Object detection 실습 예제

박석희 교수님

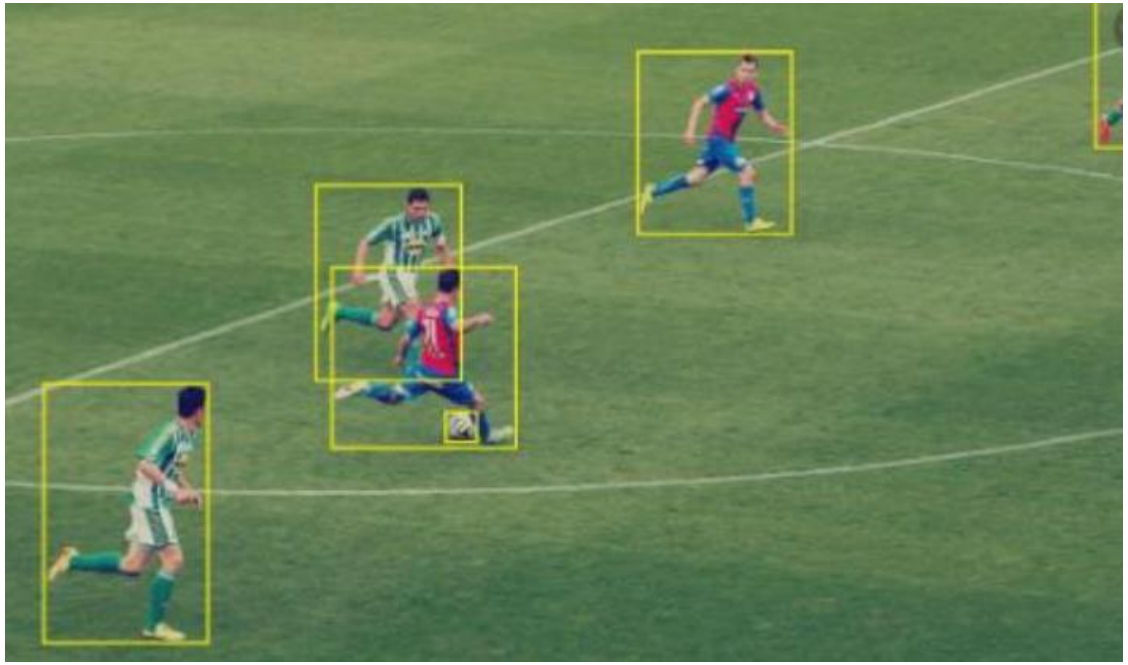
실습조교 강정훈
kjh95k@naver.com

2025.06.20.

Object detection – YOLOv5

Object detection 모델은 지도학습으로 **라벨링된 데이터를 모델에 학습시키는 과정이 필요**함

객체를 박스를 이용해 표시하고, 그 객체의 종류를 설정하는 과정을 거쳐야 함.



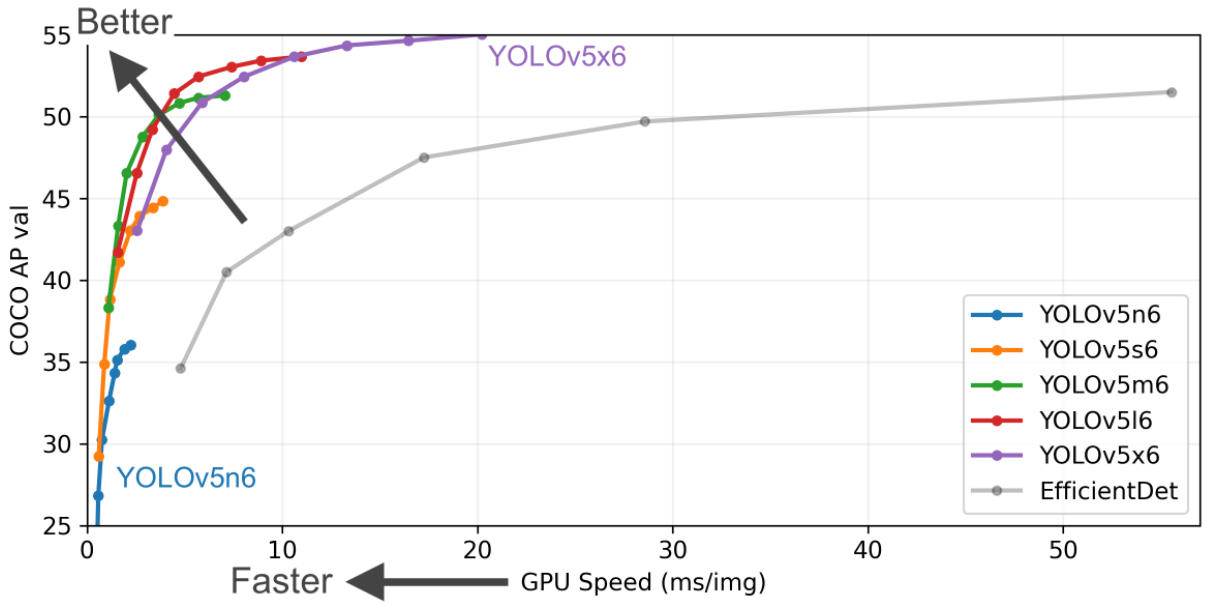
https://becominghuman.ai/what-is-the-difference-between-data-annotation-and-labeling-in-ai-ml-bc0cabe0b9d6

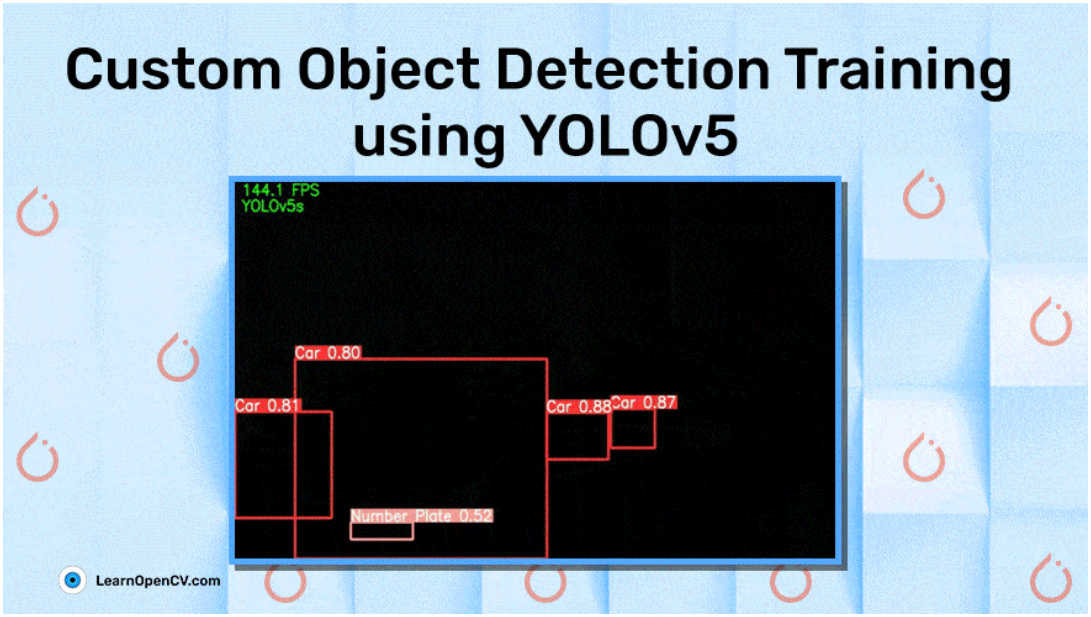
https://eehoeskrap.tistory.com/331

Object detection – YOLOv5

Object detection 모델 중 YOLOv5를 이용하여 라벨링 및 이미지 내의 객체 탐지 수행

YOLOv5에는 모델 성능에 따라 n,s,m,l,x가 있고, 그 중 **YOLOv5s를 이용하여 실습**



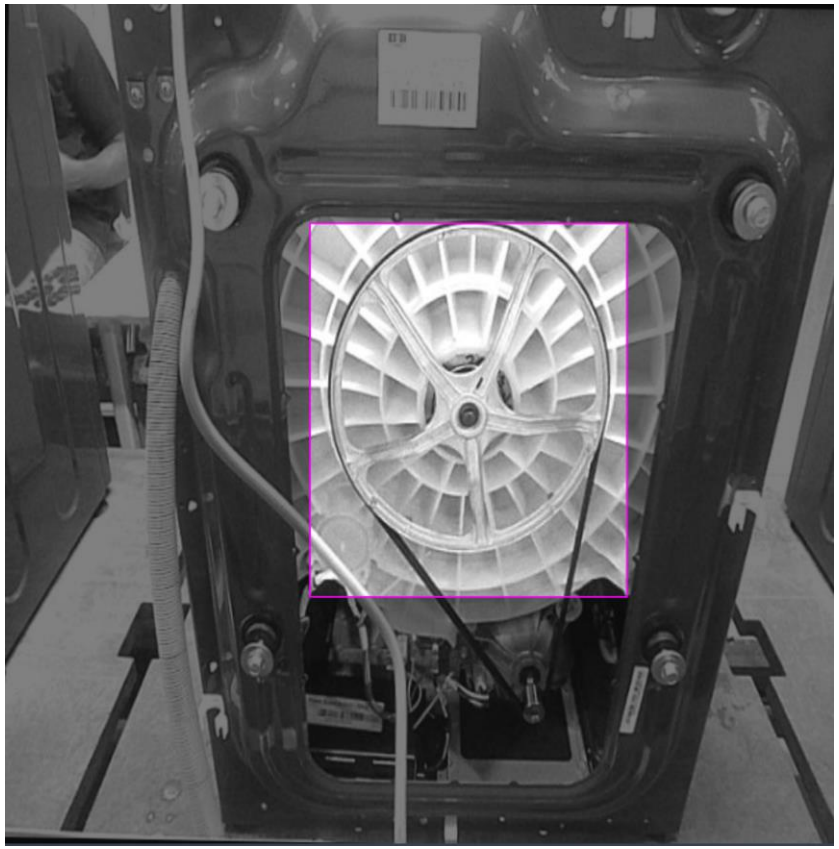


https://becominghuman.ai/what-is-the-difference-between-data-annotation-and-labeling-in-ai-ml-bc0cabe0b9d6

