 없음) 실제로는 결함이 발견된 이슈
- 양품으로 오판되어 결함에 대한 바운딩 박스 데이터가 없음
- 불량품으로 재설정 후 실제 결함 위치에 대한 바운딩 박스 처리 필요

2. 데이터 전처리 - 원본 json파일 탐색(결함 이슈 - 불량품으로 잘 못 분류)



- 파일명 상으로는 불량품으로 판정 되었으나 실제로는 결함이 없는 이슈
- 층 1과 다른 회로선을 가진 반도체 이미지에서 발생된 이슈
- 양품인데 불량품으로 오판단 된 것으로, 양품으로 처리 후 학습에서 제외 필요

2. 데이터 전처리 - 원본 json파일 탐색(패턴 1 이슈 현황)

패턴1 원본 데이터 현황		
내용	개수	비율
총 개수	136	100%
불량 개수	114	84%
양품 개수	22	16%

패턴1 실제 데이터 현황		
내용	개수	비율
총 개수	**136**	**100%**
실제 불량 개수	**118**	**87%**
실제 양품 개수	**18**	**13%**

- 양품으로 표시 되었으나 실제 불량품인 개수: 22개 중 4개

2. 데이터 전처리 - 원본 json파일 탐색(패턴 2 이슈 현황)

패턴2 원본 데이터 현황		
내용	개수	비율
총 개수	201	100%
불량 개수	165	82%
양품 개수	36	18%

패턴2 실제 데이터 현황		
내용	개수	비율
총 개수	201	100%
실제 불량 개수	162	81%
실제 양품 개수	39	19%

- 불량으로 표시 되었으나 실제 양품인 개수: 165개 중 6개
- 양품으로 표시 되었으나 실제 불량품인 개수: 36개 중 3개

2. 데이터 전처리 - 원본 json파일 탐색

json 파일 Model Summary(YOLOv5)

-> train 70%, validation 15%, test 15%로 모델링 진행

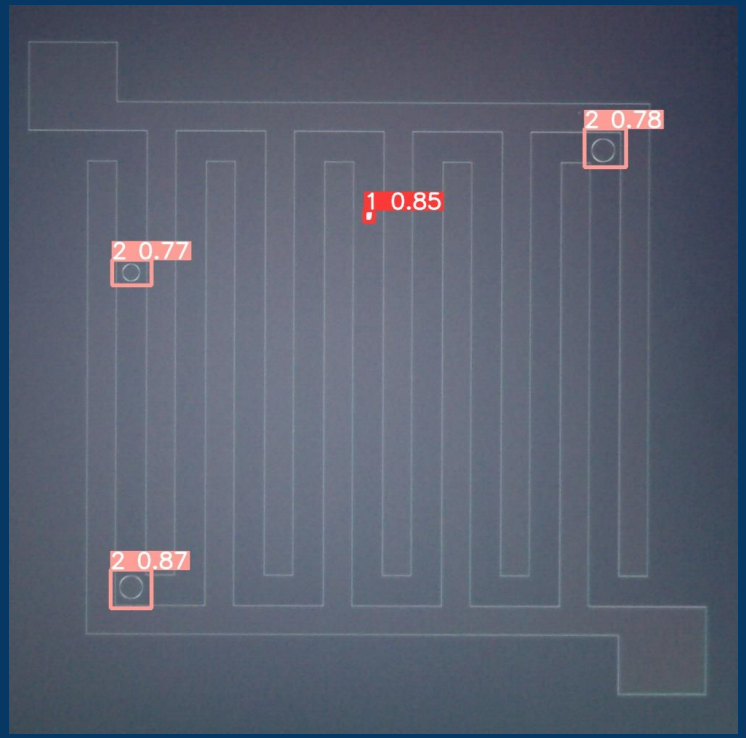
```
Model Summary: 290 layers, 20885262 parameters, 0 gradients, 48.0 GFLOPs
```

Class	Images	Labels	P	R	mAP@.5	mAP@.5:.95: 100% 1/1 [00:01<00:00, 1.72s/it]
all	41	147	0.95	0.957	0.955	0.711
1	41	60	0.881	0.615	0.666	0.334
2	41	12	0.912	1	0.963	0.623
3	41	10	0.872	1	0.995	0.864
4	41	11	0.978	1	0.995	0.808
5	41	12	0.977	1	0.995	0.714
6	41	12	1	1	0.995	0.69
7	41	8	0.967	1	0.995	0.647
8	41	14	0.982	1	0.995	0.818
9	41	8	0.977	1	0.995	0.899

