

YOLOv3 tutorial

WeGo

목 차

1. 딥러닝 준비 사항
2. Custom Dataset 생성
3. YOLOv3 훈련
4. LIMO를 활용한 객체 검출
5. 출처
6. 부록

01

팀러닝 준비 사항

01 딥러닝 준비 사항

- 딥러닝 구현 코드
 - 먼저 딥러닝 모델 및 옵티마이저, 활성화 함수 등 딥러닝이 구현된 코드가 필요합니다.
 - 딥러닝은 C/C++, 파이썬, 매트랩 등 다양한 언어로 구현할 수 있습니다.
 - 본 튜토리얼에서는 미리 구현된 파이썬 코드를 다운로드 받아 사용할 예정입니다.
- 개발 환경
 - Ubuntu 18.04
 - Pytorch v1.4.0
 - CUDA toolkit v10.2

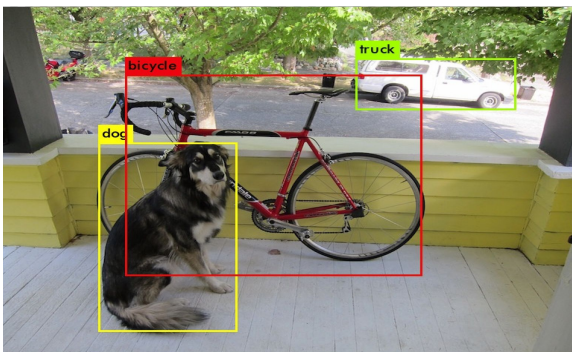
01 딥러닝 준비 사항

- 데이터셋

- 딥러닝 모델을 훈련시키기 위해 학습을 위한 훈련 데이터들이 필요합니다.
- YOLOv3의 경우, 검출할 객체가 있는 사진 + 사진 속 객체의 종류와 좌표가 담긴 라벨

- 데이터셋 획득 경로

- 오픈소스 데이터셋 활용 : 다른 사람이 제작한 데이터셋을 활용
- 커스텀 데이터셋 제작 : 목적에 맞는 데이터셋을 직접 제작
- 본 튜토리얼에서는 ROS와 LIMO를 이용하여 커스텀 데이터셋을 제작할 예정입니다.



| class | x | y | w | h |
|-------|----------|----------|----------|----------|
| 16 | 0.290698 | 0.664823 | 0.239203 | 0.559735 |
| 1 | 0.452658 | 0.480088 | 0.573090 | 0.504425 |
| 7 | 0.754983 | 0.213496 | 0.294020 | 0.174779 |

01 딥러닝 준비 사항

- 커스텀 데이터셋 생성 환경
 - 데이터셋 생성을 위해 LIMO와 ROS를 사용합니다. (LIMO와 ROS 소개는 부록 (p.51)을 참고)
 - PC에 ROS melodic이 설치되어 있다고 가정합니다.
- 커스텀 데이터셋 생성 시 주의사항
 - 커스텀 데이터 속에서 비슷한 사진들은 최소화해주는 것이 좋습니다.
 - 각 객체 종류별 사진의 수가 비슷하도록 유지하는 것이 좋습니다.
 - 하나의 사진에 하나의 객체만 있을 필요는 없습니다.

02

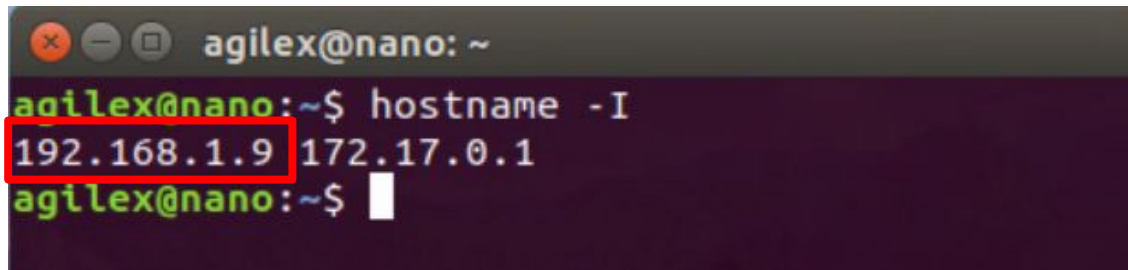
Custom Dataset 생성

02 Custom Dataset 생성

1. LIMO와 PC 통신 환경 세팅

- a. LIMO와 PC를 같은 네트워크 (와이파이)에 접속시킵니다.
- b. LIMO의 터미널 창에 다음 명령어를 입력하여 LIMO의 IP를 확인합니다. (192.168.XX.XX)

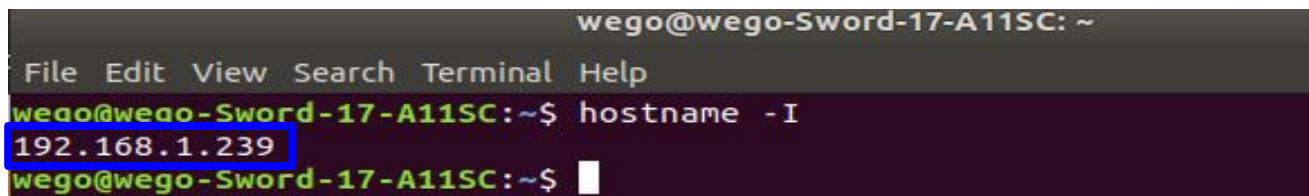
```
$ hostname -I
```



```
agilex@nano: ~  
agilex@nano:~$ hostname -I  
192.168.1.9 172.17.0.1  
agilex@nano:~$
```

- c. PC의 터미널 창에 다음 명령어를 입력하여 PC의 IP를 확인합니다. (192.168.**.**)

```
$ hostname -I
```



```
wego@wego-Sword-17-A11SC: ~  
File Edit View Search Terminal Help  
wego@wego-Sword-17-A11SC:~$ hostname -I  
192.168.1.239  
wego@wego-Sword-17-A11SC:~$
```


02 Custom Dataset 생성

1. LIMO와 PC 통신 환경 세팅

- d. LIMO의 터미널 창에 다음 명령어를 입력하여 LIMO의 .bashrc 파일을 텍스트 편집기로 엽니다.

```
$ gedit ~/.bashrc
```

- e. 다음 내용을 추가하고 저장합니다. (192.168.XX.XX는 LIMO의 IP)

```
export ROS_MASTER_URI=http://192.168.XX.XX:11311
```

```
export ROS_IP=192.168.XX.XX
```

- f. PC의 터미널 창에 다음 명령어를 입력하여 PC의 .bashrc 파일을 텍스트 편집기로 엽니다.

```
$ gedit ~/.bashrc
```

- g. 다음 내용을 추가하고 저장합니다. (192.168.XX.XX는 LIMO의 IP, 192.168.**.**는 PC의 IP)

```
export ROS_MASTER_URI=http://192.168.XX.XX:11311
```

```
export ROS_IP=192.168.**.**
```

02 Custom Dataset 생성

1. LIMO와 PC 통신 환경 세팅

```
agilex@nano: ~  
agilex@nano:~$ hostname -I  
192.168.1.9 172.17.0.1  
agilex@nano:~$ gedit ~/.bashrc  
[.bashrc (~/) - gedit]  
[Open] [Add] .bashrc  
[128]  
[129] export ROS_MASTER_URI=http://192.168.1.9:11311  
[130] export ROS_IP=192.168.1.9  
[131]  
[132]  
sh Tab Width: 4 Ln 109, Col 69
```

LIMO

```
wego@wego-Sword-17-A115  
File Edit View Search Terminal Help  
wego@wego-Sword-17-A115C:~$ hostname -I  
192.168.1.239  
wego@wego-Sword-17-A115C:~$ gedit ~/.bashrc  
[Open] [Add] *.bashrc [Save] [Menu]  
[export] ROS_IP=192.168.1.239  
[export] ROS_MASTER_URI=http://192.168.1.9:11311  
[  
sh Tab Width: 8 Ln 135, Col 1 INS
```

PC

02 Custom Dataset 생성

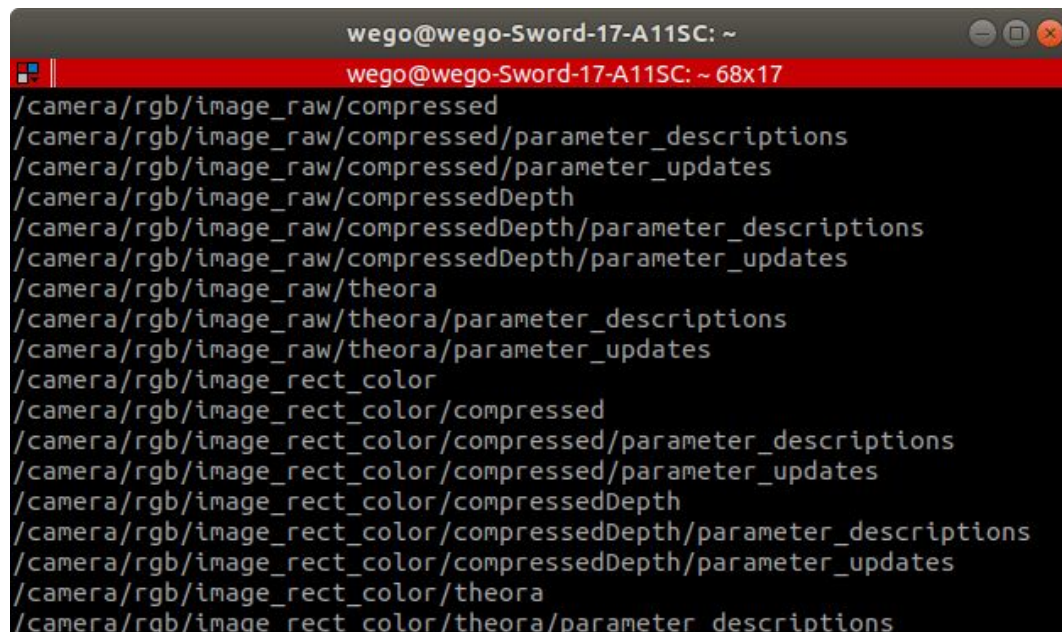
2. LIMO를 이용한 YOLOv3 훈련용 영상 녹화

- a. LIMO의 터미널 창에 다음 명령어를 입력하여 카메라를 실행합니다.

```
$ roslaunch astra_camera dabai_u3.launch
```

- b. PC의 터미널 창을 실행시킨 뒤, 다음 명령어를 입력하여 발행되고 있는 topic을 확인합니다.

```
$ rostopic list
```

