

## [프로젝트 #2]

# Object Detection(YOLOv5) 자료

2022.4.20

도규원

충북대학교 산업인공지능학과

# Table of Content

---

- 1. 공정개요 및 이슈사항
- 2. 객체 탐지 실습
  - 2.1 yolov5 설치(Installation of yolov5)
    - (1) yolov5 설치
    - (2) yolov5 시연
  - 2.2 나만의 객체탐지 모델 만들기
    - (1) 개요
    - (2) 이미지 학습(Building my own image data learning)
  - 2.3 객체 인식 실험(Detecting my own object from image data)
  - 2.4 라벨링 툴 소개
  - 2.5 Labellmg 설치
  - 2.6 Labellmg 사용법

# 1. 공정 개요 및 이슈 사항

## ■ 참치 제조 공정

- 참치 무게별 선별 → 자숙 공정(열을 가해 찌냄) → 클리닝(참치 뼈 제거) → 캔 포장
- 참치 무게 선별 작업(현재 작업자에 의한 육안으로 선별) 오류시 자숙공정에서 적절한 자숙시간 확보가 안되서(미자숙 또는 과자숙) 품질 문제 발생
- 이미지 기반 참치 길이와 무게 분석을 통해 참치 등급 선별을 하고자 함
- 이미지 상의 참치의 특정 부위(머리, 꼬리) 탐지(detection) 하여 머리와 꼬리 사이의 이미지상의 pixel 거리를 알아내고 거리에 해당하는 무게를 예측하고자 함



참치 무게 선별 작업  
(작업자 육안으로 선별)



이미지와 탐지하고자 하는 특정 위치 라벨 데이터로 학습



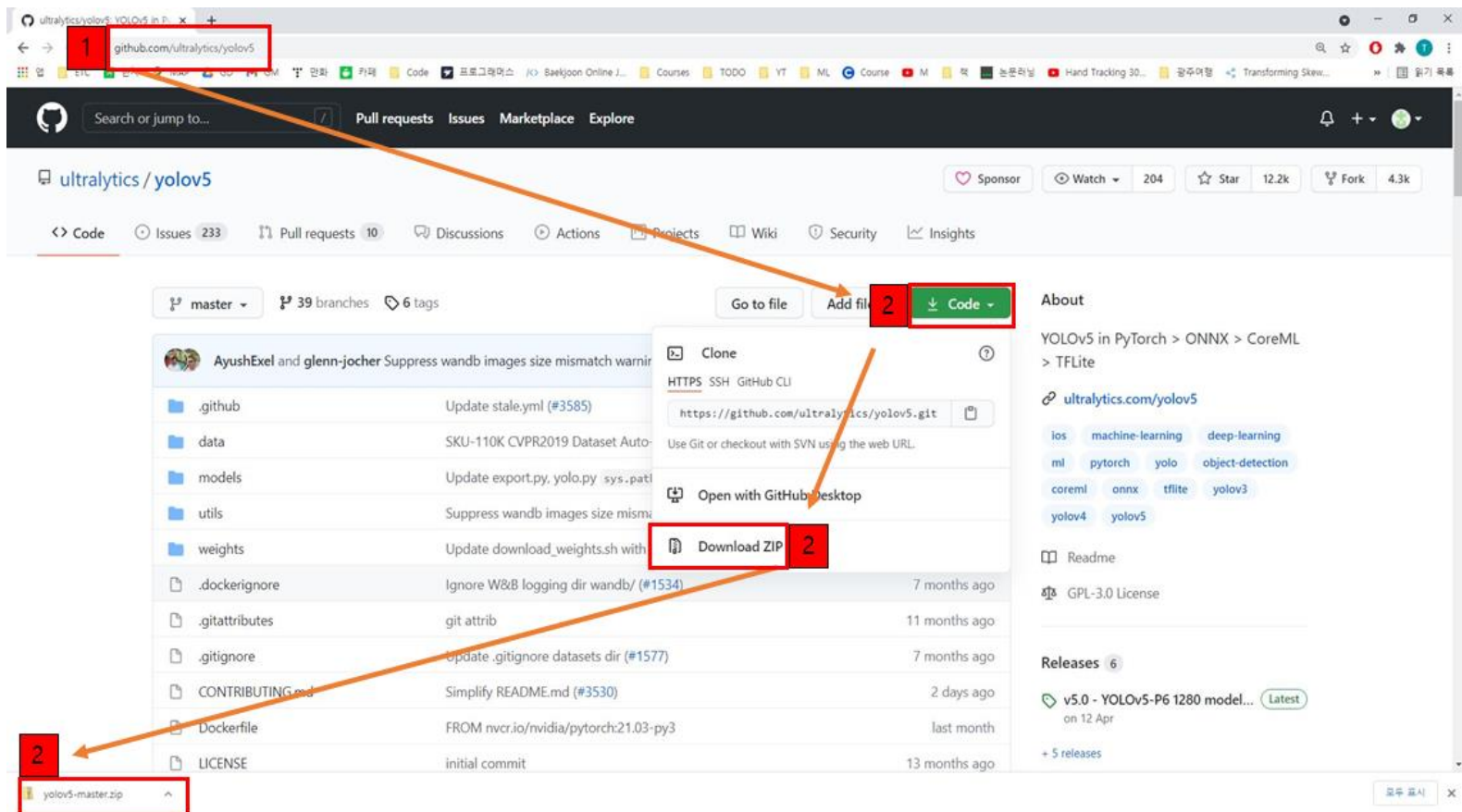
컨베이어 벨트를 통해 이동중인 참치 탐지  
및 참치 길이에 따른 등급(무게) 표시

## 2. 객체 탐지 실습

### ■ 2.1 YOLOv5 설치

#### ■ 설치 방법

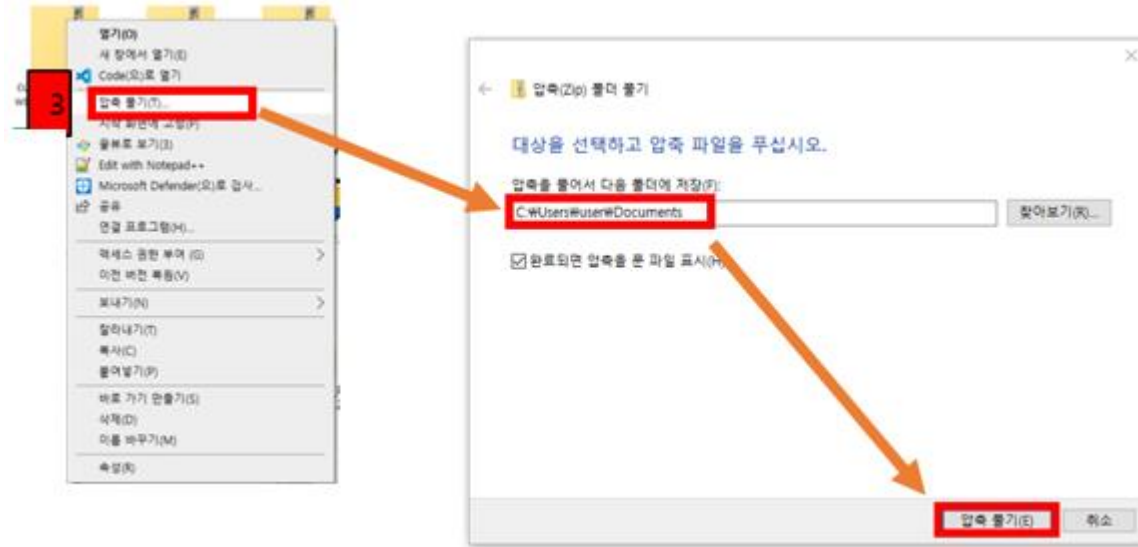
1. <https://github.com/ultralytics/yolov5> 접속
2. 우측상단 초록색 'Code' 버튼 클릭 후 'Download ZIP'으로 저장



## 2. 객체 탐지 실습

### ■ 2.1 YOLOv5 설치

#### 3. 저장된 압축 폴더를 원하시는 경로/폴더에 압축 풀기

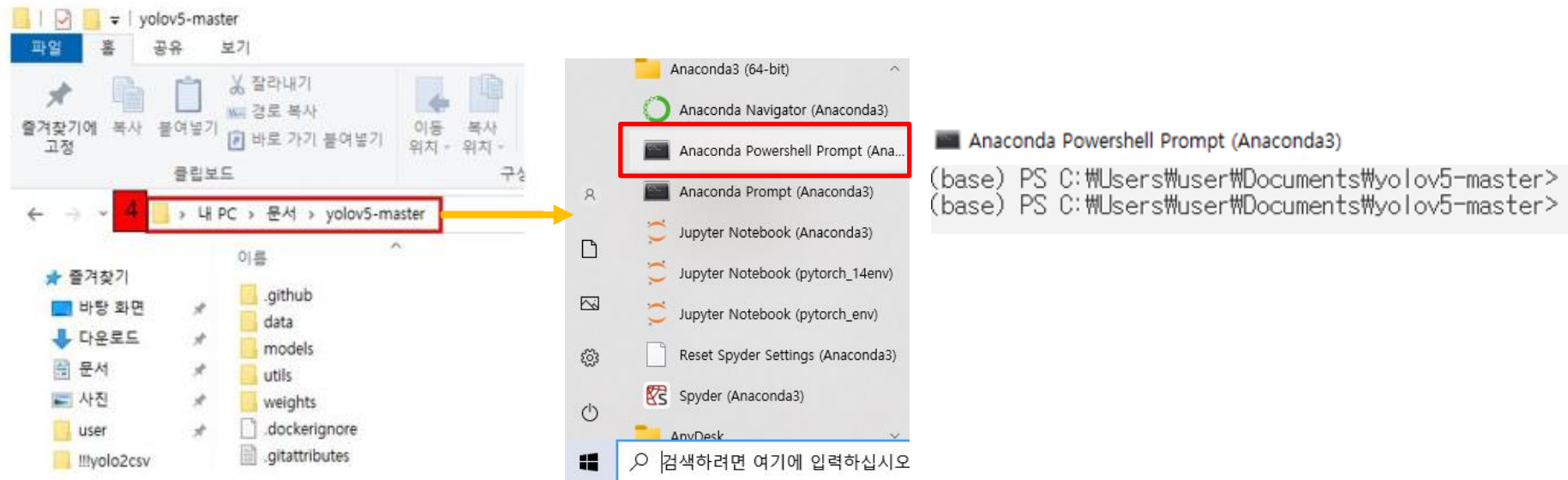


- '찾아보기'버튼으로 원하는 경로에 yolo 폴더를 저장함, 여기서는 문서(Documents)에 저장

## 2. 객체 탐지 실습

### ■ (1) YOLOv5 설치

4. 압축 해제후 Anaconda Powershell Prompt 실행후 명령 프롬프트 창이 나타남



5. 명령 프롬프트 창에 'pip install -r requirements.txt' 입력 후 실행. 실행을 하면 yolo 구현에 필요한 파이썬 패키지들을 자동으로 설치해줌

```
Anaconda Powershell Prompt (Anaconda3)  
C:\Users\User\Documents\yolo2csv>pip install -r requirements.txt
```

■ 이제 yolo를 위한 준비단계 끝

## 2. 객체 탐지 실습

### ■ (2) YOLOv5 시연

1. 다행스럽게도 올려놓은 미리 학습된 무게 (weights) 들로 어느정도 이미지나 영상 안에 있는 통상적인 물체들은 탐지가 가능함
2. yolo 폴더에 있는 /data/images 경로에 bus.jpg 와 zidane.jpg라는 파일을 넣어 객체 탐지 모델을 구현



3. yolo 폴더 내에 접속된 명령 프롬프트 창을 열고 'python detect.py'를 실행

```
Anaconda Powershell Prompt (Anaconda3)  
C:\Users\User\Documents\yolov5-master>python detect.py
```

## 2. 객체 탐지 실습

### ■ (2) YOLOv5 시연

4. 실행이 끝나면 yolo의 메인 폴더에 'runs'라는 폴더 생성 '/runs/detect/exp\*' 폴더에 Bounding box와 탐지된 객체의 이름이 명시된 사진들이 저장



올로 실행 이후의 zidane.jpg

5. yolo 로 이미지내에 있는 객체 탐지 성공!