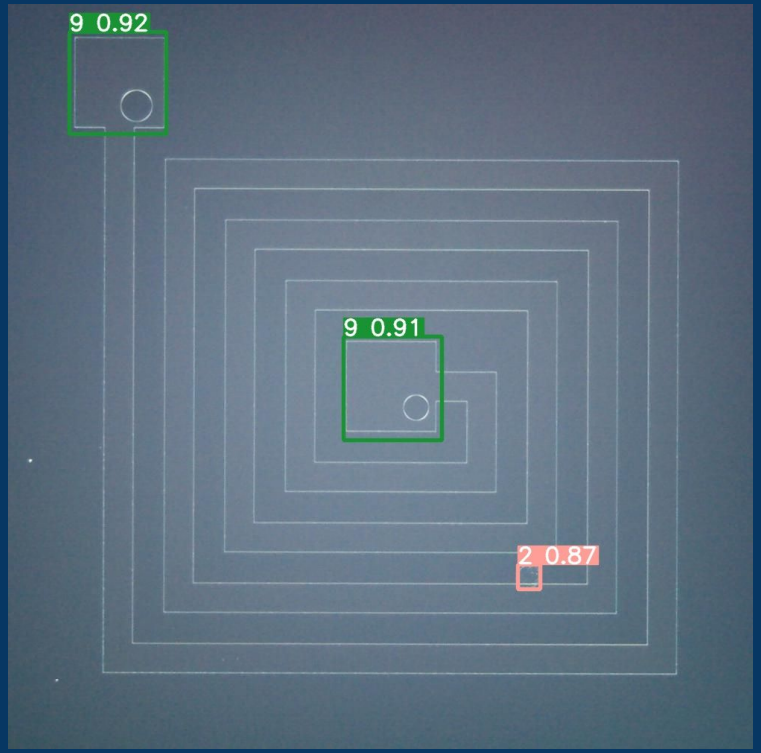
- 실제 결함 클래스는 총 9개 중 1종류(1번 클래스)
- mAP : 0.666 도출
- 작은 점 모양의 결함은 학습하지 않고, 결함이 아닌 곳, 반도체 회로선 등을 결함으로 탐지하는 등의 이슈 발생

2. 데이터 전처리 - 원본 json파일 탐색

json Inference (image)



pattern_1-1-2-09-D.jpg 이미지 결과



pattern_2-1-9-05-D.jpg이미지 결과

2. 데이터 전처리 - 전처리 진행 및 제안

json 데이터 이슈 및 전처리 시 개선사항

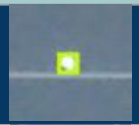
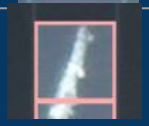
- json 파일 바운딩박스 좌표값 이슈
  1) 작은 점 형태 좌표값 없음. 작은 결함 탐지 못함
  2) 실제 결함이 아닌 곳에 좌표 설정

  -> 반도체 결함 탐지용으로 적절치 않다고 판단
  직접 전처리 진행

- 전처리 시 개선 사항
  1) 다양한 형태 결함 탐지
  2) 작은 크기의 결함 탐지

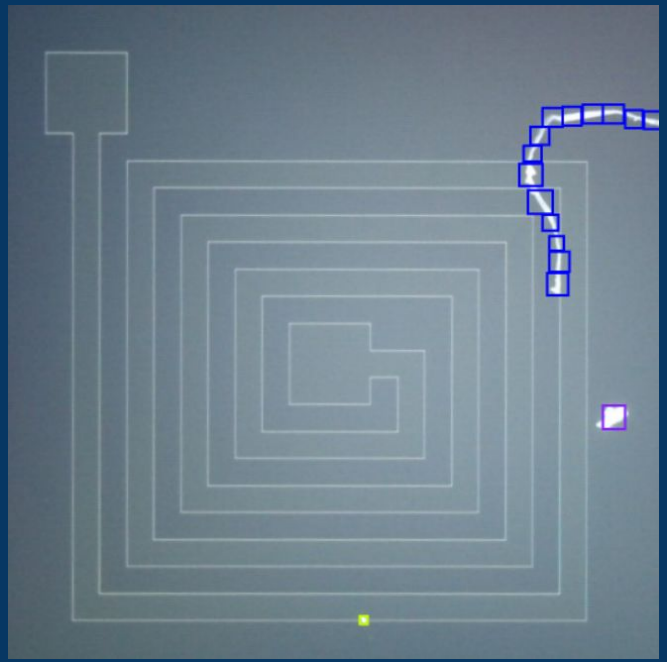
  roboflow.com(어노테이트 툴)을 사용하여 어노테이션 진행

2. 데이터 전처리 - 전처리 진행 및 제안

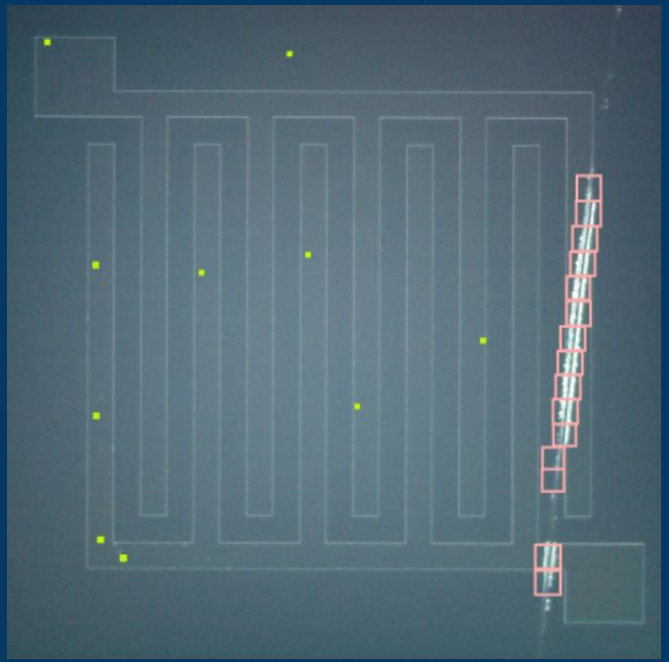
기준	어노테이션 명	예시 이미지
작은 점	dot	
큰 점	Bdot (Big dot)	
직선형 스크래치	straight	
곡선형 스크래치	curve	

