

# 2021 차량지능기초 과제 1

소프트웨어학부 20171717 최호경

## 목차

1. 자율주행 인지에 관련된 3 종 이상의 공개 Data set 조사, 정리
2. 자율주행 인지에 관련된 2 종 이상 Open Source 조사, 정리
3. 2 의 정리한 코드 중 하나 실행해서 결과 확인
4. 참고 자료

1. 자율주행 인지에 관련된 3 종 이상의 공개 Data set 조사, 정리  
작성자는 다음 3 개의 data set 을 선정하여 조사했다.

- Berkely DeepDrive[1]
- Pandaset[2]
- nuScenes[3]

각 data set 을 살펴보면서 중간에 알아야 할 내용이 있으면 언급하고 넘어가도록 하겠다.

## Berkely DeepDrive

UC Berkely 에서 제공하는 DeepDrive 는 공식 홈페이지에서 크게 5 개의 분류로 나눠 소개한다. 그 분류를 살펴보도록 하자.

## Video Data

### Video Data

Explore 100,000 HD video sequences of over 1,100-hour driving experience across many different times in the day, weather conditions, and driving scenarios. Our video sequences also include GPS locations, IMU data, and timestamps.



10 만 개의 운전자 시점 동영상 데이터가 있다. 총 재생 시간은 1,100 시간이 넘는다고 소개하고 있다. 이와 더불어 GPS 위치, IMU 데이터, 타임스탬프도 함께 제공하고 있다. 잠시 딥러닝 분야에서 feature 에 대해 설명하면 데이터가 가지고 있는 특징을 의미한다. 카드에서 설명하는 다양한 기후, 시간대, 주행 상황, GPS 위치, IMU 데이터, 그리고 타임스탬프를 비디오 데이터의 feature 라고 볼 수 있다.

### Videos

100K video clips

Size: 1.8TB md5: 253d9a2f9d89d2b09d8d93f397aecdd7

### Video Torrent

Torrent for the 100K video clips

### Video Parts

The 100K videos broken into 100 parts

### Info

The GPS/IMU information recorded along with the videos

Size: 3.9GB md5: 043811ff34b2fca6d50f37d263a65c93

비디오 데이터의 용량은 1.8TB 로 매우 크다. 그래서 공식 홈페이지에서는 직접 다운로드 외에 토렌트 링크와 영상을 18GB 씩 나눠 받을 수 있도록 video parts 링크를 제공하고 있다. 비디오를 내려받으면 train, test, val 이라는 디렉토리로 나뉘었음을 볼 수 있는데 이 3 개의 뜻에 대해 잠시 짚고 넘어가도록 하자.

Train set

신경망 모델을 학습하기 위한 용도로 분류된 data set 이다.

Validation set

Train set 으로 학습한 신경망 모델의 성능을 측정하기 위한 data set 이다. Validation set 에서 가장 결과가 좋은 모델을 선정하게 된다.

Test set

Validation set 에서 선정한 모델에 대해 예상 성능을 측정하기 위해 사용되는 data set 이다.

보통 학습을 수행할 때 전체 데이터에서 Train, validation, test set 을 6 : 2 : 2 의 비율로 나눈다고 한다.

video data 에서 받은 영상을 재생하면 홈페이지에서 소개한 대로 다양한 환경에서 기록된 주행 영상을 시청할 수 있다.