## 2. 객체 탐지 실습

### ■ (2) YOLOv5 시연

- 이전에 'python detect.py'를 실행 했을 때 bus랑 zidane 파일들의 경로도 알려주지 않고, 객체 탐지를 사용하기 위한 가중치 역시 설정을 하지 않았는데도 실행됨
- 그 이유는 detect.py 코드 안에 있음. 코드 에디터 (notepad, visual studio code, pycharm, etc.)로 detect.py를 열어 맨 아래부분에 다음과 같은 코드가 존재함

```
if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument('--weights', nargs='+', type=str, default='yolov5s.pt', help='model.pt path(s)')
    parser.add_argument('--source', type=str, default='data/images', help='file/dir/URL/glob, 0 for webcam')
    parser.add_argument('--imgsz', '--img', '--img-size', type=int, default=640, help='inference size (pixels)')
    parser.add_argument('--conf-thres', type=float, default=0.25, help='confidence threshold')
    parser.add_argument('--iou-thres', type=float, default=0.45, help='NMS IoU threshold')
    parser.add_argument('--max-det', type=int, default=1000, help='maximum detections per image')
    parser.add_argument('--device', default='', help='cuda device, i.e. 0 or 0,1,2,3 or cpu')
    parser.add_argument('--view-img', action='store_true', help='show results')
    parser.add_argument('--save-txt', action='store_true', help='save results to *.txt')
    parser.add_argument('--save-conf', action='store_true', help='save confidences in --save-txt labels')
    parser.add_argument('--save-crop', action='store_true', help='save cropped prediction boxes')
    parser.add_argument('--nosave', action='store_true', help='do not save images/videos')
    parser.add_argument('--classes', nargs='+', type=int, help='filter by class: --class 0, or --class 0 2 3')
    parser.add_argument('--agnostic-nms', action='store_true', help='class-agnostic NMS')
    parser.add_argument('--augment', action='store_true', help='augmented inference')
    parser.add_argument('--update', action='store_true', help='update all models')
    parser.add_argument('--project', default='runs/detect', help='save results to project/name')
    parser.add_argument('--name', default='exp', help='save results to project/name')
    parser.add_argument('--exist-ok', action='store_true', help='existing project/name ok, do not increment')
    parser.add_argument('--line-thickness', default=3, type=int, help='bounding box thickness (pixels)')
    parser.add_argument('--hide-labels', default=False, action='store_true', help='hide labels')
    parser.add_argument('--hide-conf', default=False, action='store_true', help='hide confidences')
    parser.add_argument('--half', action='store_true', help='use FP16 half-precision inference')
    opt = parser.parse_args()
    print(opt)
    check_requirements(exclude=('tensorboard', 'thop'))

    detect(**vars(opt))
```

## 2. 객체 탐지 실습

### ■ (2) YOLOv5 시연

#### ■ Weights

```
parser.add_argument('--weights', nargs='+', type=str, default='yolov5s.pt', help='model.pt path(s)')
```

- 3번째 줄에 있는 코드
- '--weights'란 단어 객체 탐지할 때 무게/가중치 값인데 이 값이 이미지에 있는 특정한 물체를 인식할 수 있게 해주는 패턴 인식기임
- 'default='yolov5s.pt' → default는 detect.py 파일을 실행 시 가중치를 설정하지 않는 이상 자동으로 기본값인 yolov5s.pt라는 가중치 파일로 객체 탐지를 실행함

#### ■ Source

- 4번째 줄에 있는 코드

```
parser.add_argument('--source', type=str, default='data/images', help='file/dir/URL/glob, 0 for webcam')
```

- 이번에는 '--source'라는 입력 값의 default, 즉 기본값이 'data/images'로 bus.jpg와 zidane.jpg가 들어있는 폴더 경로임
- 'python detect.py'를 실행할 때는 추가적으로 입력 값 설정이 없으면 사실 **'python detect.py --weights yolov5s.pt --source data/images'**를 내부적으로 실행함

## 2. 객체 탐지 실습

### ■ (2) YOLOv5 시연

- 임의의 훈련데이터로 물체 탐지하고자 할 때는
- 만약에 내가 직접 훈련한 가중치인 `my_weights.pt`를 이용하여 애완견 사진이 있는 `my_pet` 폴더의 이미지에 yolo를 실행하고자 하면, `my_pet` 폴더가 `yolov5-master` 폴더 아래에 있는 확인하고
- **`'python detect.py --weights my_weights.pt --source my_pet'`을 실행**

■ Anaconda Powershell Prompt (Anaconda3)

```
(base) PS C:\Users\User\Documents\yolov5-master> python detect.py --weights my_weights.pt --source my_pet,
```

## 2. 객체 탐지 실습

### ■ 2.2 나만의 객체 탐지 모델 만들기

#### ■ (1) 개요

만약에 yolo 개발자가 미리 학습해 놓은 yolov5s.pt의 가중치를 사용하지 않고 내가 원하는 특정 객체들만 인식하는 가중치를 만들고 싶다면? 예를 들어, 참치 사진만 탐지하는 특별한 모델을 만들고 싶다면?

그런 모델을 만들기 위해선 3가지 단계가 필요함

### 1. 이미지 수집

먼저 참치 탐지 모델을 만든다고 하면, 참치 사진들을 구해서 저장 해야함.

일반적으로 사진은 많을수록 탐지 모델의 정확도가 높아짐

### 2. 이미지 라벨링

학습 시간보다 더 많은 시간이 소요되는 이미지 라벨링 작업임. 이미지 라벨링이란 관심 객체 (현재 참치)가 들어있는 사진 구간에 박스(Bounding Box)를 그려서 객체의 위치 구간을 저장하는 작업임

### 3. 이미지 학습

## 2. 객체 탐지 실습

### ■ 2.2 나만의 객체 탐지 모델 만들기

#### ■ 이미지 라벨링

- 작업을 가능케 하는 여러 도구들이 있지만 이미지가 1000장 미만이면 roboflow.com에 새로운 프로젝트 생성 후 사진들 upload 후 사이트 내에서 라벨링 작업 추천함



노랑색 박스처럼 원하는 검출하고자 하는 객체의 위치 구간을 설정하는 작업

## 2. 객체 탐지 실습

---

### ■ 2.2 나만의 객체탐지 모델 만들기

#### ■ 이미지 라벨링

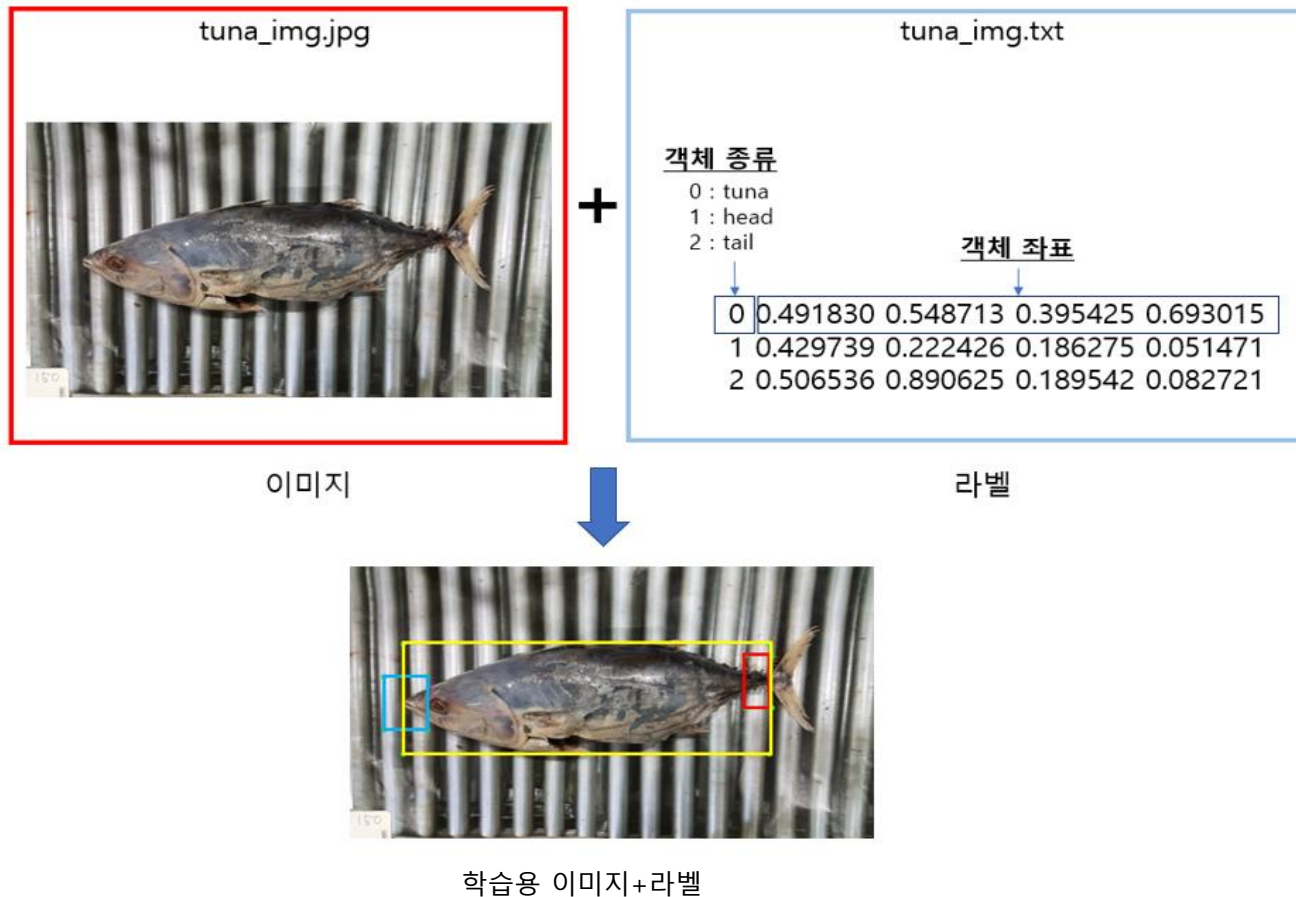
- 이미지가 1000장 이상이 넘으시면 labellmg-master라는 파이썬 프로그램을 이용해서 작업 가능함(4장에 소개됨)
- <https://github.com/tzutalin/labellmg>

## 2. 객체 탐지 실습

### ■ 2.2 나만의 객체탐지 모델 만들기

### ■ 이미지 학습

- 학습에 필요한 데이터 : **이미지** + **라벨** (이미지에서 찾고자 하는 객체 위치 정보)



## 2. 객체 탐지 실습

### ■ 2.2 나만의 객체탐지 모델 만들기

### ■ 학습

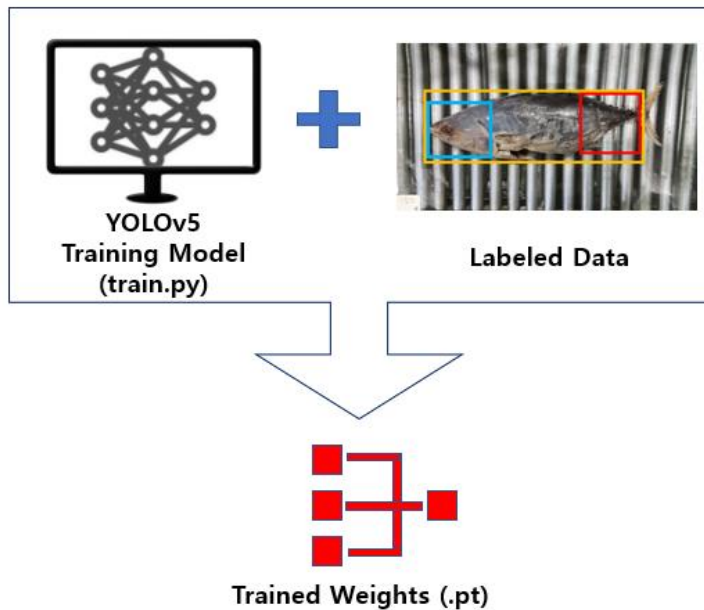
- 참치 이미지를 준비하고, 이미지만으로는 어떤 위치에 탐지하고자 하는 객체가 있는지 알 수가 없으므로 탐지하고자 하는 객체(예를 들어 참치 전체 영역, 머리, 꼬리)의 좌표를 나타내는 텍스트 파일을 만들어 줌(라벨링 작업이라고 말함)
- 라벨 파일은 1줄당 객체 종류와 객체 좌표로 구성됨 첫번째 숫자가 객체 종류를 의미 나머지 4개의 숫자가 객체 좌표를 나타냄
- 4개의 객체 좌표는 사각형 영역의 (1) 중앙점 x 축(  $(x_{\max}-x_{\min})/2$  ), (2) 중앙점 y 축(  $(y_{\max}-y_{\min})/2$  ), (3) 폭, (4) 높이를 나타냄
- 이런 이미지와 라벨 파일이 충분히 많이 있어야 하고 탐지하고자 하는 객체에 대한 라벨링이 정확하게 되어있어야 함
- 학습이 끝난 후에 모델 신뢰성 보장 가능함 ( 탐지하고자 하는 객체를 담고 있는 이미지가 충분히 많이 확보하고, 학습하여야 모델의 탐지 성능이 좋아짐 )
- 현재 기본적인 학습 횟수(epoch)는 300번으로 설정 필요에 따라서는 더 많은 횟수로 학습 가능함(600, 1200 등) 다만 많은 횟수로 학습을 하려면 GPU 가 있어야 학습하는 시간을 줄일 수 있음



