

1. 자율주행 인지에 관련된 3종 이상의 공개 Data Set 조사, 정리

- 데이터 설명, 양, 세부 요소 (Feature), 데이터 예시, 활용 예 등 데이터를 이해하는데 필요한 정보

- 구글 웨이모: <https://waymo.com/open/>

Google Waymo









Waymo 소개

웨이모(Waymo) 오픈 데이터 세트는 2019년에 발매되었으며 고해상도 센서 데이터와 1,950개의 세그먼트 라벨로 구성된 인지 데이터 셋을 제공하고 있다.

Waymo Perception Dataset 특징

웨이모의 Perception 데이터 셋은 지리적 다양성을 갖고 있다. 미국의 San Francisco, Mountain View, Los Angeles, Detroit, Seattle, Phoenix 와 같은 지역에서 수집되었다.

조건의 다양성을 갖고 있다. 다양한 환경, 물체, 및 기상 조건이 포함되어있다. 도심지나 교외와 같은 지역적인 데이터, 낮/밤과 같은 시간적 데이터, 보행자/사이클리스트와 같은 물체 인식적 데이터, 맑거나 비가 오는 때에 주행한 기상적 데이터 그리고 이들을 모두 구성하고 있는 데이터가 존재한다.

Local		Time	
Downtown	Suburban	Daylight	Night Time
			
Objects		Conditions	
Pedestrians	Cyclists	Construction	Diverse Weather
			

[그림1: 웨이모 Perception 데이터의 조건별 예시]

Perception 데이터 세트는 다양한 지역과 조건에서 10Hz (390,000 frames) 단위로 수집된 각 20초 분량의 1,950 세그먼트를 포함하고 있다.

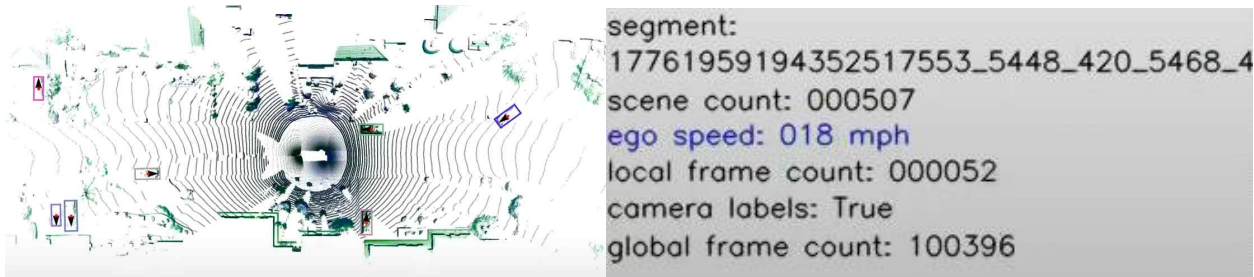
Sensor data	Labeled data
<ul style="list-style-type: none"> ● 1 mid-range lidar ● 4 short-range lidars ● 5 cameras (front and sides) ● Synchronized lidar and camera data ● Lidar to camera projections ● Sensor calibrations and vehicle poses 	<ul style="list-style-type: none"> ● Labels for 4 object classes - Vehicles, Pedestrians, Cyclists, Signs ● High-quality labels for lidar data in 1,200 segments ● 12.6M 3D bounding box labels with tracking IDs on lidar data ● High-quality labels for camera data in 1,000 segments ● 11.8M 2D bounding box labels with tracking IDs on camera data

Waymo dataset 세부 요소 및 예시 (features)

Data labels: 웨이모의 데이터 셋은 lidar 와 카메라로 측정된 데이터에 대해서 서로 독립적으로 생성된 label 데이터를 갖고 있다.

3D Lidar Labels (lidar data)

Lidar 데이터에 3D bounding box labels를 제공한다. Lidar 라벨은 차량 프레임에 있는 3D 7-DOF bounding box이며 전역적으로 고유한 tracking ID가 존재하며 차량, 보행자, 자전거, 표지판과 같은 3D 레이블이 있다.



[그림2: Waymo open Dataset visualization (with projected labels)]

2D Camera Labels (Camera images)

카메라 이미지에 2D bounding box 레이블을 제공한다. 카메라 레이블은 물체에 꼭 맞는 축 정렬 2D bounding box이며 전역적으로 고유한 추적 ID가 있다. 이때 2D bounding box는 카메라에 보이는 개체 부분만 처리한다. vehicles, pedestrians, cyclists 에는 2D 레이블이 있다. 카메라 간에 물체 추적 대응을 제공하지 않는다.



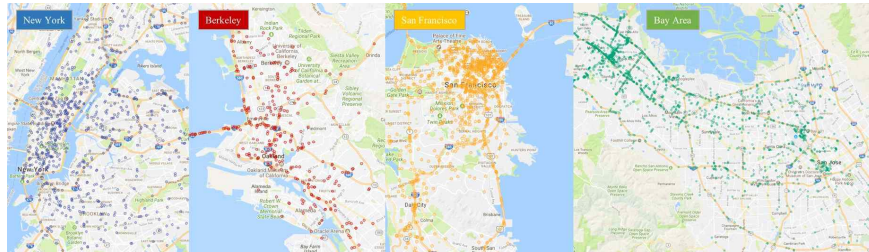
[그림3: Waymo open Dataset visualization]

- 버클리 딥드라이브: <https://bdd-data.berkeley.edu/>

UC Berkeley Deep Drive

소개

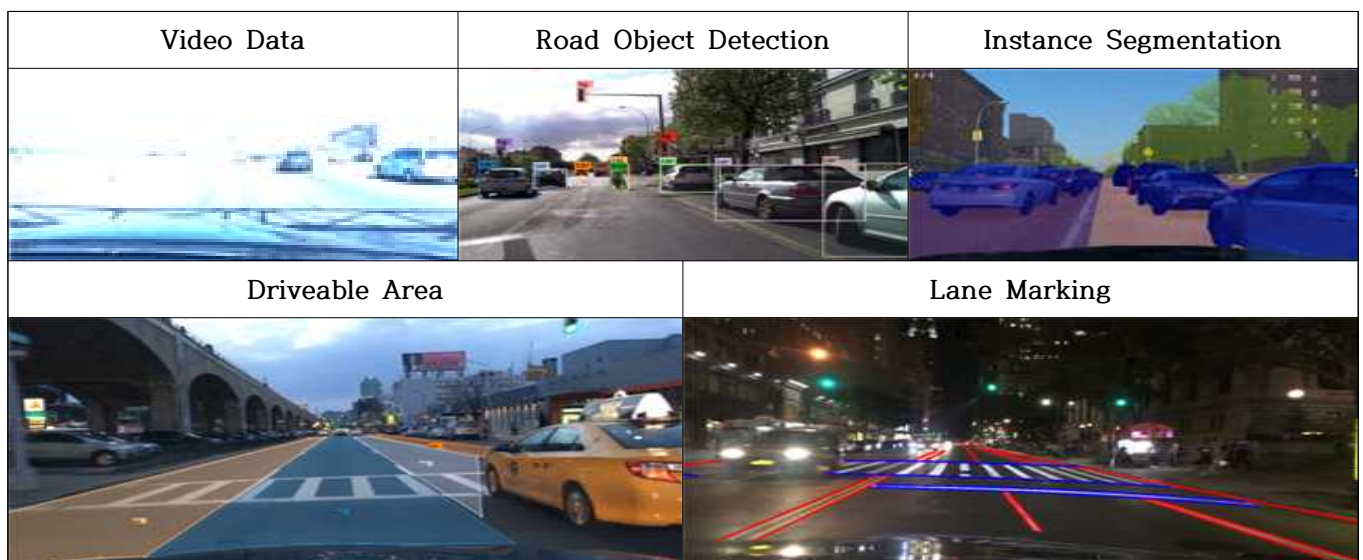
UC 버클리 인공지능 연구 실험실 (BAIR)에서 공개한 BDD100K로 불리는 운전 데이터베이스이다. BDD100K(Berkeley Deep Drive)는 40초 분량의 HD, 30 fps 고화질로 녹화된 100,000 개의 비디오 시퀀스로 구성되어 있다.