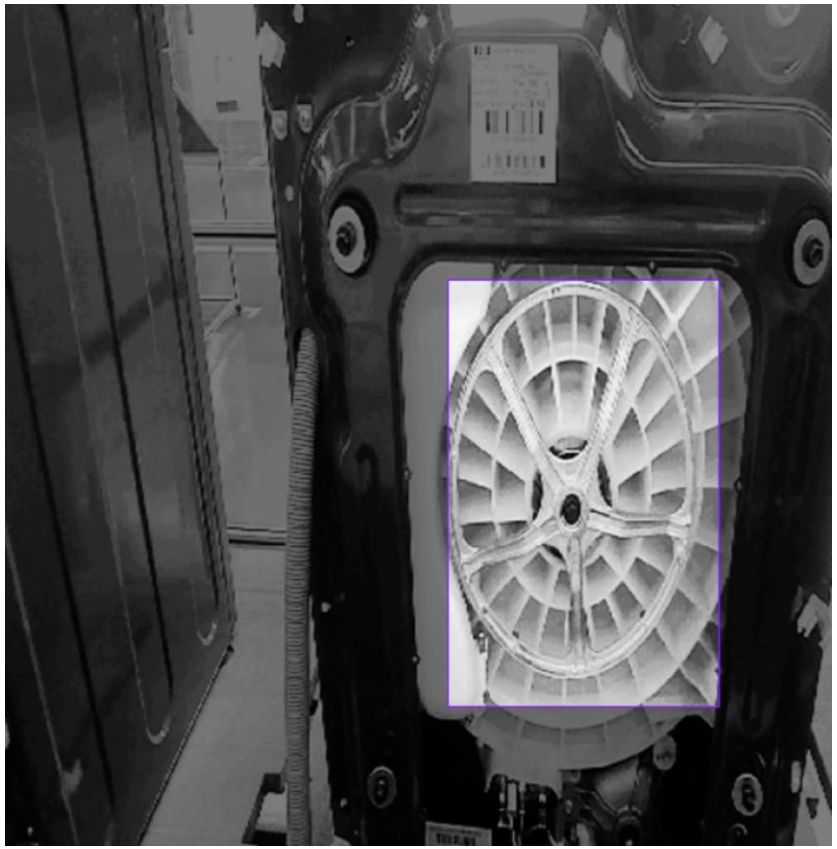
https://eehoeskrap.tistory.com/331

예제1 데이터

❖ **세탁기 pully 체결 유무 detection 예제**



체결 됨



체결 안됨

- 데이터는 세탁기 후면 사진으로, pully 체결 유무에 대해 라벨링한 데이터가 포함됨

예제1 데이터

❖ 세탁기 pully 체결 유무 detection 예제

로컬에서 압축 해제

YOLO실습파일_1.zip

↓

- test
- train
- valid

라벨링 작업

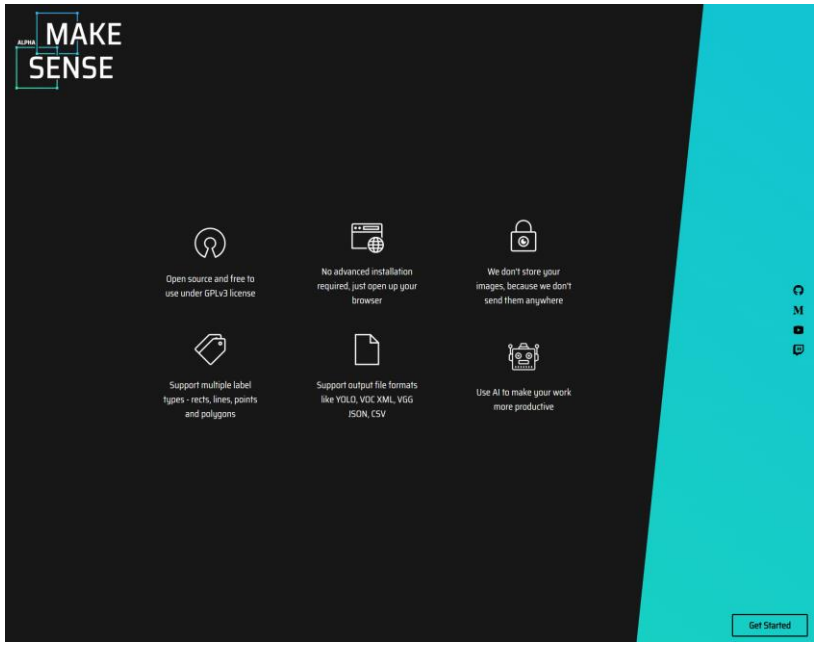
진행 순서

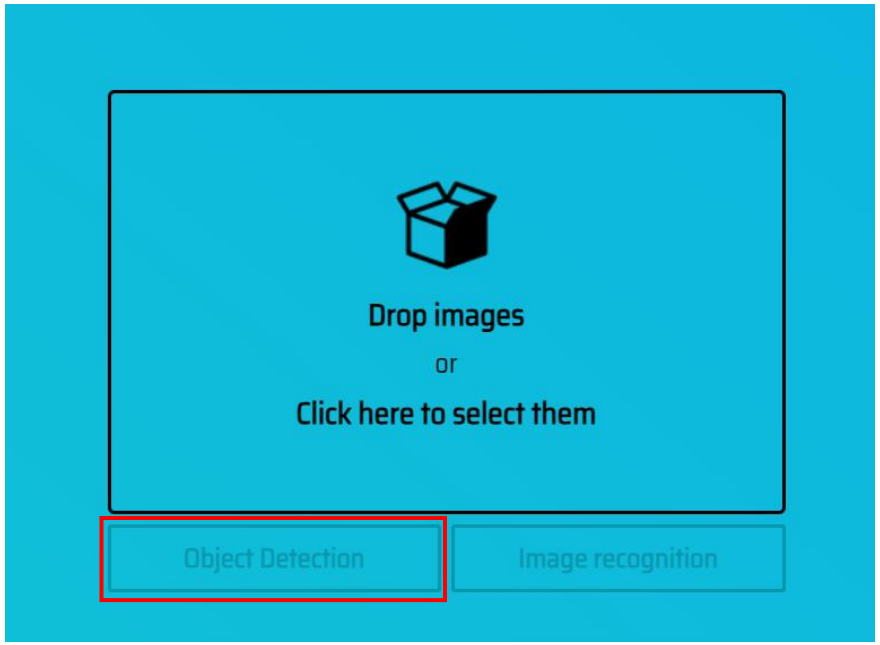
1. 컴퓨터 로컬에 다운로드 후 압축 해제
2. Labeling 작업 후 다시 압축하여 드라이브에 업로드

라벨링 작업 – 예제 1

❖ **Makesense 사이트를 이용하여 test에 사용할 데이터 셋 생성**
→ 대표적인 라벨링 툴인 labelImg를 이용하여 라벨링

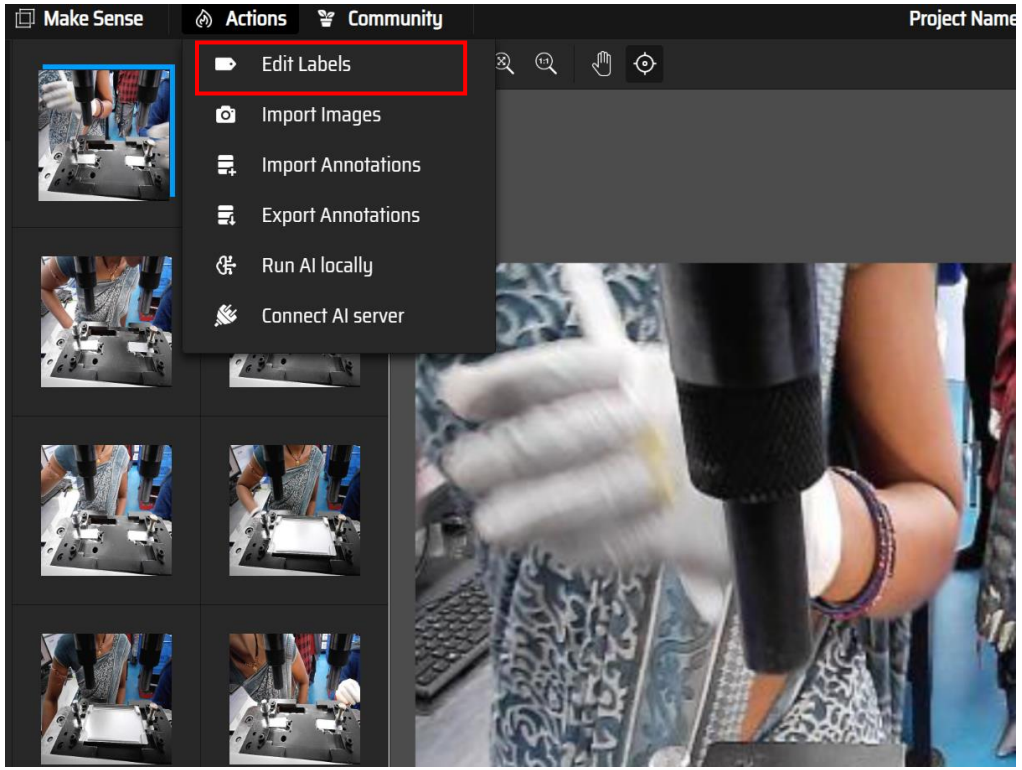
https://www.makesense.ai

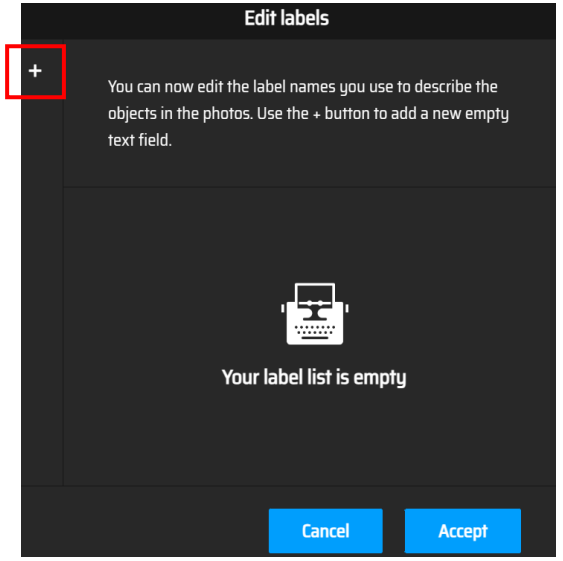


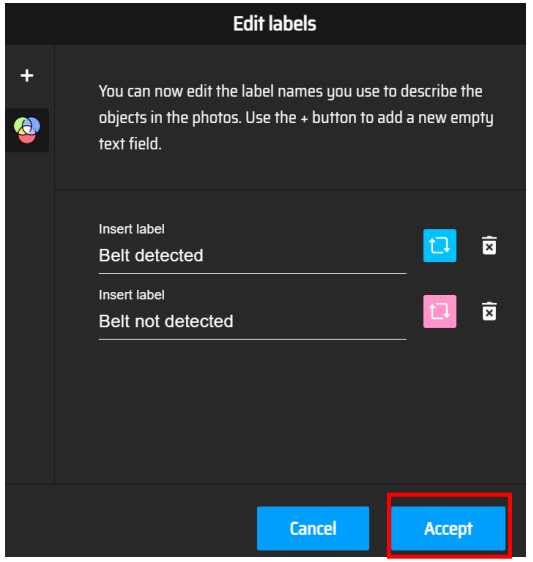


1. 사이트에 접속 및 get started 클릭
2. 라벨링할 이미지를 첨부하거나 드래그 앤 드롭 (test 데이터 투입)
3. 활성화된 Object detection 버튼 클릭