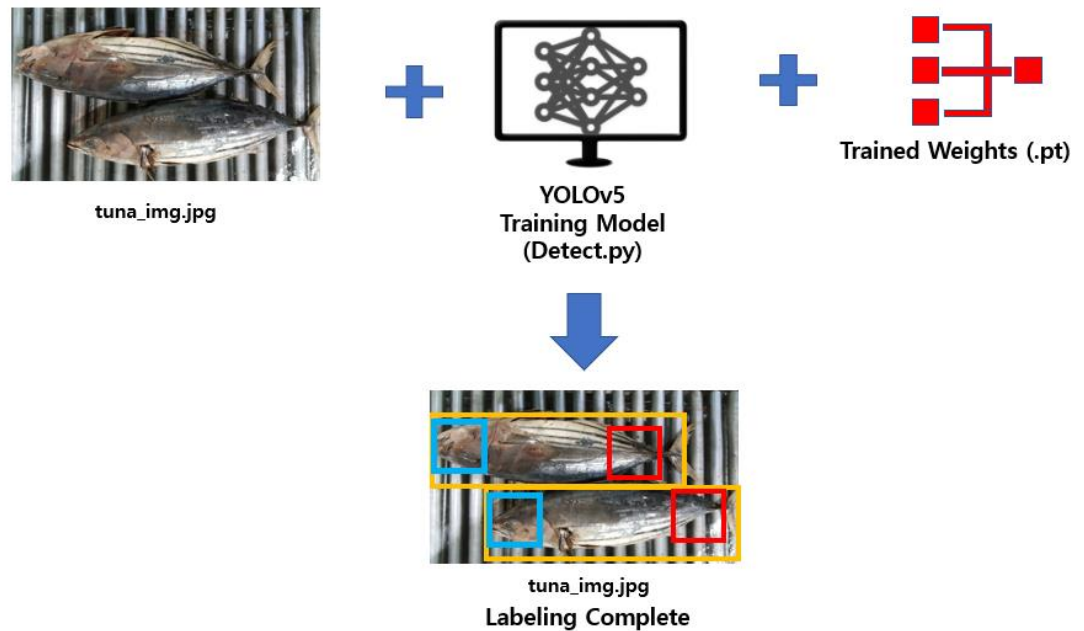
## 2. 객체 탐지 실습

- 2.2 나만의 객체탐지 모델 만들기
- YOLOv5 학습 및 객체 탐지 파이프 라인

### 학습



### 객체 탐지



## 2. 객체 탐지 실습

### ■ 2.2 나만의 객체탐지 모델 만들기

### ■ 이미지 학습

- 이미지 학습은 옐로 폴더의 train.py를 이용해서 진행됨 yolo의 homepage코드 예제임

```
python train.py --data coco.yaml --cfg yolov5s.yaml --weights '' --batch-size 64
                                yolov5m          40
                                yolov5l          24
                                yolov5x          16
```

- 먼저 python train.py 실행하고 실행할 때 추가적인 입력 값을 설정해서 넣을 수 있음
- '--cfg' 의 입력 값은 어떤 yolo 모델을 이용해서 학습할지 설정
  - x(x-large), l(large), m(medium), s(small)의 4종류로 구성
  - 아래로 갈수록 모델이 더욱 복잡해지고 학습 시간, 메모리 할당량과 연산량이 증가하는 대신 정확도가 증가함 주로 가장 작은 모델인 '--cfg yolov5s.yaml'을 사용함
- '--weights'의 입력 값은 완전 백지 상태에서 무게 가중치를 학습할지 아니면 미리 학습된 무게 가중치 값으로 시작해서 전이 학습을 진행 하느냐의 차이임. 예를 들어 --weights 다음의 구문을 따옴표 두개로 처리하면 백지 상태로 학습 한다는 거고 '--weights yolov5s.pt'로 지정하면 학습할 때 가중치들이 yolov5s.pt의 것으로 시작되어 값이 점차적으로 바뀌는 것을 말함
- '--batch-size'로 학습 데이터의 배치 크기를 설정 가능함 학습시에는 가능한 가장 큰 배치 사이즈로 학습하는 것을 추천함

## 2. 객체 탐지 실습

- 가장 중요한 '--data coco.yaml' 부분이 학습하고 싶은 이미지들과 레이블들의 경로를 설정하는 파일을 입력함
- 이 예제에서는 coco.yaml 파일 안에 훈련과 검증 데이터셋 레이블들의 경로가 설정 되어 있음
- 여기서 coco.yaml을 그대로 사용하시면 COCO Dataset이라는 유명한 이미지 데이터셋으로 학습을 하는 거니, coco.yaml의 소스 코드를 복사해서 mydata.yaml을 따로 만드신 후, 정보를 변경함
- "download:..."로 시작하는 줄은 삭제함(COCO Dataset을 웹에서 다운로드하기 위한 함수임) "train:" 옆에는 훈련 데이터셋 경로를, "val:" 옆에는 검증 데이터셋 경로를, "test: " 옆에는 테스트 데이터셋 경로를 기입함
- train, val 은 반드시 필요하며, test는 선택적이니 지워도 무방함
- "nc:" 옆에는 탐지하는 클래스 수, 예를 들어 고양이랑 강아지를 탐지하는 모델이면 2를 기록함
- "names:" 에는 클래스의 이름들을 기록 단 이미지 상에 검출하고자 하는 대상 객체를 라벨링 할 때 사용한 객체의 이름순서 그대로 기록해야 함 (라벨링 된 파일에 객체명이 텍스트 형태가 아닌 일련번호(예, 0,1,2 ...)로 저장되어 있으므로 순서가 바뀌면 결과파일에 표시되는 객체이름이 다르게 표시됨)

## 2. 객체 탐지 실습

### coco.yaml

```
coco.yaml
1 # COCO 2017 dataset http://cocodataset.org
2 # Train command: python train.py --data coco.yaml
3 # Default dataset location is next to YOLOv5:
4 #   /parent_folder
5 #   /coco
6 #   /yolov5
7
8
9 # download command/URL (optional)
10 download: bash data/scripts/get_coco.sh
11
12 # train and val data as 1) directory: path/images/, 2) file: path/images.txt, or 3) list: [path1/images/, path2/images/]
13 train: ../coco/train2017.txt # 118287 images
14 val: ../coco/val2017.txt # 5000 images
15 test: ../coco/test-dev2017.txt # 20288 of 40670 images, submit to https://competitions.codalab.org/competitions/20794
16
17 # number of classes
18 nc: 80
19
20 # class names
21 names: [ 'person', 'bicycle', 'car', 'motorcycle', 'airplane', 'bus', 'train', 'truck', 'boat', 'traffic light',
22         'fire hydrant', 'stop sign', 'parking meter', 'bench', 'bird', 'cat', 'dog', 'horse', 'sheep', 'cow',
23         'elephant', 'bear', 'zebra', 'giraffe', 'backpack', 'umbrella', 'handbag', 'tie', 'suitcase', 'frisbee',
24         'skis', 'snowboard', 'sports ball', 'kite', 'baseball bat', 'baseball glove', 'skateboard', 'surfboard',
25         'tennis racket', 'bottle', 'wine glass', 'cup', 'fork', 'knife', 'spoon', 'bowl', 'banana', 'apple',
26         'sandwich', 'orange', 'broccoli', 'carrot', 'hot dog', 'pizza', 'donut', 'cake', 'chair', 'couch',
27         'potted plant', 'bed', 'dining table', 'toilet', 'tv', 'laptop', 'mouse', 'remote', 'keyboard', 'cell phone',
28         'microwave', 'oven', 'toaster', 'sink', 'refrigerator', 'book', 'clock', 'vase', 'scissors', 'teddy bear',
29         'hair drier', 'toothbrush' ]
30
31 # Print classes
32 # with open('data/coco.yaml') as f:
33 #     d = yaml.safe_load(f) # dict
34 #     for i, x in enumerate(d['names']):
35 #         print(i, x)
36
```

삭제

학습 데이터 폴더 위치 } 필수  
검증 데이터 폴더 위치  
테스트 데이터 폴더 위치

Class 개수(탐지하고자 하는 물체의 전체 개수)

Class 종류  
(탐지하고자 하는 물체 이름)

Note: .. (점두개)로 시작하는데 이 뜻은 현재 경로에서 이전 폴더로 가는 '뒤로가기'의 표시임

## 2. 객체 탐지 실습

- cocodata.yaml 파일을 다른 이름으로 저장하여 tunadata.yaml 을 생성함.
- train과 val 키에는 각각 훈련과 검증 데이터셋의 경로를 설정하는 부분
- 훈련 데이터셋으로 1회 학습(1 epoch)을 끝나치면 같은 데이터셋을 검증용으로 사용해 훈련된 모델의 정확도를 계산됨 만약 훈련이 모두 끝나고 test 데이터셋으로 마지막 테스트 단계를 진행하고 싶으시면 "test: " 쌍점 앞에 테스트 데이터가 들어있는 폴더 경로를 지정
- 학습데이터 폴더 : 절대 경로로 지정할 때에는 폴더명 앞뒤에 ' 추가함
  - train: 'C:\Users\KYUWONDOH\Documents\yolov5-master\train\tuna'
- 검증데이터 폴더 :
  - val: 'C:\Users\KYUWONDOH\Documents\yolov5-master\validation\tuna'
- class 총 개수(number of classes)
  - nc: 3 #검출 물체 총 개수 3개
  - names: [ 'tuna', 'head', 'tail' ]
- 맨 앞에 #을 붙이면 주석 처리됨(설명을 하려는 용도로 사용됨)

## 2. 객체 탐지 실습

### ■ 훈련 데이터 및 검증 데이터

- 라벨링 된 전체 데이터 중 대부분 데이터가 학습에 사용되고, 나머지의 일부 데이터는 학습한 결과를 확인하는 검증데이터로 사용하여 학습이 진행되면서 검증데이터를 통해 학습이 제대로 되고 있는지 확인하는 용도로 사용됨
- 학습데이터와 검증데이터의 비율은 사용자가 임의로 조정가능하고, 데이터의 개수가 많지 않을 경우 전체 데이터를 여러 개의 그룹으로 분리하고 그룹간 배치를 training set , validation set 으로 번갈아 하는 형태로 하기도 함)

훈련세트 및 검증 세트



## 2. 객체 탐지 실습

### ■ (예제) 경로설정 관련

- 직접 만든 데이터셋으로 훈련하기 위해 --data 항목에 들어가는 yaml 파일을 직접 만드는 작업을 진행함. 물고기, 곤충, 강아지, 고양이, 초콜릿을 라벨링 한 데이터셋을 만들었고, 해당 데이터를 훈련, 검증 데이터셋으로 나누었음. 각각의 데이터셋은 아래의 절대 경로에 위치해 있음

- (훈련) c:\user\yolov5-master\self-train\training\_data
- (검증) c:\user\yolov5-master\self-train\validation\_data
- (테스트) 생략

위 정보를 토대로 yaml 파일은 두가지 방식으로 작성 가능함

상대경로 스타일(relative\_style.yaml)

절대경로 스타일(absolute\_style.yaml)

```
train: self-train\training_data
val: self-train\validation_data
test: self-train\test_data

nc: 5

names: ['fish', 'insect', 'dog', 'cat', 'chocolate']
```

```
train: C:\user\yolov5-master\self-train\training_data
val: C:\user\yolov5-master\self-train\validation_data
test: C:\user\yolov5-master\self-train\test_data

nc: 5

names: ['fish', 'insect', 'dog', 'cat', 'chocolate']
```

(1)상대경로를 이용한 학습 `python train.py --data relative_style.yaml --cfg yolov5s.yaml --weights '' --batch-size 64`