	KITTI	Cityscapes	ApolloScape	Mapillary	BDD100K
# Sequences	22	~50	4	N/A	100,000
# Images	14,999	5000 (+2000)	143,906	25,000	120,000,000
Multiple Cities	No	Yes	No	Yes	Yes
Multiple Weathers	No	No	No	Yes	Yes
Multiple Times of Day	No	No	No	Yes	Yes
Multiple Scene types	Yes	No	No	Yes	Yes

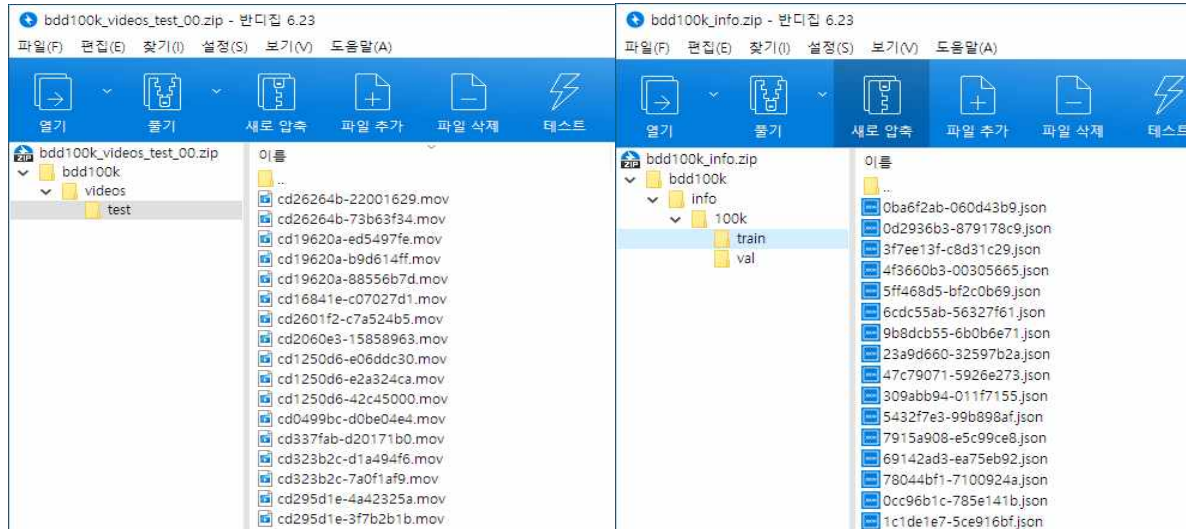
데이터 특징

- **Video Data:** 다양한 시간, 기상조건 및 운전 시나리오에서 1,100 시간 이상의 운전 경험을 담은 비디오 시퀀스를 제공하며 GPS location, IMU 데이터 및 Timestamp 등의 feature 가 포함되어 있다.
- **Road Object Detection:** Bus, Traffic light/sign, Person, Bike, Car 등과 같은 Road Object를 2D Bounding Box 형태로 설명되는 100,000 개의 이미지를 제공한다.
- **Instance Segmentation:** 픽셀 레벨과 충분한 인스턴스 레벨 설명을 통해 10,000 개의 다양한 이미지를 탐색한다.
- **Driveable Area:** 100,000 개의 이미지로부터 복잡한 drivable decision 에 대하여 학습한다.
- **Lane Markings:** driving guidance를 위한 100,000개의 다양한 형태의 차선의 marking annotations이 있다.



Deep drive 세부 요소 및 예시 (features)

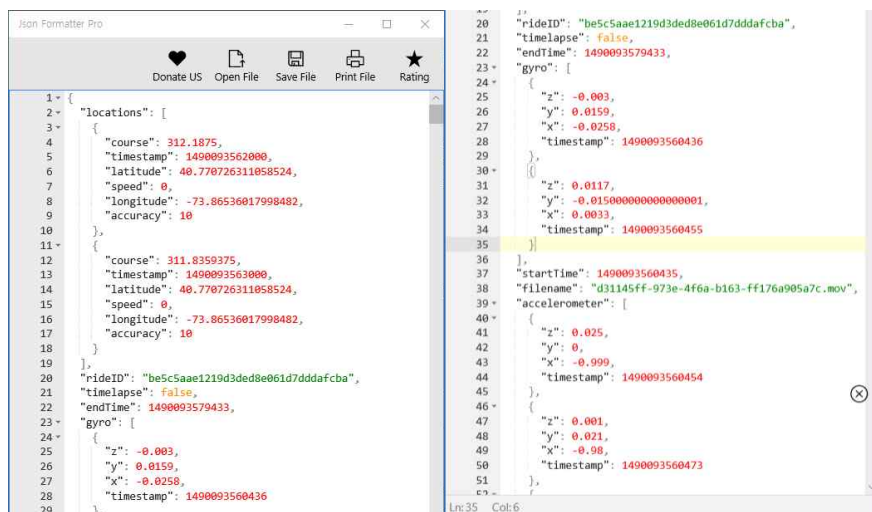
Berkeley Deep drive 공식 홈페이지에서 100K Video Clip 과 촬영시에 기록된 GPS 및 IMU features 가 기록된 자료를 확인할 수 있다. 예시로 100개의 parts 로 나뉜 Video Clip 과 Info json 데이터를 다운로드 받으면 다음과 같이 구성되어 있다.



각 Video Clip 이 16 자리의 GUID Format 으로 Naming 되어서 모든 디렉토리나 파일에 있어서 Unique 함을 보장할 수 있도록 데이터가 구성되어 있으며 각종 주행 영상을 시청할 수 있다.