라벨링 작업 – 예제 1

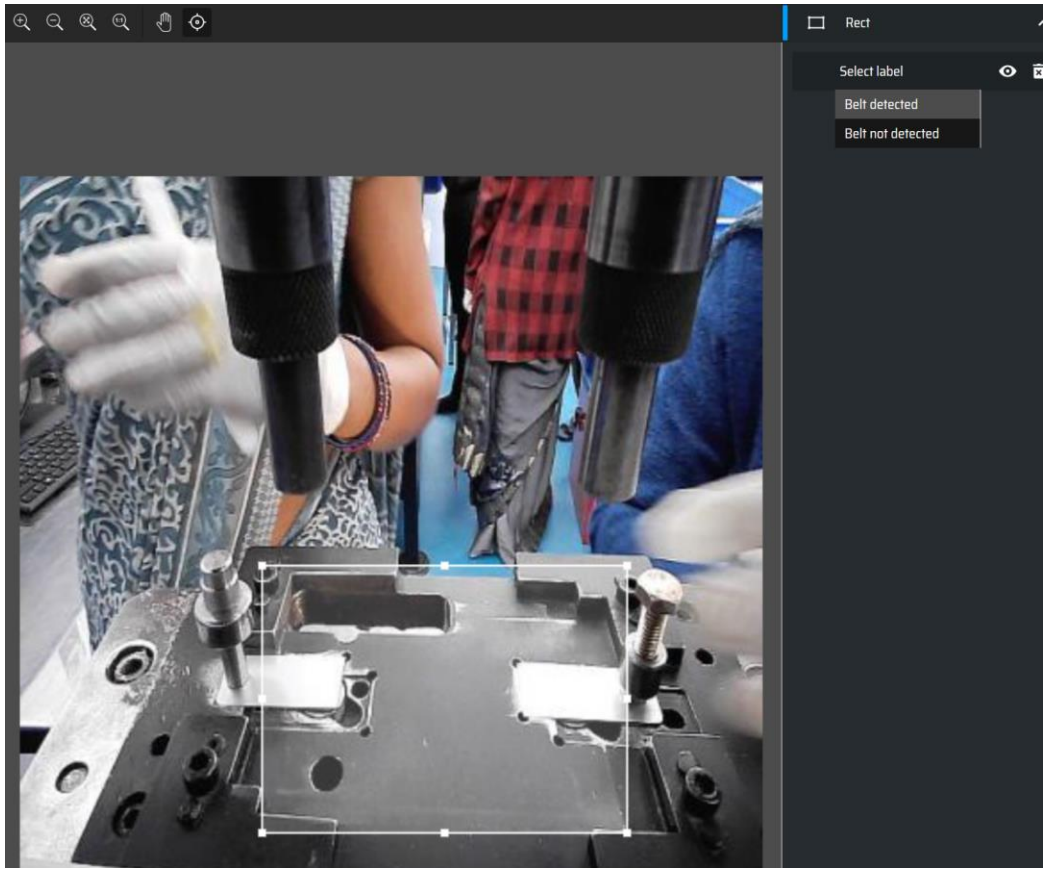


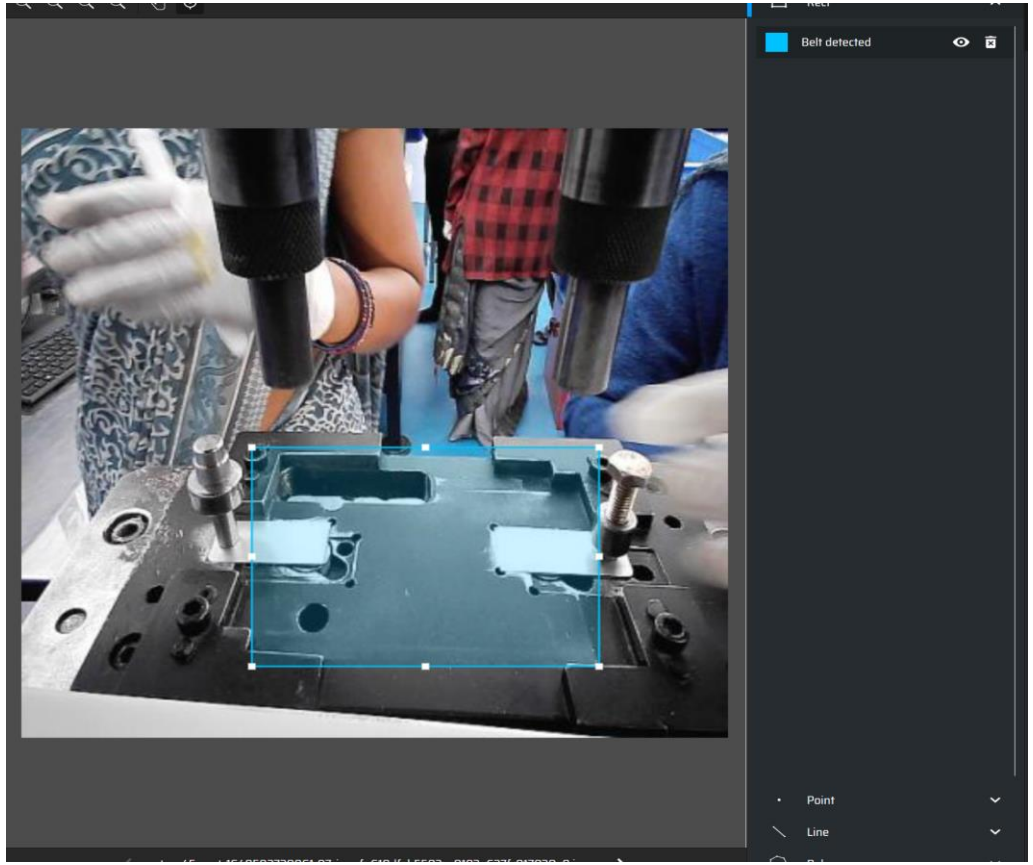




- Start project 클릭 -> 좌상단의 Actions – Edit Labels 클릭
- 좌상단의 + 버튼을 눌러 label 추가
- 원하는 라벨명을 입력한 뒤 Accept (Belt detected, Belt not detected 사용)