(2)절대경로를 이용한 학습 `python train.py --data absolute_style.yaml --cfg yolov5s.yaml --weights '' --batch-size 64`

## 2. 객체 탐지 실습

### ■ 학습데이터 폴더

- 학습할 이미지 데이터와 이미지데이터 상에 검출하고자 하는 객체의 위치정보(text 파일)가 존재해야 함 (검증데이터 폴더도 마찬가지임)

내 PC > 문서 > yolov5-master > train > tuna				tuna 검색
이름	수정한 날짜	유형	크기	
0824_0.jpg	2020-08-26 PM 02:12	JPG 파일	70KB	
0824_0.txt	2020-09-07 AM 09:39	텍스트 문서	1KB	
0824_1.jpg	2020-08-26 PM 02:12	JPG 파일	70KB	
0824_1.txt	2020-09-07 AM 09:40	텍스트 문서	1KB	
0824_2.jpg	2020-08-26 PM 02:12	JPG 파일	71KB	
0824_2.txt	2020-09-07 AM 09:46	텍스트 문서	1KB	
0824_3.jpg	2020-08-26 PM 02:12	JPG 파일	67KB	
0824_3.txt	2020-09-07 AM 09:46	텍스트 문서	1KB	
0824_4.jpg	2020-08-26 PM 02:12	JPG 파일	68KB	
0824_4.txt	2020-09-07 AM 09:47	텍스트 문서	1KB	
0824_5.jpg	2020-08-26 PM 02:12	JPG 파일	68KB	
0824_5.txt	2020-09-07 AM 09:54	텍스트 문서	1KB	
0824_6.jpg	2020-08-26 PM 02:12	JPG 파일	78KB	
0824_6.txt	2020-09-07 AM 09:55	텍스트 문서	1KB	
0824_7.jpg	2020-08-26 PM 02:12	JPG 파일	79KB	
0824_7.txt	2020-09-07 AM 09:55	텍스트 문서	1KB	
0824_8.jpg	2020-08-26 PM 02:12	JPG 파일	78KB	
0824_8.txt	2020-09-07 AM 09:56	텍스트 문서	1KB	
0824_9.jpg	2020-08-26 PM 02:12	JPG 파일	77KB	
0824_9.txt	2020-09-07 AM 09:57	텍스트 문서	1KB	
0824_10.jpg	2020-08-26 PM 02:12	JPG 파일	78KB	
0824_10.txt	2020-09-07 AM 10:01	텍스트 문서	1KB	
0824_11.jpg	2020-08-26 PM 02:12	JPG 파일	78KB	
0824_11.txt	2020-09-07 AM 10:02	텍스트 문서	1KB	



## 2. 객체 탐지 실습

### ■ 이미지 학습 방법

- 학습을 위해서는 yolo폴더의 train.py 파일을 사용함

■ Anaconda Powershell Prompt (Anaconda3)

C:\Users\WKYUWONDOH\Documents\yolov5-master\python train.py -data tunadata.yaml -cfg yolov5s.yaml -weights yolov5s.pt -batch-size 24

■ Anaconda Powershell Prompt (Anaconda3)

```
github: skipping check (not a git repository), for updates see https://github.com/ultralytics/yolov5
YOLOv5 2021-6-15 torch 1.9.0+cpu CPU

Namespace(adam=False, artifact_alias='latest', batch_size=24, bbox_interval=-1, bucket='', cache_images=False, cfg='models\yolov5s.yaml', data='data\tunadata.yaml', device='', entity=None, epochs=300, evolve=False, exist_ok=False, global_rank=-1, hyp='data\hyp.scratch.yaml', image_weights=False, img_size=[640, 640], label_smoothing=0.0, linear_lr=False, local_rank=-1, multi_scale=False, name='exp', noautoanchor=False, nosave=False, notest=False, project='runs/train', quad=False, rect=False, resume=False, save_dir='runs\tain\exp13', save_period=1, single_cls=False, sync_bn=False, total_batch_size=24, upload_dataset=False, weights='', workers=8, world_size=1)

tensorboard: Start with 'tensorboard --logdir runs/train' view at http://localhost:6006/
2021-06-21 10:45:59.758007: W tensorflow/stream_executor/platform/default/dso_loader.cc:55] Could not load dynamic library 'cudart64_101.dll'; dlerror: cudart64_101.dll not found
2021-06-21 10:45:59.758782: I tensorflow/stream_executor/cuda/cudart_stub.cc:29] Ignore above cudart dlerror if you do not have a GPU set up on your machine.
hyperparameters: lr=0.01, lr_f=0.2, momentum=0.937, weight_decay=0.0005, warmup_epochs=3.0, warmup_momentum=0.8, warmup_bias_lr=0.1, box=0.05, cls=0.5, cls_pw=1.0, obj=1.0, obj_pw=1.0, iou_t=0.2, anchor_t=4.0, fl_gamma=0.0, hsv_h=0.015, hsv_s=0.7, hsv_v=0.4, degrees=0.0, translate=0.1, scale=0.5, shear=0.0, perspective=0.0, flipud=0.0, flipud=0.5, mosaic=1.0, mixup=0.0
wandb: Install Weights & Biases for YOLOv5 logging with 'pip install wandb' (recommended)
Overriding model.yaml nc=80 with nc=3

   from  n  params module  arguments
   --  --  --  --  --
0      1  3520  models.common.Focus  [3, 32, 3]
1      1  18560  models.common.Conv  [32, 64, 3, 2]
2      1  18816  models.common.C3  [64, 64, 1]
3      1  73984  models.common.Conv  [64, 128, 3, 2]
4      1  156928  models.common.C3  [128, 128, 3]
5      1  295424  models.common.Conv  [128, 256, 3, 2]
6      1  625152  models.common.C3  [256, 256, 3]
7      1  1180672  models.common.Conv  [256, 512, 3, 2]
8      1  656896  models.common.SPP  [512, 512, [5, 9, 13]]
9      1  1182720  models.common.C3  [512, 512, 1, False]
10     1  131584  models.common.Conv  [512, 256, 1, 1]
11     1  0  torch.nn.modules.upsampling.Upsample  [None, 2, 'nearest']
12     1  0  models.common.Concat  [1]
13     1  361984  models.common.C3  [512, 256, 1, False]
14     1  33024  models.common.Conv  [256, 128, 1, 1]
15     1  0  torch.nn.modules.upsampling.Upsample  [None, 2, 'nearest']
16     1  0  models.common.Concat  [1]
17     1  90880  models.common.C3  [256, 128, 1, False]
18     1  147712  models.common.Conv  [128, 128, 3, 2]
19     1  0  models.common.Concat  [1]
20     1  296448  models.common.C3  [256, 256, 1, False]
21     1  590336  models.common.Conv  [256, 256, 3, 2]
22     1  0  models.common.Concat  [1]
23     1  1182720  models.common.C3  [512, 512, 1, False]
24    [17, 20, 23] 1  21576  models.yolo.Detect  [3, [[10, 13, 16, 30, 33, 23], [30, 61, 62, 45, 59, 119], [116, 90, 156, 198, 373, 326]], [128, 256, 512]]

C:\Python38\lib\site-packages\torch\functional.py:718: UserWarning: Named tensors and all their associated APIs are an experimental feature and subject to change. Please do not use them for anything important until they are released as stable. (Triggered internally at ..\..\core\tensorimpl.h:1156.)
  return torch.max_pool2d(input, kernel_size, stride, padding, dilation, ceil_mode)
Model Summary: 283 layers, 7068936 parameters, 7068936 gradients, 16.4 GFLOPs

Scaled weight_decay = 0.0005625000000000001
Optimizer groups: 62 .bias, 62 conv.weight, 59 other
train: Scanning 'C:\Users\WKYUWONDOH\Documents\yolov5-master\train\tuna.cache' images and labels... 50 found, 0 missing, 0 empty, 0 corrupted: 100%
val: Scanning 'C:\Users\WKYUWONDOH\Documents\yolov5-master\train\tuna.cache' images and labels... 50 found, 0 missing, 0 empty, 0 corrupted: 100%
Plotting labels...
```

## 2. 객체 탐지 실습

### ■ 학습 진행 화면

#### ■ Anaconda Powershell Prompt (Anaconda3)

```
val: New cache created: C:\Users\WKYUWONDOH\Documents\yolov5-master\val\imagetuna.cache
Plotting labels...

autoanchor: Analyzing anchors... anchors/target = 4.79, Best Possible Recall (BPR) = 1.0000
Image sizes 640 train, 640 test
Using 8 dataloader workers
Logging results to runs\train\exp15
Starting training for 300 epochs...
```

Epoch	gpu_mem	box	obj	cls	total	labels	img_size	
0/299	OG	0.1086	0.06317	0.03821	0.21	48	640: 100%	9/9 [1:10:03<00:00, 467.11s/i]
	Class	Images	Labels	P	R	mAP@.5	mAP@.5: .95:	100%  1/1 [00:06<00:00, 6.87s/it]
	all	30	273	0.0013	0.122	0.00331	0.000542	
1/299	OG	0.1029	0.06689	0.0364	0.2062	57	640: 100%	9/9 [03:45<00:00, 25.06s/it]
	Class	Images	Labels	P	R	mAP@.5	mAP@.5: .95:	100%  1/1 [00:04<00:00, 4.52s/it]
	all	30	273	0.00331	0.122	0.0046	0.00052	
2/299	OG	0.09969	0.06833	0.03425	0.2023	41	640: 100%	9/9 [03:47<00:00, 25.23s/it]
	Class	Images	Labels	P	R	mAP@.5	mAP@.5: .95:	100%  1/1 [00:03<00:00, 3.37s/it]
	all	30	273	0.67	0.118	0.00256	0.000306	
3/299	OG	0.09842	0.0709	0.03292	0.2022	53	640: 100%	9/9 [03:49<00:00, 25.48s/it]
	Class	Images	Labels	P	R	mAP@.5	mAP@.5: .95:	100%  1/1 [00:03<00:00, 3.28s/it]
	all	30	273	0.67	0.118	0.00244	0.00034	
4/299	OG	0.09689	0.06997	0.0317	0.1986	31	640: 100%	9/9 [03:49<00:00, 25.51s/it]
	Class	Images	Labels	P	R	mAP@.5	mAP@.5: .95:	100%  1/1 [00:03<00:00, 3.29s/it]
	all	30	273	0.67	0.111	0.00215	0.000279	
5/299	OG	0.09678	0.073	0.03084	0.2006	57	640: 100%	9/9 [03:49<00:00, 25.46s/it]
	Class	Images	Labels	P	R	mAP@.5	mAP@.5: .95:	100%  1/1 [00:03<00:00, 3.68s/it]

- 1번 학습이(epoch) 끝날 때 마다 생성된 weight 파일을 이용하여 앞서 설정한 yaml 파일에 설정된 validation 경로의 이미지 파일을 입력으로 추론한 결과와 실제 정답(labeling)과의 오차를 확인함
- 학습이 끝난 후 이 오차가 가장 작은(실제 이미지상의 물체와 가장 근접하게 추정) weight 파일을 best.pt 파일로 저장함

## 2. 객체 탐지 실습

### ■ 학습 진행 화면

- 매 epoch 마다 표시되는 내용의 뜻
- 훈련지표
  - Epoch – 현재 학습의 epoch 진행 상황
  - Gpu\_mem – 그래픽 카드의 메모리 할당량. 그래픽 카드가 아닌 CPU로 학습할 시 0G로 표시됨
  - Box,obj,cls,total – yolo가 계산하는 3가지 손실 값, total 은 3가지 손실 값의 합
    - 학습 과정은 이 손실 값을 최소화하는 것
  - Img\_size : 훈련용 데이터셋은 다양한 해상도의 사진이 모여 있을 수 있음 학습을 진행 할 때는 이미지가 해상도가 같아야 함 학습 진행 할 때는 (640x640) 으로 조정됨
- 검증지표
  - Class, Images, Labels로 시작하는 줄은 epoch마다 학습이 끝난 시점에서 진행하는 검증단계임
  - Class – all 이라고 되어있는데 전체 객체를 이용하여 검증한다는 것
  - Images – 검증에 사용된 이미지 개수
  - Labels – 검증에 사용된 라벨 개수
  - P - Precision의 약자, 특정 클래스를 예측하였을 때 그 예측이 맞는 비율을 나타냄
  - R – Recall의 약자, 실제 존재하는 클래스 중에 올바르게 예측한 비율을 나타내는 지표
  - Precision과 Recall은 서로 반비례함