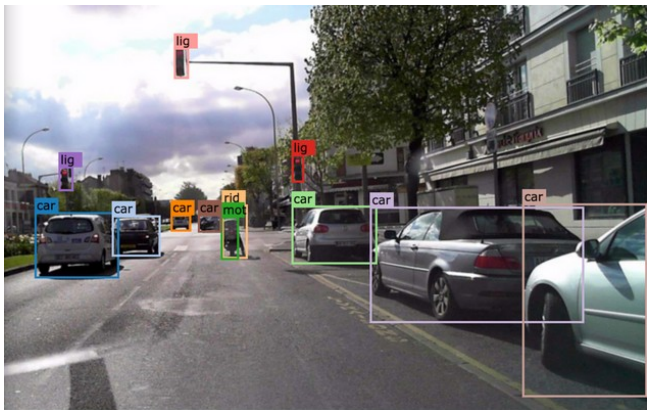


그 다음으로 3.9GB 용량의 info 데이터를 살펴보자.

```
"accelerometer": [
  {
    "y": 0.008,
    "timestamp": 1503833985936,
    "z": -0.058,
    "x": 0.963
  },
  {
    "y": 0.108,
    "timestamp": 1503833985956,
    "z": -0.168,
    "x": 1.0130000000000001
  },
  {
    "y": 0.008,
    "timestamp": 1503833985976,
    "z": -0.058,
    "x": 0.963
  }
],
"locations": [
  {
    "timestamp": 1503833985000,
    "longitude": -73.78046082528799,
    "course": 309.861083984375,
    "latitude": 40.647425175693314,
    "speed": 6.929999828338623,
    "accuracy": 5.0
  },
  {
    "timestamp": 1503833986000,
    "longitude": -73.78051880853288,
    "course": 309.861083984375,
    "latitude": 40.64745994045979,
    "speed": 5.670000076293945,
    "accuracy": 5.0
  },
  {
    "timestamp": 1503833987000,
    "longitude": -73.78057121146711,
    "course": 309.861083984375,
    "latitude": 40.647494959540206,
    "speed": 5.670000076293945,
    "accuracy": 5.0
  }
],
"gps": [
  {
    "timestamp": 1503828255000,
    "altitude": 16.3800048828125,
    "longitude": -73.83105439140817,
    "vertical accuracy": 3.0,
    "horizontal accuracy": 10.0,
    "latitude": 40.677577927746505,
    "speed": 12.680000305175781
  },
  {
    "timestamp": 1503828256000,
    "altitude": 16.3800048828125,
    "longitude": -73.83105439140817,
    "vertical accuracy": 3.0,
    "horizontal accuracy": 10.0,
    "latitude": 40.677577927746505,
    "speed": 12.680000305175781
  },
  {
    "timestamp": 1503828257000,
    "altitude": 16.3800048828125,
    "longitude": -73.83105439140817,
    "vertical accuracy": 3.0,
    "horizontal accuracy": 10.0,
    "latitude": 40.677577927746505,
    "speed": 12.680000305175781
  }
]
```

동영상 촬영 당시 센서에 의해 기록된 feature 가 담겨있다. Info 데이터를 통해 자동차가 어떤 위치에 어떤 상태인지 관찰할 수 있다. 센서 데이터를 이용하면 현재 자동차의 구동 상태를 인지하여 다음에 해야 할 행동 (정지, 주행, 유턴 등)에 영향을 미치도록 할 수 있을 것으로 보인다. 비디오 데이터는 다음에 소개할 항목들에서 이미지로써 더 자세히 annotation 되는 것을 확인할 수 있다.

## Road Object Detection



## Road Object Detection

2D Bounding Boxes annotated on 100,000 images for bus, traffic light, traffic sign, person, bike, truck, motor, car, train, and rider.

사람, 사물에 대해 bounding box 로 주석을 매긴 10 만개의 이미지를 제공한다. Images 링크에서 6.5GB 상당의 이미지를 내려받을 수 있다.

### Images

It has two subfolders. 1) 100K labeled key frame images extracted from the videos at 10th second 2) 10K key frames for full-frame semantic segmentation.

Size: 6.5GB md5: b538e3731a132e28dae37f18c442c51e

### Labels

Annotations of road objects, lanes, and drivable areas in JSON format released in 2018. Details at [Github repo](#). We revised the detection annotations in 2020 and released them as *Detection 2020 Labels* in the list. You are recommended to use the new labels. This detection annotation set is kept for comparison with legacy results.