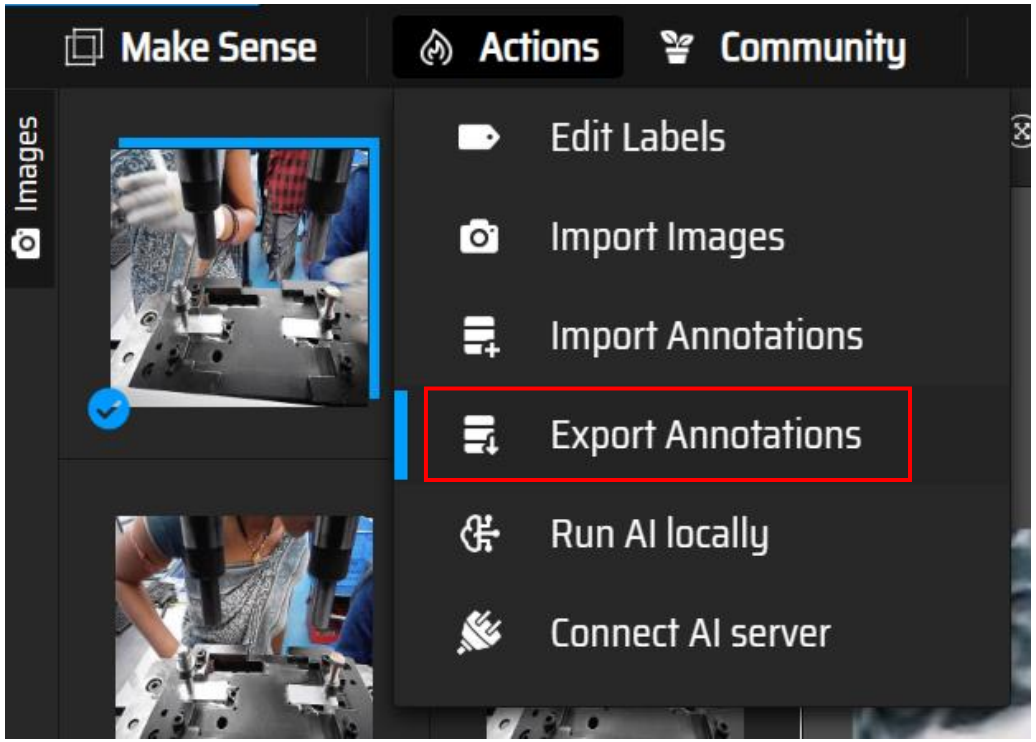
라벨링 작업 – 예제 1

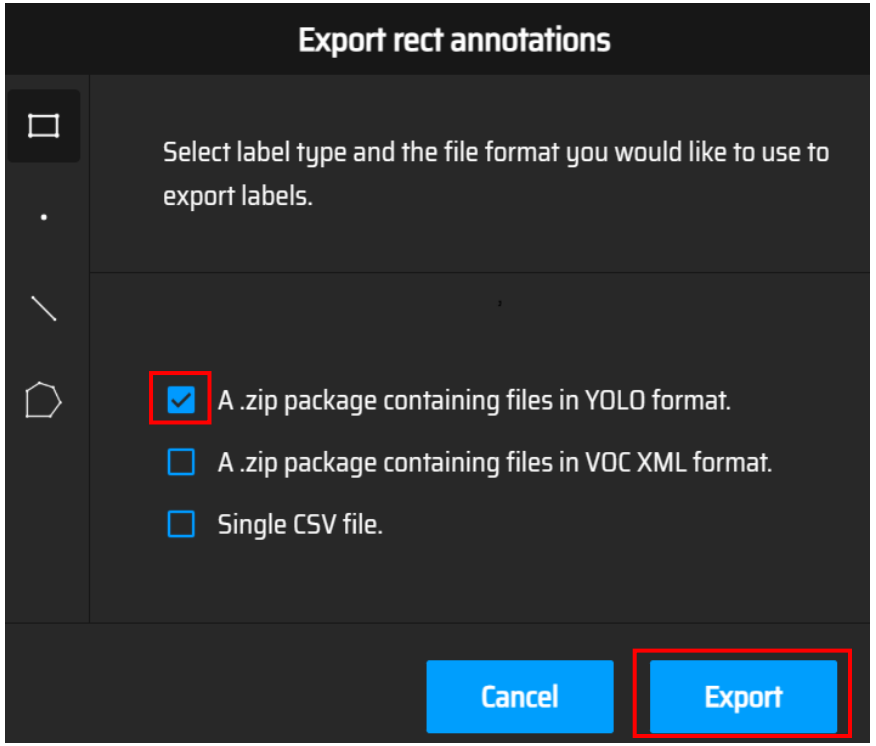




- 사진에서 원하는 위치에 드래그한 뒤 우측의 label을 선택
- 라벨을 선택하면 색깔이 입혀진 박스 형태로 변하고 라벨이 지정됨

라벨링 작업 – 예제 1





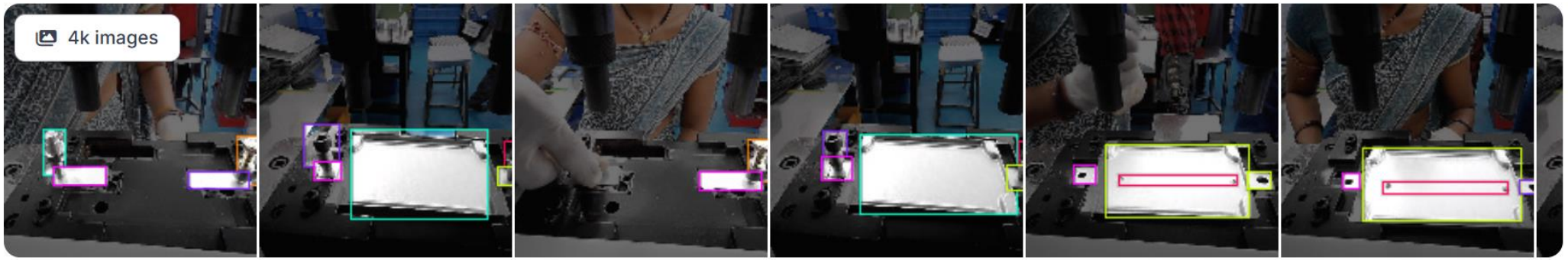
- 라벨링을 모두 완료한 뒤 Actions - Export Annotations을 클릭
- YOLO format으로 file을 export하면 label 데이터를 추출 가능

라벨링 작업 – 예제 1

사진과 라벨의 파일 이름이 일치해야 인식됨
(기본적으로 동일하게 추출됨)

사진은 test 폴더 내에 **images** 폴더와 **labels** 폴더에 각각 사진 파일과 labeling 파일 (txt)을 저장
다시 압축 후 드라이브에 파일 저장

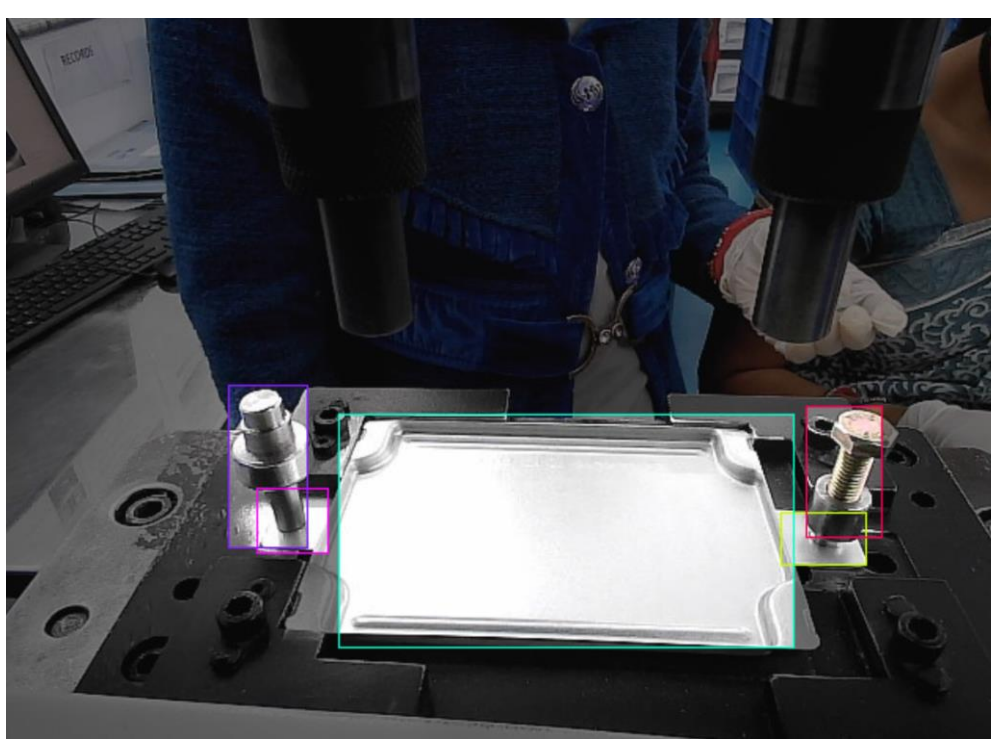
예제 2 데이터

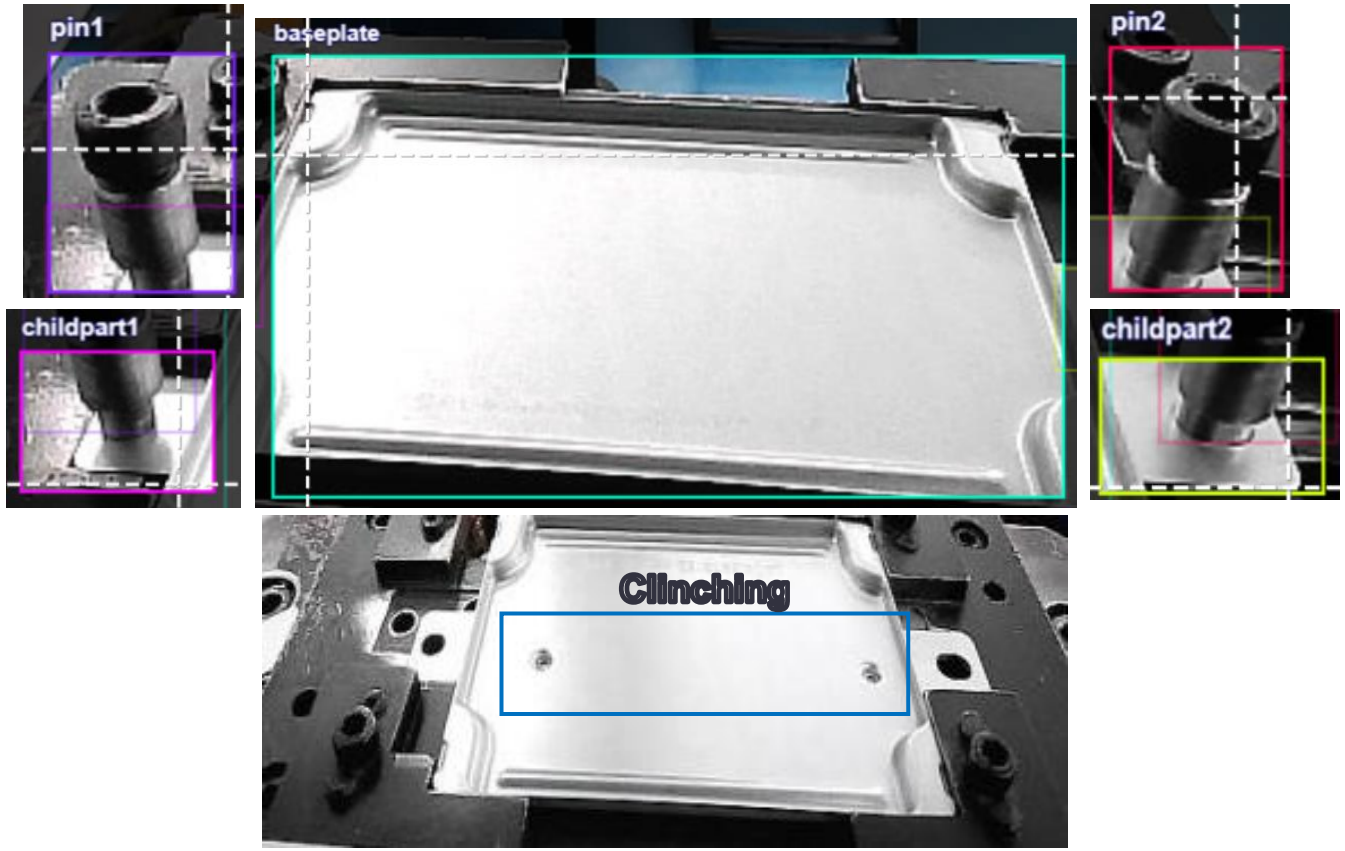


한 이미지 내에 여러 객체를 포함한 경우에 대해 라벨링 및 평가
예제 1과 동일하게 makesense 사이트에서 작업 진행

https://universe.roboflow.com/im-settwo/topcam12

라벨링 작업 – 예제 2





Labeling 종류는 6가지로 baseplate, childpart1, childpart2, clinching, pin1, pin2 6가지에 대해 라벨링 수행