- 형태가 명확하게 구분되는 결함으로 선정
- 종류별로 어노테이트 이름 설정, 총 4개로 분류

2. 데이터 전처리 - 전처리 진행 및 제안

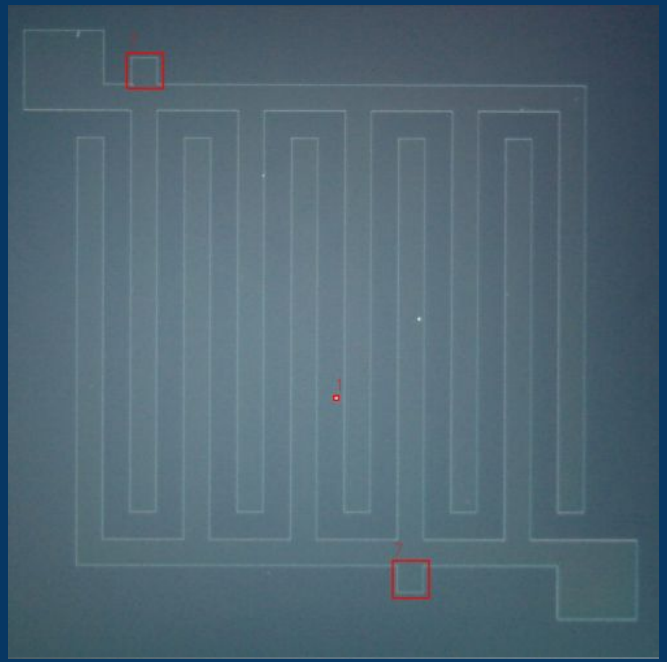


curve, Bdot, dot
바운딩박스 처리 이미지

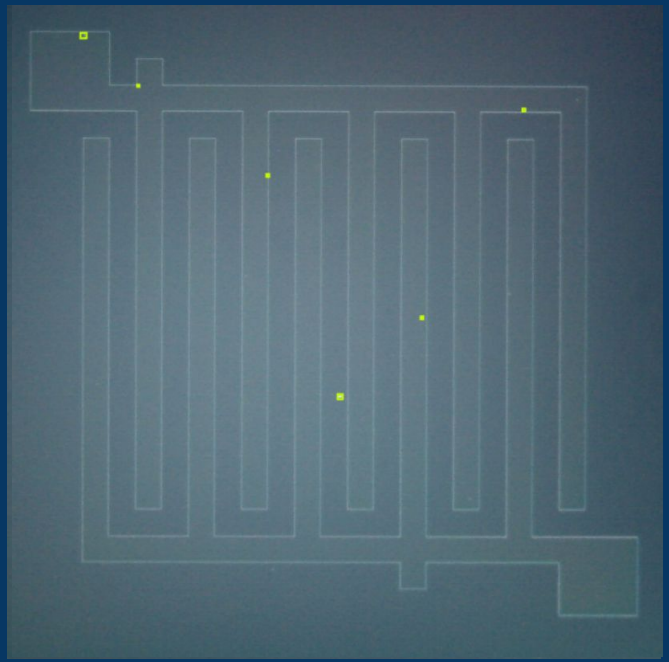


straight, dot
바운딩박스 처리 이미지

2. 데이터 전처리 - 전처리 진행 및 제안



json 데이터 상
바운딩박스 처리 결과



전처리 후
바운딩박스 처리 결과

목차

3. 학습 및 평가

1) 모델 학습 및 평가 결과

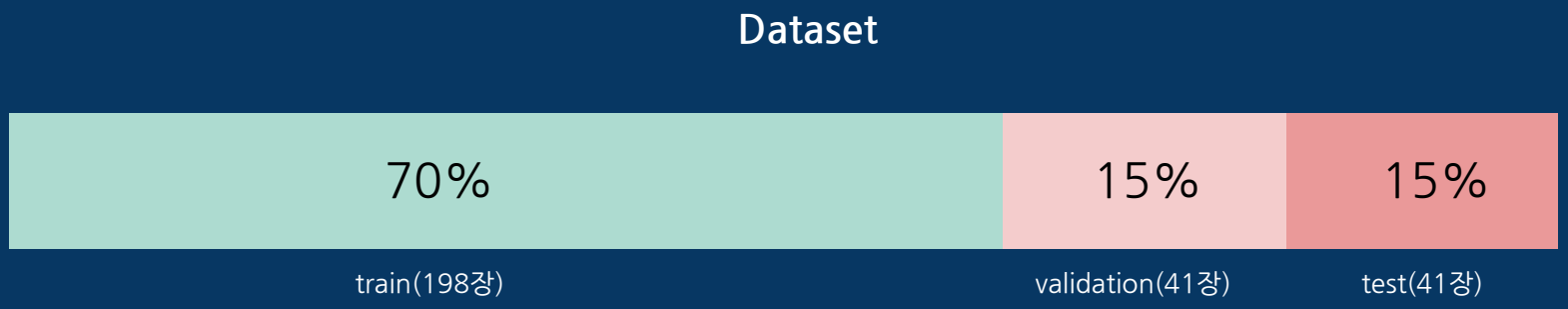
2) 프로그램 사용 방법

3) 솔루션 활용 방안

3. 학습 및 평가 - 모델 학습 및 평가 결과

