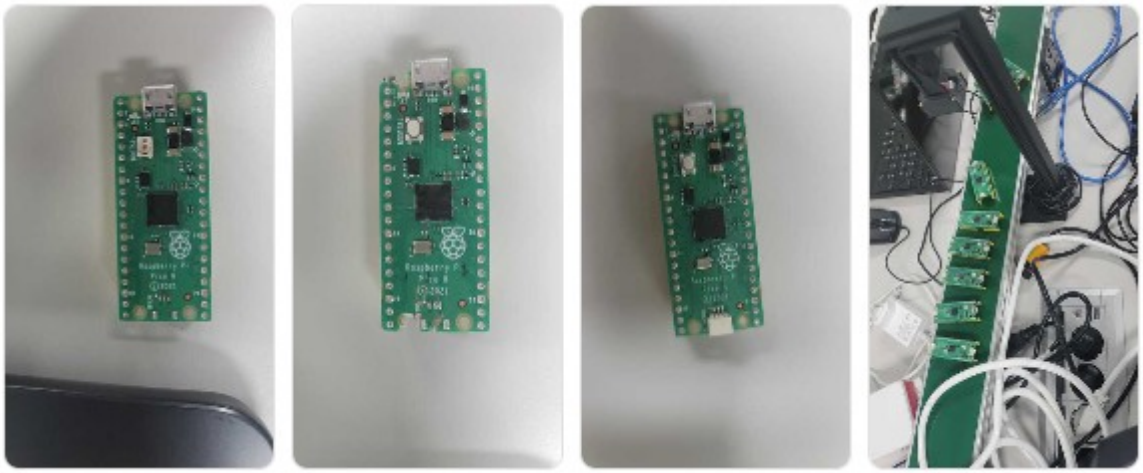
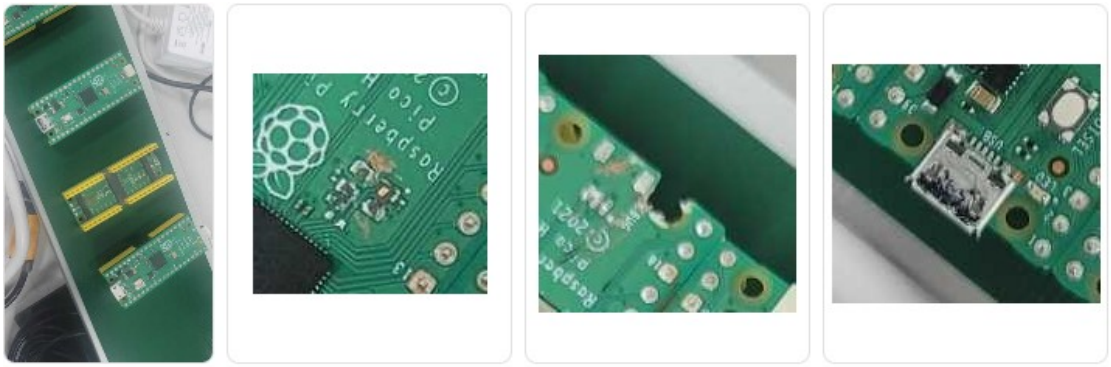
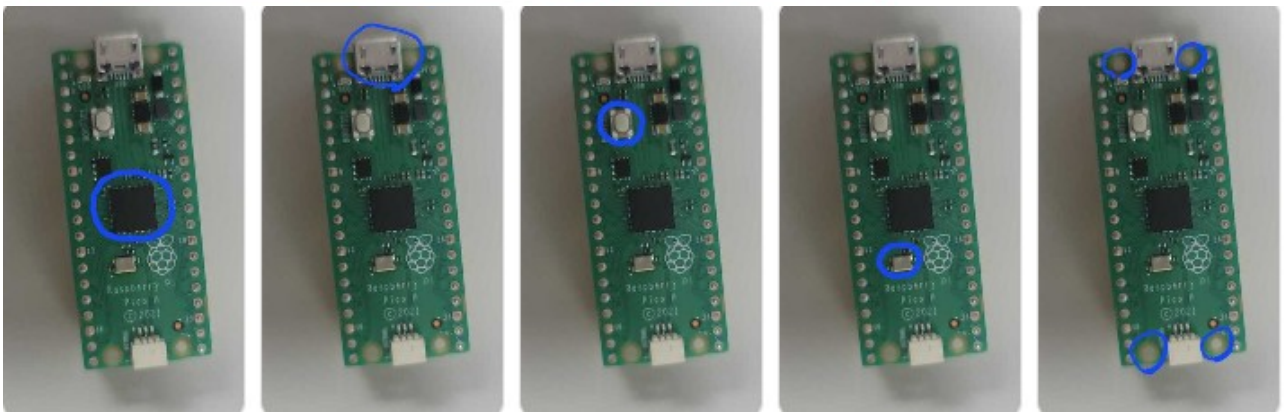
<컨베이어 벨트 기반 라즈베리 파이 pico 보드의 부품 불량 검사 시스템>