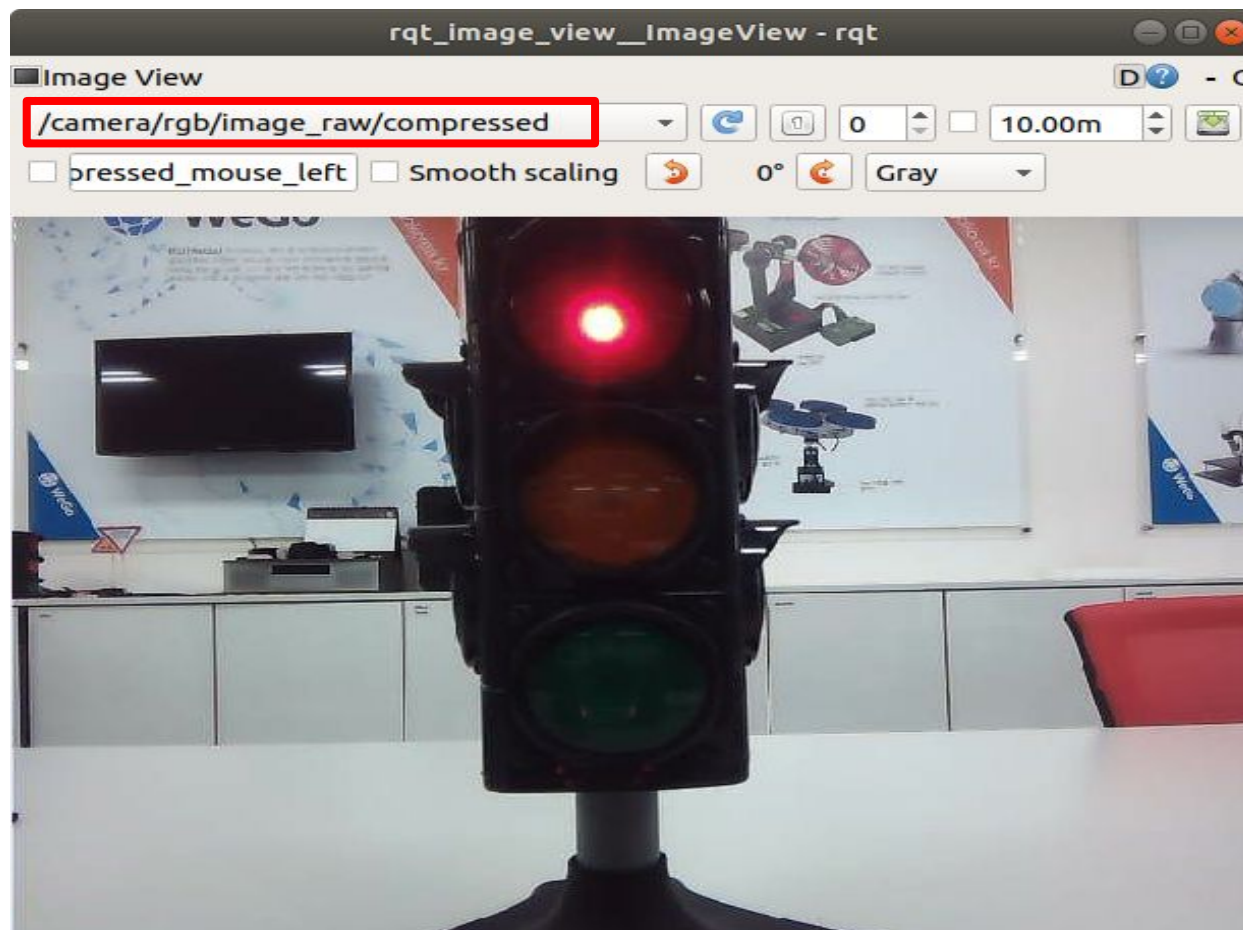


```
wego@wego-Sword-17-A115C: ~  
wego@wego-Sword-17-A115C: ~ 68x17  
/camera/rgb/image_raw/compressed  
/camera/rgb/image_raw/compressed/parameter_descriptions  
/camera/rgb/image_raw/compressed/parameter_updates  
/camera/rgb/image_raw/compressedDepth  
/camera/rgb/image_raw/compressedDepth/parameter_descriptions  
/camera/rgb/image_raw/compressedDepth/parameter_updates  
/camera/rgb/image_raw/theora  
/camera/rgb/image_raw/theora/parameter_descriptions  
/camera/rgb/image_raw/theora/parameter_updates  
/camera/rgb/image_rect_color  
/camera/rgb/image_rect_color/compressed  
/camera/rgb/image_rect_color/compressed/parameter_descriptions  
/camera/rgb/image_rect_color/compressed/parameter_updates  
/camera/rgb/image_rect_color/compressedDepth  
/camera/rgb/image_rect_color/compressedDepth/parameter_descriptions  
/camera/rgb/image_rect_color/compressedDepth/parameter_updates  
/camera/rgb/image_rect_color/theora  
/camera/rgb/image_rect_color/theora/parameter_descriptions
```

02 Custom Dataset 생성

2. LIMO를 이용한 YOLOv3 훈련용 영상 녹화

c. rqt를 통해 LIMO 카메라 영상을 확인하실 수 있습니다.



02 Custom Dataset 생성

2. LIMO를 이용한 YOLOv3 훈련용 영상 녹화

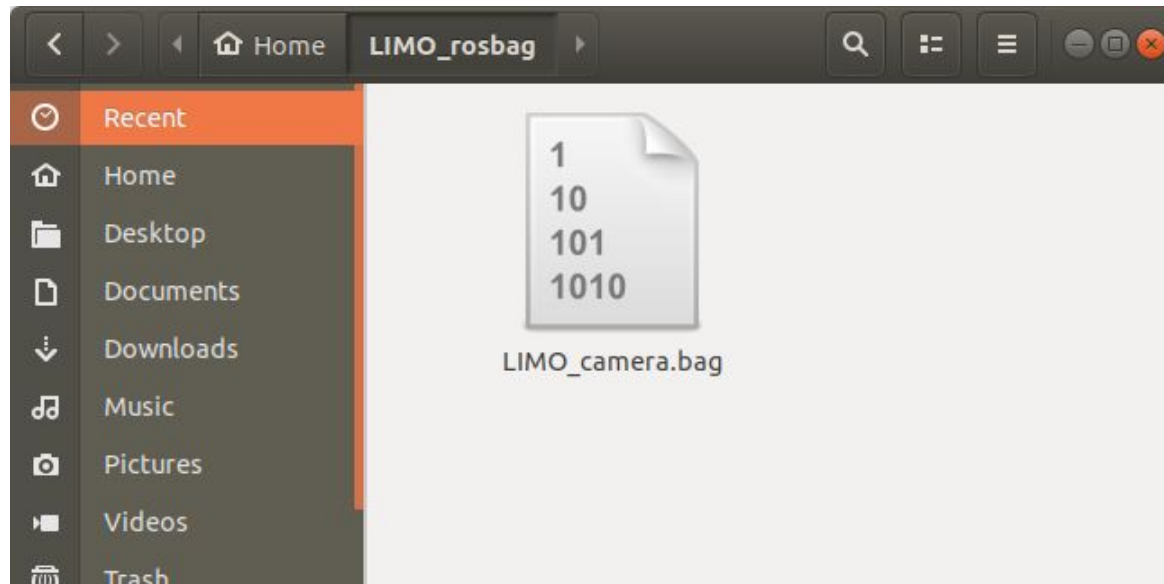
d. rosbag을 통해 카메라 토픽을 녹화합니다.

```
$ rosbag record [camera topic] -o [bag 파일 이름]
```

예시) **/camera/rgb/image_raw/compressed** 토픽 메시지를 녹화해서 **LIMO_camera.bag** 파일에 저장

```
$ rosbag record /camera/rgb/image_raw/compreessed -o LIMO_camera.bag
```

e. LIMO를 주행하며 영상을 녹화 후 저장합니다.



3. bag 파일에서 이미지 추출

- a. 파이썬에서 rosbag 모듈을 사용하여 bag 파일을 불러올 수 있습니다.
- b. `read_messages(topics=['topic'])` 기능을 통해 bag 파일 속 원하는 topic 메시지를 추출할 수 있습니다.
- c. 추출한 카메라 영상 메시지를 `cv_bridge`를 통해 `cv2` 이미지로 변환한 뒤, `cv2.imwrite` 기능을 이용하여 이미지를 저장합니다.

02 Custom Dataset 생성

3. bag 파일에서 이미지 추출

```
import os
import cv2

import rosbag
from cv_bridge import CvBridge

def main():

    outputdir = os.getcwd()+'/images'
    img_topic = "/camera/rgb/image_raw/compressed"

    if not os.path.isdir(outputdir):
        os.makedirs(outputdir)

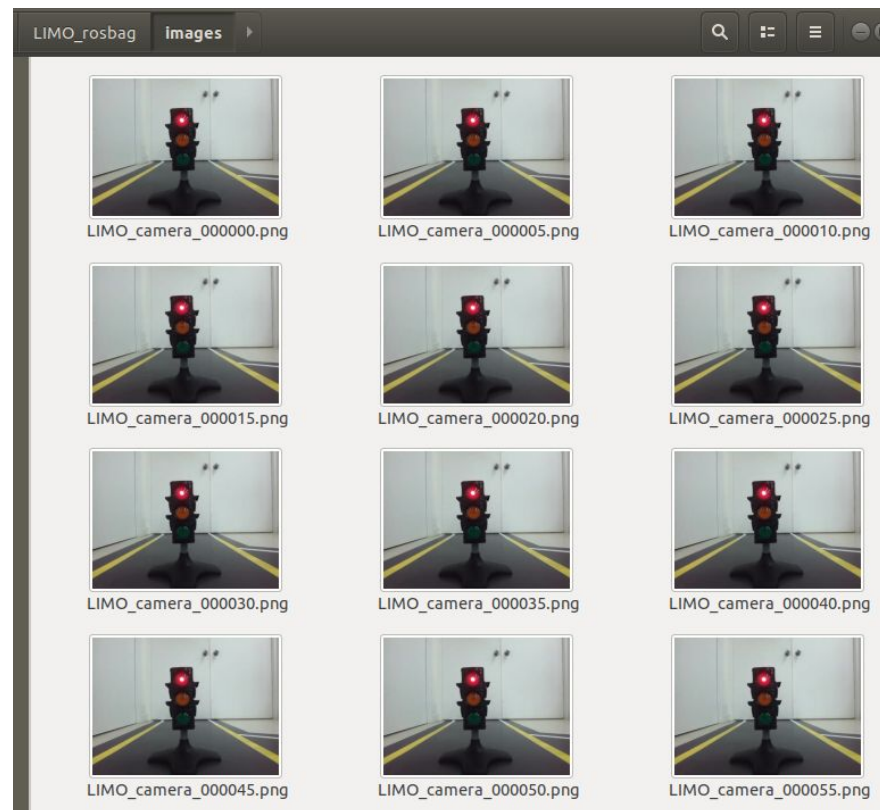
    bagfile = "LIMO_camera.bag"
    basefile = os.path.splitext(os.path.basename(bagfile))[0]
    bag = rosbag.Bag(bagfile, 'r')
    bridge = CvBridge()
    count = 0
    for _, msg, t in bag.read_messages(topics = [img_topic]):
        if count % 5 == 0:
            filename = basefile + "_{:06}.png".format(count)
            img = bridge.compressed_imgmsg_to_cv2(msg)
            cv2.imwrite(os.path.join(outputdir, filename), img)
            print("\tWrote image " + filename)
            count += 1
    bag.close()

if __name__ == '__main__':
    main()
```