학습 비율 설정 및 모델 출력

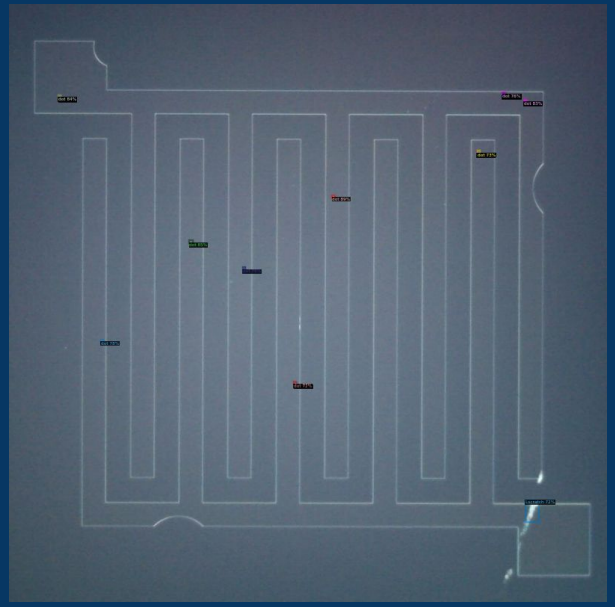
- 총 학습 이미지 개수 : 280

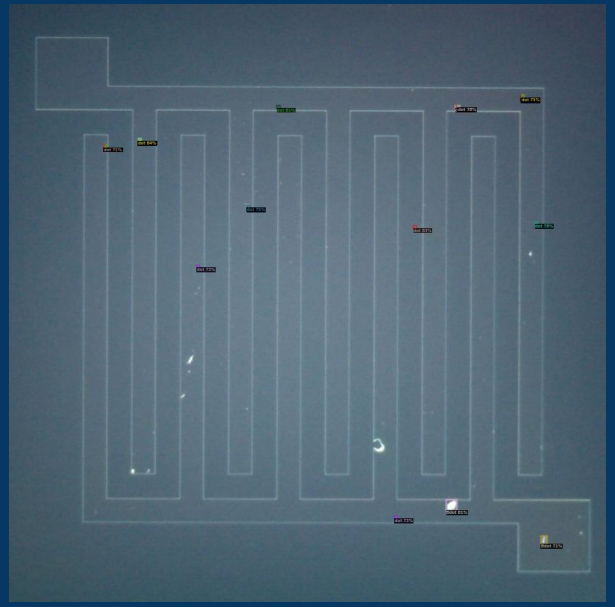


- 본 프로젝트에서는 총 4개의 모델 사용. 각 모델 용으로 데이터 형식 변환하여 모델링 진행
- 총 13회차에 걸친 모델링 학습 진행

3. 학습 및 평가 - 모델 학습 및 평가 결과

Detectron2

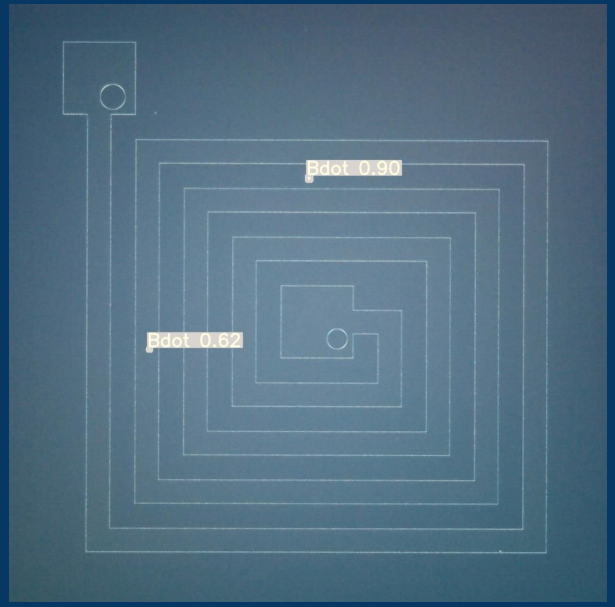


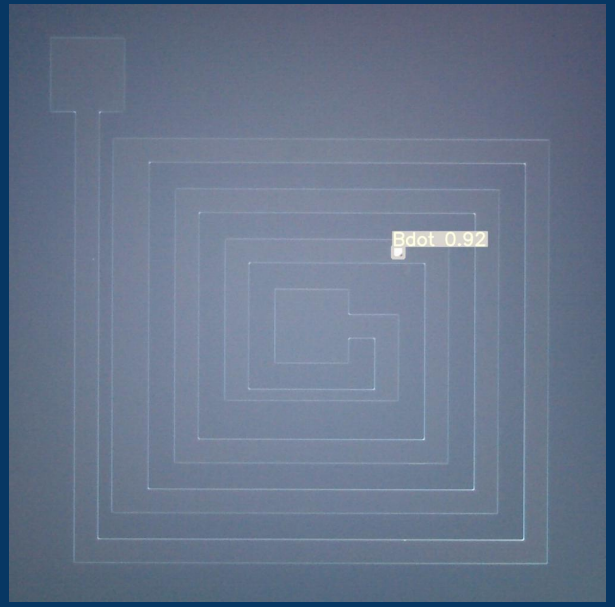


- Detectron2 모델 사용, 최대 객체 탐지율 0.92 도출
- 결함 중 명확하게 보이는 결함을 탐지 못하는 문제 발생
- 프로그램의 취지를 고려, 다른 모델 탐색