## 2. 객체 탐지 실습

### ■ 학습 진행 화면

- [mAP@.5](#) / [mAP@.5:.95](#) : mAP 는 mean Average Precision의 약자임
  - [mAP@.5](#) 실제 라벨링된 구간의 50%이상이 겹치는 탐지구간만 포함해서 Precision과 Recall 을 분석한 결과
  - [mAP@.5:.95](#) 50%에서 95%의 겹치는 구간을 이용해서 Precision과 Recall을 분석한 결과
  - 둘 다 최댓값이 1이고, 높을수록 좋은 모델임
- 학습 과정 중에 참고용 지표를 표시해줌으로써 진행 상황을 나타내줌

## 2. 객체 탐지 실습

- 전체 학습 횟수 : 300 회
- 학습이 끝난 weight 파일은 yolov5 설치한 디렉토리 아래 runs/train/exp\*/weights/ 에 생성됨

```
Anaconda Powershell Prompt (Anaconda3)
294/299   OG  0.02377  0.02634  0.0007381  0.05085  59  640: 100% | 9/9 [04:15<00:00, 28.43s/it]
          Class Images Labels P R mAP@.5 mAP@.5:.95: 100% | 1/1 [00:03<00:00, 3.48s/it]
          all 30 273 0.737 0.679 0.635 0.336

Epoch    gpu_mem  box  obj  cls  total  labels  img_size
295/299   OG  0.02443  0.02473  0.0008502  0.05001  37  640: 100% | 9/9 [04:24<00:00, 29.34s/it]
          Class Images Labels P R mAP@.5 mAP@.5:.95: 100% | 1/1 [00:03<00:00, 3.63s/it]
          all 30 273 0.772 0.713 0.673 0.347

Epoch    gpu_mem  box  obj  cls  total  labels  img_size
296/299   OG  0.02396  0.02591  0.0007601  0.05063  42  640: 100% | 9/9 [04:26<00:00, 29.64s/it]
          Class Images Labels P R mAP@.5 mAP@.5:.95: 100% | 1/1 [00:03<00:00, 3.73s/it]
          all 30 273 0.762 0.698 0.655 0.337

Epoch    gpu_mem  box  obj  cls  total  labels  img_size
297/299   OG  0.02317  0.02624  0.0007147  0.05012  45  640: 100% | 9/9 [04:23<00:00, 29.23s/it]
          Class Images Labels P R mAP@.5 mAP@.5:.95: 100% | 1/1 [00:03<00:00, 3.51s/it]
          all 30 273 0.7 0.649 0.598 0.32

Epoch    gpu_mem  box  obj  cls  total  labels  img_size
298/299   OG  0.02456  0.02597  0.0008958  0.05143  45  640: 100% | 9/9 [04:26<00:00, 29.58s/it]
          Class Images Labels P R mAP@.5 mAP@.5:.95: 100% | 1/1 [00:03<00:00, 3.61s/it]
          all 30 273 0.726 0.676 0.632 0.337

Epoch    gpu_mem  box  obj  cls  total  labels  img_size
299/299   OG  0.0246  0.02746  0.0008017  0.05286  69  640: 100% | 9/9 [04:26<00:00, 29.62s/it]
          Class Images Labels P R mAP@.5 mAP@.5:.95: 100% | 1/1 [00:03<00:00, 3.65s/it]
          all 30 273 0.792 0.744 0.716 0.358
          mouth 30 96 0.998 1 0.996 0.78
          tail 30 88 0.555 0.455 0.392 0.0837
          tuna 30 89 0.822 0.778 0.76 0.212

300 epochs completed in 20.909 hours.

Optimizer stripped from runs\train\exp15\weights\last.pt, 14.4MB
Optimizer stripped from runs\train\exp15\weights\best.pt, 14.4MB
```

last.pt , best.pt 두개의 weight 파일이 생성됨

두개 파일중에 best.pt 파일이 가장 좋은 성능(검증용 파일에 있는 라벨링된 정답데이터와 비교 하였을 경우 성능이 가장 좋은 weight 값임)을 보인 것이므로 best.pt 파일을 사용하여 객체 탐지시 사용함

## 2. 객체 탐지 실습

### ■ 2.3 객체 인식 실험

#### ■ 실행

- 학습한 weight 파일은 runs/train/exp\*/weights/best.pt 파일을 yolov5 디렉토리(yolov5-master)로 이동함
- 테스트하고자 하는 파일은 yolov5 디렉토리 아래에 inference/images 디렉토리에 이미지 파일을 저장한다
- `python detect.py --weights best.pt --source inference/images`

#### ■ Anaconda Powershell Prompt (Anaconda3)

```
C:\Users\K\YUWONDOH\Documents\yolov5-master>python detect.py --weights best.pt --source inference/tuna_test
Namespace(agnostic_nms=False, augment=False, classes=None, conf_thres=0.25, device='', exist_ok=False, half=False, hide_conf=False, hide_labels=False, imgsz=640, iou_thres=0.45, line_thickness=3, max_det=1000, name='exp', nosave=False, project='runs/detect', save_conf=False, save_crop=False, save_txt=False, source='inference/tuna_test', update=False, view_img=False, weights=['best.pt'])
YOLOv5 2021-6-15 torch 1.9.0+cpu CPU

Fusing layers...
C:\Python38\lib\site-packages\torch\nn\functional.py:718: UserWarning: Named tensors and all their associated APIs are an experimental feature and subject to change. Please do not use them for anything
  important until they are released as stable. (Triggered internally at ..\c10\core\tensorimpl.h:1156.)
  return torch.max_pool2d(input, kernel_size, stride, padding, dilation, ceil_mode)
Model Summary: 224 layers, 7059304 parameters, 0 gradients, 16.3 GFLOPs
image 1/523 C:\Users\K\YUWONDOH\Documents\yolov5-master\inference\tuna_test\20200911_110922.jpg: 640x384 1 mouth, 1 tail, 1 tuna, Done. (0.109s)
image 2/523 C:\Users\K\YUWONDOH\Documents\yolov5-master\inference\tuna_test\20200911_110942.jpg: 640x384 1 mouth, 1 tail, 1 tuna, Done. (0.088s)
image 3/523 C:\Users\K\YUWONDOH\Documents\yolov5-master\inference\tuna_test\20200911_110948.jpg: 640x384 1 mouth, 1 tail, 1 tuna, Done. (0.084s)
image 4/523 C:\Users\K\YUWONDOH\Documents\yolov5-master\inference\tuna_test\20200911_110959.jpg: 640x384 1 mouth, 1 tail, 1 tuna, Done. (0.085s)
image 5/523 C:\Users\K\YUWONDOH\Documents\yolov5-master\inference\tuna_test\20200911_111022.jpg: 640x384 1 mouth, 1 tail, 1 tuna, Done. (0.078s)
image 6/523 C:\Users\K\YUWONDOH\Documents\yolov5-master\inference\tuna_test\20200911_111030.jpg: 640x384 1 mouth, 1 tail, 1 tuna, Done. (0.081s)
image 7/523 C:\Users\K\YUWONDOH\Documents\yolov5-master\inference\tuna_test\20200911_111042.jpg: 640x384 1 mouth, 1 tail, 1 tuna, Done. (0.083s)
image 8/523 C:\Users\K\YUWONDOH\Documents\yolov5-master\inference\tuna_test\20200911_111052.jpg: 640x384 1 mouth, 1 tail, 1 tuna, Done. (0.083s)
image 9/523 C:\Users\K\YUWONDOH\Documents\yolov5-master\inference\tuna_test\20200911_111105.jpg: 640x384 1 mouth, 1 tail, 1 tuna, Done. (0.083s)
image 10/523 C:\Users\K\YUWONDOH\Documents\yolov5-master\inference\tuna_test\20200911_111133.jpg: 640x384 1 mouth, 1 tail, 1 tuna, Done. (0.087s)
image 11/523 C:\Users\K\YUWONDOH\Documents\yolov5-master\inference\tuna_test\20200911_111140.jpg: 640x384 1 mouth, 1 tail, 1 tuna, Done. (0.091s)
image 12/523 C:\Users\K\YUWONDOH\Documents\yolov5-master\inference\tuna_test\20200911_111140.jpg: 640x384 1 mouth, 1 tail, 1 tuna, Done. (0.082s)
image 13/523 C:\Users\K\YUWONDOH\Documents\yolov5-master\inference\tuna_test\20200911_111143.jpg: 640x384 1 mouth, 1 tail, 1 tuna, Done. (0.093s)
image 14/523 C:\Users\K\YUWONDOH\Documents\yolov5-master\inference\tuna_test\20200911_111143.jpg: 640x384 1 mouth, 1 tail, 1 tuna, Done. (0.090s)
image 15/523 C:\Users\K\YUWONDOH\Documents\yolov5-master\inference\tuna_test\20200911_111145.jpg: 640x384 1 mouth, 1 tail, 1 tuna, Done. (0.081s)
image 16/523 C:\Users\K\YUWONDOH\Documents\yolov5-master\inference\tuna_test\20200911_111150.jpg: 640x384 1 mouth, 1 tail, 1 tuna, Done. (0.083s)
```

## 2. 객체 탐지 실습

### ■ 2.3 객체 인식 실험

- 완료되면 "Results saved to runs/detect/exp\*" 결과 이미지가 저장된 디렉토리 표시됨
- detect.py 라는 프로그램을 실행할 때마다 runs/detect/ 아래의 exp\* 번호가 늘어나는 형태로 detection 한 결과 이미지 파일이 저장됨(아래 그림에서 runs/detect/exp13 폴더에 저장됨)

Anaconda Powershell Prompt (Anaconda3)

```
image 508/523 C:\Users\K\YUWONDOH\Documents\yolov5-master\inference\tuna_test\20200911_144050.jpg: 640x384 1 mouth, 2 tails, Done. (0.085s)
image 509/523 C:\Users\K\YUWONDOH\Documents\yolov5-master\inference\tuna_test\20200911_144059.jpg: 640x384 1 mouth, 1 tail, 1 tuna, Done. (0.085s)
image 510/523 C:\Users\K\YUWONDOH\Documents\yolov5-master\inference\tuna_test\20200911_144102.jpg: 640x384 1 mouth, 1 tail, 1 tuna, Done. (0.085s)
image 511/523 C:\Users\K\YUWONDOH\Documents\yolov5-master\inference\tuna_test\20200911_144105.jpg: 640x384 1 mouth, 1 tail, 1 tuna, Done. (0.083s)
image 512/523 C:\Users\K\YUWONDOH\Documents\yolov5-master\inference\tuna_test\20200911_144109.jpg: 640x384 1 mouth, 2 tails, 1 tuna, Done. (0.090s)
image 513/523 C:\Users\K\YUWONDOH\Documents\yolov5-master\inference\tuna_test\20200911_144223.jpg: 640x384 1 mouth, 1 tail, 1 tuna, Done. (0.085s)
image 514/523 C:\Users\K\YUWONDOH\Documents\yolov5-master\inference\tuna_test\20200911_144226.jpg: 640x384 1 mouth, 1 tail, 1 tuna, Done. (0.085s)
image 515/523 C:\Users\K\YUWONDOH\Documents\yolov5-master\inference\tuna_test\20200911_144230.jpg: 640x384 1 mouth, 1 tail, 1 tuna, Done. (0.085s)
image 516/523 C:\Users\K\YUWONDOH\Documents\yolov5-master\inference\tuna_test\20200911_144238.jpg: 640x384 1 mouth, 1 tail, 1 tuna, Done. (0.087s)
image 517/523 C:\Users\K\YUWONDOH\Documents\yolov5-master\inference\tuna_test\20200911_144242.jpg: 640x384 1 mouth, 1 tail, 1 tuna, Done. (0.083s)
image 518/523 C:\Users\K\YUWONDOH\Documents\yolov5-master\inference\tuna_test\20200911_144248.jpg: 640x384 1 mouth, 1 tail, Done. (0.081s)
image 519/523 C:\Users\K\YUWONDOH\Documents\yolov5-master\inference\tuna_test\20200911_144304.jpg: 640x384 2 mouths, 2 tails, 2 tunas, Done. (0.092s)
image 520/523 C:\Users\K\YUWONDOH\Documents\yolov5-master\inference\tuna_test\20200911_144315.jpg: 640x384 2 mouths, 2 tails, 2 tunas, Done. (0.083s)
image 521/523 C:\Users\K\YUWONDOH\Documents\yolov5-master\inference\tuna_test\20200911_144324.jpg: 640x384 2 mouths, 1 tail, 2 tunas, Done. (0.090s)
image 522/523 C:\Users\K\YUWONDOH\Documents\yolov5-master\inference\tuna_test\20200911_144330.jpg: 640x384 2 mouths, 1 tail, 2 tunas, Done. (0.083s)
image 523/523 C:\Users\K\YUWONDOH\Documents\yolov5-master\inference\tuna_test\20200911_144337.jpg: 640x384 2 mouths, 2 tails, 2 tunas, Done. (0.084s)
Results saved to runs\detect\exp13
Done. (149.216s)
```