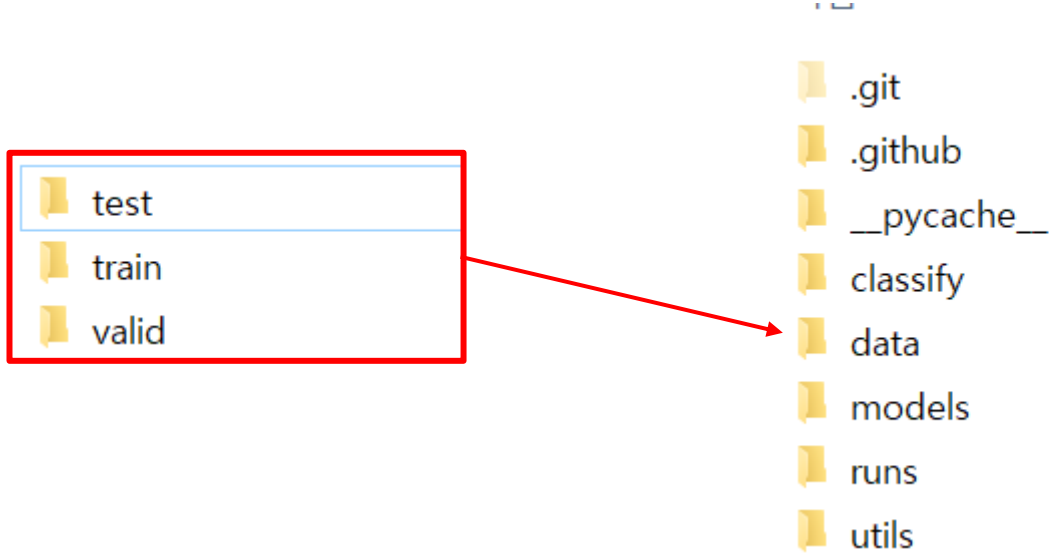
라벨링 명 순서대로 작성하여 사용할 것

라벨링 작업 – 예제 2

예제 1 -> test 데이터에 대해 라벨링 해보기

예제 2 -> train 데이터를 일부 라벨링 해보고 test 데이터에 대해 정확도 확인하기

구글 드라이브에 업로드



yolov5/data 폴더로 train, valid, test 폴더 옮기고 .zip 한 뒤 구글 드라이브에 Mydrive 폴더에

dataset 폴더를 만들어 폴더 안에 .zip 파일을 업로드

객체 탐지 수행 – 예제 1

```
[1]  # 구글 드라이브 마운트
     from google.colab import drive
     import os

     drive.mount('/content/drive/')
```

```
Mounted at /content/drive/
```

해당 .zip 파일명 넣기

```
[2]  %cd /content/drive/MyDrive/dataset

     !unzip -qq "/content/drive/MyDrive/dataset/''.zip"
```

구글 드라이브에 올린 파일을 사용하기 위해 마운트

Dataset 폴더에 넣은 zip 파일을 해제

객체 탐지 수행 – 예제 1

```
# YOLOv5 모델 가져오기
!git clone https://github.com/ultralytics/yolov5
```

```
Cloning into 'yolov5'...
remote: Enumerating objects: 17493, done.
remote: Counting objects: 100% (3/3), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 17493 (delta 1), reused 0 (delta 0), pack-reused 17490 (from 3)
Receiving objects: 100% (17493/17493), 16.56 MiB | 18.39 MiB/s, done.
Resolving deltas: 100% (11996/11996), done.
```

```
[8]  %cd yolov5
     %pip install -qr requirements.txt # install dependencies
     %pip install -q roboflow
     %pip install opencv-python matplotlib pyyaml torch

     import torch
     import yaml
     from IPython.display import Image, clear_output  # to display images
     import cv2
     import matplotlib.pyplot as plt

     print(f"Setup complete. Using torch {torch.__version__} ({torch.cuda.get_device_properties(0).name if torch.cuda.is_available() else 'CPU'}
```

학습을 위한 필수 요소들 설치

객체 탐지 수행 – 예제 1

Yaml 파일 – 학습 및 평가에 사용할 이미지 데이터의 주소를 포함

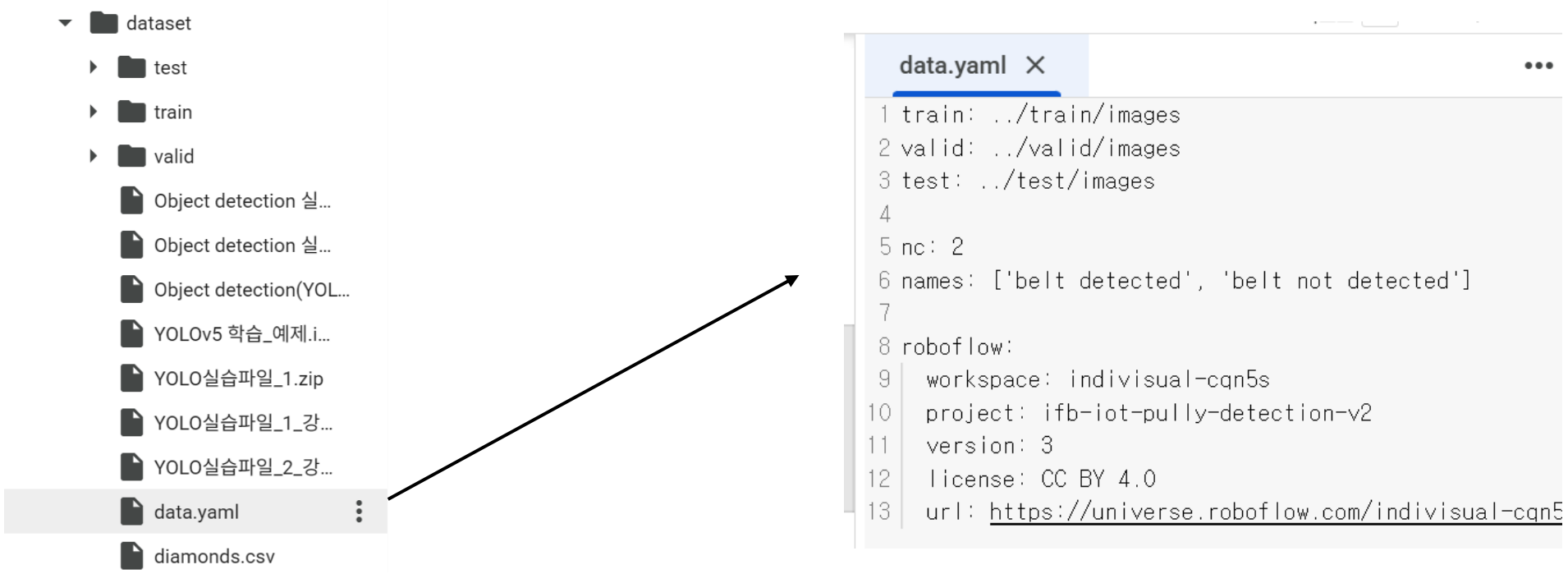
```
[10]  # 데이터셋 yaml 파일 확인
      with open(data_yaml) as f:
          film = yaml.load(f, Loader=yaml.FullLoader)
          display(film)

      # yaml 파일에 저장된 label 이름과 train, valid 파일 경로를 출력
```

```
{'train': '../train/images',
 'valid': '../valid/images',
 'test': '../test/images',
 'nc': 2,
 'names': ['belt detected', 'belt not detected'],
```

Yaml 파일에 포함된 데이터셋 경로와 class 수, class명을 현재 경로에 맞게 지정 해야함

객체 탐지 수행 – 예제 1

Yaml 파일 – 학습 및 평가에 사용할 이미지 데이터의 주소를 포함



해당 데이터의 train, val, test의 위치를 설정해주어야 학습 및 detection이 가능

Yaml 파일에서 labeling 이름과 labeling 목록 수를 설정

초기 설정

학습 파라미터 확인 및 조절

```python
def parse_opt(known=False):
    parser = argparse.ArgumentParser()
    parser.add_argument('--weights', type=str, default=ROOT / 'yolov5s.pt', help='initial weights path')
    parser.add_argument('--cfg', type=str, default='', help='model.yaml path')
    parser.add_argument('--data', type=str, default=ROOT / 'data/coco128.yaml', help='dataset.yaml path')
    parser.add_argument('--hyp', type=str, default=ROOT / 'data/hyps/hyp.scratch-low.yaml', help='hyperparameters path')
    parser.add_argument('--epochs', type=int, default=300, help='total training epochs')
    parser.add_argument('--batch-size', type=int, default=16, help='total batch size for all GPUs, -1 for autobatch')
    parser.add_argument('--imgsz', '--img', '--img-size', type=int, default=640, help='train, val image size (pixels)')
    parser.add_argument('--rect', action='store_true', help='rectangular training')
    parser.add_argument('--resume', nargs='?', const=True, default=False, help='resume most recent training')
    parser.add_argument('--nosave', action='store_true', help='only save final checkpoint')
    parser.add_argument('--noval', action='store_true', help='only validate final epoch')
    parser.add_argument('--noautoanchor', action='store_true', help='disable AutoAnchor')
    parser.add_argument('--noplots', action='store_true', help='save no plot files')
    parser.add_argument('--evolve', type=int, nargs='?', const=300, help='evolve hyperparameters for x generations')
    parser.add_argument('--bucket', type=str, default='', help='gsutil bucket')
    parser.add_argument('--cache', type=str, nargs='?', const='ram', help='--cache images in "ram" (default) or "disk"')
    parser.add_argument('--image-weights', action='store_true', help='use weighted image selection for training')
    parser.add_argument('--device', default='', help='cuda device, i.e. 0 or 0,1,2,3 or cpu')
    parser.add_argument('--multi-scale', action='store_true', help='vary img-size +/- 50%%')
    parser.add_argument('--single-cls', action='store_true', help='train multi-class data as single-class')
    parser.add_argument('--optimizer', type=str, choices=['SGD', 'Adam', 'AdamW'], default='SGD', help='optimizer')
    parser.add_argument('--sync-bn', action='store_true', help='use SyncBatchNorm, only available in DDP mode')
    parser.add_argument('--workers', type=int, default=8, help='max dataloader workers (per RANK in DDP mode)')
    parser.add_argument('--project', default=ROOT / 'runs/train', help='save to project/name')
    parser.add_argument('--name', default='exp', help='save to project/name')
    parser.add_argument('--exist-ok', action='store_true', help='existing project/name ok, do not increment')
    parser.add_argument('--quad', action='store_true', help='quad dataloader')
    parser.add_argument('--cos-lr', action='store_true', help='cosine LR scheduler')
    parser.add_argument('--label-smoothing', type=float, default=0.0, help='Label smoothing epsilon')
    parser.add_argument('--patience', type=int, default=100, help='EarlyStopping patience (epochs without improvement)')
    parser.add_argument('--freeze', nargs='+', type=int, default=[0], help='Freeze layers: backbone=10, first3=0 1 2')
    parser.add_argument('--save-period', type=int, default=-1, help='Save checkpoint every x epochs (disabled if < 1)')
    parser.add_argument('--seed', type=int, default=0, help='Global training seed')
    parser.add_argument('--local_rank', type=int, default=-1, help='Automatic DDP Multi-GPU argument, do not modify')
```