Size: 107MB md5: e21be3e7d6a07ee439faf61e769667e4

설명에 의하면 비디오에서 10 초 단위로 캡처한 10 만개의 labeled 이미지와, semantic segmentation 을 위한 1 만개의 이미지로 구성되어있다. 또한 이와 관련하여 json 형식으로 작성된 label 정보를 받을 수 있다.



100k 이미지에 대한 각각 label 데이터가 json 포맷으로 제공된다. 데이터 설명에서 Label 과 semantic segmentation 이라는 용어가 나오는데 이에 대해 알아보자.

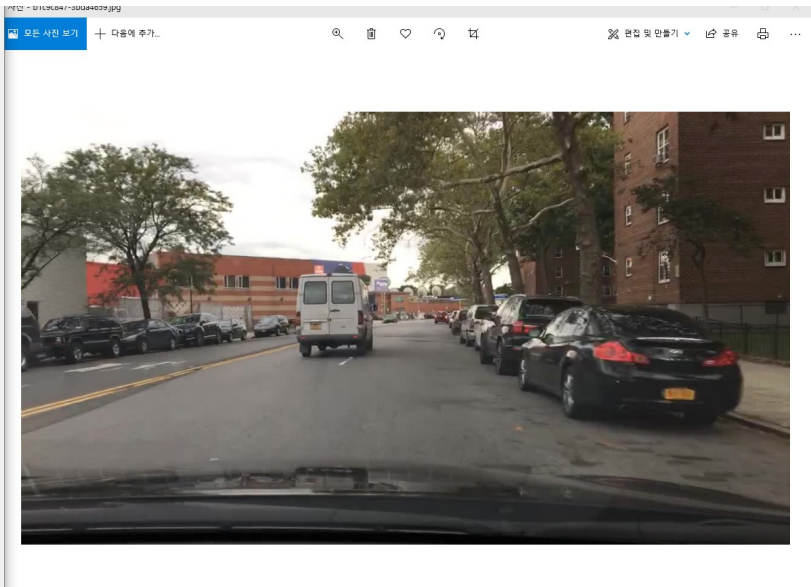


## Label



label은 예측하고자 하는 대상 항목이라는 설명을 찾아볼 수 있다. 쉽게 말해 위의 사진처럼 이미지에서 식별하려는 객체들을 사각형으로 주석을 달아놓은 것이다. 머신러닝에서 데이터의 종류와 특징을 명시하고 지도학습을 하는 용도로 쓰인다고 한다.