## 2. 객체 탐지 실습

### ■ 2.3 객체 인식 실험



runs/detect/exp\*  
폴더의 객체 검출 결과 화면

사각형 Box 형태로 검출  
박스 상단에 검출된 물체의 이름(tuna, head, tail)과  
해당물체일 확률값 표시됨



## 2. 객체 탐지 실습

### ■ 2.3 객체 인식 실험

### ■ WebCam 에서 실시간 물체 인식 실험

- Webcam 설치된 PC/notebook 에서 실시간으로 입력되는 영상에 대해서 물체 인식이 가능함

```
$ python detect.py --source OPTION
```

Replace OPTION with your selection, to detect from:

- **Webcam** : (OPTION = 0) *For live object detection from your connected webcam*
- **Image** : (OPTION = filename.jpg) *Create a copy of the image with an object detection overlay*
- **Video** : (OPTION = filename.mp4) *Create a copy of the video with an object detection overlay*
- **Directory** : (OPTION = directory\_name/) *Create a copy of all file with an object detection overlay*

< yolo 공식사이트의 문서 내용중 발췌 <https://docs.ultralytics.com> >

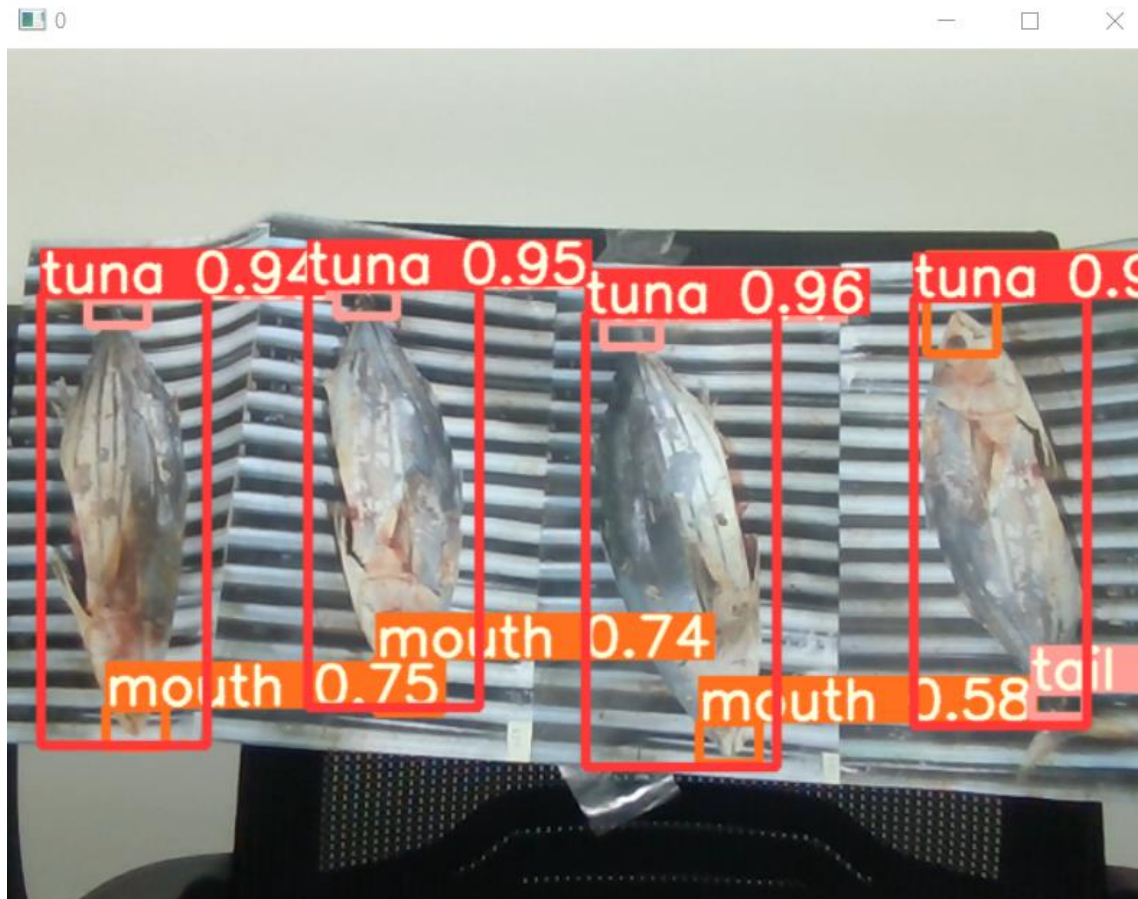
- 앞서 학습한 참치 이미지를 객체인식 테스트하고자 할 경우 Anaconda Powershell prompt 에서(학습한 weight 파일 이름이 best.pt 일 경우)

- `python detect.py --weights best.pt --source 0`

```
Anaconda Powershell Prompt (anaconda3)  
(base) PS C:\Users\WYUWONDOH\Documents\yolov5-master> python --weights best.pt --source 0
```

## 2. 객체 탐지 실습

- 2.3 객체 인식 실험
- Webcam 을 통해 입력되는 실시간 영상에서 물체 검출 화면 예시



## 2. 객체 탐지 실습

### ■ 2.4 라벨링 툴 소개

### ■ 이미지 라벨링 프로그램

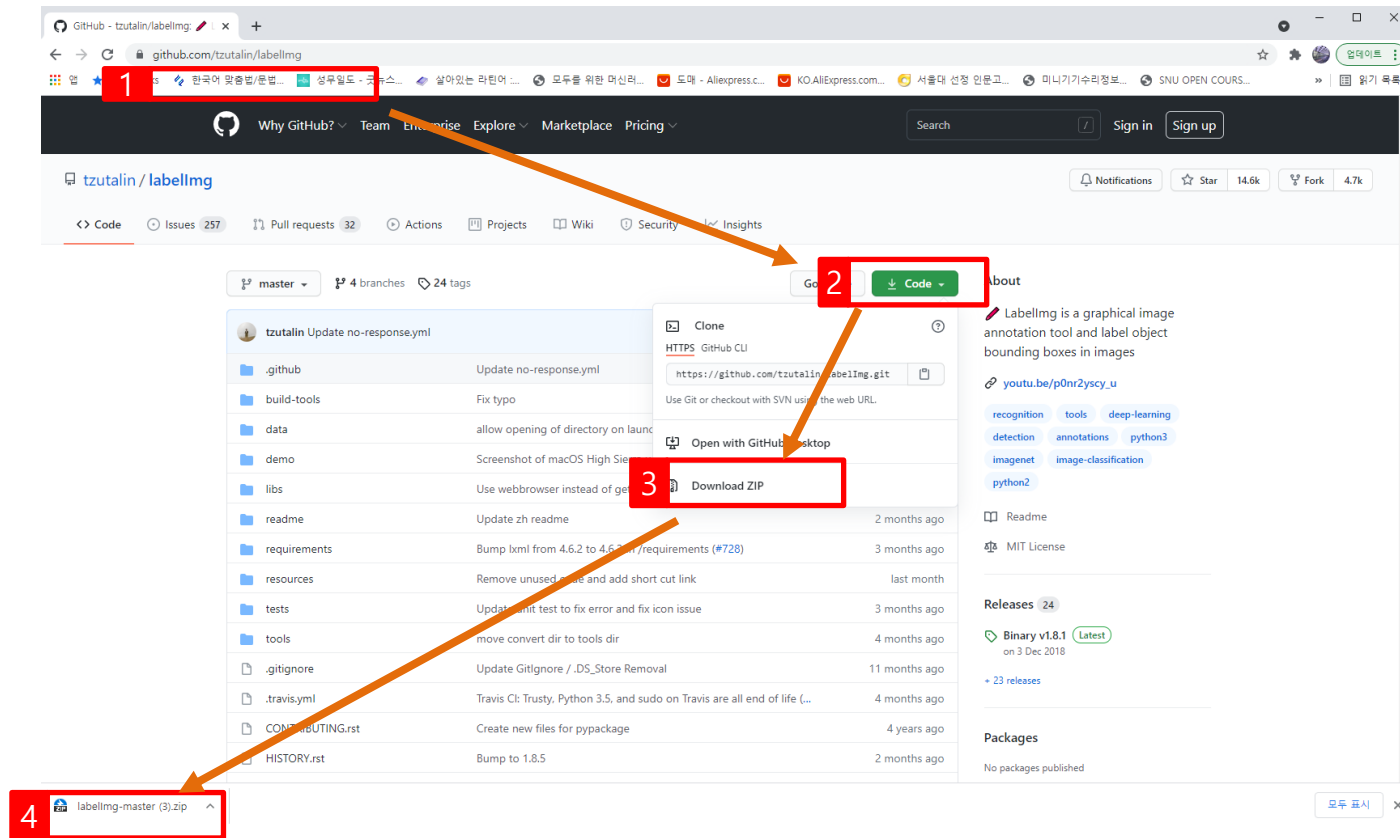
- 이미지상에 검출하고자 하는 대상의 위치 정보를 표시하는 툴
- 모델학습에 사용되는 데이터에 정답을 달아놓는 과정임
- 많은 데이터가 필요하기 때문에, 사람이 직접 라벨링 작업을 수행하는 데 많은 시간이 필요함
- 그런데도 라벨링 작업이 중요한 이유는 좋은 머신러닝 모델이 좋은 결과를 이끄는 만큼 좋은 학습 데이터가 좋은 결과 만들게 됨

## 2. 객체 탐지 실습

### ■ (1) Labellmg 설치

#### ■ 설치방법

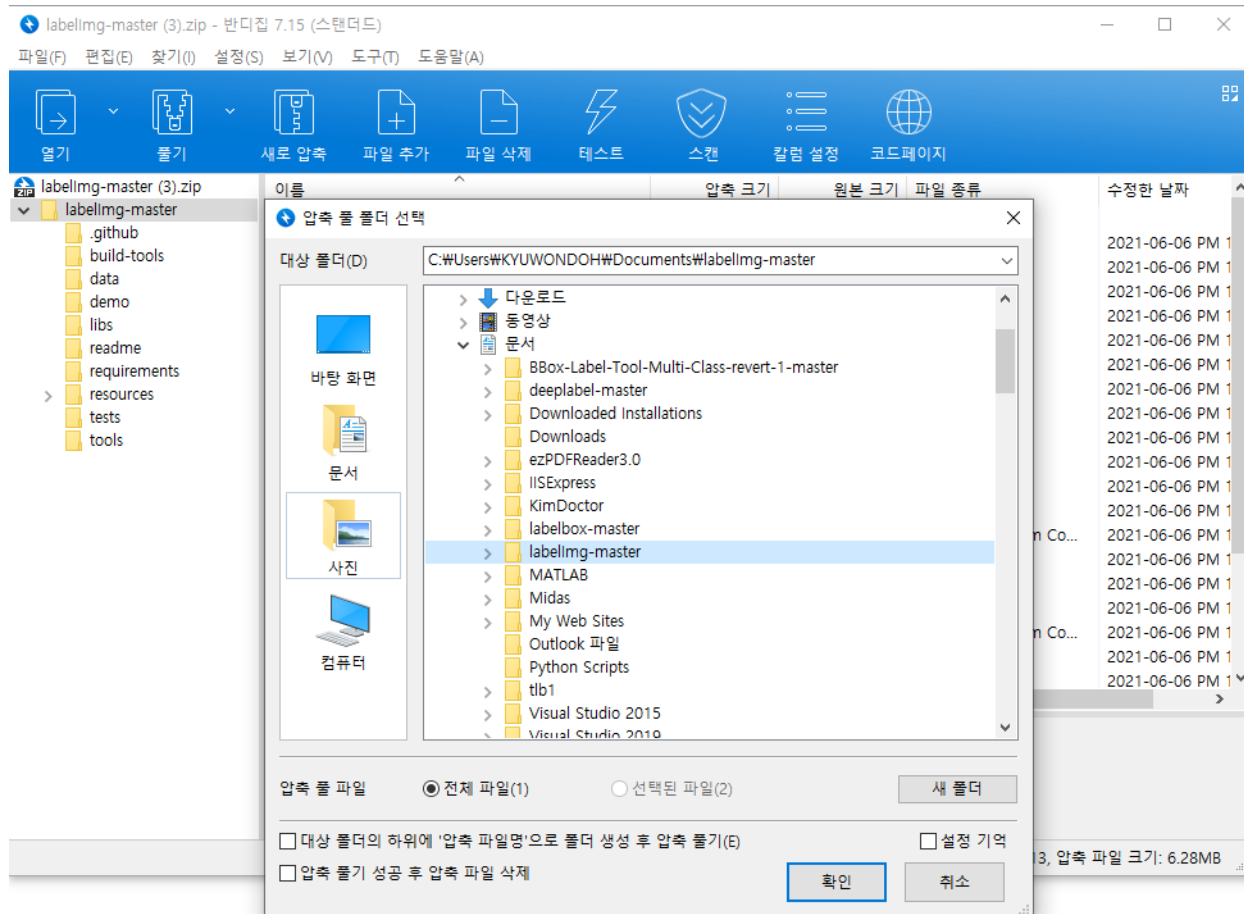
1. <https://github.com/tzutalin/labellmg> 접속
2. 우측 상단 초록색 'code" 버튼 클릭 후