02 Custom Dataset 생성

3. bag 파일에서 이미지 추출

```
wego@wego-Sword-17-A11SC:~/LIMO_rosbag$ python bagfile_to_imgs.py
Wrote image LIMO_camera_000000.png
Wrote image LIMO_camera_000005.png
Wrote image LIMO_camera_000010.png
Wrote image LIMO_camera_000015.png
Wrote image LIMO_camera_000020.png
Wrote image LIMO_camera_000025.png
Wrote image LIMO_camera_000030.png
Wrote image LIMO_camera_000035.png
Wrote image LIMO_camera_000040.png
Wrote image LIMO_camera_000045.png
Wrote image LIMO_camera_000050.png
Wrote image LIMO_camera_000055.png
Wrote image LIMO_camera_000060.png
Wrote image LIMO_camera_000065.png
Wrote image LIMO_camera_000070.png
Wrote image LIMO_camera_000075.png
Wrote image LIMO_camera_000080.png
```



02 Custom Dataset 생성

3. bag 파일에서 이미지 추출 시 주의 사항

- a. LIMO의 카메라는 1초당 30장의 사진을 publish합니다. 이 경우, 사진간 촬영 간격이 짧아 비슷한 사진이 과도하게 생성됩니다.
- b. 예시 코드에서 'count % (숫자)' 부분의 숫자를 조정하여 이미지 추출 간격을 조절할 수 있습니다. 예시 코드의 경우 카메라 사진 5장마다 1장의 이미지를 저장합니다.

```
for _, msg, t in bag.read_messages(topics = [img_topic]):  
    if count % 5 == 0:
```

- c. 이미지 추출이 모두 완료된 후에도 사진 상의 객체의 위치와 각도가 비슷한 사진들은 직접 제거해주시는 것이 좋습니다.

02 Custom Dataset 생성

4. 이미지 라벨링

- a. 라벨링은 파이썬 패키지인 `labelImg`를 사용합니다.
- b. 터미널 창에서 다음 명령어를 입력하여 `labelImg` 를 설치합니다.

```
$ sudo apt install python3-pip  
$ python3 -m pip install --upgrade pip  
$ python3 -m pip install labelImg
```

- c. 다음 명령어를 통해 Custom Dataset 디렉토리를 만듭니다.

```
$ cd ~  
$ mkdir -p custom/images custom/labels
```

- d. 터미널 창을 `custom` 디렉토리로 이동하고 `custom.names` 파일을 생성합니다.

```
$ cd custom  
$ gedit custom.names
```

02 Custom Dataset 생성

4. 이미지 라벨링

- e. custom.names 파일에 검출하고 싶은 객체들의 이름을 저장합니다.
 - i. custom.names에 저장하는 객체의 이름은 객체 검출 결과 표시되는 이름입니다. 객체들을 구분할 수 있는 이름을 저장합니다.

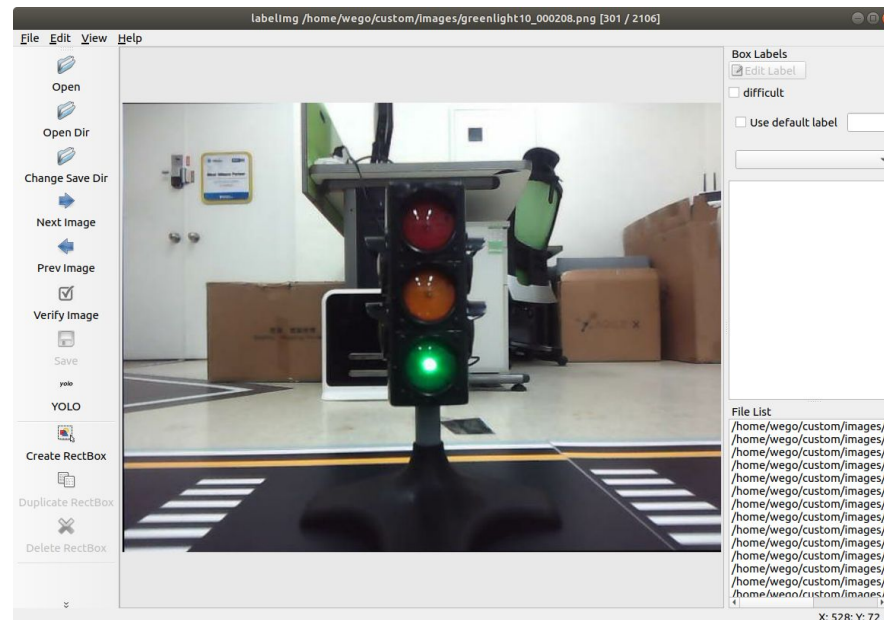


02 Custom Dataset 생성

4. 이미지 라벨링

- f. YOLOv3를 학습시킬 이미지들을 custom/images 디렉토리에 저장합니다.
- g. 터미널 창에서 다음 명령어를 입력하여 labelImg 를 실행합니다. (터미널의 현재 작업 위치가 custom 디렉토리에 위치한 상태에서 실행합니다.)

```
$ labelImg images custom.names
```

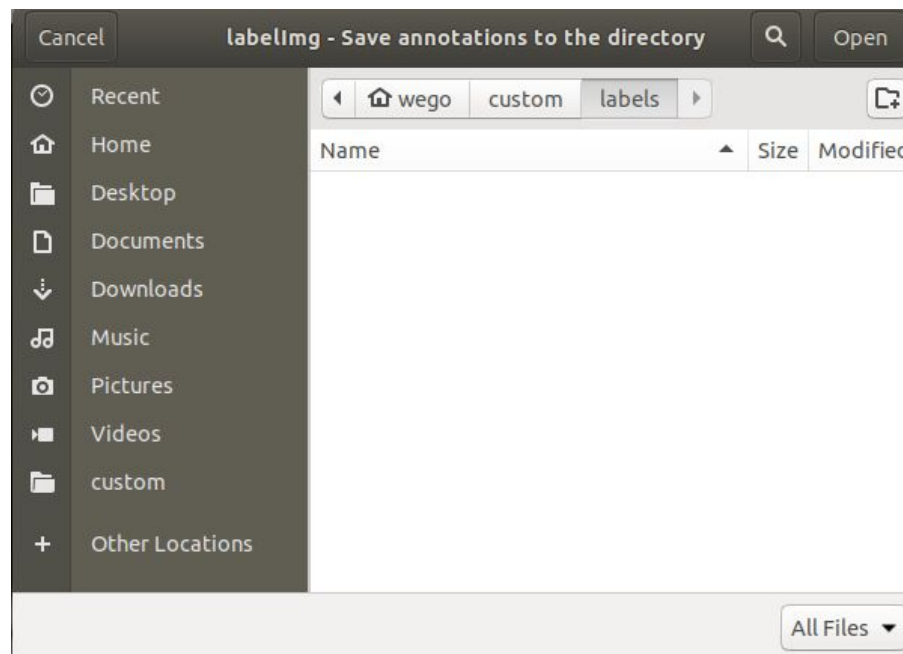
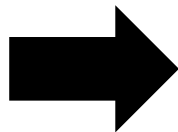
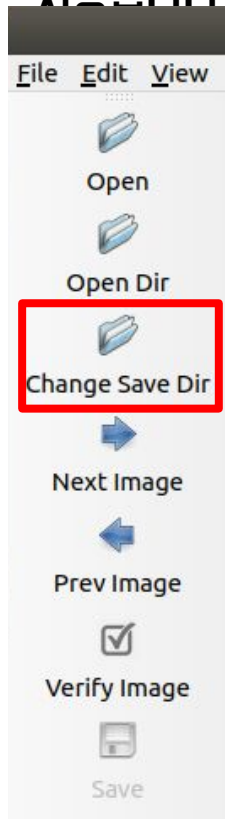


02 Custom Dataset 생성

4. 이미지 라벨링

- h. 왼쪽 메뉴바에서 'Change Save Dir'을 누르고 custom 디렉토리 안의 labels 디렉토리를 열어줍니다. 그러면 생성되는 label 텍스트 파일들이 custom/labels 디렉토리에

저장됩니다



02 Custom Dataset 생성

4. 이미지 라벨링

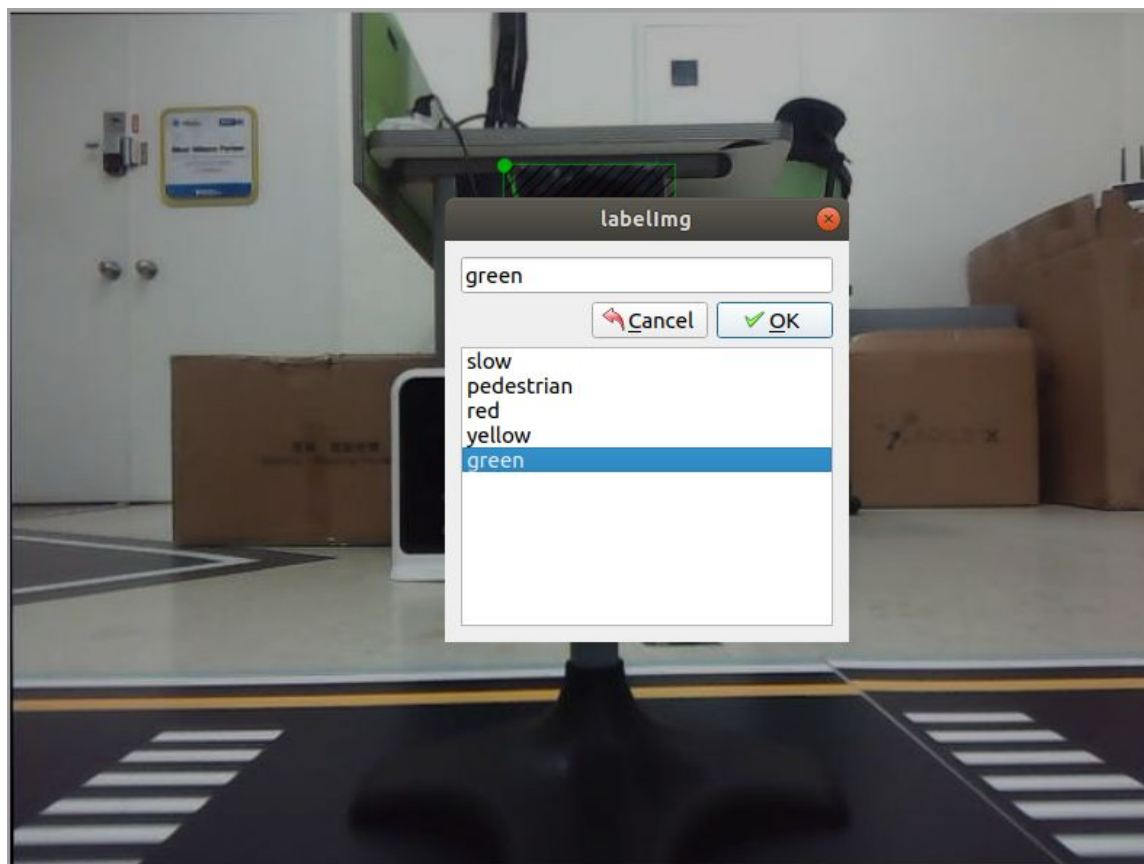
- i. 'w'키를 누르면 사진 위에 드래그하여 상자를 그릴 수 있습니다. 검출하고 싶은 객체의 위치에 최대한 가깝게 상자를 그려줍니다.