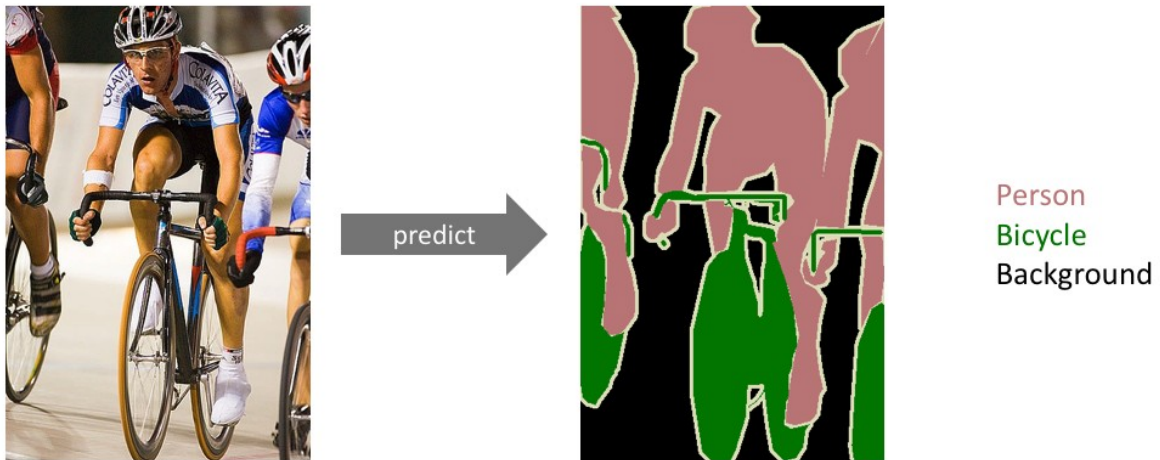
```
{
  "name": "b1c9c847-3bda4659.jpg",
  "attributes": {
    "weather": "undefined",
    "scene": "city street",
    "timeofday": "daytime"
  },
  "timestamp": 10000,
  "labels": [
    {
      "category": "traffic sign",
      "attributes": {
        "occluded": false,
        "truncated": false,
        "trafficLightColor": "none"
      },
      "manualShape": true,
      "manualAttributes": true,
      "box2d": {
        "x1": 623.59743,
        "y1": 308.229303,
        "x2": 635.355495,
        "y2": 315.088176
      },
      "id": 92
    },
    {
      "category": "car",
      "attributes": {
        "occluded": true,
        "truncated": false,
        "trafficLightColor": "none"
      },
      "manualShape": true,
      "manualAttributes": true,
      "box2d": {
        "x1": 325.726422,
        "y1": 342.943609,
        "x2": 376.678043,
        "y2": 384.09684
      }
    }
  ]
}
```



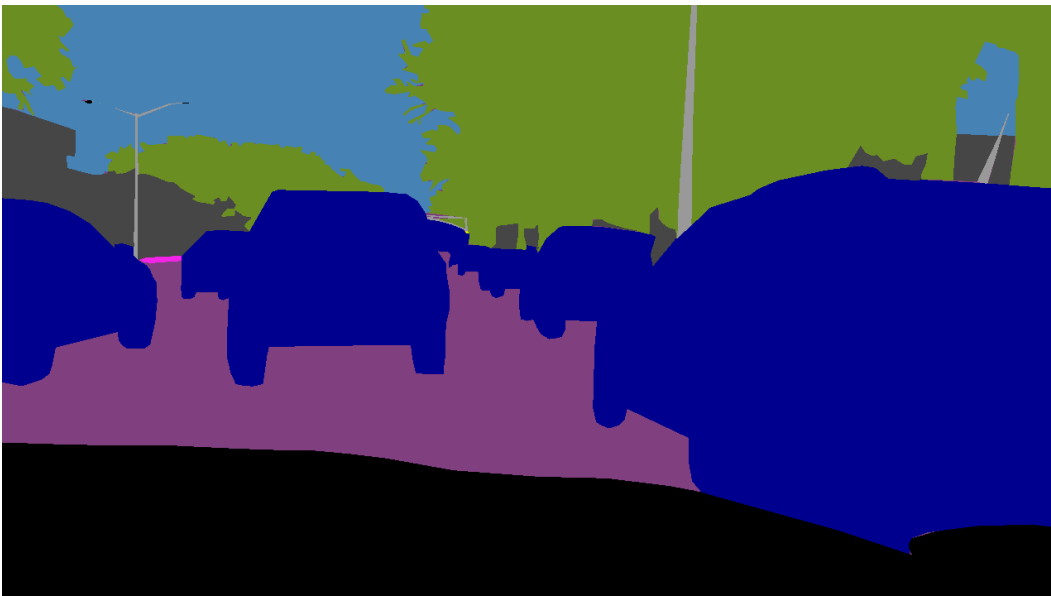
DeepDrive 에서 제공되는 데이터를 보면 특정 사진에 대응하는 json 데이터를 보면 사진에서 확인할 수 있는 전체적인 상황과 사진 안에 보이는 사물들에 대해 어떤 속성을 가진 객체인지 태그하고 관련 attribute(feature)를 제공하고 있다. DeepDrive 의 label 데이터의 포맷은 공식 깃허브 repository 에서 볼 수 있다.[4] 이미지와 그에 대응되는 label 데이터는 자동차로 하여금 주행 환경과, 어떤 사물이 있는지 인지하기 위한 학습 용도로 사용할 수 있을 것이다.