## 2. 객체 탐지 실습

### ■ (1) Labellmg 설치

#### 3. 압축파일 풀기



- 원하는 경로에 Labellmg-master 파일을 저장. 여기서는 문서(Documents) 에 저장

## 2. 객체 탐지 실습

### ■ (1) Labellmg 설치

4. Anaconda Powershell Prompt 실행
5. Labellmg-master 가 설치된 경로로 이동

■ Anaconda Powershell Prompt (anaconda3)

```
(base) PS C:\Users\K\Documents\labellmg-master  
(base) PS C:\Users\K\Documents\labellmg-master>
```

6. 프로그램에 실행시 필요한 파일 설치

```
conda install pyqt=5
```

```
conda install -c anaconda lxml
```

```
pyrcc5 -o libs/resources.py resources.qrc
```

## 2. 객체 탐지 실습

### ■ (1) Labellmg 설치

#### 7. Labellmg-master 가 설치된 경로로 이동

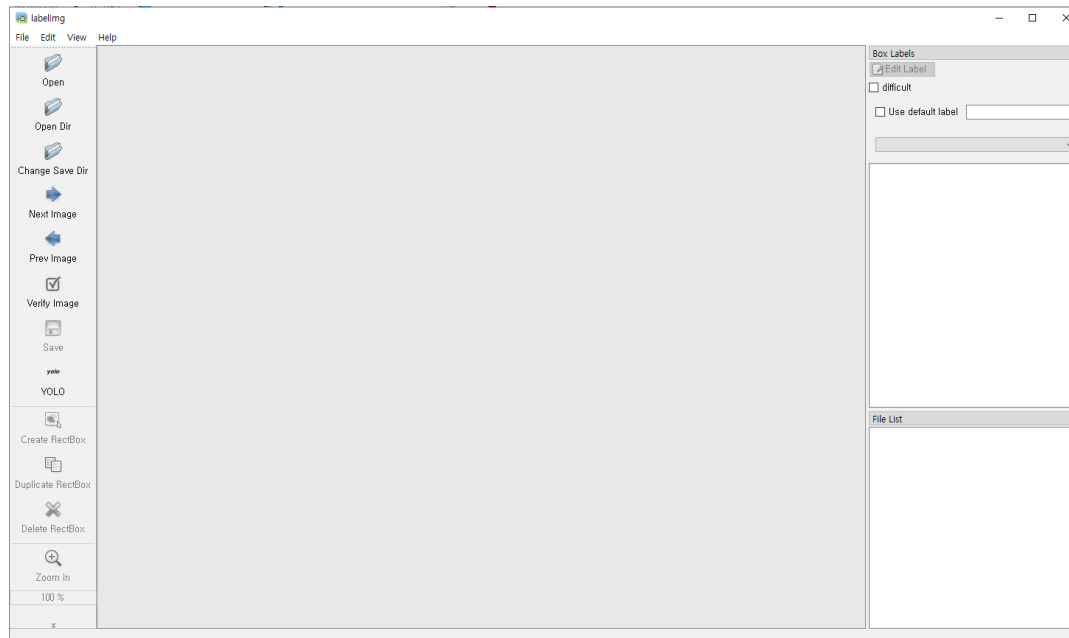
Anaconda Powershell Prompt (anaconda3)

```
(base) PS C:\Users\KYLWONDOH> cd documents\labellmg-master  
(base) PS C:\Users\KYLWONDOH\documents\labellmg-master>
```

#### 8. python labellmg.py 입력

선택 Anaconda Powershell Prompt (anaconda3)

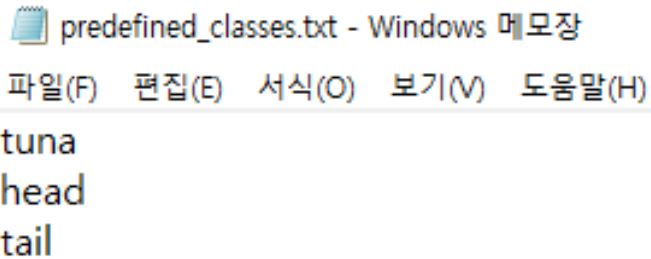
```
(base) PS C:\Users\KYLWONDOH> cd documents\labellmg-master  
(base) PS C:\Users\KYLWONDOH\documents\labellmg-master> python labellmg.py
```



## 2. 객체 탐지 모델 실습

### ■ (2) Labellmg 사용법

- labellmg-master 설치 파일 아래에 data 폴더의 predefined\_classes.txt 파일 열기
- 검출할 객체의 class 명 이 예제에서는 3개의 클래스로 tuna, head, tail 을 입력한다



```
predefined_classes.txt - Windows 메모장
파일(F)  편집(E)  서식(O)  보기(V)  도움말(H)
tuna
head
tail
```

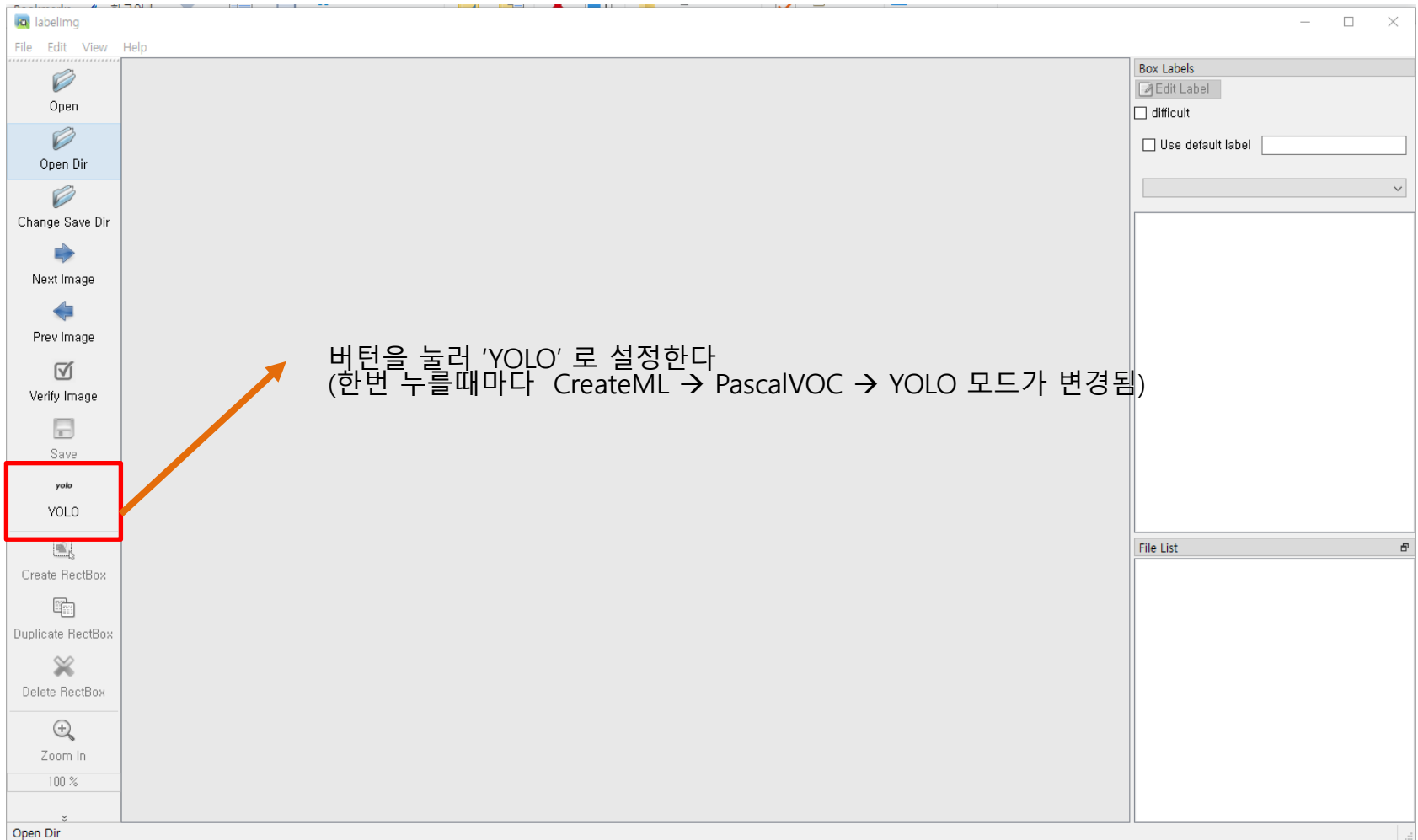
- labellmg 프로그램을 재실행 한다.



## 2. 객체 탐지 실습

### ■ (2) Labellmg 사용법

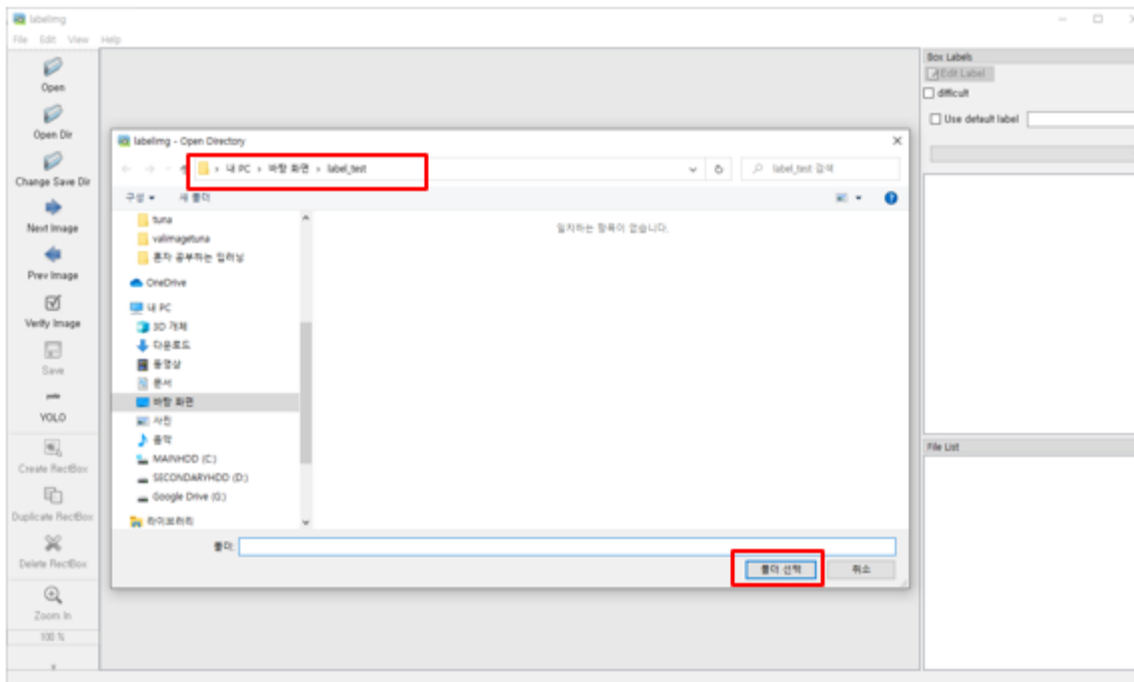
- 왼쪽 중간부분에 YOLO 라고 설정되어 있는지 확인. 다른것으로되어 있으면 클릭하여 YOLO 가 표시될때까지 클릭함