라즈베리 파이 재단(Raspberry Pi Foundation)**에서 개발한 초소형, 저전력 마이크로컨트롤러 보드

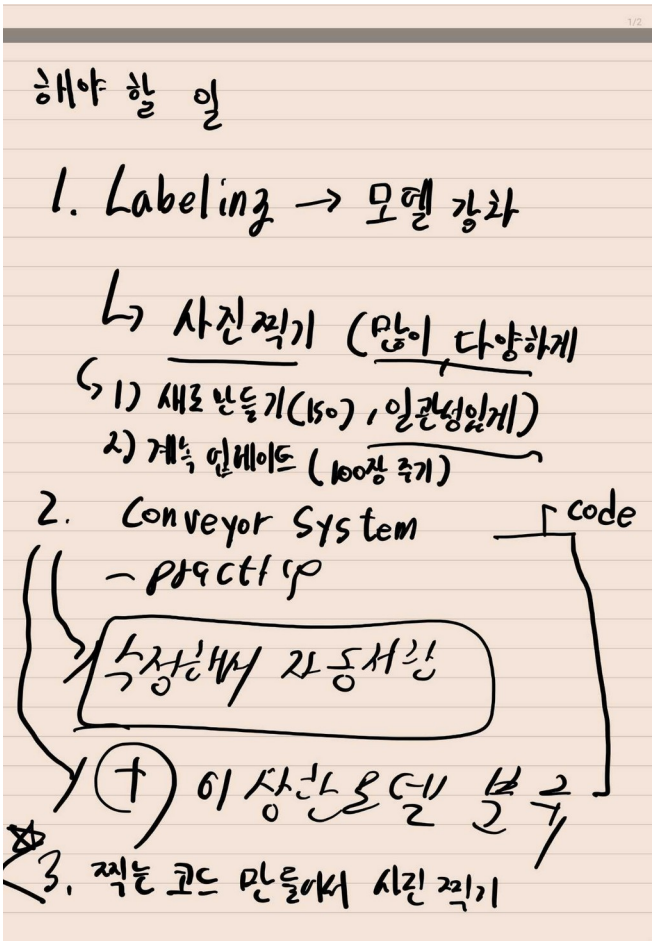
학습시키기 좋은 데이터의 조건:
다양하고, 일관성있고, 정확해야함

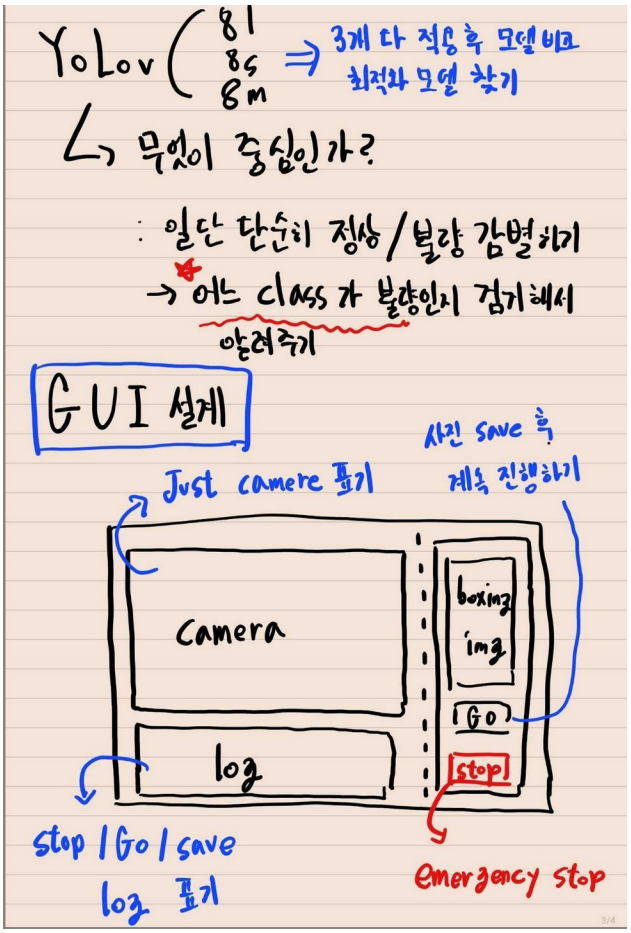






<프로젝트를 하기전에 한 생각들>





객체 향상 테스트

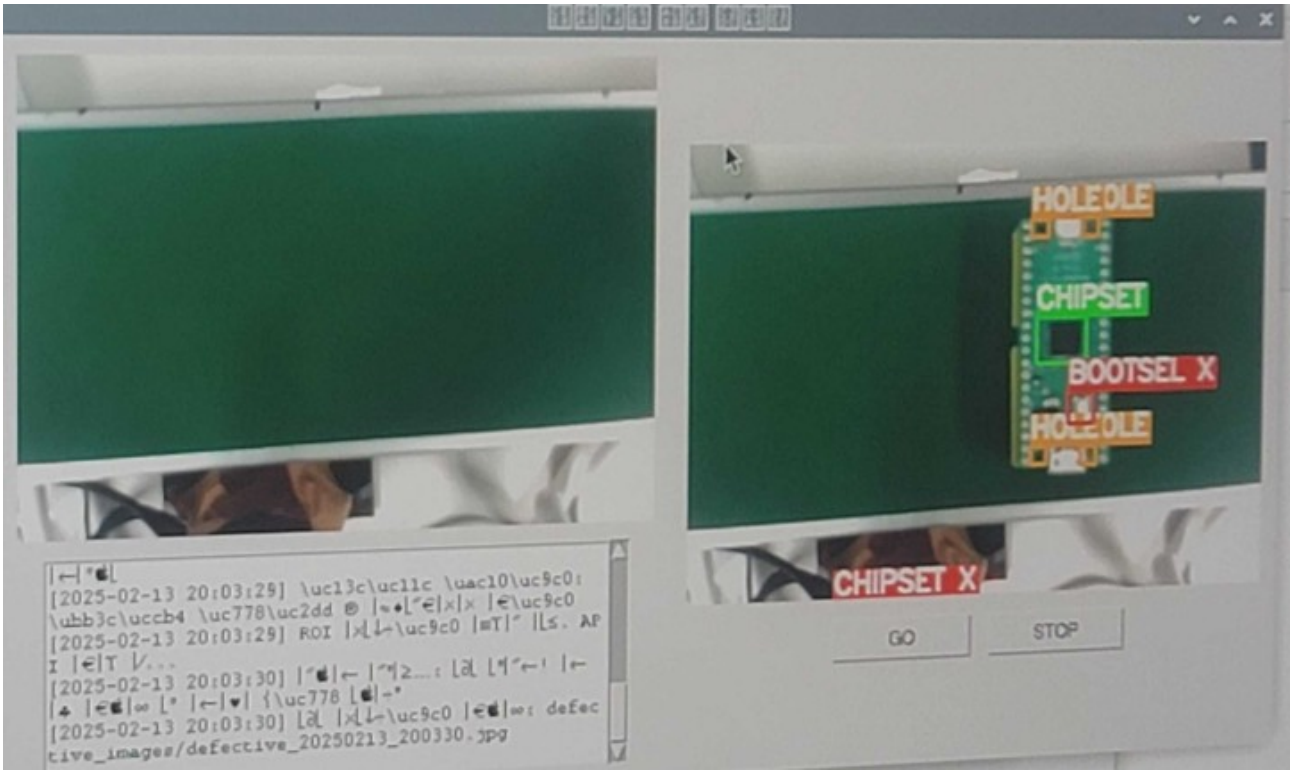
>> E-3-test RASPBERRY 진행 상황

빠른 테스트를 해보기 위해 100개의 라벨링 이미지로 진행(라벨링 대충휘리릭)
첫날 학습시킨 모델이 mAP값이 34.9% 였으나 새로 생성하여 라벨링 하고 학습시킨 결과 mAP 66.4%로 성능 향상(YOLOv6-N에서 YOLOv6-M으로 욜로모델 변경 이슈이지 않을까 생각됨)