## Semantic segmentation

인터넷 블로그에서 설명하는 Semantic segmentation 은 “이미지 내에 있는 물체들을 의미 있는 단위로 분할해내는 것”[5]이라고 설명하고 있다.

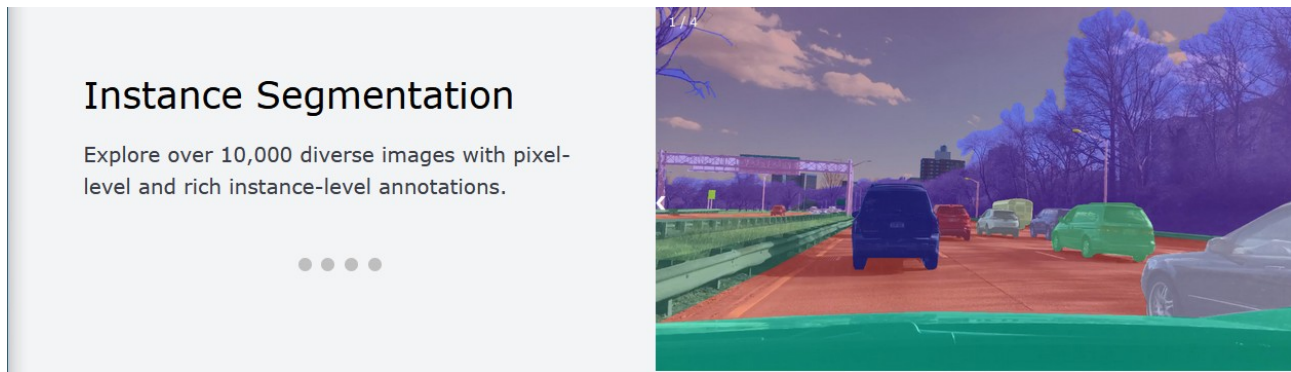


Semantic segmentation 에 대한 예시[6] 사람, 자전거, 배경을 색깔로 구분하는 것을 볼 수 있다. Semantic segmentation 은 과거에는 컴퓨터가 하기에 불가능한 영역이었지만 CNN 의 등장으로 인해 높은 수준으로 클래스 분할을 할 수 있게 되었다. DeepDrive 에서 제공되는 이미지에서 10k 는 바로 이런 Semantic segmentation 을 따로 학습하기 위해 분리한 것으로 보인다. 그 외에도 1.2GB 분량의 segmentation data set 을 제공하는데 열어보면 다음과 같은 그림을 볼 수 있다.



도로는 분홍색, 자동차는 파란색, 나무는 녹색, 기타 배경은 회색으로 구분되어있다. Semantic segmentation 은 자율주행에서 상당히 중요한 역할을 한다고 한다. pytorchhair 에서는 “자율 주행 자동차가 정면에 위치한 대상이 사람인지, 자동차인지, 횡단보도인지 신호등인지 정확하게 구분하지 못하면 상황에 따른 적절한 판단을 내릴 수 없습니다. 따라서 Semantic Segmentation 의 정확도와 속도를 모두 높이기 위해 많은 연구가 이루어지고 있습니다.”[7]라고 semantic segmentation 의 중요성을 설명하고 있다. 즉, 자율주행 분야에서 카메라를 통한 인지 기술 중 가장 중요한 부분이라는 것이다.