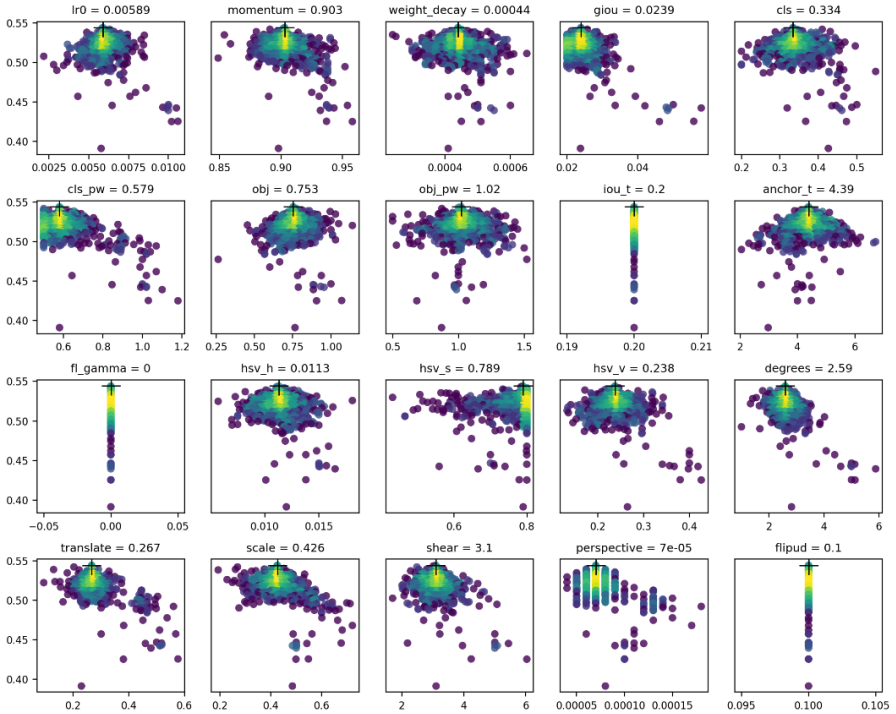
train.py 파일 내에서 학습과 관련된 파라미터의 기본값을 확인 및 변경 가능
→ 설정 안할 시 train.py 파일의 기본값으로 실행됨

초기 설정

부가 기능) 모델의 하이퍼 파라미터 자동 설정 기능 – evolution

```
# Evolve hyperparameters (optional)
else:
    # Hyperparameter evolution metadata (mutation scale 0-1, lower_limit, upper_limit)
    meta = {
        'lr0': (1, 1e-5, 1e-1),  # initial learning rate (SGD=1E-2, Adam=1E-3)
        'lrf': (1, 0.01, 1.0),  # final OneCycleLR learning rate (lr0 * lrf)
        'momentum': (0.3, 0.6, 0.98),  # SGD momentum/Adam beta1
        'weight_decay': (1, 0.0, 0.001),  # optimizer weight decay
        'warmup_epochs': (1, 0.0, 5.0),  # warmup epochs (fractions ok)
        'warmup_momentum': (1, 0.0, 0.95),  # warmup initial momentum
        'warmup_bias_lr': (1, 0.0, 0.2),  # warmup initial bias lr
        'box': (1, 0.02, 0.2),  # box loss gain
        'cls': (1, 0.2, 4.0),  # cls loss gain
        'cls_pw': (1, 0.5, 2.0),  # cls BCELoss positive_weight
        'obj': (1, 0.2, 4.0),  # obj loss gain (scale with pixels)
        'obj_pw': (1, 0.5, 2.0),  # obj BCELoss positive_weight
        'iou_t': (0, 0.1, 0.7),  # IoU training threshold
        'anchor_t': (1, 2.0, 8.0),  # anchor-multiple threshold
        'anchors': (2, 2.0, 10.0),  # anchors per output grid (0 to ignore)
        'fl_gamma': (0, 0.0, 2.0),  # focal loss gamma (efficientDet default gamma=1.5)
        'hsv_h': (1, 0.0, 0.1),  # image HSV-Hue augmentation (fraction)
        'hsv_s': (1, 0.0, 0.9),  # image HSV-Saturation augmentation (fraction)
        'hsv_v': (1, 0.0, 0.9),  # image HSV-Value augmentation (fraction)
        'degrees': (1, 0.0, 45.0),  # image rotation (+/- deg)
        'translate': (1, 0.0, 0.9),  # image translation (+/- fraction)
        'scale': (1, 0.0, 0.9),  # image scale (+/- gain)
        'shear': (1, 0.0, 10.0),  # image shear (+/- deg)
        'perspective': (0, 0.0, 0.001),  # image perspective (+/- fraction), range 0-0.001
        'flipud': (1, 0.0, 1.0),  # image flip up-down (probability)
        'fliplr': (0, 0.0, 1.0),  # image flip left-right (probability)
        'mosaic': (1, 0.0, 1.0),  # image mixup (probability)
        'mixup': (1, 0.0, 1.0),  # image mixup (probability)
        'copy_paste': (1, 0.0, 1.0)}  # segment copy-paste (probability)
```

python train.py ..... --weights yolov5s.pt **--evolve**



모델에 적절한 hyperparameter를 유전 알고리즘을 이용해 자동으로 내 데이터 셋에 적합하게 설정하는 기능

그 때 탐색할 hyperparameter 범위를 직접 설정할 수 있음

초기 설정

부가 기능) 이미지 비율이 정사각형이 아닌경우

python train.py ..... --weights yolov5s.pt **--rect**

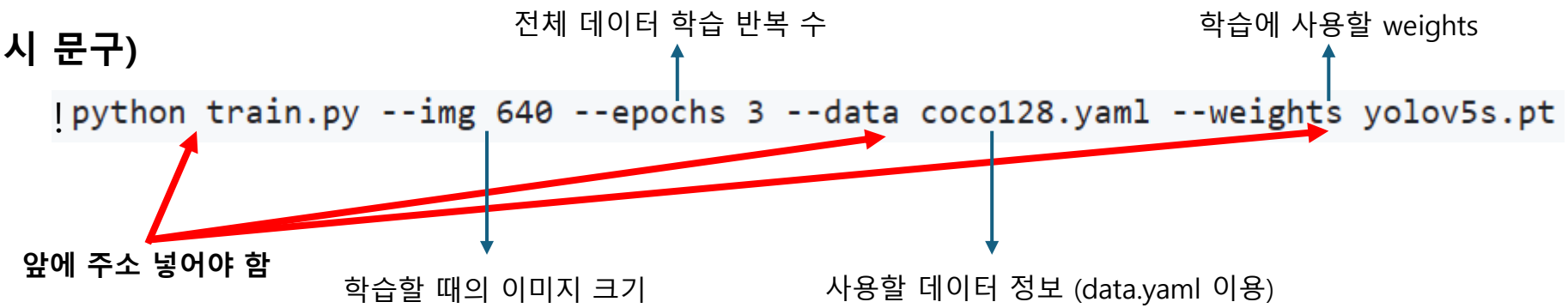
기본적으로 학습 이미지는 정사각형으로 학습되고, 직사각형 이미지의 경우 정사각형 비율로 일부 왜곡되어 학습됨.

--rect를 사용할 경우 직사각형 이미지의 가로 세로 중 긴 부분에 맞춰서 직사각형으로 학습됨
➔ ex) --img 1080 –rect =>(1080x720)으로 학습됨

객체 탐지 수행

객체 탐지 수행

예시 문구)



Ex) !python ~/train.py --img 320 --batch 5 --epochs 3 --device cpu --data ~/data.yaml --weights ~/yolov5s.pt

train.py 파일을 실행시켜 yolov5s.pt를 통해 학습 수행

추가로 사용할 수 있는 요소

--batch => 한 번에 학습하는 이미지(데이터) 수 → 학습 정확도와 관련
--cache => 학습 데이터 기록 → 학습 과정 확인을 위해 사용

⋮

객체 탐지 수행

객체 탐지 수행

```
      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
  0% 0/9 [00:00<?, ?it/s]/content/drive/MyDrive/yolov5/train.py:412: FutureWarning: `torch.cuda.amp.autocast(args...)` is deprecat
  with torch.cuda.amp.autocast(amp):
      99/99      7.15G   0.008235    0.00388  0.0001499        136        416:  11% 1/9 [00:00<00:03,  2.07it/s]/content/drive/MyD
  with torch.cuda.amp.autocast(amp):
      99/99      7.15G   0.009012   0.003488  0.0001471        114        416:  22% 2/9 [00:00<00:03,  2.25it/s]/content/drive/MyD
  with torch.cuda.amp.autocast(amp):
      99/99      7.15G   0.008838   0.003678  0.0001616        125        416:  33% 3/9 [00:01<00:02,  2.22it/s]/content/drive/MyD
  with torch.cuda.amp.autocast(amp):
      99/99      7.15G   0.008537    0.00362  0.0001593        130        416:  44% 4/9 [00:01<00:02,  2.34it/s]/content/drive/MyD
  with torch.cuda.amp.autocast(amp):
      99/99      7.15G   0.008389   0.003598  0.0001573        133        416:  56% 5/9 [00:02<00:01,  2.36it/s]/content/drive/MyD
  with torch.cuda.amp.autocast(amp):
      99/99      7.15G   0.008276   0.003576  0.0001563        114        416:  67% 6/9 [00:02<00:01,  2.25it/s]/content/drive/MyD
  with torch.cuda.amp.autocast(amp):
      99/99      7.15G    0.00818    0.00354  0.0001536        114        416:  78% 7/9 [00:03<00:00,  2.28it/s]/content/drive/MyD
  with torch.cuda.amp.autocast(amp):
      99/99      7.15G   0.008227   0.003563  0.0001562        127        416:  89% 8/9 [00:03<00:00,  2.35it/s]/content/drive/MyD
  with torch.cuda.amp.autocast(amp):
      99/99      7.15G   0.008292   0.003513  0.0001621         49        416: 100% 9/9 [00:03<00:00,  2.43it/s]
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 1/1 [00:00<00:00,  1.04it/s]
                   all        114        114      0.441          1      0.775       0.66

100 epochs completed in 0.147 hours.
Optimizer stripped from runs/train/exp/weights/last.pt, 14.3MB
Optimizer stripped from runs/train/exp/weights/best.pt, 14.3MB

Validating runs/train/exp/weights/best.pt...
Fusing layers...
Model summary: 157 layers, 7015519 parameters, 0 gradients, 15.8 GFLOPs
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 1/1 [00:00<00:00,  1.06it/s]
                   all        114        114      0.399          1      0.885      0.734
          belt detected       114        110      0.786          1      0.981      0.755
      belt not detected       114          4     0.0122          1      0.788      0.714
Results saved to runs/train/exp
```