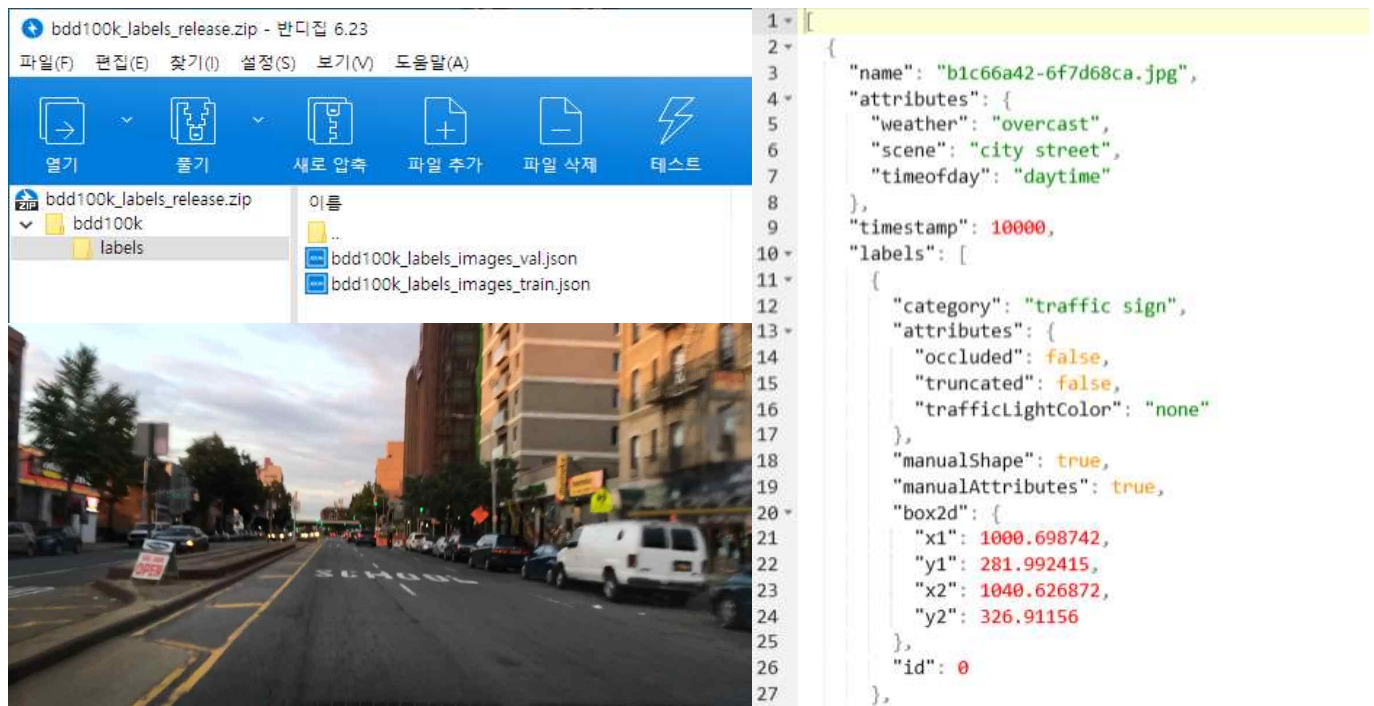


각 Json 파일에는 동영상이 촬영되는 동안 GPS, IMU 등에 기록된 feature 가 location, rideID, timelapse, endTime, gyro 분류로 표현되어있다. 자세한 사항은 다음과 같다.



Road Object Detection

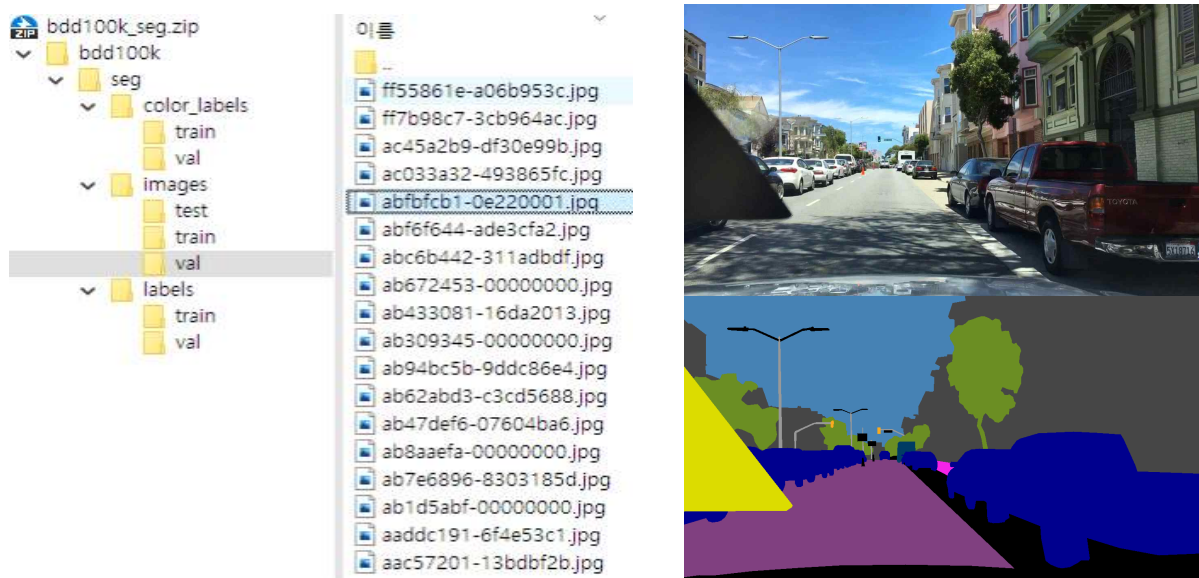
Video 나 Image 에 대한 설명으로 화면에 보이는 Bus, Traffic light/sign, Person, Bike, Car 등과 같은 Road Object 에 대하여 2D Bounding Box 형태로 Annotaion 해둔 데이터 Label을 Json 데이터로 제공한다.



Instance Segmentation

컴퓨터 비전 분야에서 Image segmentation 은 이미지의 영역을 분할해서 각 Object 에 알맞게 합쳐주는 것을 의미한다. 대표적으로 같은 class 의 object를 같은 색으로 표현해주는 Segmantic segmentation 과 같은 class 여도 서로 다른 instance 로 구분지어주는 Instance Segmentation 기법이 존재한다.

Deep drive dataset에서는 픽셀 레벨과 충분한 인스턴스 레벨 설명을 통해 10,000 개의 다양한 이미지를 탐색한 Instance segementaion 데이터를 제공하며 자세한 사항은 다음과 같다.



Driveable Area & Lane Marking

자동차가 주행할 때 주행가능한 영역과 차선을 인식하는 것은 매우 중요한 일이다. Lane Marking 은 두 가지 유형으로 나뉩니다. 수직 차선 표시(빨간색)는 차선의 주행 방향을 따라 표현되며 병렬 차선 표시(파란색)는 차선의 차량이 정지할 수 있도록 표현된다.



도로를 주행할 때 각종 Lane Marking 과 같은 교통 요소에만 의존할 수 없다. 도로 위에는 여러 Object 가 존재하기 때문에 어느 영역에서 주행해야 하는지를 이해하는 것이 중요하다. Deep drive 는 Driveable Area 에 관하여 세분화된 Annotation을 제공하며 direct drivable(빨간색) 과 alternative drivable(파란색) 두 가지 범주로 구분한다. direct drivable 은 자신의 차량이 도로 우선권을 갖고 있어서 해당 지역에서 계속 운전할 수 있음을 의미한다. alternative drivable 은 자신의 차량이 표시된 지역에서 운전할 수 있지만 도로 우선권이 다른 차량에 속할 수 있기 때문에 조심해야 함을 의미한다.



- Pandaset: <https://pandaset.org/>

Pandaset

소개

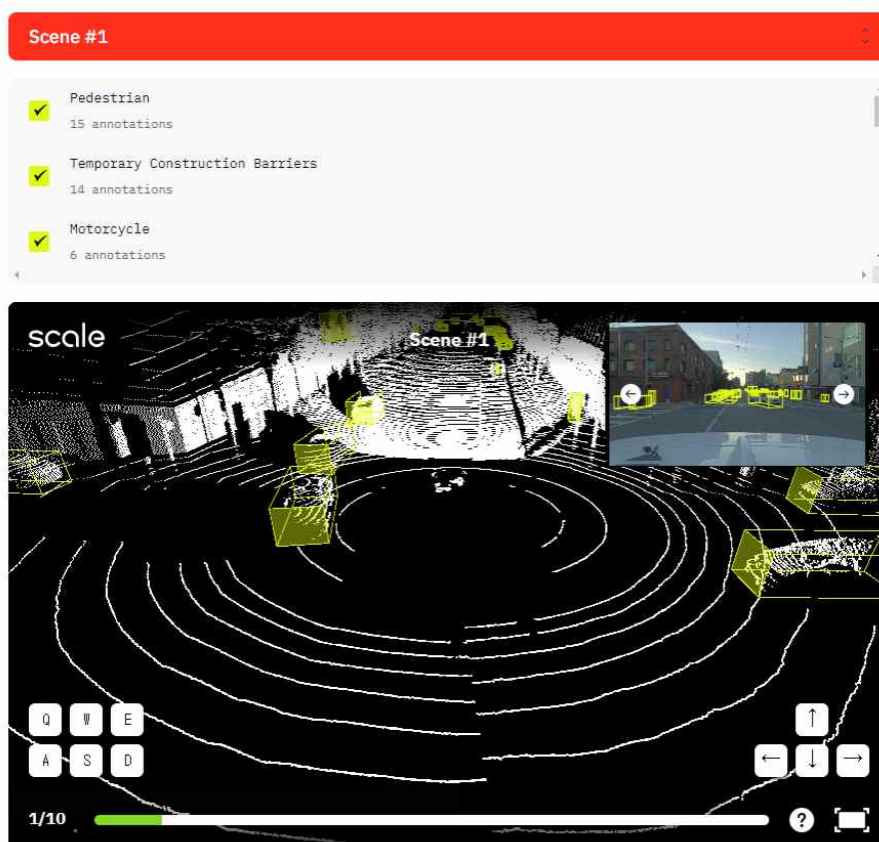
PandaSet은 Autonomous driving 과 Machine learning 분야에서 연구 개발을 촉진하고 발전시키는 것을 목표로 한다. 학문적, PandaSet은 상업적 용도로 제공되는 최초 오픈소스 형태의 AV 데이터 세트이며 Hesai의 LiDAR 센서와 Scale AI의 고품질 데이터 Annotation 으로 구성되어 제공된다.