## Instance Segmentation



Instance segmentation에서는 pixel-level, rich instance-level로 주석이 달린 1만개의 이미지를 제공한다고 설명한다. 먼저 Instance segmentation에 대해 알아보자. Instance Segmentation은 semantic segmentation의 클래스 분할에서 더 나아가 각각의 instance도 구별하는 것이다. 다만 실제로 찾아보면 형형색색으로 masking된 이미지를 제공하지 않는데 이는 앞서 본 label 데이터에서 id 프로퍼티로 각 객체를 구분할 수 있기 때문인 것으로 보인다.

## Driveable Area & Lane Markings

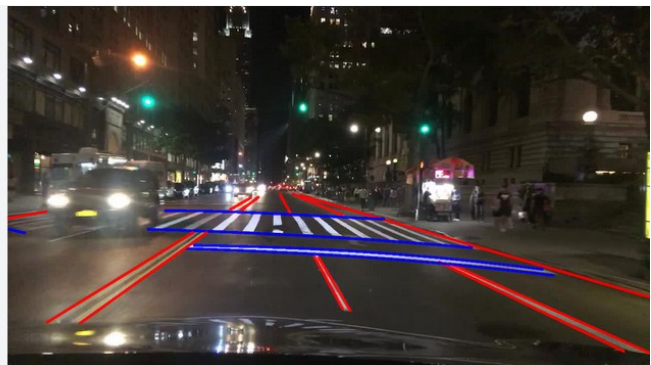


### Driveable Area

Learn complicated drivable decision from 100,000 images.

### Lane Markings

Multiple types of lane marking annotations on 100,000 images for driving guidance.



Driveable Area와 Lane Markings는 각각 자동차가 주행할 수 있는 영역과 차선에 대한 annotation이다. 모름지기 자동차라면 차선을 지키며 운전할 수 있는 곳으로 주행해야 하므로 자율주행의 인지에서 가장 기본적인 부분이다. 해당 주석에 대한 정보는 label에서 찾아볼 수 있는데, 이미지에서 각 영역에 해당하는 정점 좌표를 제공하고 있다. Driveable area의 attribute는 areaType, lane markings의 attribute는 laneDirection, laneStyle, laneType을 가지고 있다.

지금까지 DeepDrive에 대해 알아보았다. DeepDrive는 운전자 시점의 영상 데이터와 자동차가 갖추고 있는 기본적인 센서들의 데이터를 제공하고 있다. 센서 장비가 적은 자동차에 대해 훌륭한 학습 데이터가 될 수 있다고 생각한다.



## Pandaset

Pandaset 은 scale.com 에서 제공하는 자율주행 data set 이다. 공식 홈페이지에 들어가면 어떤 데이터를 제공하는지 보여주고 있다.

**PandaSet aims to promote and advance research and development in autonomous driving and machine learning. The first open-source AV dataset made available for both academic and commercial use, PandaSet combines Hesai's best-in-class LiDAR sensors with Scale AI's high-quality data annotation. It features:**

- 48,000+ camera images
- 16,000+ LiDAR sweeps
- 100+ scenes
- 28 annotation classes
- 37 semantic segmentation labels
- Full sensor suite: 1x mechanical spinning LiDAR (Pandar64), 1x forward-facing LiDAR (PandarGT), 6x cameras, On-board GPS/IMU

앞서 살펴본 DeepDrive 와 비교하여 이미지의 제공량이 적지만 6 방향으로 촬영되었으며 DeepDrive 에 없는 lidar 데이터를 포함한 것을 볼 수 있다.

# Pandaset Tutorial:

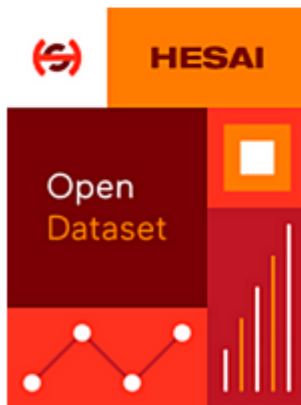
Our team hopes to enable researchers to study challenging urban driving situations using the full sensor suite of a real self-driving-car. Click below to access the **Pandaset**

**Devkit** for best practice on how to apply the dataset to your project:

Pandaset DevKit

## Get Started:

### Your Unique Dataset Access in Three Parts






1. [Part 1](#)

2. [Part 2](#)

3. [Part 3](#)

*(your unique URL expires 48 hours after submitting the download form)*

data set 사용을 신청하면 입력한 이메일 주소로 3 개의 링크로 data set 을 다운로드 할 수 있다.