3. 학습 및 평가 - 모델 학습 및 평가 결과

YOLO R

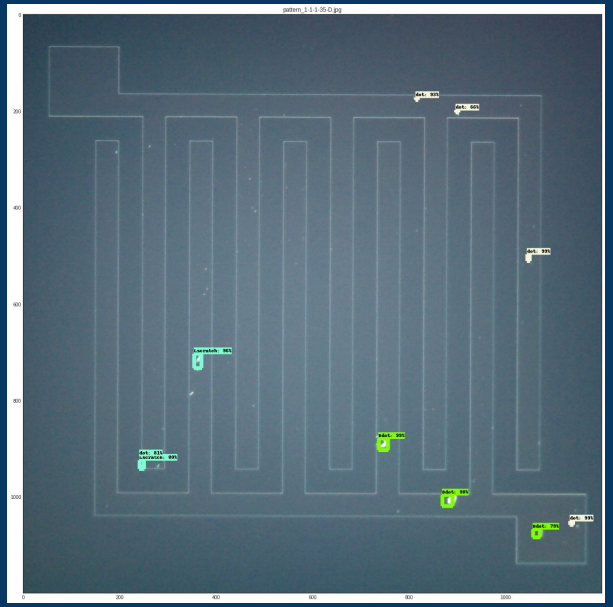


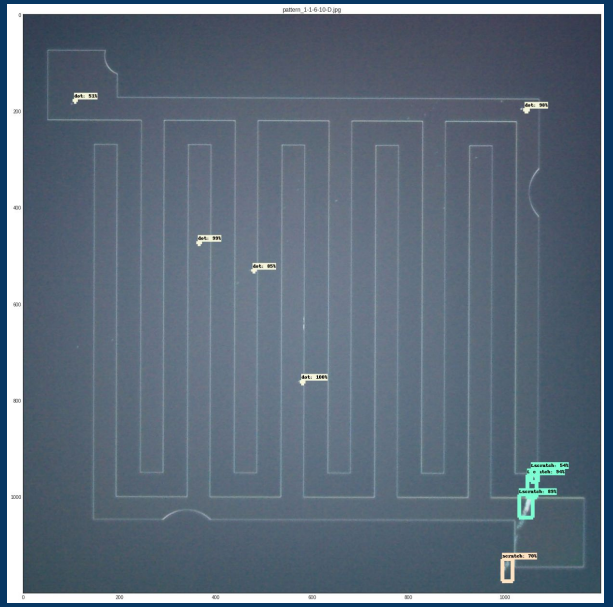


- YOLO R 사용 결과, 개별 결함 최대 객체 탐지율 0.92 도출
- 작은 크기의 결함은 잘 탐지하지 못하는 문제 발생
- 결함 탐지율을 높이기 위해 타 모델 탐색

3. 학습 및 평가 - 모델 학습 및 평가 결과

MobileNetSSDv2





- FasterRCNN 기반의 mobileNetSSDv2 사용. 개별 결함 최대 객체 탐지율 0.99 도출
- 전반적인 탐지율은 상승 했으나 모델링 학습 1회 시 18시간 소요.
- 반도체 불량품 판단 공정 시 적절하지 않다고 판단. 프로그램의 이용성을 고려, 다른 모델 탐색

3. 학습 및 평가 - 모델 학습 및 평가 결과

YOLOv5

```
[ ]  img_size = 600
     bacth_size = 32
     epochs = 400
     weights_size = 'm' # s,m,l,x
     name = 'yolov5'
     name_fix = name+weights_size
     data_path = '/content/yolov5/data.yaml'
     yalm_path = '/content/yolov5/models/{}.yaml'
     weights_path = '/content/yolov5/{}.pt'

     name = 'dot_{}_results'
```

```
[ ]  # train
     !python3 /content/yolov5/train.py --patience 0 --img {img_size} --batch {bacth_size} --epochs {epochs} --data {data_path} ₩
     --cfg {yalm_path.format(name_fix)} --weights {weights_path.format(name_fix)} --name {name.format(name_fix)}
```