카테고리에 불량/정품을 추가해 보았으나 차이가 없어 차후 프로젝트에서 삭제
불량은 라벨링 하지 않고 정식 규격만 라벨링 하였는데 불량이 정식규격이라며 오검출을 뱉어냄
논의 끝에 새로운 프로젝트 생성

>> E-3-RASPBERRY 프로젝트

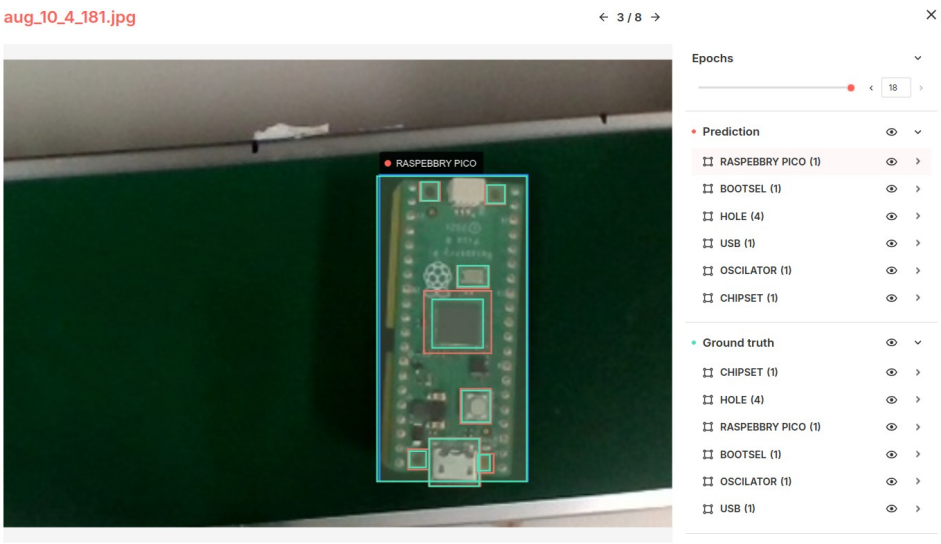
라벨링을 앞 테스트 보다 공들였는데 오토 라벨링의 인식이 좀 잘 안되고 있음
각 class별 불량 클래스를 추가로 넣어 정품과 불량을 따로 학습 시킨 결과 모델 학습 리포트상으로 오검출 감소
160개 정도의 이미지를 돌렸지만 mAP는 35.7로 낮음
- hole 클래스의 경우 이미지가 너무 다양하여 판별을 잘 못하고 있어 추가 학습 자료 필요할 것 같음(젤 많은 부분을 차지하고 있어서 정확도가 낮게 나오지 않았나 생각됨)

첫날 만든 라벨링을 가져다 쓰는것도 좋지 않을까 생각이 됨
(두개 합치는 방법?있으면 좋을것 같음)