	<b>pandaset_pandaset_2.zip</b> 8분 남음 — 819 MB / 9.8 GB (18.5 MB/초)	×
	<b>pandaset_pandaset_1.zip</b> 13분 남음 — 989 MB / 15.7 GB (18.5 MB/초)	×
	<b>pandaset_pandaset_0.zip</b> 13분 남음 — 1.5 / 16.0 GB (17.4 MB/초)	×

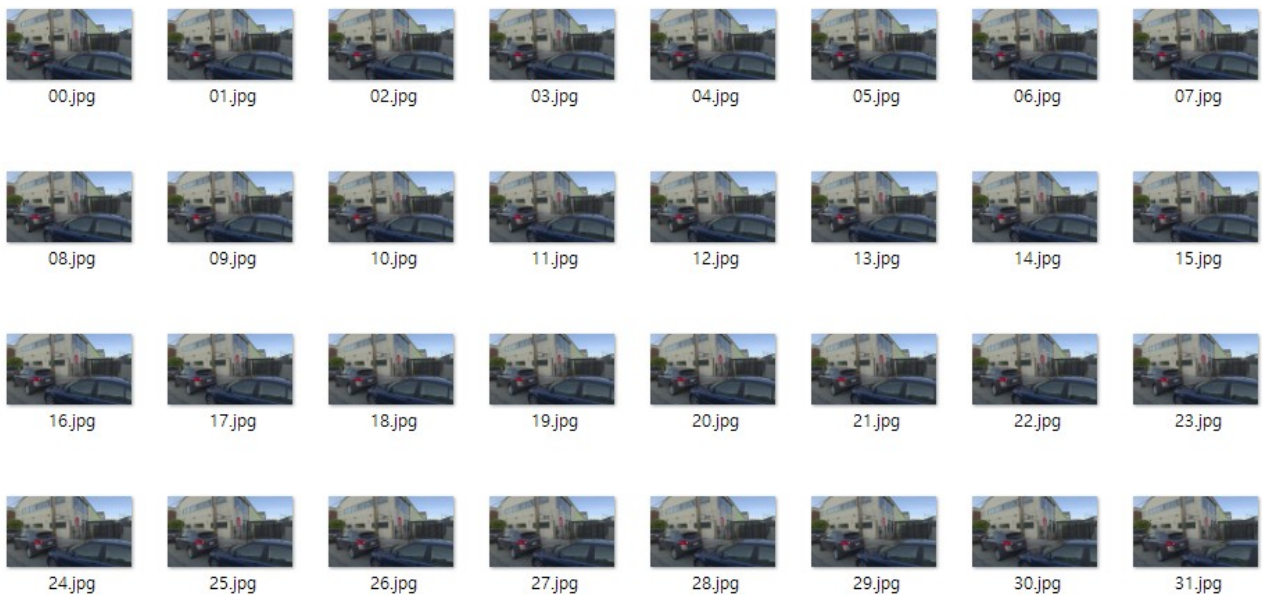
압축 용량은 40GB 를 넘는다.