02 Custom Dataset 생성

4. 이미지 라벨링

- j. 검출할 객체의 이름을 선택하고 'OK' 버튼을 누릅니다. 이후 'Ctrl+S' 버튼을 누르면 라벨 텍스트 파일이 labels 디렉토리에 저장됩니다.



02 Custom Dataset 생성

4. 이미지 라벨링

- k. 'a' / 'd' 키로 사진을 이전 / 다음 사진으로 넘어갈 수 있습니다.
- l. 한 사진에 검출해야할 객체가 여러 개 있다면 상자를 여러 개 그려주시면 됩니다.



03

YOLOv3 훈련

1. YOLOv3 파이썬 코드 다운로드

a. 터미널 창에 다음 명령어를 실행합니다.

```
$ cd ~
```

```
$ git clone https://github.com/WeGo-Robotics/PyTorch-YOLOv3.git
```

```
$ cd PyTorch-YOLOv3
```

```
$ python -m pip install -r requirements.txt
```

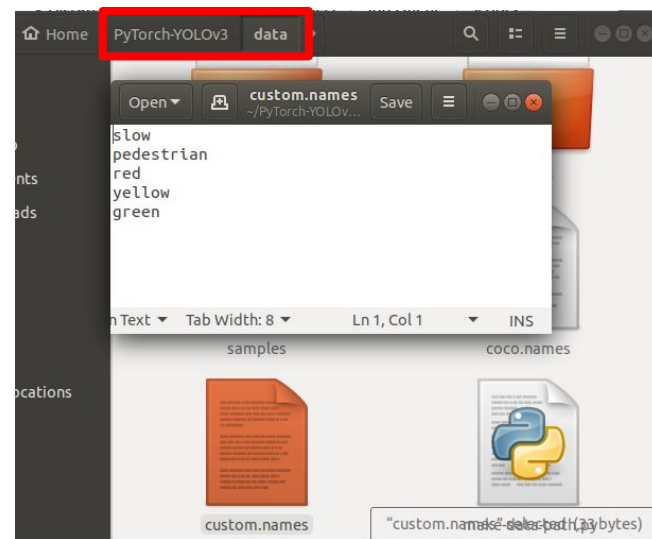
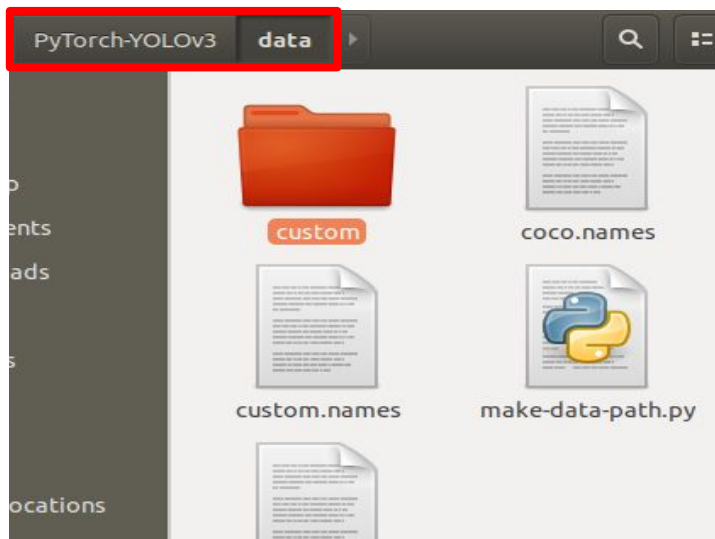
```
$ sh ./weights/download_weights.sh
```

b. 다운로드한 파이썬 코드 구조에 대한 설명은 부록 (p.54)을 참고해주시기 바랍니다.

c. requirements.txt 파일에는 딥러닝 코드 실행을 위한 파이썬 패키지가 정리되어 있습니다.

2. Custom Dataset 이동

- 준비한 Custom Dataset을 Pytorch-YOLOv3/data 디렉토리에 저장합니다.
- custom 디렉토리에 생성한 custom.names 파일을 data 디렉토리에 복사, 붙여넣기합니다.

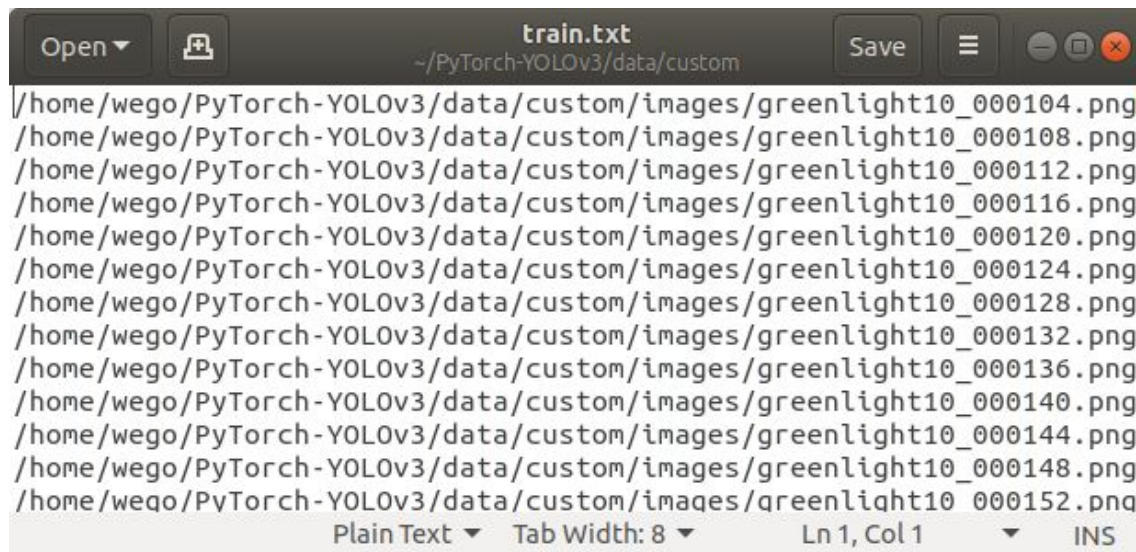
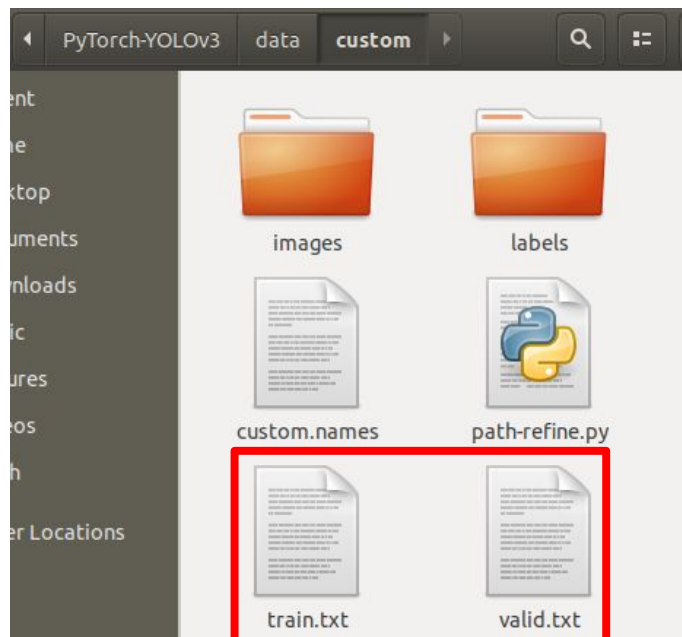


2. Custom Dataset 이동

c. 터미널 창의 현재 작업 위치를 Pytorch-YOLOv3에 위치한 채로 다음 명령어를 입력합니다.

```
$ python data/make-data-path.py
```

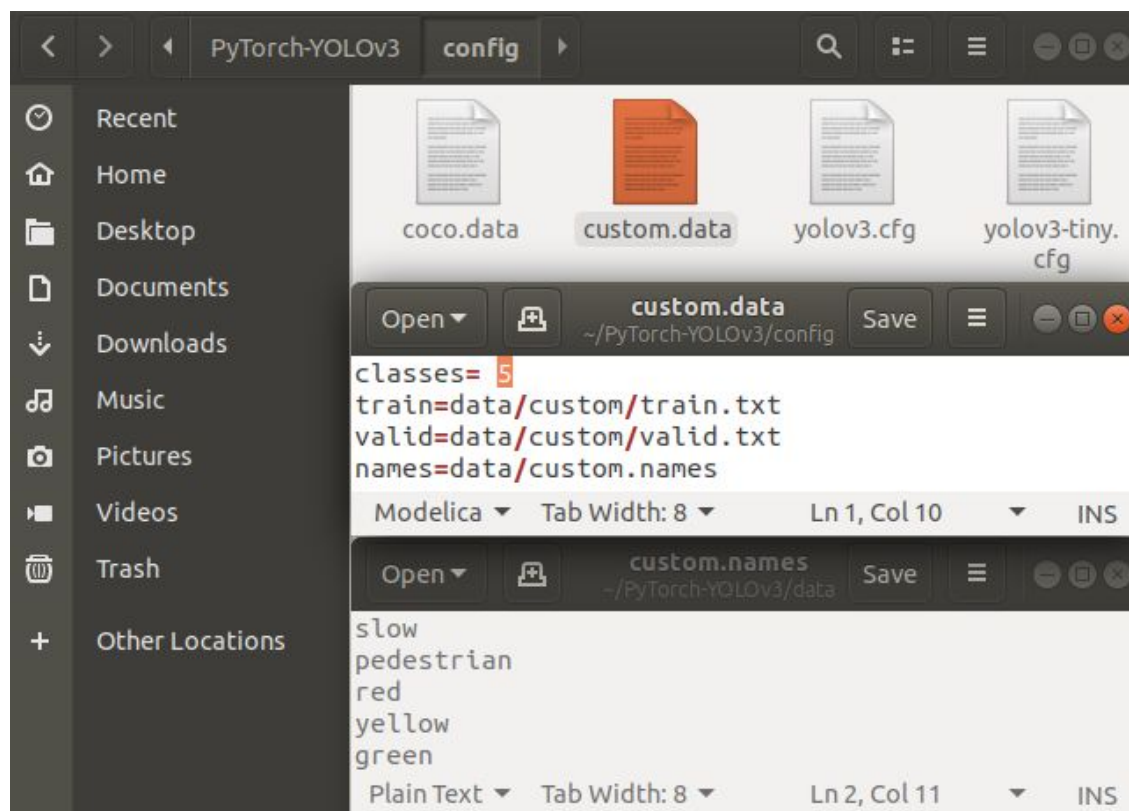
i. 위 명령어는 Custom 디렉토리 내부에 훈련 이미지들의 절대경로가 담긴 train.txt파일과 valid.txt 파일을 생성합니다. 디버깅 코드는 해당 경로를 통해 훈련 데이터의 위치를 파악합니다.