## 2. 객체 탐지 실습

### ■ (2) Labellmg 사용법

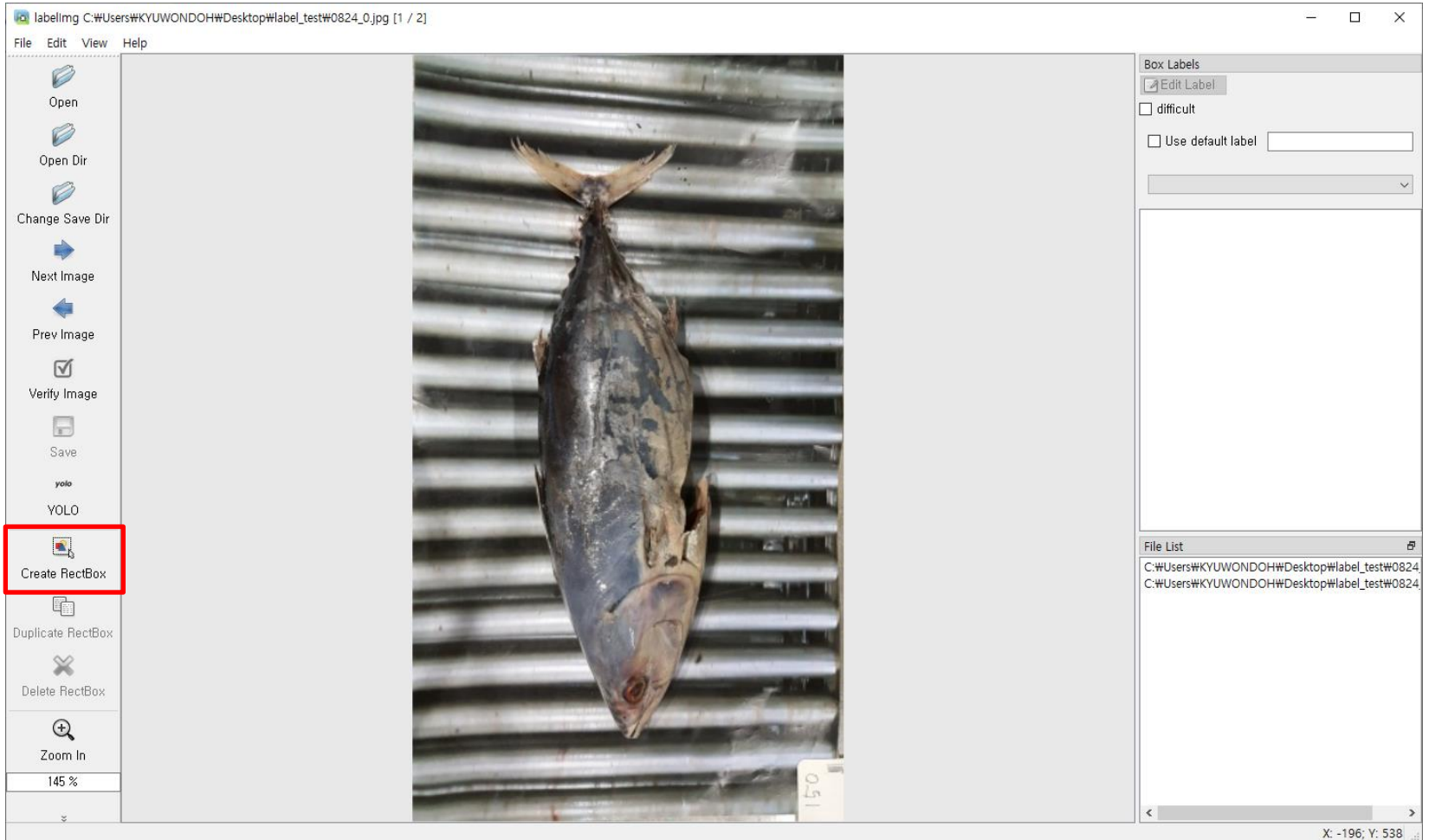
- 왼쪽 상단의 Open Dir 클릭, 라벨링을 하고자 하는 폴더 위치로 이동



## 2. 객체 탐지 실습

### ■ (2) Labellmg 사용법

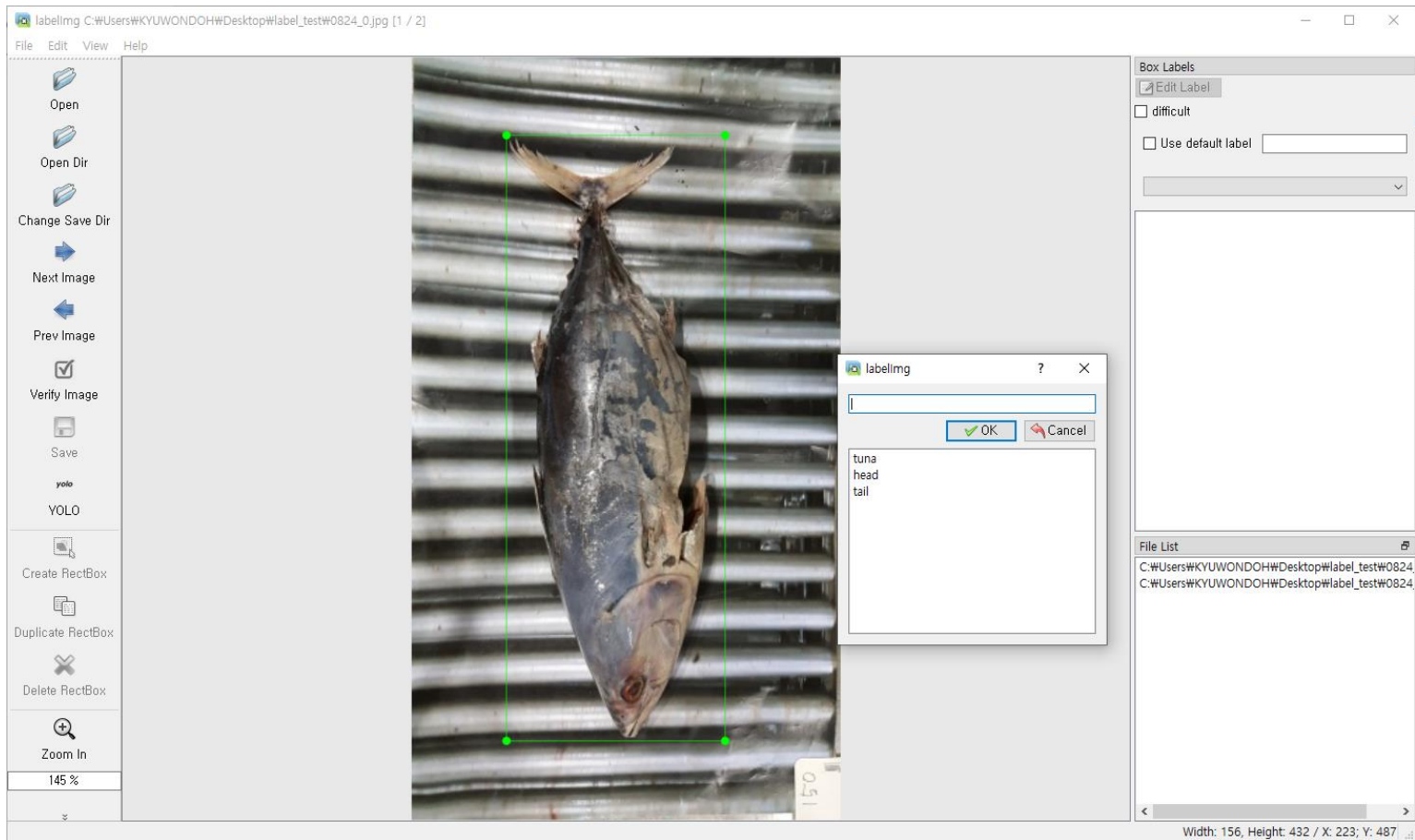
- Create RectBox 클릭



## 2. 객체 탐지 실습

### ■ (2) Labellmg 사용법

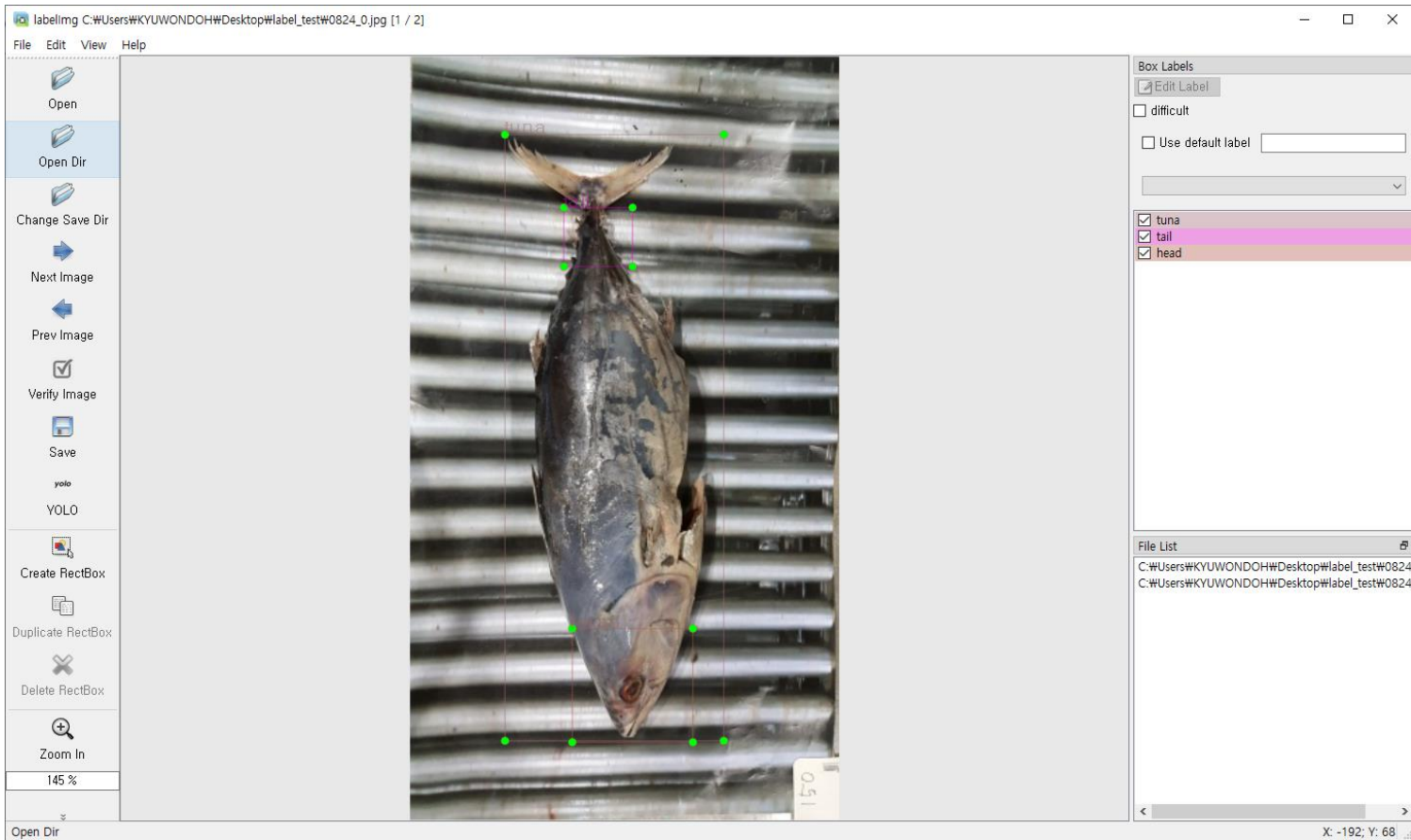
- 좌측마우스로 클릭 후 드래그 하여 영역 지정하면 해당영역을 어떤 class로 지정하는 팝업 창 뜨고 선택하여 OK 버튼 누르면 라벨링 완료됨



## 2. 객체 탐지 실습

### ■ (2) Labellmg 사용법

- 라벨링 후 라벨데이터 수정하려면 오른쪽 중간에 라벨데이터에서 마우스 오른쪽 클릭후 Edit Label 클릭하고 새로운 라벨을 선택함



## 2. 객체 탐지 실습

### ■ (2) Labellmg 사용법

- Save 버튼을 눌러 라벨링 된 결과 파일을 저장함(파일이름은 이미지 파일 이름.txt)