<ul style="list-style-type: none">▪ 48,000+ camera images▪ 16,000+ LiDAR sweeps▪ 100+ scenes of 8s each	<ul style="list-style-type: none">▪ 28 annotation classes▪ 37 semantic segmentation labels▪ Full sensor suite: 1x mechanical spinning LiDAR, 1x forward-facing LiDAR, 6x cameras, On-board GPS/IMU
---	--

데이터 특징

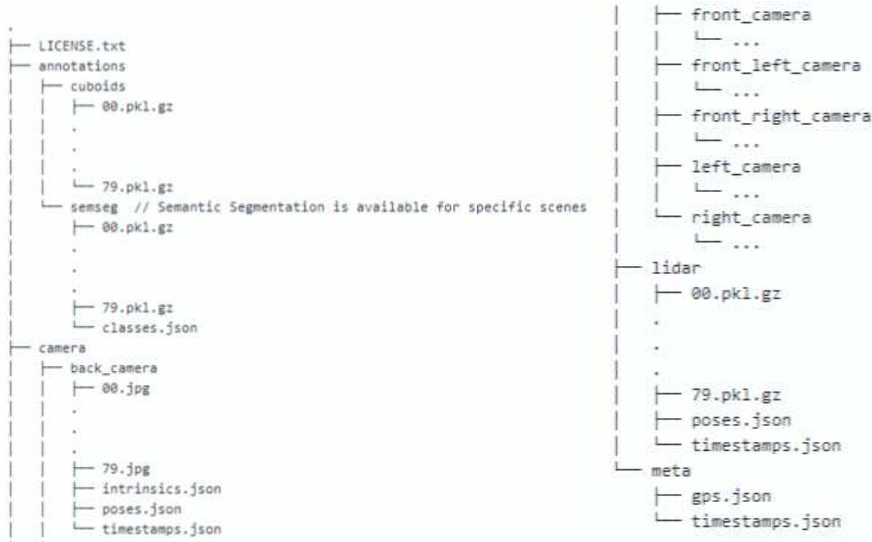
도시 환경에서의 복잡한 운전 시나리오를 반영한다. 가파른 언덕, 건설 현장, 혼잡한 교통 및 보행자, 아침, 점심, 저녁과 같은 다양한 시간대 및 조명 조건이 포함되어 있으며 San Francisco와 Palo Alto부터 San Mateo까지의 지리적 위치에서 수집되었다.

PandaSet에는 28개의 object class 에 대한 3D Bounding boxes 와 활동, 가시성, 위치, 포즈와 관련된 풍부한 클래스 속성 집합이 포함되어 있다. 데이터 집합에는 연기, 자동차 배기, 식물 및 구동 가능한 표면을 포함하여 37개의 의미 체계 라벨이 있는 Point Cloud 세분화도 포함된다.



Pandaset 세부 요소 및 예시 (features)

Pandaset 의 파일 및 디렉토리 구조는 다음과 같다.



Pandaset 는 데이터셋 뿐만 아니라 Python 기반의 devkit API를 제공해주어서 각 Video Sequence 에 관한 센서 데이터에 손쉽게 접근할 수 있다.

Lidar Feature	Camera Feature
<pre> >>> pc0 = seq002.lidar[0] >>> print(pc0) x y z i t d index 0 -75.131138 -79.331690 3.511804 7.0 1.557540e+09 0 1 -112.588306 -118.666002 1.423499 31.0 1.557540e+09 0 2 -42.085902 -44.384891 0.593491 7.0 1.557540e+09 0 3 -27.329435 -28.795053 -0.403781 0.0 1.557540e+09 0 4 -6.196208 -6.621082 1.130009 3.0 1.557540e+09 0 ... 166763 27.670526 17.159726 3.778677 25.0 1.557540e+09 1 166764 27.703935 17.114063 3.780626 27.0 1.557540e+09 1 166765 27.560664 16.955518 3.767948 18.0 1.557540e+09 1 166766 27.384433 16.783824 3.752670 22.0 1.557540e+09 1 166767 27.228821 16.626038 3.739154 20.0 1.557540e+09 1 [166768 rows x 6 columns] </pre>	<pre> >>> s1 = slice(None, None, 5) # Equivalent to [::5] >>> camera_obj = seq002.camera['front_camera'] >>> pcs = camera_obj[s1] >>> poses = camera_obj.poses[s1] >>> timestamps = camera_obj.timestamps[s1] >>> intrinsics = camera_obj.intrinsics </pre>
Metadata Feature (GPS/Timestamps)	Cuboids Annotation Feature
<pre> >>> pose0 = seq002.gps[0] >>> print(pose0['lat']) 37.776089291519924 >>> print(pose0['long']) -122.39931707791749 </pre>	<pre> >>> cuboids0 = seq002.cuboids[0] # Returns the cuboid annotations for the fi >>> print(cuboids0.columns) Index(['uid', 'label', 'yaw', 'stationary', 'camera_used', 'position.x', 'position.y', 'position.z', 'dimensions.x', 'dimensions.y', 'dimensions.z', 'attributes.object_motion', 'cuboids.sibling_id', 'cuboids.sensor_id', 'attributes.rider_status', 'attributes.pedestrian_behavior', 'attributes.pedestrian_age'], dtype='object') </pre>
Semantic Segmentation	
<pre> >>> semseg0 = seq002.semseg[0] # Returns the semantic segmentation for the first LiDAR >>> print(semseg0.columns) Index(['class'], dtype='object') >>> print(seq002.semseg.classes) {'1': 'Smoke', '2': 'Exhaust', '3': 'Spray or rain', '4': 'Reflection', '5': 'Vegetati </pre>	

정리

앞서 소개한 3가지 Open dataset(Waymo, Deep drive, Pandaset) 이 제공하려고 하는 데이터들에 공통적인 키워드가 존재하는 것으로 보인다. 우선, 공도를 운행할 때 시간, 기상, 위치 등의 주변 환경에 영향을 받기 때문에 외부 환경적인 요소를 데이터 셋에 반영하고자 한 흔적이 보인다. 인공지능 모델을 학습시킬 때 특정한 환경에서만 인지를 잘하하는 것이 아닌 좀더 다양한 운전환경에 알맞은 인지 결과를 얻는데 도움이 될 것으로 사료된다.