03 YOLOv3 훈련

3. config/custom.data 파일 수정

- config 디렉토리의 custom.data 파일의 classes 부분을 수정해야 합니다.
- data/custom.names 파일 안에 적혀있는 class의 개수를 적어줍니다.



4. GPU 사용량 확인

- GPU를 딥러닝에 사용하는 경우, 딥러닝 모델과 학습데이터가 GPU 메모리에 적재됩니다.
- 다음 명령어를 통해 GPU 실시간 사용량을 파악합니다.

```
$ watch -n 0.5 nvidia-smi
```

```
wego@wego-Sword-17-A11SC: ~  
File Edit View Search Terminal Help  
Every 0.5s: nvidia-smi      wego-Sword-17-A11SC: Wed Apr 27 10:56:34 2022  
Wed Apr 27 10:56:34 2022  
+-----+  
| NVIDIA-SMI 470.103.01    Driver Version: 470.103.01    CUDA Version: 11.4    |  
+-----+  
| GPU   Name                Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |  
| Fan   Temp  Perf    Pwr:Usage/Cap|      Memory-Usage | GPU-Util  Compute M. |  
|=====  
| 0   NVIDIA GeForce ...   Off      00000000:01:00.0 Off |                  N/A |  
| N/A   41C    P8      6W /  N/A      9MiB / 3911MiB      0%      Default  
|=====  
+-----+  
+-----+  
| Processes:                                     GPU Memory  
|  GPU   GI    CI        PID   Type   Process name                  Usage  
|=====  
| 0     N/A   N/A         1094    G   /usr/lib/xorg/Xorg              4MiB  
| 0     N/A   N/A         1990    G   /usr/lib/xorg/Xorg              4MiB  
+-----+
```

GPU 메모리 사용량 GPU 연산량

4. GPU 사용량 확인

- c. 사용할 수 있는 GPU 메모리보다 많은 데이터를 GPU에 적재하려고 하면,
CUDA out of memory 에러가 발생합니다

```
RuntimeError: CUDA out of memory Tried to allocate 20.00 MiB (GPU 0; 3.82 GiB total capacity; 2.68 GiB already allocated; 16.44 MiB free; 2.71 GiB reserved in total by PyTorch)
```

5. batch 사이즈 조정

- a. 터미널의 작업 위치를 Pytorch-YOLOv3에 위치시킨 뒤, 다음 명령어를 입력합니다.

```
$ python yolo/train.py -e 1
```

- b. config 디렉토리의 yolov3-tiny.cfg 파일의 batch=XX의 값을 2배씩 올리면서 정상 실행 여부를 확인합니다.
- c. CUDA out of memory 에러가 발생하면 마지막으로 정상 실행된 값으로 저장해주세요.



```
[net]
# Testing
#batch=1
#subdivisions=1
# Training
batch=32
subdivisions=2
```


6. YOLOv3 훈련

- a. 터미널의 작업 위치를 Pytorch-YOLOv3에 위치시킨 뒤, 다음 명령어를 입력합니다.

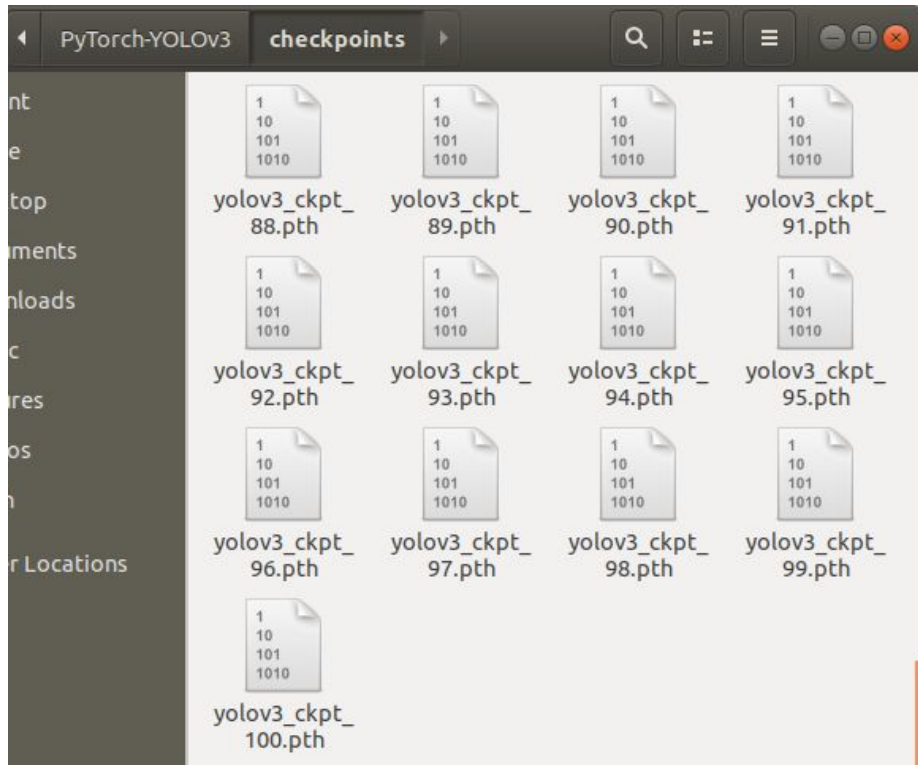
```
$ python yolo/train.py
```

```
$ python yolo/train.py -e (epoch)
```

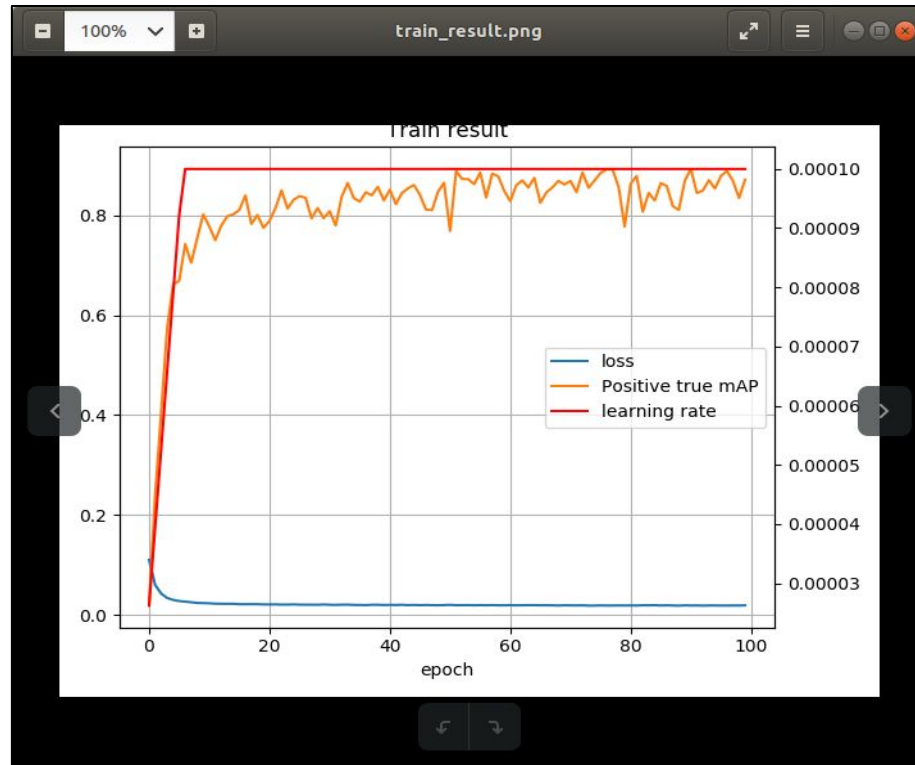
- b. `-e` 옵션으로 에포크를 지정할 수 있습니다. `-e (epoch)` 옵션이 없으면 기본적으로 100 에포크를 훈련시킵니다.
- c. 학습이 종료되고나면 학습 결과 그래프는 `train_results` 디렉토리에, 각 에포크별 모델 가중치는 `checkpoints` 디렉토리에 저장됩니다.

03 YOLOv3 훈련

6. YOLOv3 훈련



checkpoints 디렉토리



train_results 디렉토리

7. YOLOv3 구조 변경

- a. YOLOv3를 커스텀 데이터셋으로 훈련시키는 경우, class 개수에 맞춰 YOLOv3의 구조를 변경시키는 경우가 있습니다. 이 경우 config 디렉토리 속 cfg 파일을 변경해야 합니다.
- b. 본 튜토리얼에서는 딥러닝 모델 설계에 대해 다루지 않았기 때문에 임의 조정시 코드의 실행이 실패할 수도 있습니다.
- c. 또한, 구조를 변경하지 않아도 충분한 학습 결과를 얻을 수 있기때문에 본 튜토리얼에서는 커스텀 데이터셋에 맞춰 YOLOv3의 구조를 변경하는 법에 대해서는 설명드리지 않습니다.