- YOLOv5m 모델 사용 500epochs로 모델 추론 진행

3. 학습 및 평가 - 모델 학습 및 평가 결과

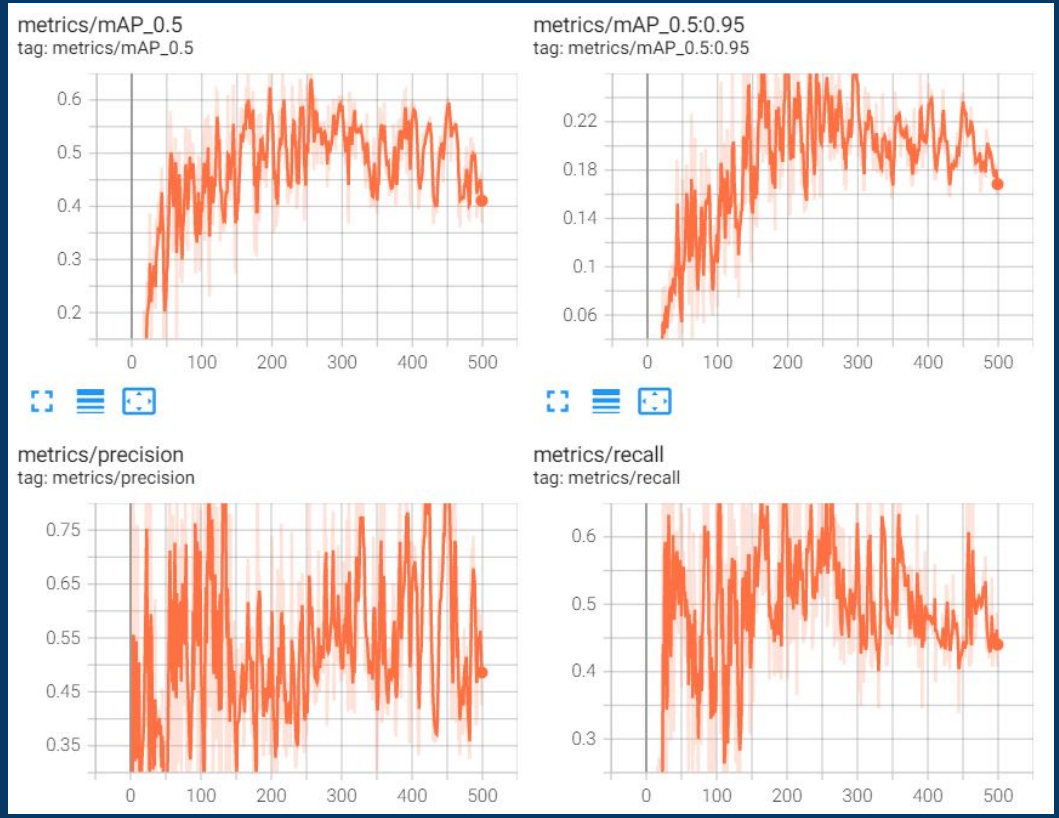
YOLOv5 Model Summary

```
500 epochs completed in 10.729 hours.
Optimizer stripped from runs/train/dot_yolov5m_results/weights/last.pt, 42.2MB
Optimizer stripped from runs/train/dot_yolov5m_results/weights/best.pt, 42.2MB

Validating runs/train/dot_yolov5m_results/weights/best.pt...
Fusing layers...
Model Summary: 290 layers, 20865057 parameters, 0 gradients, 48.0 GFLOPs
               Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95: 100% 1/1 [00:04<00:00,  4.93s/it]
                 all         41        113      0.525      0.681      0.622      0.351
                Bdot         41          2          1      0.819      0.995      0.746
               curve         41         17      0.407      0.647       0.34     0.0962
                 dot         41         91      0.437      0.593      0.486       0.16
            straight         41          3      0.255      0.667      0.665        0.4
Results saved to runs/train/dot_yolov5m_results
```

- YOLOv5를 500 epochs로 모델링 진행
- 클래스 별 최대 mAP : 0.995 도출 (Bdot)
- 성능 및 학습 속도를 고려하여 YOLOv5를 최종 사용 모델로 선정