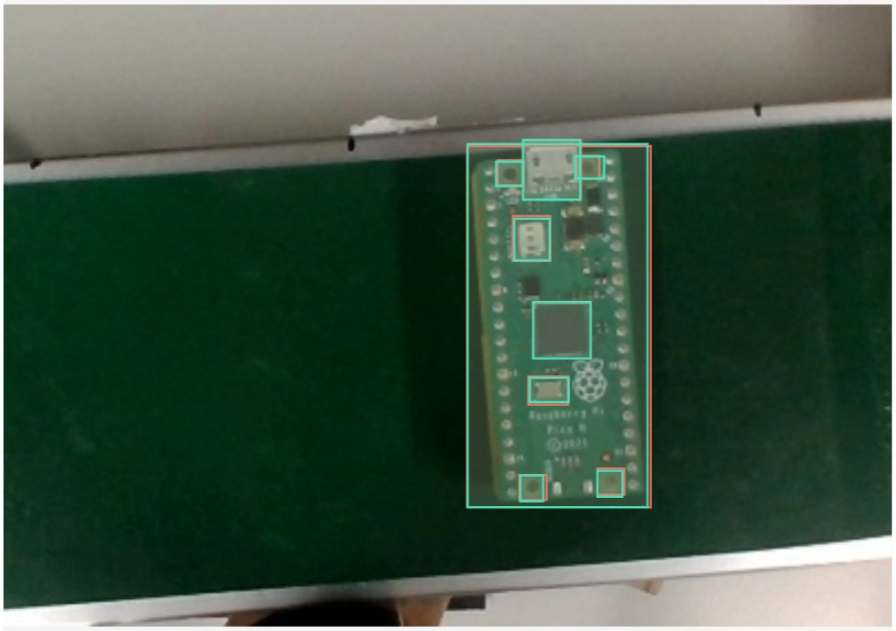
<실제로 컨베이어를 돌려본 결과>

mAP가 적게 나온이유
우리의 다른조와 다르게 class가 12개
다른조는 6개(우리는 2배나 많음)
불량에 대한 class를 ~x이렇게 지정하였는데 특정 class에 대해 표본이 압도적으로 부족하다
그리고 다른조와 mAP가 그렇게 높지 않아 과적합에서 이점이 있음
위의 사진을 보면 mAP의 수치와 별개로 인식을 잘함ㅋ
3번째 사진을 보면 CLASS6개로 돌렸고 라벨링도 기준없이 했는데 mAP도 높은데 인식을 못함ㅋㅋㅋ
4번째 사진은 개념설명ㅋ



aug_6_3_28.jpg

← 4 / 8 →



Epochs

18

Prediction

- USB (1)
- RASPEBBRY PICO X (1)
- BOOTSEL X (1)
- OSCILATOR (1)
- HOLE (4)
- CHIPSET (1)

Ground truth

- OSCILATOR (1)
- HOLE (4)
- BOOTSEL X (1)
- USB (1)
- CHIPSET (1)
- RASPEBBRY PICO X (1)

🔎 mAP와 Overfitting의 연관성

훈련 데이터와 검증 데이터에서의 mAP 변화를 보면 **과적합 여부를 판단**할 수 있어.

상태	훈련 mAP (Train mAP)	검증 mAP (Validation mAP)	설명
정상 학습 (Generalization)	✅ 높음	✅ 높음	모델이 잘 학습됨
과적합 (Overfitting)	✅ 매우 높음	❌ 낮음	훈련 데이터에는 잘 맞지만, 새로운 데이터에서는 성능 저하
과소적합 (Underfitting)	❌ 낮음	❌ 낮음	모델이 충분히 학습되지 않음

📌 과적합이 발생하면?

- **훈련 데이터의 mAP는 계속 증가하지만, 검증 데이터의 mAP가 감소**함
- 즉, **훈련 데이터에는 높은 정확도를 보이지만, 실제 테스트에서 성능이 낮아짐**

✅ mAP 그래프를 통한 과적합 감지

훈련 과정에서 **mAP 그래프**를 보면 과적합 여부를 판단할 수 있어.

1. **정상적인 학습**
   - 훈련 mAP와 검증 mAP가 **비슷한 경향으로 증가**
   - 모델이 일반화(generalization) 능력을 잘 가짐