8. YOLOv3 객체 검출

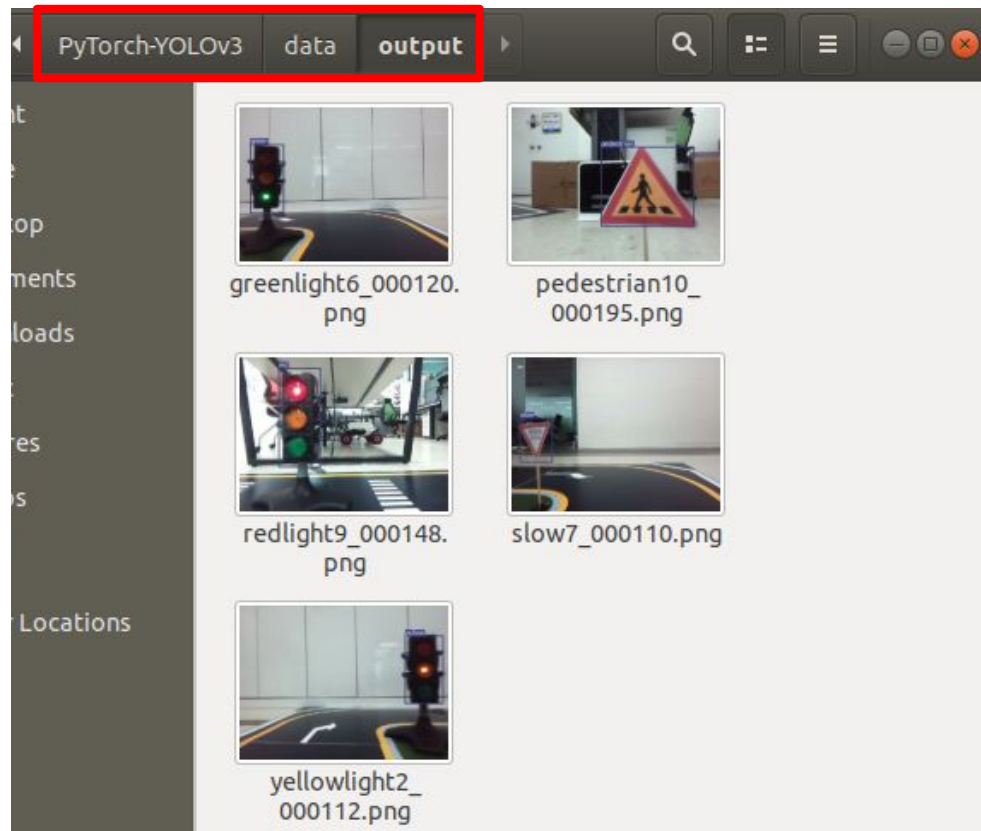
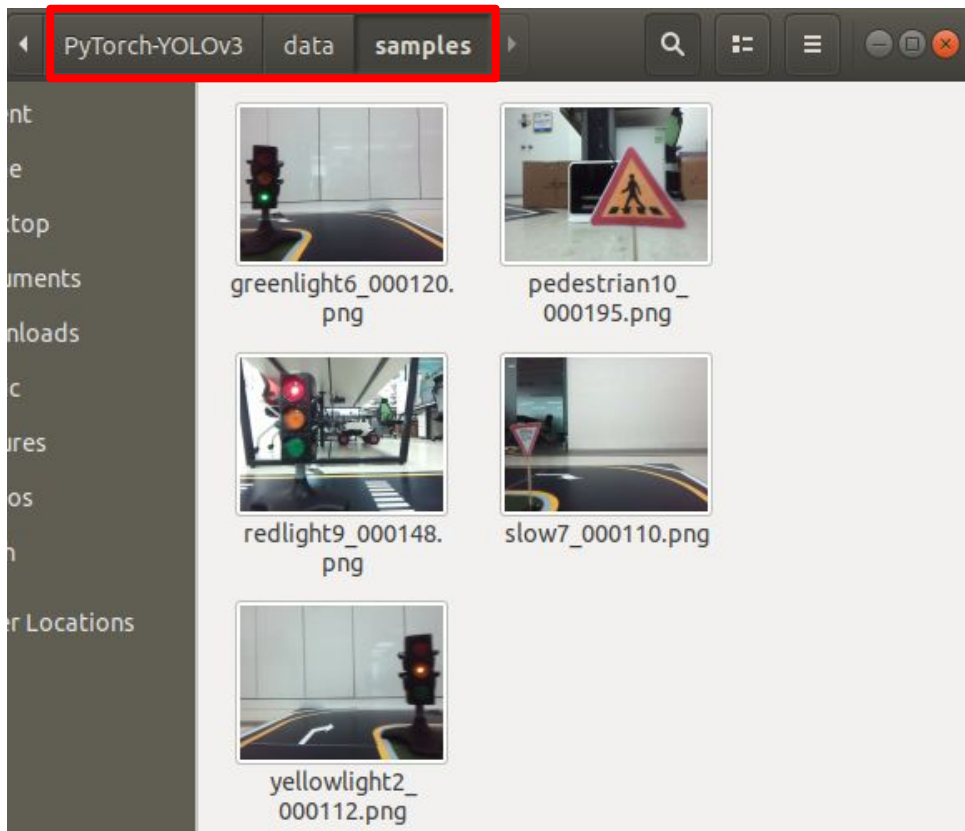
- a. data 디렉토리 안에 samples 디렉토리를 생성합니다.
- b. samples 디렉토리 안에 객체를 검출하고 싶은 사진들을 저장합니다.
- c. 터미널 창의 작업 위치를 Pytorch-YOLOv3에 위치한 채로. 다음 명령어를 실행합니다.

```
$ python yolo/detect.py --weights "checkpoints/yolov3_ckpt_(epoch).pth"
```

- i. "checkpoints/yolov3_ckpt_(epoch).pth"은 객체 검출에 사용하실 가중치 파일의 이름입니다.
- d. 위 코드 실행시 samples 디렉토리에 있는 모든 사진에 대해 객체 검출을 시행합니다.
- e. 객체 검출 결과는 data/output 디렉토리에 저장됩니다.

03 YOLOv3 훈련

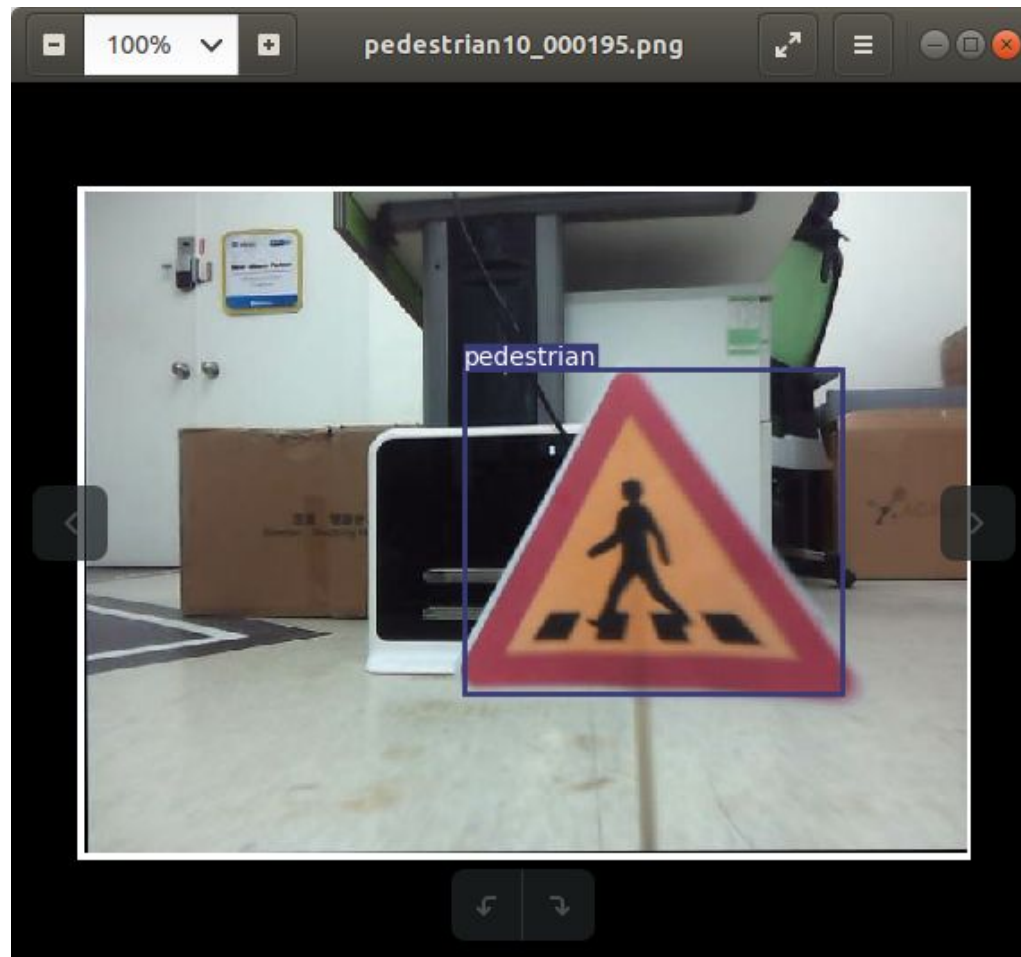
8. YOLOv3 객체 검출



03 YOLOv3 훈련

8. YOLOv3 객체 검출

a. 예시) 객체 검출 결과



9. YOLOv3 객체 검출 실패시 검토사항

a. 데이터의 수가 너무 적은 경우

b. 비슷한 데이터가 너무 많은 경우

i. a, b의 경우 모델이 제대로 학습하지 못한 경우입니다. 데이터의 양을 늘리거나 비슷한 데이터를 줄이고 다른 형태의 데이터를 추가할 필요가 있습니다.

c. 객체 검출 추정치가 낮을 경우

i. YOLOv3는 객체를 검출할 때, 검출한 객체가 정말로 객체인지에 대한 추정치를 계산합니다. 추정치가 특정 기준치 이하라면, 해당 객체를 무시합니다. 기준치를 낮춘다면 객체를 더 잘 검출하지만, 정확도가 낮아집니다.

ii. 아래 명령어를 통해 기준치를 새로 지정합니다. (기본 기준치는 0.7입니다.)

```
$ python yolo/detect.py --weights "checkpoints/yolov3_ckpt_(epoch).pth" --conf_thres (기준치)
```

04

LIMO를 활용한 객체 검출

04 LIMO를 활용한 객체 검출

1. YOLOv3-ROS 코드 다운로드

a. 터미널 창에 다음 명령어를 실행합니다.

```
$ cd ~  
$ mkdir -p yolo_ws/src  
$ cd yolo_ws/src  
$ git clone https://github.com/WeGo-Robotics/yolov3-pytorch-ros.git  
$ cd ..  
$ catkin_make && source devel/setup.bash
```

b. `.bashrc`에 `source ~/yolo_ws/devel/setup.bash` 를 추가합니다.

```
$ gedit ~/.bashrc
```



```
Open ▾ .bashrc Save ≡ - □ ×  
~/  
  
export ROS_IP=192.168.1.239  
export ROS_MASTER_URI=http://192.168.1.9:11311  
  
source yolo_ws/devel/setup.bash  
  
sh ▾ Tab Width: 8 ▾ Ln 134, Col 1 ▾ INS
```