또한 데이터 셋은 Video Sequence 나 Image 와 같은 콘텐츠 데이터 제공에서 더 나아가 Lidar 의 경우 3D Lidar Labels 을 제공하고 Camera 의 경우 2D Lidar Labels을 제공하고 있다. Label 은 어느 영상이나 이미지에서 식별하고자 하는 객체에 사각형(혹은 입체폴)으로 Annotation 해둔 것이다. 주로 머신러닝 분야에서 데이터를 통한 지도학습을 구현하는 목적으로 응용된다.

자율주행 인지 데이터 셋에 대해서 컴퓨터 비전 기술을 활용하여 차선 혹은 도로를 인식하고 화면 속에 보이는 주변 object를 구분할 수 있도록 데이터 셋을 제공하고 있다. 주로 포함되어 있는 데이터는 Segmantic 데이터로 화면속에 보이는 부분에 대해서 Object 별로 서로 다른 색으로 칠하여 화면속 환경을 인지할 수 있도록 도와준다. Segmentic 데이터는 자율주행 인지 분야에서 인지 속도 및 정확도를 향상시키는데 응용된다.

2. 자율주행 인지에 관련된 2종 이상 Open Source 조사, 정리

- 코드 설명, 구성, 활용 및 결과 등 코드를 이해하는데 필요한 정보

- ImageAI: <https://github.com/OlafenwaMoses/ImageAI>

Image AI

소개

Image AI 라이브러리에 포함된 Deep learning 및 컴퓨터 비전 기능을 사용하여 개발자가 간단한 몇줄의 코드작성만으로 인지 시스템을 구축할 수 있도록 구현된 오픈소스 Python 라이브러리이다.

특징

- ImageAI는 image prediction, custom image prediction, object detection, video detection, video object tracking 및 image predictions trainings을 위한 최첨단 기계 학습 알고리즘을 지원한다.
- 현재 ImageNet-1000 dataset에서 학습된 4가지 Machine Learning 알고리즘을 사용하여 image prediction 및 training을 지원한다.
- COCO dataset에서 학습된 RetinaNet, YOLOv3 및 TinyYOLOv3을 사용하여 개체 감지, 비디오 감지 및 개체 추적을 지원한다.
- ImageAI를 사용하면 새로운 개체의 탐지 및 인식을 수행하기 위해 사용자 지정 모델을 학습할 수 있다.

분석

Image AI 코드를 실행하기 전 다음과 같은 Dependency 설치가 필요하다.

- Python 3.7.6
- Tensorflow 2.4.0
- OpenCV
- Keras 2.4.3

```
pip install tensorflow==2.4.0
```

```
pip install keras==2.4.3 numpy==1.19.3 pillow==7.0.0 scipy==1.4.1 h5py==2.10.0 matplotlib==3.3.2
```

```
opencv-python keras-resnet==0.2.0
```

Image AI를 설치하기 위해서 다음과 같은 Command line을 통해 설치할 수 있다.

```
pip install imageai --upgrade
```

Image AI가 제공하는 기능 중 테스트 해볼 기능은 다음과 같다.




Image Prediction	Object Detection	Video Object Detection & Tracking
		
<pre>convertible : 52.459555864334106 sports_car : 37.61284649372101 pickup : 3.1751200556755066 car_wheel : 1.817505806684494 minivan : 1.7487050965428352</pre>	<pre>person : 91.94094364151 person : 73.6182637718465 laptop : 90.24320840835571 laptop : 73.688167338029 laptop : 95.16398318661316 person : 87.1031939983978</pre>	

Image Prediction

Image AI 는 Image Prediction을 위해 4개의 Machine learning 모델을 지원하며 각 다음과 같은 특징을 갖고 있다.

MobileNetV2 : (Size = 4.82 mb, fastest prediction time and moderate accuracy)
ResNet50 by Microsoft Research : (Size = 98 mb, fast prediction time and high accuracy)
InceptionV3 by Google Brain team : (Size = 91.6 mb, slow prediction time and higher accuracy)
DenseNet121 by Facebook AI Research : (Size = 31.6 mb, slower prediction time and highest accuracy)

이들 중 비교적 빠른 처리시간 과 높은 정확성을 지니고 있는 ResNet50 모델을 사용하여 테스트 소스 (FirstPrediction.py)를 Run 해본다.

```
from imageai.Classification import ImageClassification
import os

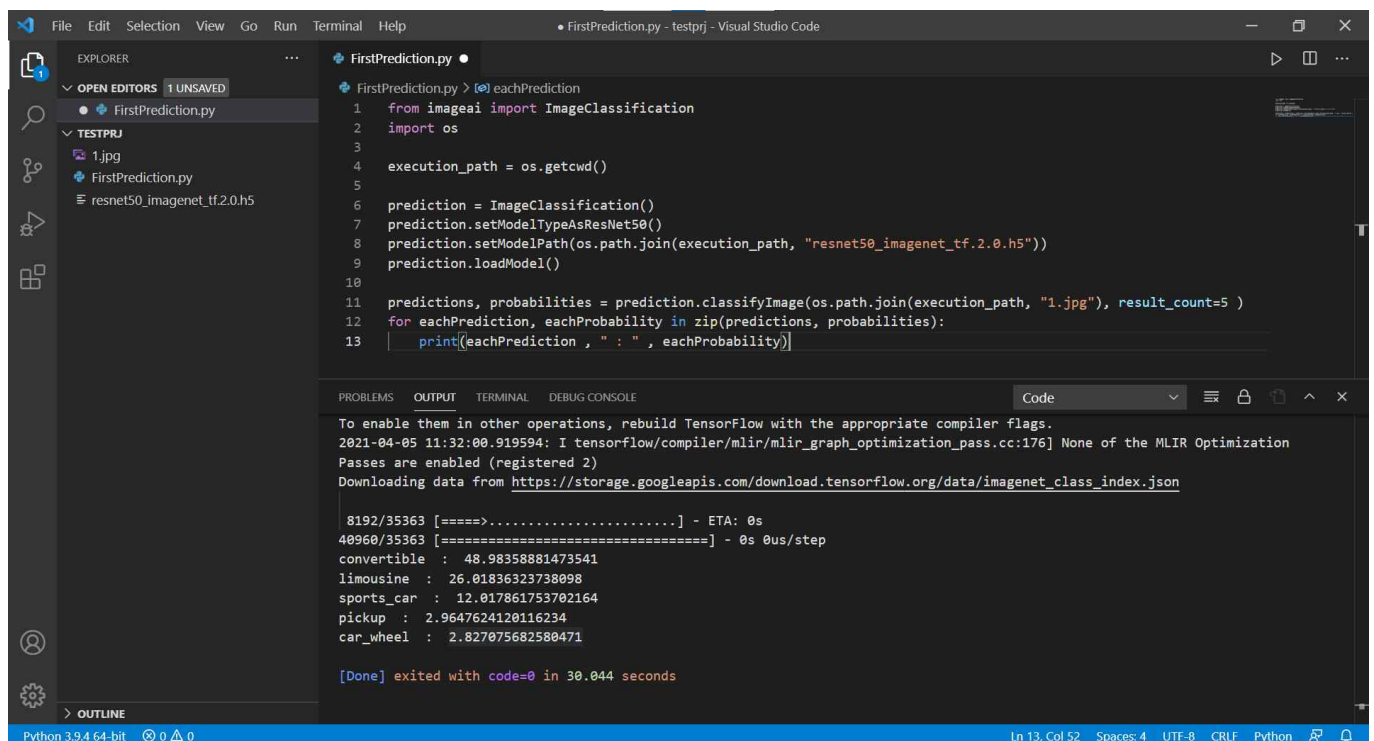
execution_path = os.getcwd()

prediction = ImageClassification()
prediction.setModelTypeAsResNet50()
prediction.setModelPath(os.path.join(execution_path, "resnet50_imagenet_tf.2.0.h5"))
prediction.loadModel()

predictions, probabilities = prediction.classifyImage(os.path.join(execution_path, "1.jpg"), result_count=5 )
for eachPrediction, eachProbability in zip(predictions, probabilities):
    print(eachPrediction , " : " , eachProbability)
```

imageai에서 제공하는 ImageClassification 객체를 사용하면 사용하고 싶은 모델을 설정하여 불러올 수 있다. Microsoft Research 에서 제공한 ResNet50을 사용하였기 때문에 객체에 setModelTypeAsResNet50 method를 통해 모델 설정을 해주었음을 확인할 수 있다. classifyImage method를 통해 분류하고자 하는 이미지를 넣어보면 prediction 과 probabilities를 제공한다.