높은 성능의 GPU 사용 시 더 큰 크기의 이미지 학습 및 한 번에 학습 가능한 데이터 수를 크게 할 수 있음

* 저장된 위치 잘 확인하기

객체 탐지 수행

```
# 텐서보드 실행
%load_ext tensorboard
%tensorboard --logdir runs
```

Tensorboard를 실행시키면 학습 과정 및 mAP, F1 score 등 다양한 학습 결과를 확인 가능

객체 탐지 수행

학습 모델 이용 탐지 수행

탐지를 수행할 이미지 폴더

```
!python detect.py --weights runs/train/exp{train_exp_num}/weights/best.pt --img 416 --conf 0.3 --source {test_data_dir}
```

학습된 모델의 최고 성능의 weights 사용

Confidence threshold : 객체를 탐지할 최소 정확도
Ex) 0.3 보다 작은 정확도를 가질 경우 객체가 없는 것으로 판단

```
@smart_inference_mode()
def run(
    weights=ROOT / "yolov5s.pt",  # model path or triton URL
    source=ROOT / "data/images",  # file/dir/URL/glob/screen/0(webcam)
    data=ROOT / "data/coco128.yaml",  # dataset.yaml path
    imgsz=(640, 640),  # inference size (height, width)
    conf_thres=0.25,  # confidence threshold
    iou_thres=0.45,  # NMS IOU threshold
    max_det=1000,  # maximum detections per image
    device="",  # cuda device, i.e. 0 or 0,1,2,3 or cpu
    view_img=False,  # show results
    save_txt=False,  # save results to *.txt
    save_csv=False,  # save results in CSV format
    save_conf=False,  # save confidences in --save-txt labels
    save_crop=False,  # save cropped prediction boxes
    nosave=False,  # do not save images/videos
    classes=None,  # filter by class: --class 0, or --class 0 2 3
    agnostic_nms=False,  # class-agnostic NMS
    augment=False,  # augmented inference
```

학습된 모델을 이용하여 탐지를 수행하는 코드

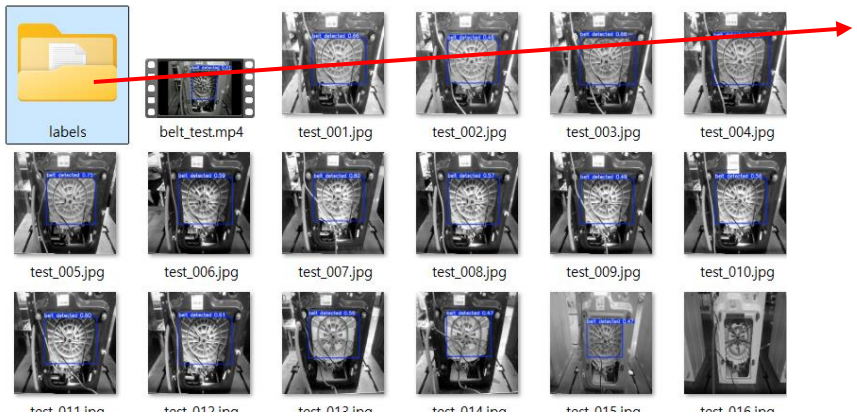
Detect.py 파일에서 다양한 하이퍼 파라미터를 조절 가능

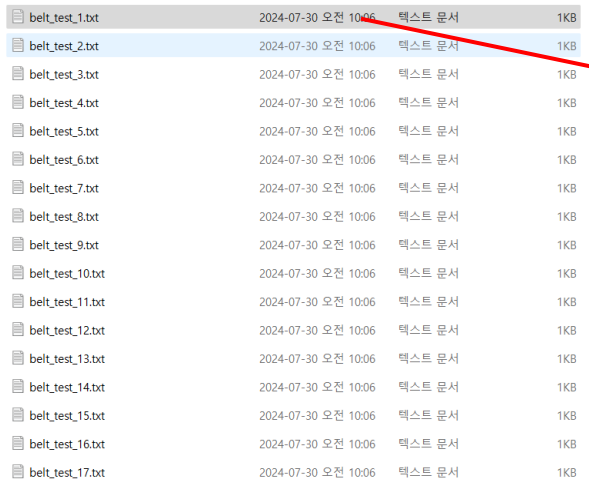
Ex)
conf_thres => 설정 안할 경우 기본값 0.25
Iou_thres -> 객체로 탐지하기 위한 최소 IoU 값
Max_det -> 한 이미지 내에서 탐지가능한 최대 객체 수
Save_txt -> True로 할 경우 탐지 결과를 txt 파일로 저장 가능

객체 탐지 수행

부가 기능 – save-txt

```
1  -conf 0.3 --source C:/Users/USER/Desktop/yolov5-master/data/test/images --save-txt
```



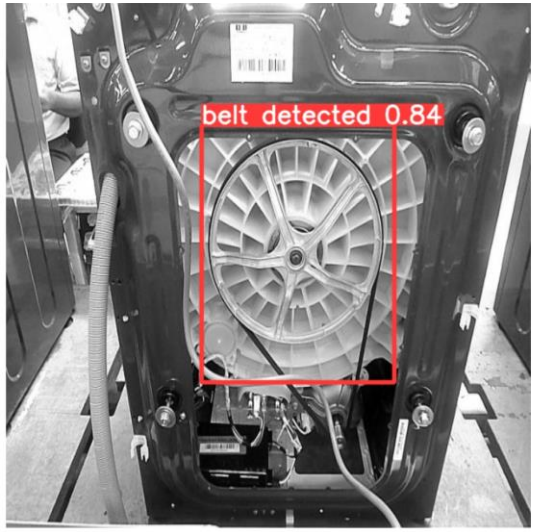


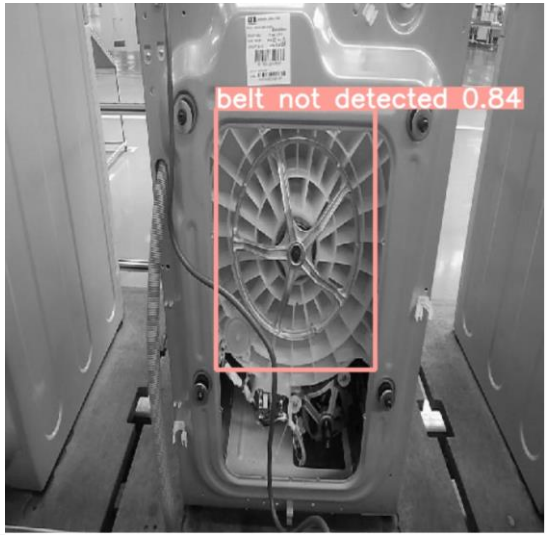
detect.py 기능을 사용해 탐지를 수행할 때 탐지 결과를 사진과 더불어 숫자 데이터로 얻고 싶을 때 사용

객체 탐지 수행

runs 폴더 내의 detect 폴더에 detect.py를 통해 탐지한 결과가 저장됨

```
for imageName in glob.glob('/content/drive/MyDrive/yolov5/runs/detect/exp' + str(test_exp_num) + '/*.jpg'):
    display(Image(filename=imageName))
    print("₩n")
```