YOLOv6-M-pico yolo_s test ›

Trained • 1 Endpoint

pico yolo_s test ›

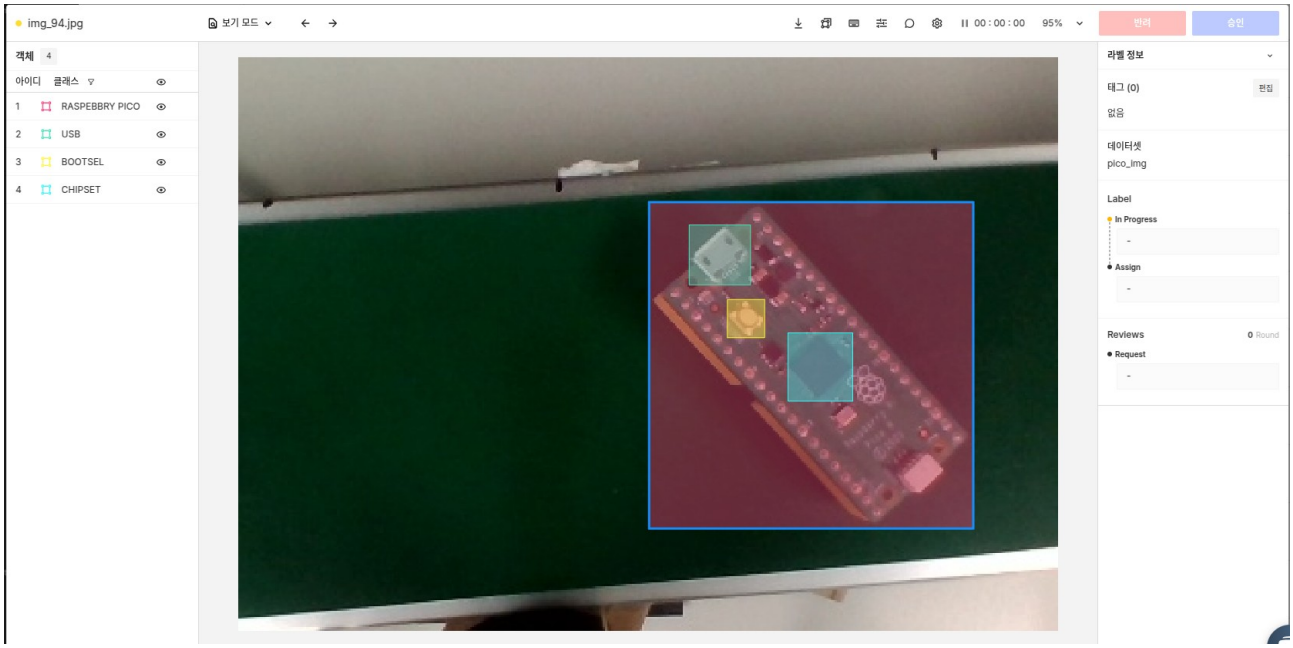
Object detection

YOLOv6-M ›

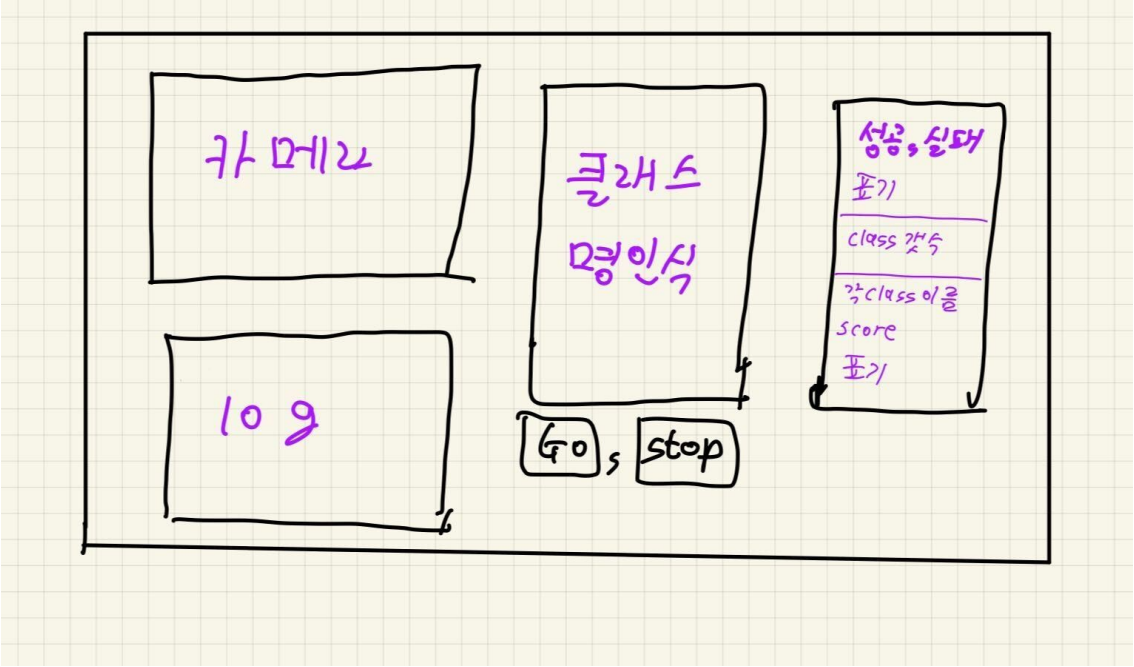