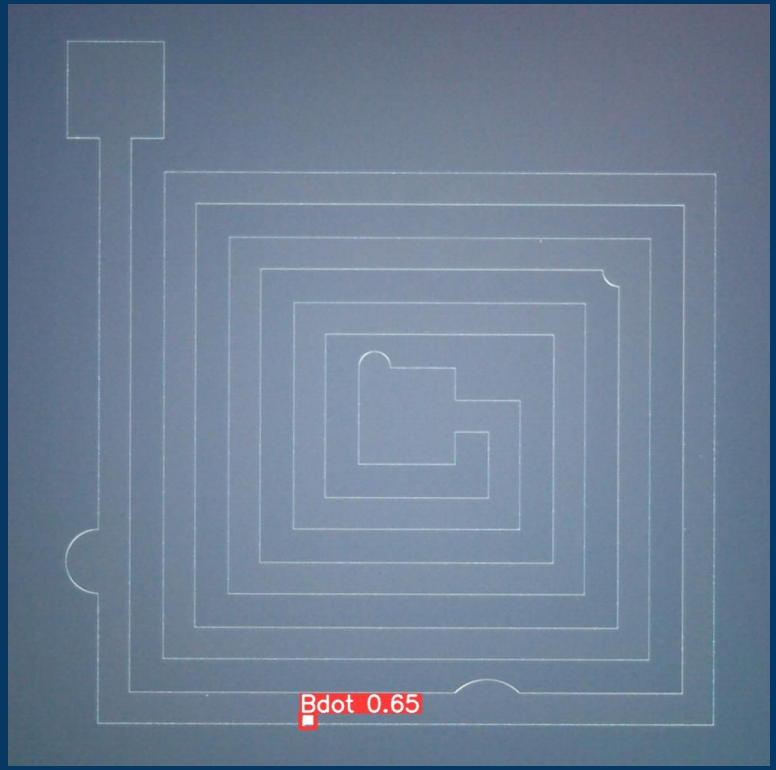
3. 학습 및 평가 - 모델 학습 및 평가 결과

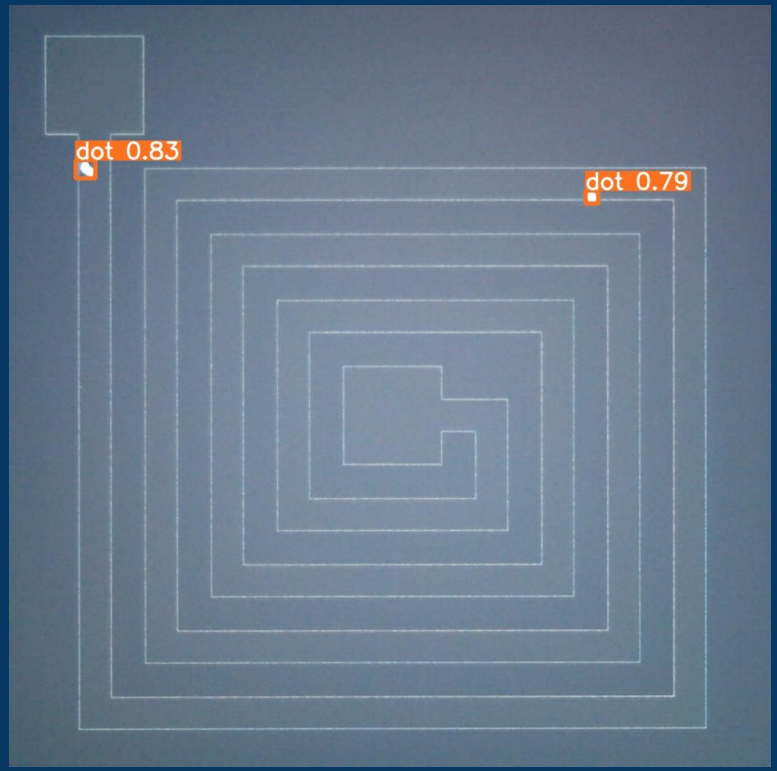
YOLOv5 TensorBoard



3. 학습 및 평가 - 모델 학습 및 평가 결과

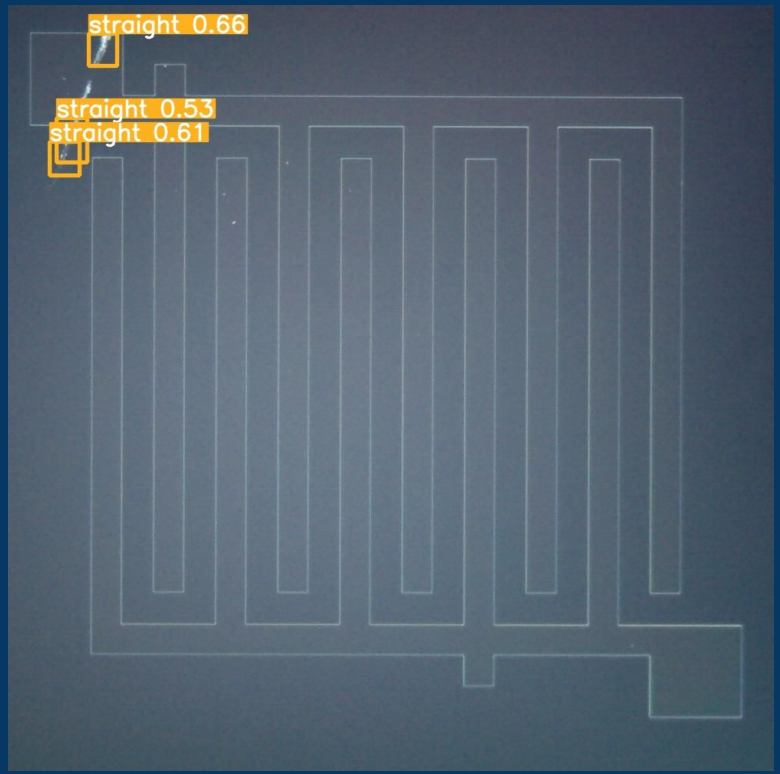
YOLOv5 Inference (image)