위에서 언급한 자동차 사진에 대한 실행 결과는 다음과 같다.



```
File Edit Selection View Go Run Terminal Help
FirstPrediction.py - testprj - Visual Studio Code

EXPLORER
1 UNSAVED
FirstPrediction.py
TESTPRJ
1.jpg
FirstPrediction.py
resnet50_imagenet_tf.2.0.h5

FirstPrediction.py
1 from imageai import ImageClassification
2 import os
3
4 execution_path = os.getcwd()
5
6 prediction = ImageClassification()
7 prediction.setModelTypeAsResNet50()
8 prediction.setModelPath(os.path.join(execution_path, "resnet50_imagenet_tf.2.0.h5"))
9 prediction.loadModel()
10
11 predictions, probabilities = prediction.classifyImage(os.path.join(execution_path, "1.jpg"), result_count=5 )
12 for eachPrediction, eachProbability in zip(predictions, probabilities):
13     print(eachPrediction , " : " , eachProbability)]

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
Code
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
2021-04-05 11:32:00.919594: I tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:176] None of the MLIR Optimization
Passes are enabled (registered 2)
Downloading data from https://storage.googleapis.com/download.tensorflow.org/data/imagenet_class_index.json

8192/35363 [=====] - ETA: 0s
40960/35363 [=====] - 0s 0us/step
convertible : 48.98358881473541
limousine : 26.01836323738098
sports_car : 12.017861753702164
pickup : 2.9647624120116234
car_wheel : 2.827075682580471

[Done] exited with code=0 in 30.044 seconds
```


Object Detection

Imageai 는 Object detection을 위해 3개의 Machine learning 모델을 지원하며 각 다음과 같은 특징을 갖고 있다.

RetinaNet : (Size = 145 mb, high performance and accuracy, with longer detection time)
YOLOv3 : (Size = 237 mb, moderate performance and accuracy, with a moderate detection time)
TinyYOLOv3 : (Size = 34 mb, optimized for speed and moderate performance, with fast detection time)

이들 중 비교적 중간 사양의 처리시간과 정확성을 지니고 있는 YOLOv3 모델을 사용하여 테스트 소스 (FirstObjectDetection.py) 는 다음과 같다.

```
from imageai.Detection import ObjectDetection
import os

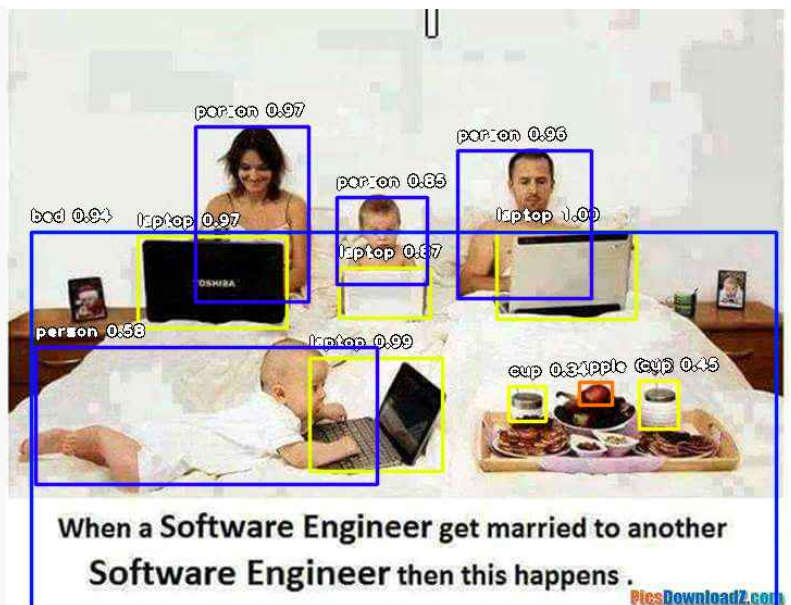
execution_path = os.getcwd()

detector = ObjectDetection()
detector.setModelTypeAsYOLOv3()
detector.setModelPath( os.path.join(execution_path , "yolo.h5"))
detector.loadModel()
detections = detector.detectObjectsFromImage(input_image=os.path.join(execution_path , "image2.jpg"), output

for eachObject in detections:
    print(eachObject["name"] , " : " , eachObject["percentage_probability"] , " : " , eachObject["box_points"]
    print("-----")
```

imageai에서 제공하는 ObjectDetection 객체를 사용하면 사용하고 싶은 모델을 설정하여 불러올 수 있다. 위의 경우 YOLOv3 모델을 사용하였기 때문에 setModelTypeAsYOLOv3 method를 활용하여 모델을 설정하고 있다. detectObjectsFromImage를 통해 인식하고자 하는 이미지를 넣으면 분류된 객체의 이름과 확률 그리고 박스의 좌표를 바운딩된 박스의 좌표와 함께 갱신된 이미지 파일을 리턴한다.

```
laptop : 87.32235431671143 : (306, 238, 390, 284)
-----
laptop : 96.86298966407776 : (121, 209, 258, 293)
-----
laptop : 98.6301600933075 : (279, 321, 401, 425)
-----
laptop : 99.78572130203247 : (451, 204, 579, 285)
-----
bed : 94.02391314506531 : (23, 205, 708, 553)
-----
apple : 48.03136885166168 : (527, 343, 557, 364)
-----
cup : 34.09906327724457 : (462, 347, 496, 379)
-----
cup : 44.65090036392212 : (582, 342, 618, 386)
-----
person : 57.70219564437866 : (27, 311, 341, 437)
-----
person : 85.26121377944946 : (304, 173, 387, 253)
-----
person : 96.33603692054749 : (415, 130, 538, 266)
-----
person : 96.95255160331726 : (174, 108, 278, 269)
```



Video Object Detection & Tracking

imageai 는 Video Object Detection & Tracking을 위해 3개의 Machine learning 모델을 지원하며 각 다음과 같은 특징을 갖고 있다.

RetinaNet : (Size = 145 mb, high performance and accuracy, with longer detection time)
YOLOv3 : (Size = 237 mb, moderate performance and accuracy, with a moderate detection time)
TinyYOLOv3 : (Size = 34 mb, optimized for speed and moderate performance, with fast detection time)

이들 중 비교적 높은 정확성을 보여주지만 처리시간이 오래걸리는 RetinaNet 모델을 사용하여 테스트 소스 (FirstVideoObjectDetection.py) 는 다음과 같다.

```
from imageai.Detection import VideoObjectDetection
import os

execution_path = os.getcwd()

detector = VideoObjectDetection()
detector.setModelTypeAsRetinaNet()
detector.setModelPath( os.path.join(execution_path , "resnet50_coco_best_v2.0.1.h5"))
detector.loadModel()

video_path = detector.detectObjectsFromVideo(input_file_path=os.path.join(execution_path, "traffic.mp4"),
                                             output_file_path=os.path.join(execution_path, "traffic_detected")
                                             , frames_per_second=20, log_progress=True)

print(video_path)
```

imageai에서 제공하는 VideoObjectDetection 객체를 사용하면 사용하고 싶은 모델을 설정하여 불러올 수 있다. 위의 경우 RetinaNet 모델을 사용하였기 때문에 setModelTypeAsRetinaNet method를 활용하여 모델을 설정하고 있다.

detectObjectsFromVideo를 통해 인식하고자 하는 동영상상을 넣으면 분류된 객체의 이름과 확률 그리고 2D 바운딩 박스가 포함된 동영상 파일을 리턴한다.