YOLOv6

mAP 66.4% ›

2/12/25 3:13 PM

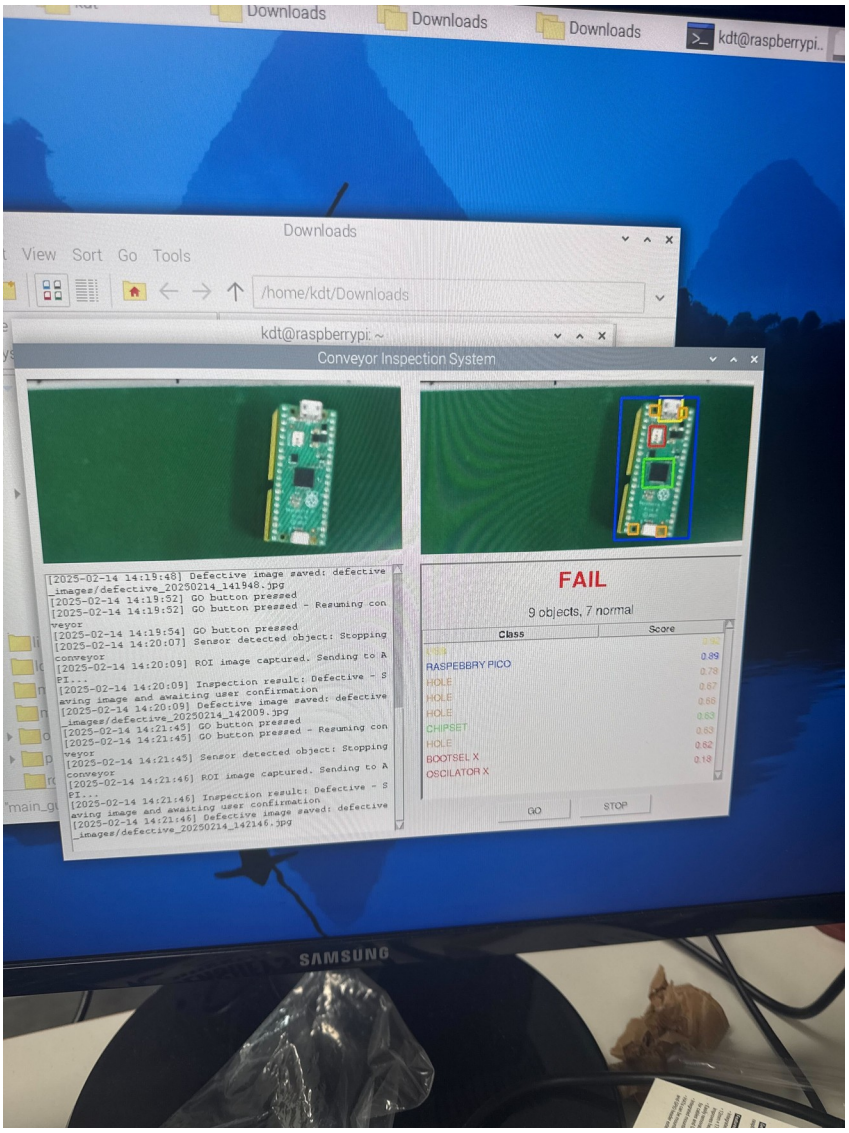


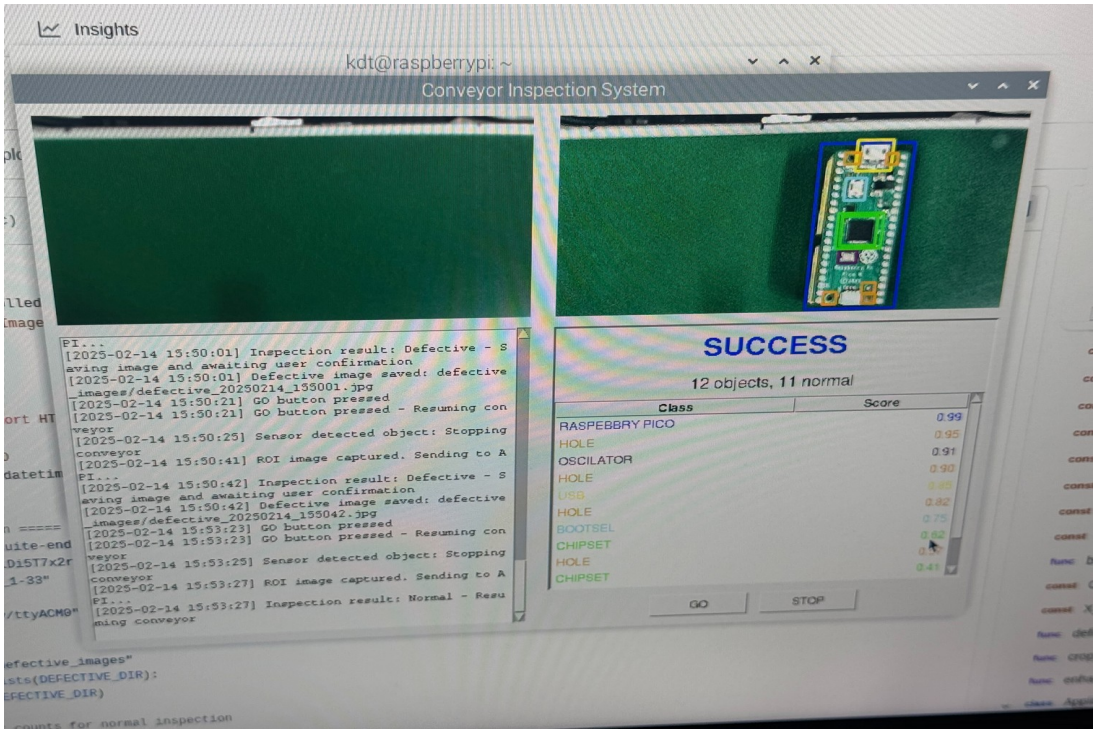
< 라벨링을 500장 해보고 100장을 선별해서 증강된 이미지를 오토라벨링 돌려보았을때의 결과>