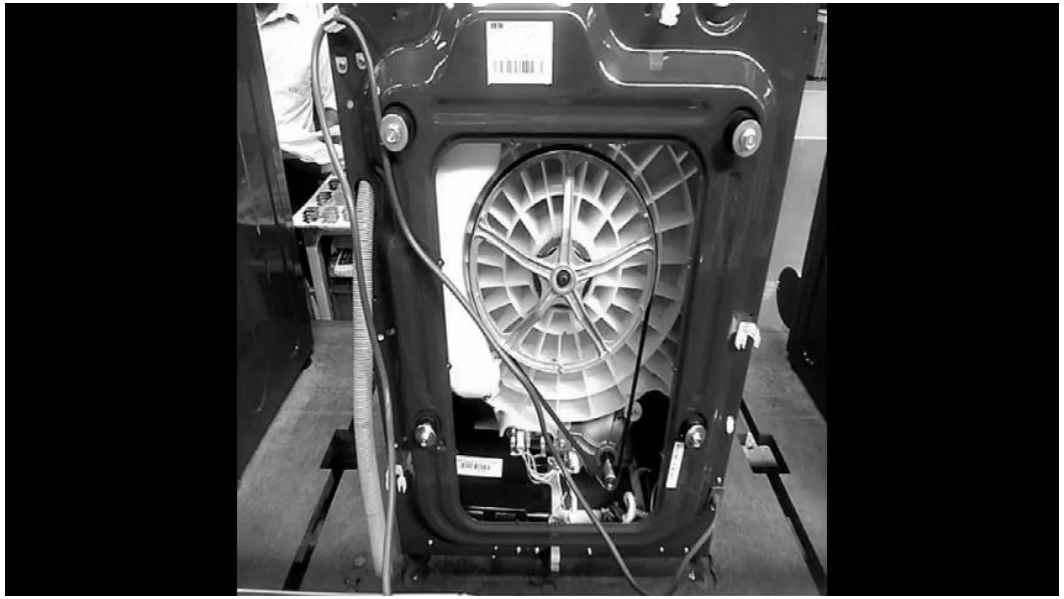


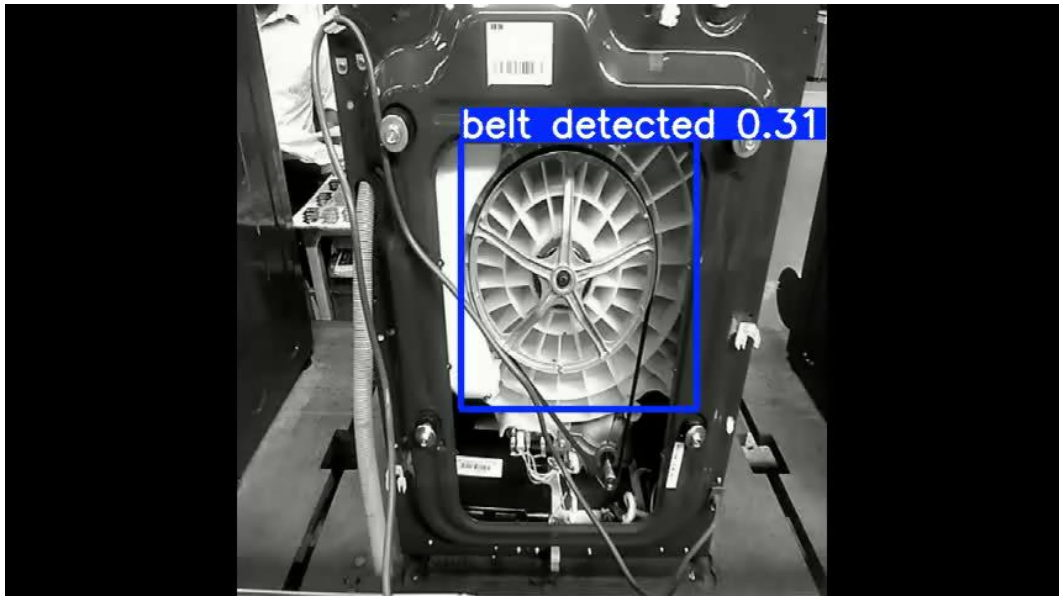


Detect를 수행한 폴더를 불러와 이미지 확인

객체 탐지 수행

영상에 대해 탐지 수행

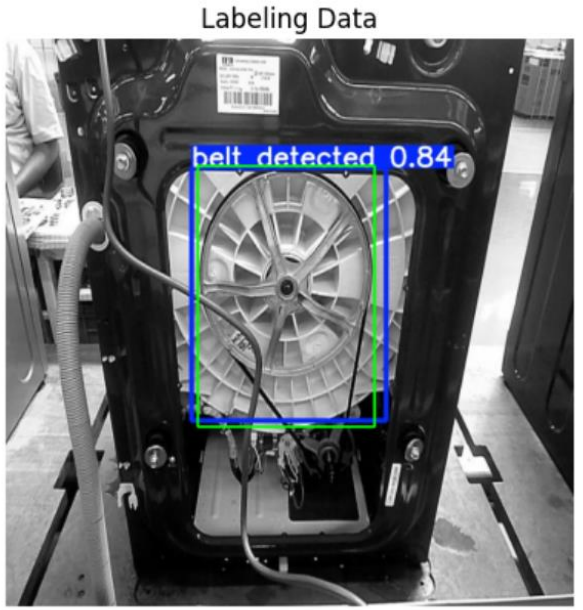


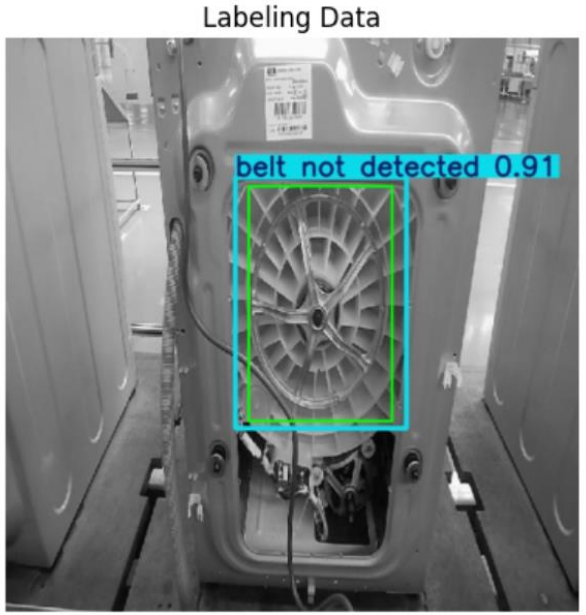


detect.py 파일을 실행시켜 동일한 코드로 영상에 대해서도 탐지를 수행할 수 있음

객체 탐지 수행

```python
import os
import yaml
import cv2
import matplotlib.pyplot as plt
from IPython.display import Image, display
import glob

# 바운딩 박스 그리기 함수
def draw_boxes(image, boxes, color, labels=None):
    for i, box in enumerate(boxes):
        x1, y1, x2, y2 = map(int, box)
        cv2.rectangle(image, (x1, y1), (x2, y2), color, 2)
        if labels:
            cv2.putText(image, labels[i], (x1, y1 - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.9, color, 2)
    return image

def load_labels(label_file, width, height):
    with open(label_file, 'r') as file:
        labels = [list(map(float, line.split())) for line in file.readlines()]
    boxes = [convert_box_format(label, width, height) for label in labels]
    return boxes

def convert_box_format(box, width, height):
    x_center, y_center, w, h = box[1:5]
    x1 = (x_center - w / 2) * width
    y1 = (y_center - h / 2) * height
    x2 = (x_center + w / 2) * width
    y2 = (y_center + h / 2) * height
    return [x1, y1, x2, y2]

# 테스트 결과 시각화
```





탐지 결과와 labeling 결과 비교

→ Detect 결과는 사진에 bounding box가 붙은 채로 출력되기 때문에 탐지 정확도를 수치화해서 출력하는 방법은 아님
detect.py의 save_txt 파라미터를 true로 하면 bounding box의 txt를 추출하여 직접적인 비교에 사용할 수 있음

객체 탐지 수행

예제 2에 대해서도 동일하게 학습 및 테스트 결과 비교

```
# [3] 데이터 압축 해제

!unzip /content/drive/MyDrive/dataset/DIC_YOLO2.zip -d /content/drive/MyDrive/dataset
```

```
  inflating: /content/drive/MyDrive/dataset/DIC_subset/labels/top_45_part_1640593655024-999_jpg.rf.d4c6c4502485f8a466195a0909af5286.txt
  inflating: /content/drive/MyDrive/dataset/DIC_subset/labels/top_45_part_1640593656426-303_jpg.rf.e0c75ce0f4823273ceff2c6b9369d400.txt
  inflating: /content/drive/MyDrive/dataset/DIC_subset/labels/top_45_part_1640593659515-0981_jpg.rf.68b30ddce2fc33da00a3f0bf37c37a60.txt
  inflating: /content/drive/MyDrive/dataset/DIC_subset/labels/top_45_part_1640593660118-4849_jpg.rf.d41ad2909e4bdf8d02e837e88117bbe7.txt
  inflating: /content/drive/MyDrive/dataset/DIC_subset/labels/top_45_part_1640593661427-491_jpg.rf.64cad1b2c3ea46592b9a86f27179933e.txt
  inflating: /content/drive/MyDrive/dataset/DIC_subset/labels/top_45_part_1640593662774-424_jpg.rf.5c9e3c3bb86323fe068beccb2eebe4a8.txt
  inflating: /content/drive/MyDrive/dataset/DIC_subset/labels/top_45_part_1640593667054-5051_jpg.rf.d7b963be952229fe6ac62174bf3b4b08.txt
  inflating: /content/drive/MyDrive/dataset/DIC_subset/labels/top_45_part_1640593687916-6719_jpg.rf.66c78beb383b7e2ce23e074b824a4b42.txt
```

DIC_YOLO2.zip 파일을 dataset 폴더에 넣고 압축해제

객체 탐지 수행

```python
# [4] Train/Validation 자동 분할 (8:2)
import os, random, shutil

img_dir = "/content/drive/MyDrive/dataset/DIC_subset/images"
lbl_dir = "/content/drive/MyDrive/dataset/DIC_subset/labels"
train_out = "/content/drive/MyDrive/dataset/DIC_split/images/train"
val_out = "/content/drive/MyDrive/dataset/DIC_split/images/val"
train_lbl_out = "/content/drive/MyDrive/dataset/DIC_split/labels/train"
val_lbl_out = "/content/drive/MyDrive/dataset/DIC_split/labels/val"

os.makedirs(train_out, exist_ok=True)
os.makedirs(val_out, exist_ok=True)
os.makedirs(train_lbl_out, exist_ok=True)
os.makedirs(val_lbl_out, exist_ok=True)

images = [f for f in os.listdir(img_dir) if f.endswith('.jpg')]
random.seed(42)
random.shuffle(images)
split_idx = int(0.8 * len(images))

for i, fname in enumerate(images):
    label = fname.replace(".jpg", ".txt")
    if i < split_idx:
        shutil.copy(os.path.join(img_dir, fname), os.path.join(train_out, fname))
        shutil.copy(os.path.join(lbl_dir, label), os.path.join(train_lbl_out, label))
    else:
        shutil.copy(os.path.join(img_dir, fname), os.path.join(val_out, fname))
        shutil.copy(os.path.join(lbl_dir, label), os.path.join(val_lbl_out, label))
```

train, validation 데이터를 8:2로 나누어 저장

객체 탐지 수행

[5] `data.yaml` 수정 실습

아래는 수정 예시입니다. `DIC_split` 폴더를 기준으로 학습/검증 데이터를 참조하고, `test` 는 원본에서 사용합니다.

```
train: ../train/images
val: ../valid/images       => 실제 데이터가 있는 주소로
test: ../test/images

nc: 2  -> class 수에 맞게
names: ['belt detected', 'belt not detected']  -> 사용할 class 명으로
```

```
- baseplate
- childpart1
- childpart2
- clinching
- pin1
- pin2
```

1번 실습 데이터에서 사용하던 data.yaml 파일을 수정하여 현재 class 명과 개수, 데이터 위치에 맞게 수정

Class 명의 순서는 : baseplate, childpart1, childpart2, clinching, pin1, pin2 순서대로 작성

객체 탐지 수행

```
# [6] 학습 실행  // 지우고 제공
!python train.py --img 640 --batch 4 --epochs 30 --data /content/drive/MyDrive/dataset/data.yaml --weights yolov5s.pt --name DIC_small_train
```

train.py를 실행시켜 학습 수행 -> 테스트 이미지에 대해 예측 -> 예측 결과 확인

감사합니다