04 LIMO를 활용한 객체 검출

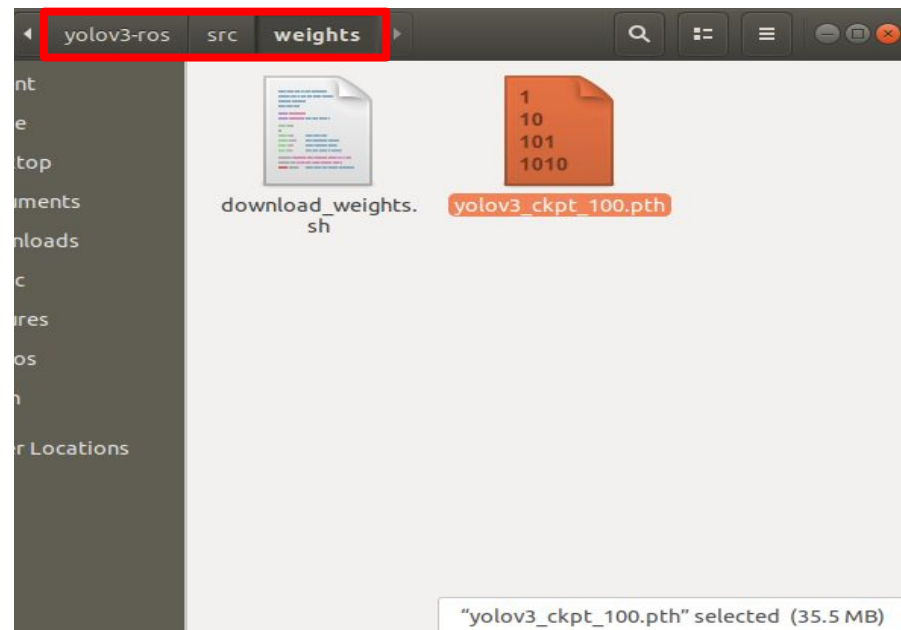
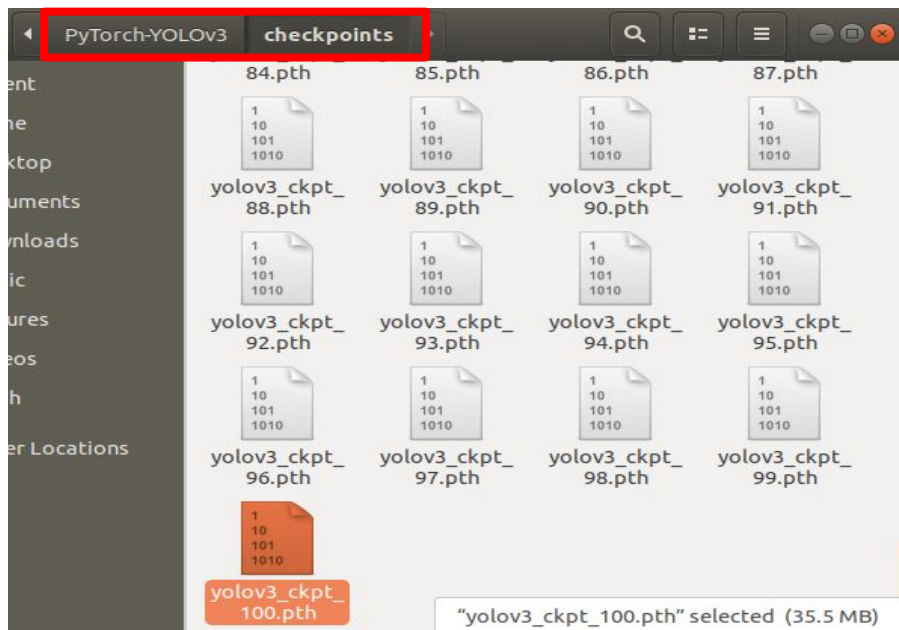
2. YOLOv3 가중치, custom.names 이동

- a. 터미널 창에 다음 명령어를 입력하여 yolov3-ros/src 디렉토리를 GUI로 실행합니다.

```
$ roscd yolov3-ros/src
```

```
$ nautilus .
```

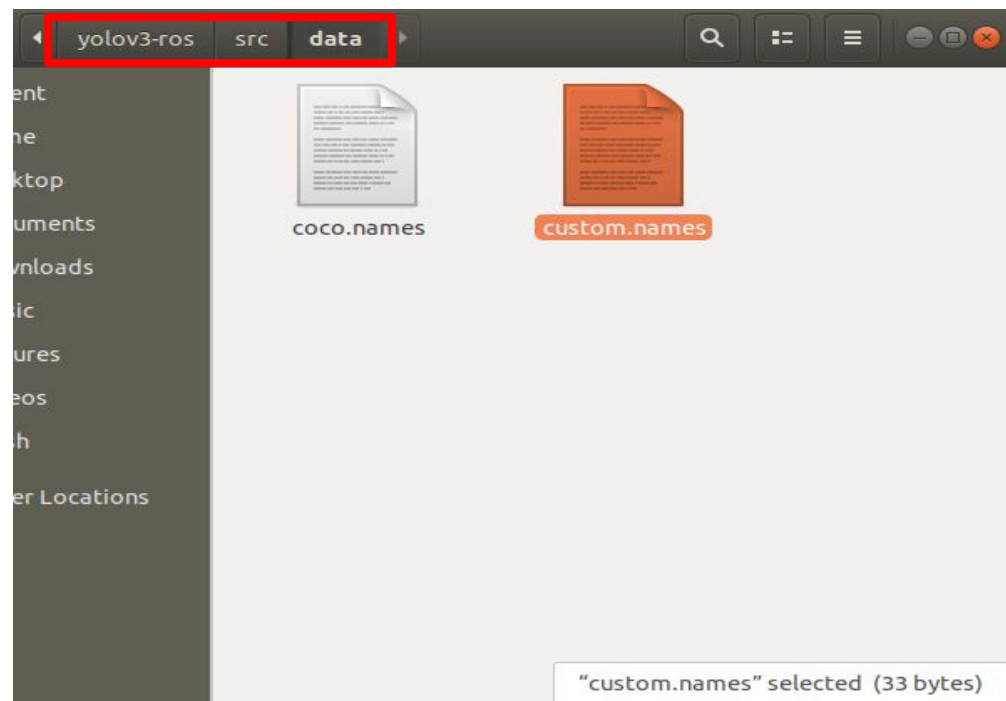
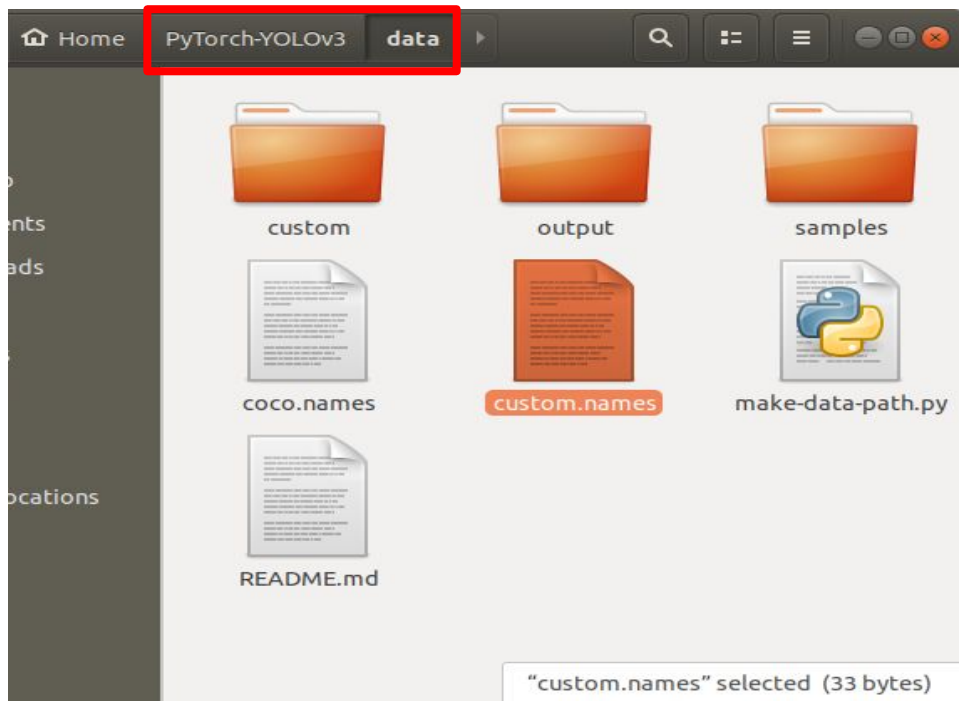
- b. src/weights 디렉토리에 훈련시킨 YOLOv3 가중치 파일을 저장합니다.



04 LIMON을 활용한 객체 검출

2. YOLOv3 가중치, custom.names 이동

c. src/data 디렉토리에 custom.name 파일을 저장합니다.



3. launch 파일 수정

- a. 터미널 창에 다음 명령어를 실행하여 launch 파일의 텍스트 편집기를 실행합니다.

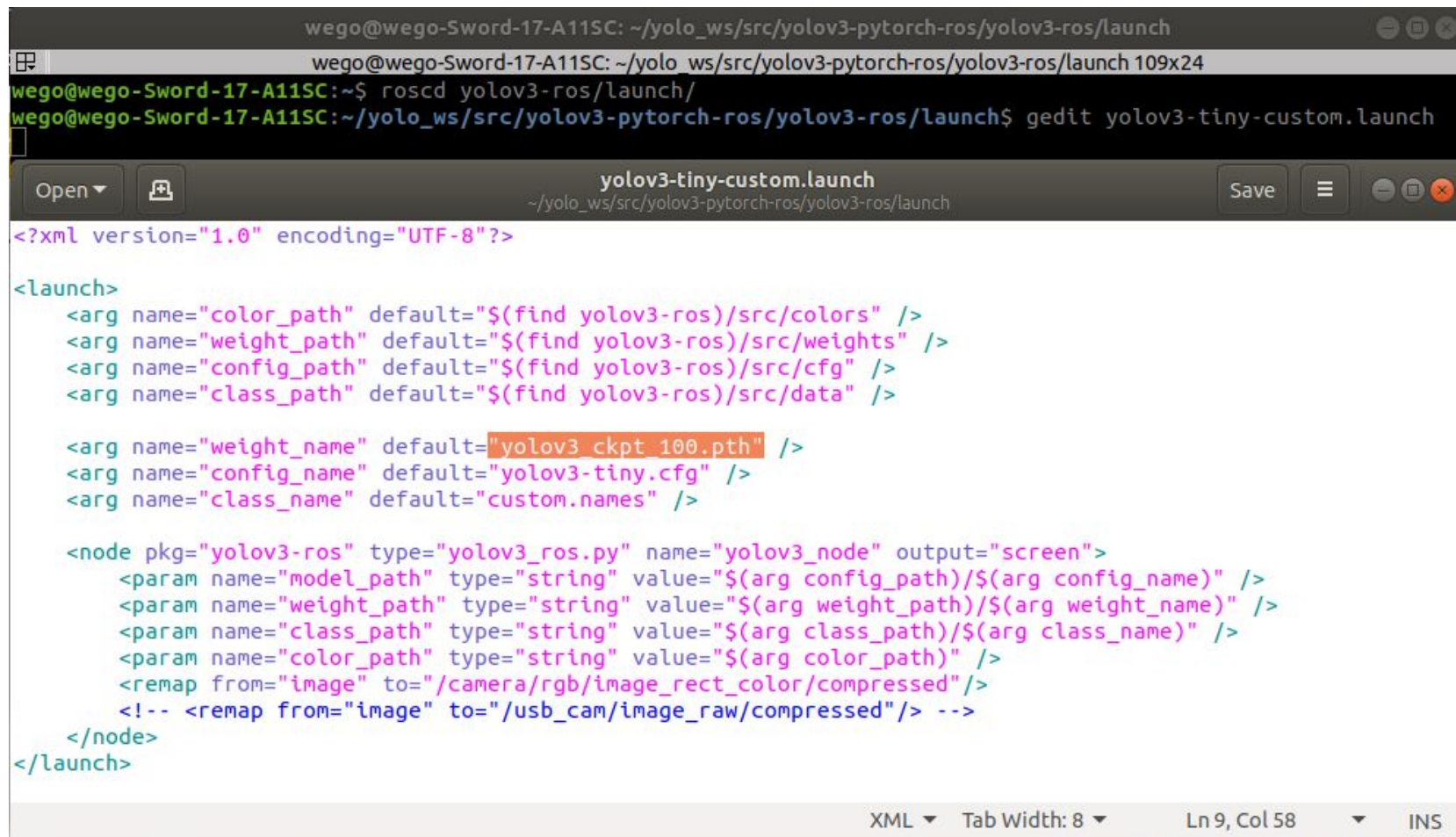
```
$ roscd yolov3-ros/launch
```

```
$ gedit yolov3-tiny-custom.launch
```

- b. 파일 중반부 `<arg name="weight_name" default="yolov3_ckpt_100.pth"/>` 에서 **"yolov3_ckpt_100.pth"** 부분을 weights 디렉토리에 저장한 YOLOv3 가중치 파일명으로 변경합니다.