- cvlib: <https://www.cvlib.net/>

cvlib

소개

Python 기반에 간단하고 고 수준의 사용하기 쉬운 오픈 소스 컴퓨터 비전 라이브러리이다. 많은 기능을 담고있는 라이브러리도 필요하지만 때로는 쉽고 빠른 실험을 염두해 둔 라이브러리가 필요하다. cvlib 는 이와 같이 빠르게 개발하여 프로토타입을 확인할 수 있는데 중점을 두고 개발되었다. 적은 시간을 투자하여 아이디어를 구현할 수 있다는 cvlib 의 장점은 연구를 수행하는 데 핵심적인 요소가 될 것이다.

특징

cvlib 는 Deep learning 라이브러리 인 Keras 에 많은 영향을 받았으며 아래와 같은 특징을 지닌다.

- simplicity
- user friendliness
- modularity and
- extensibility

분석

cvlib 코드를 실행하기 전 다음과 같은 Dependency 설치가 필요하다.

<ul style="list-style-type: none">▪ OpenCV▪ Tensorflow
<pre>pip install opencv-python tensorflow</pre>

cvlib 를 설치하기 위해서 다음과 같은 Command line을 통해 설치할 수 있다.

<pre>pip install cvlib</pre>

cvlib에서 지원하는 기능은 다음과 같다.

- | |
|--|
| <ul style="list-style-type: none">▪ Face Detection: detect_face() 함수의 호출을 통해 얼굴 인식 기능을 제공한다.▪ Gender Detection: detect_gender() 함수의 호출을 통해 성별을 인식하는 기능을 제공한다.▪ Object Detection: detect_common_objects()를 통해 YOLOv3 모델에 기반한 물체 인식을 제공한다. |
|--|

Face Detection

cvlib 는 간단한 detect_face() 함수의 호출을 통해 이미지에서 얼굴을 감지할 수 있다. 소스 코드는 아래와 같다.

<pre>import cvlib as cv faces, confidences = cv.detect_face(image)</pre>
--

코드가 실행되면 bounding box corners 와 감지된 얼굴에 상응하는 가능성 정보를 리턴하며 샘플 출력은 다음과 같다.

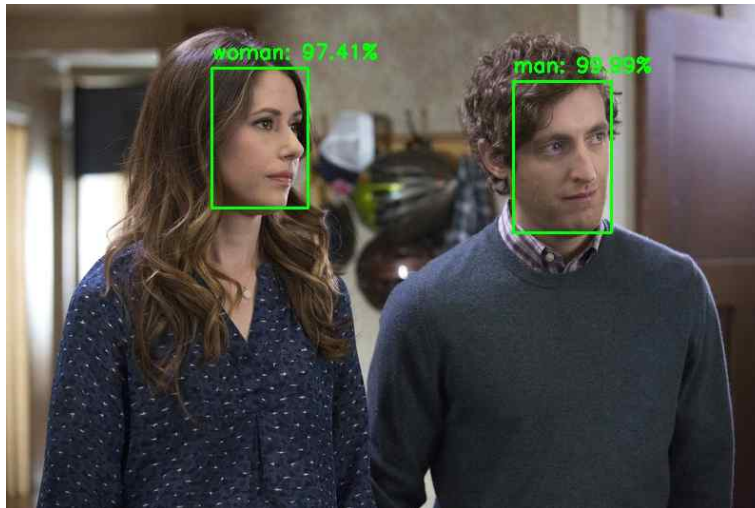


Gender Detection

face detection을 실행하여 이미지에서 얼굴이 인식되면 detect_gender() 함수를 활용해서 감지된 얼굴에 상응하는 성별을 확인할 수 있다. 소스 코드는 다음과 같다.

```
label, confidence = cv.detect_gender(face)
```

코드가 실행되면 감지된 얼굴에 상응하는 성별 정보(man, woman)가 포함된 label을 리턴하며 샘플 출력은 다음과 같다.



Object detection

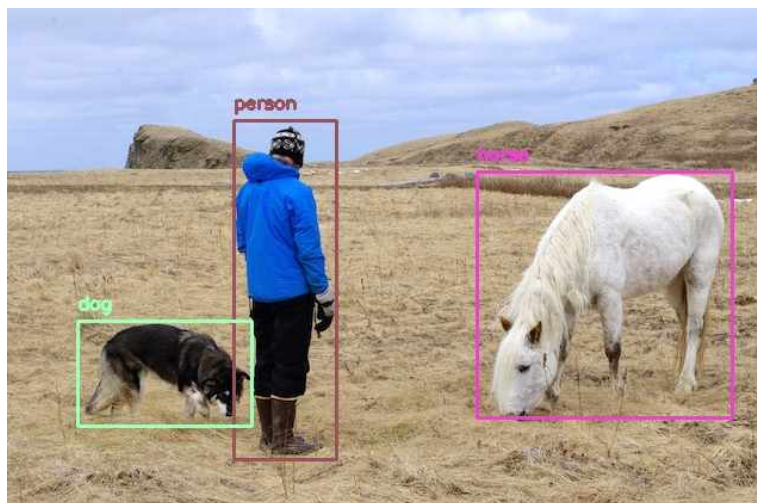
이미지에서 object 들을 인식하기 위하여 YOLO 모델을 활용한 detect_common_objects()를 호출함으로써 구현할 수 있다. 소스 코드는 다음과 같다.

```
import cvlib as cv
from cvlib.object_detection import draw_bbox

bbox, label, conf = cv.detect_common_objects(img)

output_image = draw_bbox(img, bbox, label, conf)
```

코드가 실행되면 이미지에서 감지된 object 들에 대한 Bounding box 좌표, Label, 확률 점수가 리턴된다. 샘플 출력은 다음과 같다.



3. 2)의 정리한 코드 중 하나 실행해서 결과 확인

- 구현 환경 (시스템, lib, docker 등 실행 환경) 및 실행에 관련된 코드 설명 등

시스템 환경

Operating System: Windows 10 Pro (20H2) Processor: Intel(R) Xeon(R) CPU E3-1230 V2 @ 3.30GHz 3.70GHz Graphic Unit: AMD Radeon HD 7850 GDDR5 1GB Memory: DDR3 12GB (1,333Mhz)
Python Version: Python 3.9.4 dependency: opencv, tensorflow Machine Learning Model: YOLOv3 IDE: Visual Studio Code

Source Code

```
C:\Users> smart > Desktop > object_detection.py
1 # object detection example
2 # usage: python object_detection.py <input_image>
3
4 # import necessary packages
5 import cvlib as cv
6 from cvlib.object_detection import draw_bbox
7 import sys
8 import cv2
9
10 # read input image
11 image = cv2.imread(sys.argv[1])
12
13 # apply object detection
14 bbox, label, conf = cv.detect_common_objects(image)
15
16 print(bbox, label, conf)
17
18 # draw bounding box over detected objects
19 out = draw_bbox(image, bbox, label, conf)
20
21 # display output
22 # press any key to close window
23 cv2.imshow("object_detection", out)
24 cv2.waitKey()
25
26 # save output
27 cv2.imwrite("object_detection.jpg", out)
28
29 # release resources
30 cv2.destroyAllWindows()
```

Python 프로그램이 실행되면 프로그램이 있는 경로에서 argument 로 입력한 파일명을 통해 입력 이미지를 불러온다.

cvlib에서 제공하는 detect_common_object 의 parameter 로 이미지를 전달하여 detection을 수행한다.