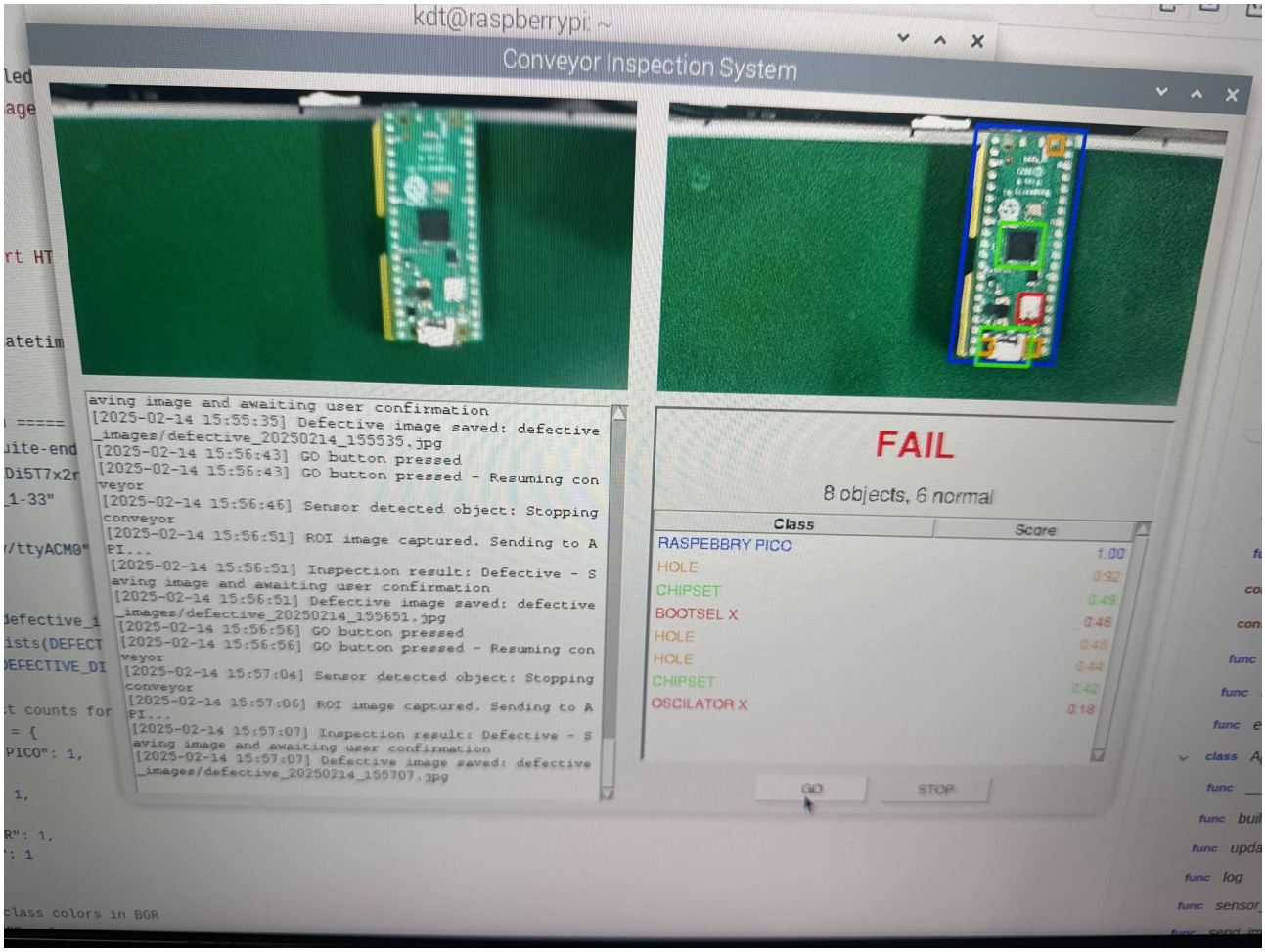


<최종 GUI 설계>

<아래부터 우리의 최종 라즈베리피코 검증타임>

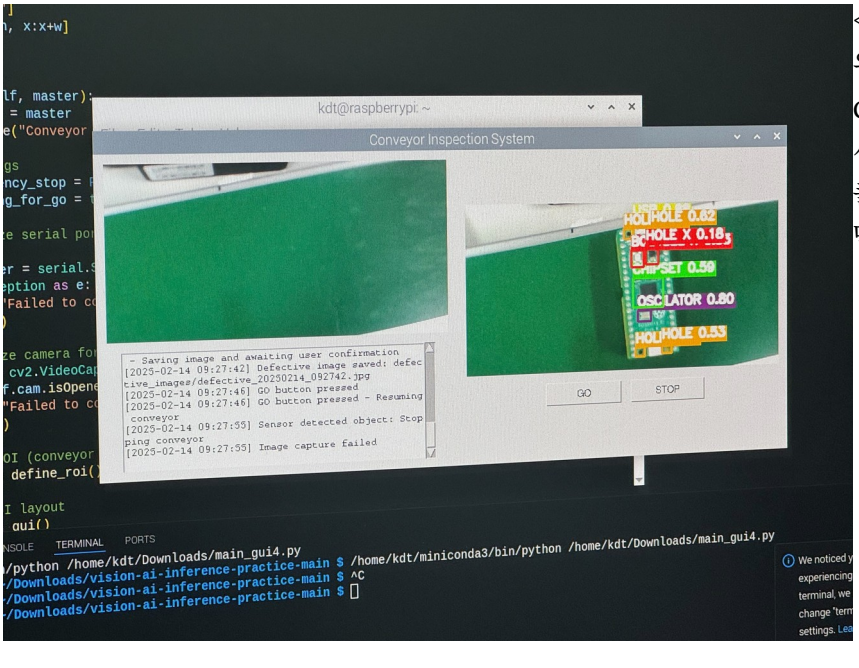






<시연영상> : https://www.youtube.com/shorts/RzP7vA0Al8I

27.py는 데이터 증강 파이썬 파일임
→ 데이터 증강으로는 거리는 불가. 환경 만들기가 어려움 그래서 데이터 증강으로는 크기,빛 이런 ,각도의 요소들만 사용
결국에 6주차 프로그램때는 거리가 중요(각도도 중요하지만 6주차에서는 데이터 증강이 크게 중요하지 않았음..)



<최종 시연 전에는>
왼쪽 사진과 같이 중구난방으로 class들이 보여서 최종에는 시연사진에서 볼수 있듯이 클래스를 따로 보여주는 gui를 만들었었음

코드 간단 분석

1. 프로젝트 개요

프로젝트 명:

컨베이어 벨트 기반 라즈베리 파이 pico 보드의 부품 불량 검사 시스템

주요 목적:

- 컨베이어 벨트 위에서 부품을 실시간으로 검사하여 정상품과 불량품을 판별하는 시스템 구축
- 딥러닝 기반 객체 검출 API (YOLO V6-M by SUPERB AI)를 활용하여 부품의 상태를 분석
- 결과에 따라 컨베이어 벨트를 제어하고, 불량품 이미지 저장 및 사용자 확인 기능을 제공

---

2. 25.py – JSON 불러오기 및 API 통신

주요 역할:

- **이미지 파일 읽기:** 지정된 이미지 파일을 바이너리 형태로 읽어들임
- **API 요청:** 읽어들인 이미지 데이터를 딥러닝 모델 API로 전송하여 객체 탐지 결과를 받아옴
- **JSON 파싱 및 출력:** API로부터 받은 JSON 형식의 응답을 파싱하여 결과를 출력

주요 코드 흐름:

1. **파일 열기 및 데이터 읽기**
    - 이미지 파일 경로 지정 후, 바이너리 모드로 열어 데이터를 읽음
2. **API 요청 보내기**
    - `requests.post`를 사용하여 API 엔드포인트에 이미지 데이터를 전송
    - HTTP 기본 인증을 통해 액세스 키와 사용자명을 함께 전송
3. **응답 처리**
    - 응답을 JSON 형식으로 파싱하여 출력 (예: 불량 여부 및 객체 정보)

요약: 25.py는 통신 과정을 통해 API로부터 부품 검출 결과(JSON)를 받아오는 간단한 테스트 스크립트입니다.
25

---

3. 26.py – JSON 기반 이미지에 Bounding Box 그리기

주요 역할:

- **API 요청 및 JSON 결과 받기:** 25.py와 유사하게 이미지 파일을 API로 보내고 객체 탐지 결과를 JSON으로 받아옴
- **이미지 불러오기:** OpenCV를 사용하여 원본 이미지를 읽어들임
- **객체 시각화:**
    - API 응답에 포함된 각 객체의 좌표(box), 클래스, 신뢰도(score)를 활용
    - 클래스별로 미리 정의된 색상을 사용해 이미지에 사각형(bounding box)과 텍스트(객체명 및 신뢰도)를 표시

주요 코드 흐름:

1. **API 요청 및 결과 파싱**
    - 이미지 파일을 읽고 API로 전송하여 JSON 응답을 받아옴
2. **이미지 불러오기 & 색상 매핑**
    - 클래스별 색상 정보를 이용하여 시각적 구분 제공
3. **반복문을 통해 객체별 박스 그리기 및 텍스트 출력**
    - 각 객체의 좌표 정보를 이용해 OpenCV의 `rectangle` 함수로 사각형 표시
    - 객체명을 포함한 텍스트를 이미지에 삽입

요약: 26.py는 25.py의 API 결과를 바탕으로 이미지에 객체 경계 상자와 라벨을 표시해 시각화하는 역할을 수행합니다.
26

---

4. main_gui11.py – 통합 GUI 및 컨베이어 벨트 제어 시스템

주요 역할:

- **실시간 영상 처리:**
    - 카메라에서 실시간으로 영상을 받아와 관심 영역(ROI)을 추출
    - ROI에 대해 전처리(예: CLAHE, 감마 보정, 샤프닝) 적용
- **API 연동 및 결과 평가:**
    - 전처리된 ROI 이미지를 API로 전송하여 객체 탐지 결과를 받아옴
    - 받은 JSON 결과를 바탕으로 bounding box를 그리고, 검출 객체의 수 및 신뢰도를 평가하여 정상/불량 여부 판별
- **GUI 구성:**
    - Tkinter를 이용한 사용자 인터페이스 구성 (실시간 영상, 결과 이미지, 로그, 상태 표시, 테이블 등)
    - GO와 STOP 버튼을 통한 컨베이어 벨트 제어 및 사용자 입력 처리
- **하드웨어 제어:**
    - 시리얼 포트를 통해 센서 신호 수신 및 컨베이어 벨트의 정지/재개 제어

- 불량품으로 판별되면 해당 이미지를 저장하고, 사용자의 확인(Go 버튼 입력)을 대기

주요 코드 흐름 및 기능 설명:

1. **초기 설정 및 GUI 구축 (build_gui 함수):**
   - 왼쪽 패널: 실시간 카메라 영상 및 로그 창
   - 오른쪽 패널: API 결과를 시각화한 이미지와 상세 정보(상태, 객체 개수, 검출 결과 테이블)
2. **ROI 및 이미지 전처리:**
   - `define_roi`와 `crop_to_roi` 함수로 관심 영역 설정
   - `enhance_image` 함수를 통해 영상의 대비, 밝기, 선명도 개선
3. **API 연동:**
   - `send_image_to_api` 메소드로 전처리된 이미지를 API에 전송하고 JSON 결과 획득
4. **결과 시각화 및 평가:**
   - `draw_boxes` 함수로 bounding box를 그리고, `update_detailed_info`로 GUI 테이블 갱신
   - `evaluate_result` 함수에서 검출된 객체들의 수와 클래스 정보를 비교하여 정상 여부 판정
5. **컨베이어 벨트 제어 및 사용자 상호작용:**
   - `sensor_loop`에서 시리얼 포트로 센서 신호를 읽고, 정지 후 이미지 캡처
   - 불량품 판별 시 이미지를 저장하고, 사용자 입력(Go 버튼) 대기 후 재개

요약: main_gui11.py는 26.py의 객체 검출 및 시각화 기능을 포함하여, 컨베이어 벨트 하드웨어 제어와 사용자 인터페이스(GUI)를 통합한 전체 검사 시스템의 핵심 애플리케이션입니다.
main_gui11

---

5. 결론

- **25.py:** API와의 통신을 통해 이미지 데이터를 보내고, JSON 형식의 객체 탐지 결과를 받아오는 역할을 합니다.
- **26.py:** 25.py의 결과를 활용하여 이미지에 bounding box와 라벨을 그려 객체 검출 결과를 시각화합니다.
- **main_gui11.py:** 실시간 영상 처리, API 연동, 결과 평가, GUI 구성, 그리고 컨베이어 벨트 제어 기능을 통합하여 최종 검사 시스템을 구현합니다.

이 세 스크립트를 조합함으로써, 컨베이어 벨트 위에서 실시간 부품 불량 검사가 가능하며, 불량 판정 시 자동으로 이미지 저장 및 사용자 확인 대기 등 다양한 기능을 수행할 수 있습니다.