04 LIMO를 활용한 객체 검출

3. launch 파일 수정



The screenshot shows a terminal window and a code editor. The terminal window displays the following commands and output:

```
wego@wego-Sword-17-A11SC: ~/yolo_ws/src/yolov3-pytorch-ros/yolov3-ros/launch
wego@wego-Sword-17-A11SC: ~/yolo_ws/src/yolov3-pytorch-ros/yolov3-ros/launch 109x24
wego@wego-Sword-17-A11SC:~$ roscd yolov3-ros/launch/
wego@wego-Sword-17-A11SC:~/yolo_ws/src/yolov3-pytorch-ros/yolov3-ros/launch$ gedit yolov3-tiny-custom.launch
```

The code editor shows the content of the `yolov3-tiny-custom.launch` file:

```
<?xml version="1.0" encoding="UTF-8"?>

<launch>
  <arg name="color_path" default="$(find yolov3-ros)/src/colors" />
  <arg name="weight_path" default="$(find yolov3-ros)/src/weights" />
  <arg name="config_path" default="$(find yolov3-ros)/src/cfg" />
  <arg name="class_path" default="$(find yolov3-ros)/src/data" />

  <arg name="weight_name" default="yolov3_ckpt_100.pth" />
  <arg name="config_name" default="yolov3-tiny.cfg" />
  <arg name="class_name" default="custom.names" />

  <node pkg="yolov3-ros" type="yolov3_ros.py" name="yolov3_node" output="screen">
    <param name="model_path" type="string" value="$(arg config_path)/$(arg config_name)" />
    <param name="weight_path" type="string" value="$(arg weight_path)/$(arg weight_name)" />
    <param name="class_path" type="string" value="$(arg class_path)/$(arg class_name)" />
    <param name="color_path" type="string" value="$(arg color_path)" />
    <remap from="image" to="/camera/rgb/image_rect_color/compressed"/>
    <!-- <remap from="image" to="/usb_cam/image_raw/compressed"/> -->
  </node>
</launch>
```

The status bar at the bottom of the code editor shows: XML Tab Width: 8 Ln 9, Col 58 INS.

04 LIMO를 활용한 객체 검출

4. 실시간 객체 검출

- a. LIMO의 터미널 창에 다음 명령어를 입력하여 카메라 데이터를 퍼블리시합니다.

```
$ roslaunch astra_camera dabai_u3.launch
```

- b. PC의 터미널 창에서 다음 명령어를 입력하여 YOLOv3를 실행합니다.

```
$ roslaunch yolov3-ros yolov3-tiny-custom.launch
```

- c. PC에서 새로운 터미널 창을 실행시키고 다음 명령어를 입력하여 rqt_image_view를 실행합니다.

```
$ rqt_image_view
```

- d. 이미지 토픽을 /yolov3_detect/compressed 로 변경합니다.

04 LIMO를 활용한 객체 검출

4. 실시간 객체 검출



05

출처

- Andrew Ng. Machine Learning.
<https://www.coursera.org/learn/machine-learning>
- 파이토치 공식 홈페이지. <https://pytorch.org/>
- 딥 러닝을 이용한 자연어 처리 입문. <https://wikidocs.net/book/2155>
- Darknet 공식 홈페이지. <https://pjreddie.com/darknet/yolo/>
- NVIDIA 공식 블로그. <https://blogs.nvidia.co.kr/>
- YOLOv1 : <https://arxiv.org/pdf/1506.02640.pdf>
- YOLOv2 : <https://arxiv.org/pdf/1612.08242.pdf>
- YOLOv3 : <https://arxiv.org/pdf/1804.02767.pdf>

06



ROS와 LIMO 소개



- ROS

- ROS는 메시지를 프로그램끼리 주고 받을 수 있도록 해주는 로봇 개발용 프레임워크입니다.
- 카메라나 라이다 등 센서 데이터 역시 메시지에 담을 수도 있습니다.
- ROS는 발행 (publish)되고 있는 메시지들을 녹화하고 다시 재생할 수 있는 rosbag 기능을 지원합니다.

 ROS

- LIMO
 - LIMO는 로봇 교육 및 연구 개발용 4륜 구동 ROS 개발 플랫폼입니다.
 - 4-Wheel Drive Mode, Ackermann Mode, Track-type Mode, Mecanum wheel Mode의 4가지 움직임 모델을 제공합니다.
 - Jetson Nano, 2D LiDAR, Depth Camera, IMU가 장착되어 있습니다.



YOLOv3 코드 구조 분석

부록 YOLOv3 코드 구조 분석

- YOLOv3 학습용 파이썬 코드 트리
 - 다운로드한 YOLOv3 학습용 코드의 구조와 각각의 역할을 알아보겠습니다.

```
PyTorch-YOLOv3/  
├── checkpoints  
│   └── README.md  
├── config  
│   ├── coco.data  
│   ├── custom.data  
│   ├── yolov3.cfg  
│   └── yolov3-tiny.cfg  
├── cuda-test.py  
├── data  
│   ├── coco.names  
│   ├── custom.names  
│   ├── make-data-path.py  
│   └── README.md  
├── LICENSE  
├── README.md  
├── requirements.txt  
├── train_results  
│   └── README.md  
├── weights  
│   ├── darknet53.conv.74  
│   ├── download_weights.sh  
│   ├── yolov3-tiny.weights  
│   └── yolov3.weights  
├── yolo  
│   ├── detect.py  
│   ├── models.py  
│   ├── test.py  
│   ├── train.py  
│   └── utils
```

부록 YOLOv3 코드 구조 분석

- YOLOv3 학습용 파이썬 코드 트리
 - config 디렉토리 : YOLOv3의 구조 및 훈련 데이터의 정보를 저장하는 디렉토리입니다.
 - cfg 파일 : cfg 파일에는 YOLOv3의 구조가 저장되어 있습니다. cfg 파일을 기반으로 파이썬 코드가 YOLOv3를 구축합니다.
 - custom.data 파일 : custom.data 파일은 학습시킬 데이터에 대한 정보가 담겨 있습니다. custom.data 파일을 기반으로 파이썬 코드가 학습시킬 데이터들의 경로 및 데이터의 종류를 파악합니다.

```
PyTorch-YOLOv3/  
├── checkpoints  
│   └── README.md  
├── config  
│   ├── coco.data  
│   ├── custom.data  
│   ├── yolov3.cfg  
│   └── yolov3-tiny.cfg  
├── cuda-test.py  
├── data  
│   ├── coco.names  
│   ├── custom.names  
│   ├── make-data-path.py  
│   └── README.md  
├── LICENSE  
├── README.md  
├── requirements.txt  
├── train_results  
│   └── README.md  
├── weights  
│   ├── darknet53.conv.74  
│   ├── download_weights.sh  
│   ├── yolov3-tiny.weights  
│   └── yolov3.weights  
└── yolo  
    ├── detect.py  
    ├── models.py  
    ├── test.py  
    ├── train.py  
    └── utils
```


- YOLOv3 학습용 파이썬 코드 트리
 - data 디렉토리 : 학습시킬 데이터를 저장하는 디렉토리입니다.
 - custom.names 파일 : 검출하고 싶은 객체의 이름이 저장되어 있는 파일입니다.
 - make-data-path.py : 훈련시킬 데이터들의 경로를 담은 txt 파일을 생성하는 파이썬 코드입니다.

```
PyTorch-YOLOv3/  
├── checkpoints  
│   └── README.md  
├── config  
│   ├── coco.data  
│   ├── custom.data  
│   ├── yolov3.cfg  
│   └── yolov3-tiny.cfg  
├── cuda-test.py  
├── data  
│   ├── coco.names  
│   ├── custom.names  
│   ├── make-data-path.py  
│   └── README.md  
├── LICENSE  
├── README.md  
├── requirements.txt  
├── train_results  
│   └── README.md  
├── weights  
│   ├── darknet53.conv.74  
│   ├── download_weights.sh  
│   ├── yolov3-tiny.weights  
│   └── yolov3.weights  
└── yolo  
    ├── detect.py  
    ├── models.py  
    ├── test.py  
    ├── train.py  
    └── utils
```

부록 YOLOv3 코드 구조 분석

- YOLOv3 학습용 파이썬 코드 트리
 - train_results 디렉토리 : YOLOv3 훈련 결과를 시각화한 이미지가 저장되는 디렉토리입니다.
 - weights 디렉토리 : 미리 학습된 YOLOv3의 가중치가 저장되어 있는 디렉토리입니다. 미리 학습된 가중치를 기반으로 새로운 데이터를 학습시키면 훨씬 빠르고 정확하게 YOLOv3가 학습합니다. 이를 '전이학습'이라고 합니다.

```
PyTorch-YOLOv3/  
├── checkpoints  
│   └── README.md  
├── config  
│   ├── coco.data  
│   ├── custom.data  
│   ├── yolov3.cfg  
│   └── yolov3-tiny.cfg  
├── cuda-test.py  
├── data  
│   ├── coco.names  
│   ├── custom.names  
│   ├── make-data-path.py  
│   └── README.md  
├── LICENSE  
├── README.md  
├── requirements.txt  
├── train_results  
│   └── README.md  
├── weights  
│   ├── darknet53.conv.74  
│   ├── download_weights.sh  
│   ├── yolov3-tiny.weights  
│   └── yolov3.weights  
└── yolo  
    ├── detect.py  
    ├── models.py  
    ├── test.py  
    ├── train.py  
    └── utils
```

부록 YOLOv3 코드 구조 분석

- YOLOv3 학습용 파이썬 코드 트리
 - yolo 디렉토리 : YOLOv3의 학습 실행 코드가 저장되어 있는 디렉토리입니다.
 - checkpoints 디렉토리 : 학습시킨 YOLOv3의 가중치가 저장되는 디렉토리입니다.

```
PyTorch-YOLOv3/  
├── checkpoints  
│   └── README.md  
├── config  
│   ├── coco.data  
│   ├── custom.data  
│   ├── yolov3.cfg  
│   └── yolov3-tiny.cfg  
├── cuda-test.py  
├── data  
│   ├── coco.names  
│   ├── custom.names  
│   ├── make-data-path.py  
│   └── README.md  
├── LICENSE  
├── README.md  
├── requirements.txt  
├── train_results  
│   └── README.md  
├── weights  
│   ├── darknet53.conv.74  
│   ├── download_weights.sh  
│   ├── yolov3-tiny.weights  
│   └── yolov3.weights  
└── yolo  
    ├── detect.py  
    ├── models.py  
    ├── test.py  
    ├── train.py  
    └── utils
```



WeGo Robotics

Tel. 031 – 229 – 3553

Fax. 031 – 229 – 3554



제품 문의: go.sales@wego-robotics.com

기술 문의: go.support@wego-robotics.com