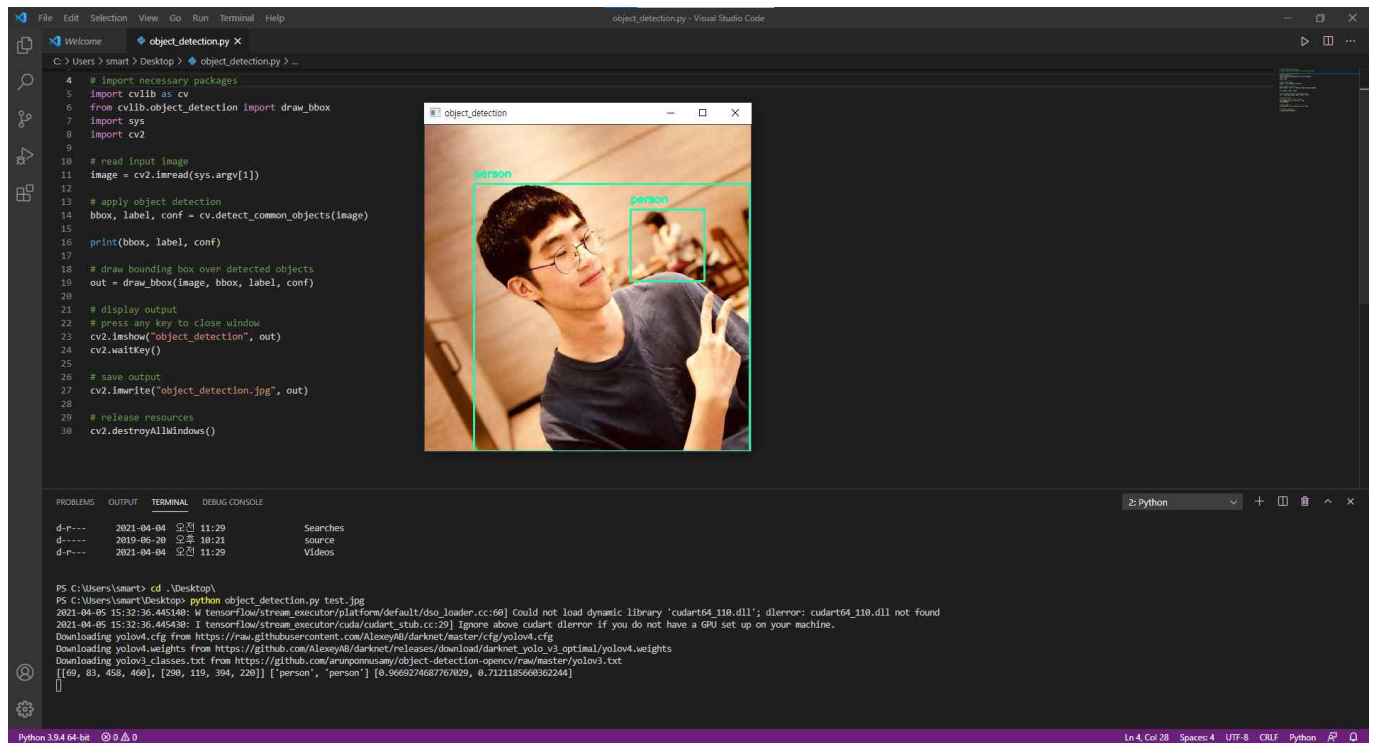
detection 작업이 완료되면 bounding box와 식별된 객체의 이름으로 이루어진 label 그리고 확률정보를 콘솔창에 출력한다.

Bounding Box 가 표시된 이미지를 모니터에 팝업창의 형태로 띄운 후 저장한다.

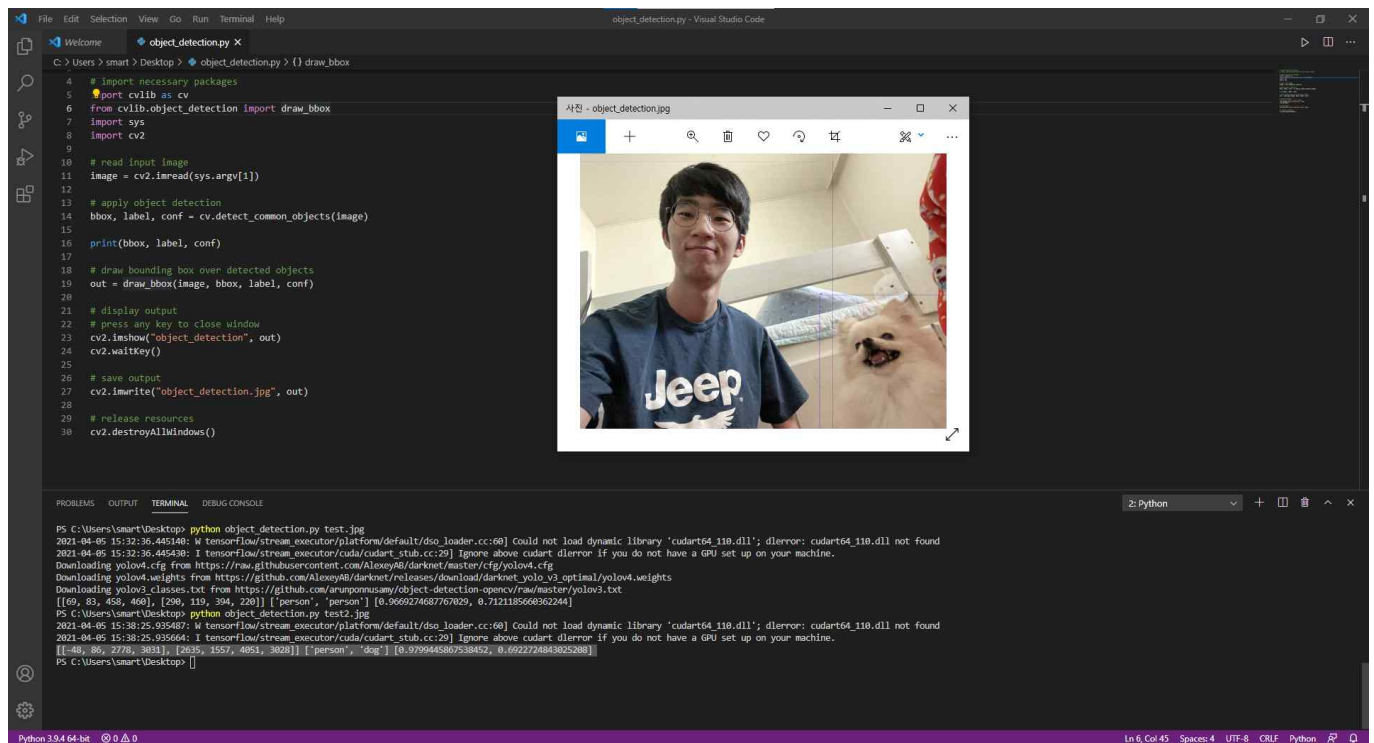
Github Repository

소스 코드 및 실행 결과 이미지 셋은 아래와 같은 Github Repository 에 오픈되어있다.

@smart8612: Intelligence-for-Vehicles (<https://github.com/smart8612/Intelligence-for-Vehicles>)



현재 필자가 SNS에서 사용하는 프로필 이미지에 대하여 Detection을 수행하였다. 카메라에 배경에 블러 효과가 들어갔음에도 불구하고 후방에 앉아있는 사람을 인식해내고 있다.



실험을 위해 필자가 키우고 있는 애완견과 함께 사진을 찍어본 후 같은 모델에서 Detection 작업을 수행해 보았는데 사람과 강아지를 인식하는데 성공하였다.