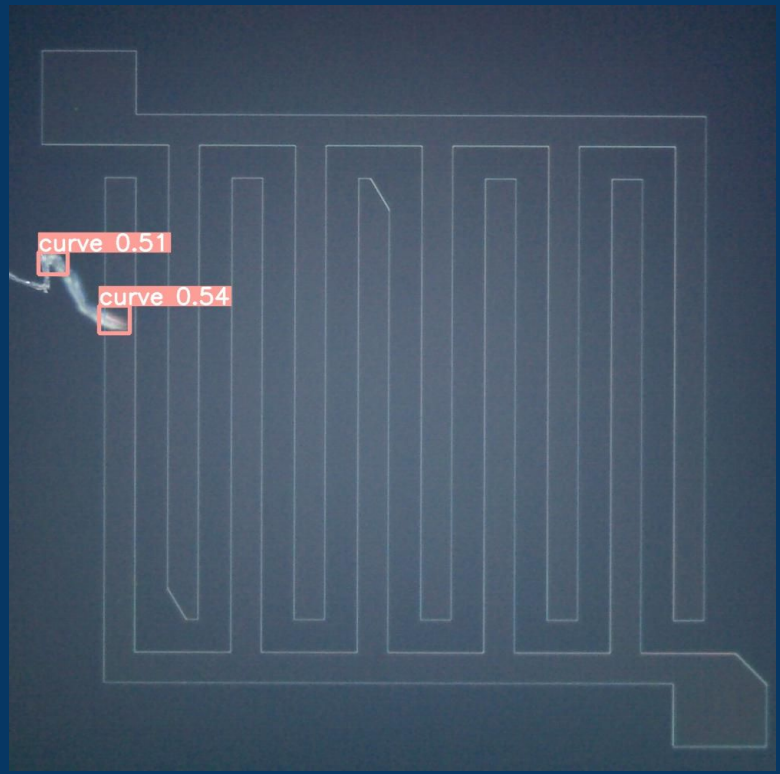




3. 학습 및 평가 - 모델 학습 및 평가 결과

YOLOv5 Inference (image)





3. 학습 및 평가 - 모델 학습 및 평가 결과

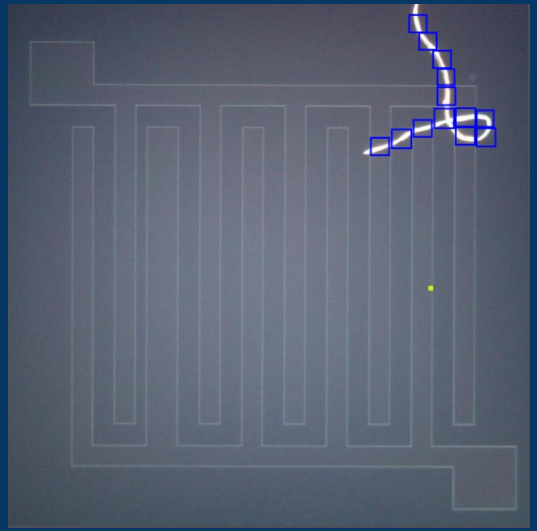
전처리 전, 후 결함 클래스 개선

	원본 json 데이터 (전처리 전)	YOLOv5 모델링결과 (전처리 후)
클래스 이름	1~9	dot, Bdot, Straight, curve
결함 종류	1개	4개
특징	클래스 1의 결함만 실제 결함 클래스 2-9는 실제결함이 아닌 회로 상 특징	점, 선, 크기로 결함 분류
결과	반도체 회로를 결함으로 탐지 작은 결함 탐지 불가	**크기와 형태에 상관없이 반도체 결함 탐지 가능**

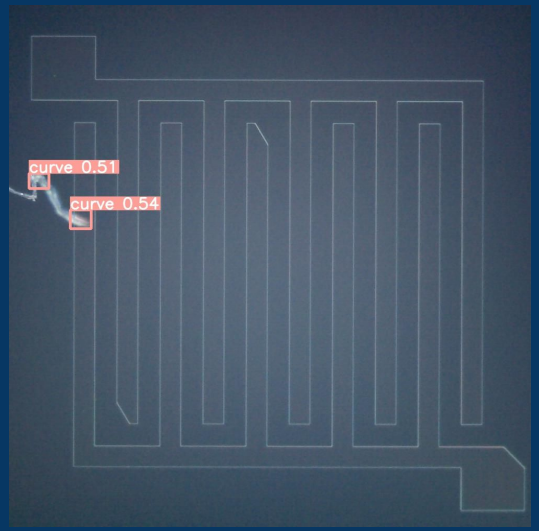
3. 학습 및 평가 - 모델 학습 및 평가 결과

한계점 및 향후 개선안

- curve 클래스 mAP 향상



curve와 dot 결함을 학습한 이미지



curve 결함이 탐지된 test 이미지

- curve 결함이 있는 이미지가 5장 밖에 없으며 곡선형태로 이루어져 부분별로 다양한 형태를 띄고 있음
- test의 결함 모양과 학습한 결함 모양과 다른 결함 모양이 대부분, 다양한 부분을 학습 시킴
- mAP 낮게 도출 되었으나 전체 중 부분적으로 결함을 탐지함

3. 학습 및 평가 - 모델 학습 및 평가 결과

한계점 및 향후 개선안

- 사용자가 더 쉽게 사용할 수 있도록 GUI 프로그램으로 구현
- Django 또는 Windows Executable으로
  cmd 접속 없이 프로그램 내에서 바로 불량품 탐지 결과가 나오도록 개선

3. 학습 및 평가 - 프로그램 사용 방법

detect.py 사용법

1. semiconductor.zip 파일 다운후 C드라이브에 압축풀기

2. cmd 입력창에 명령어 입력

1) 경로 변경 : cd C://yolov5
2) 필요 라이브러리 설치 : pip install -r requirements.txt
3) 스크립트 파일 실행 : python detect.py

3. 반도체 이미지 결함 탐지결과 저장

- 저장 경로: C://yolov5/runs/detect/result

3. 학습 및 평가 - 솔루션 활용 방안

1. AI 기반 반도체 장비 불량품 탐지 프로그램 제공

2. 사용자가 불량품 탐지가 필요한
반도체 이미지 입력 시 자동으로 불량품 예측

3. 제작 공정상 시간, 인적, 경제적 **자원 절약**

4. **반도체 수율 향상**으로 이어질 것으로 예상

감사합니다