📁 annotations	2020-04-07 오전 5:36	파일 폴더	
📁 camera	2020-04-02 오후 4:39	파일 폴더	
📁 lidar	2020-04-07 오전 1:23	파일 폴더	
📁 meta	2020-04-07 오후 5:05	파일 폴더	
📄 LICENSE.txt	2020-04-27 오후 4:57	텍스트 문서	6KB

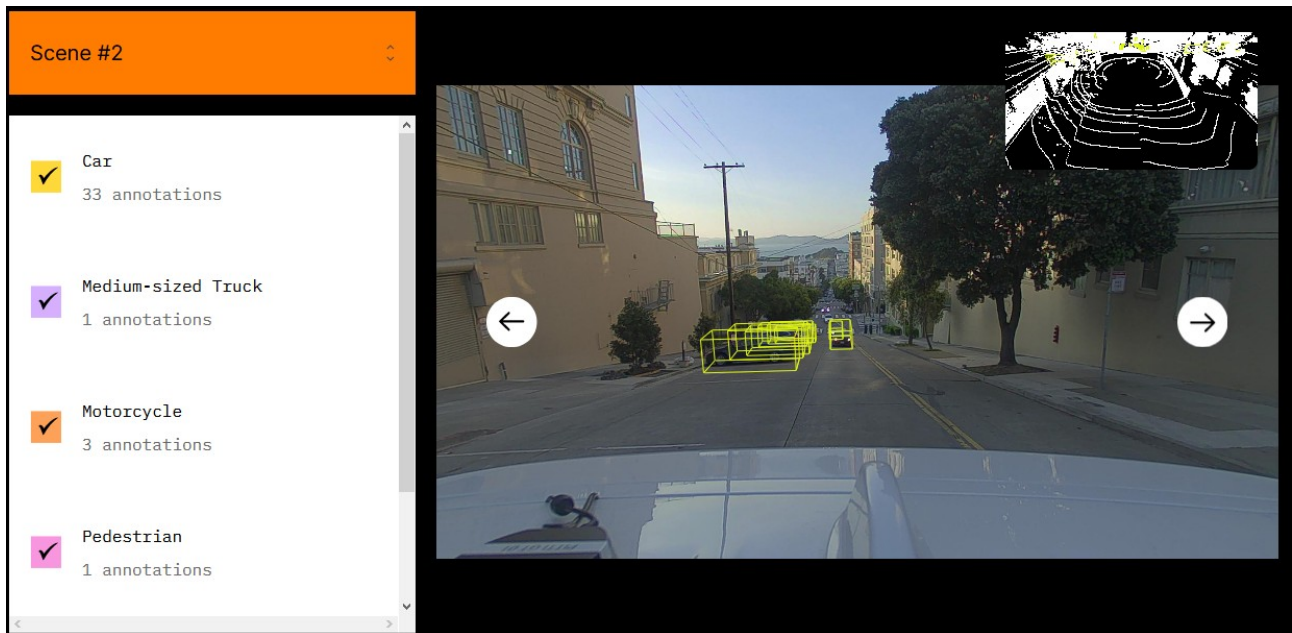
다운로드한 data set 을 열어보면 수많은 디렉터리로 구성되어 있는데 그중 한곳으로 들어가면 annotation 데이터, 카메라 이미지, lidar 데이터, gps 와 타임스탬프가 있는 meta 데이터로 분류되어 있다. 이미지와 meta 를 제외한 모든 데이터는 파이썬 pickle 확장자인 pkl 로 저장되어 있어 파이썬에 최적화되었음을 알 수 있다.

📁 back_camera	2020-04-07 오전 6:56	파일 폴더
📁 front_camera	2020-04-07 오전 6:56	파일 폴더
📁 front_left_camera	2020-04-07 오전 6:56	파일 폴더
📁 front_right_camera	2020-04-07 오전 6:56	파일 폴더
📁 left_camera	2020-04-07 오전 6:56	파일 폴더
📁 right_camera	2020-04-07 오전 6:56	파일 폴더

먼저 이미지를 살펴보면 전방 3 방향, 왼쪽, 오른쪽, 그리고 후방 시점으로 촬영된 이미지를 볼 수 있다.



디렉터리마다 79 장의 이미지가 들어있다. DeepDrive 보다는 확실히 적은 분량이다. 하지만 6 방향에 대한 시야를 제공하기 때문에 자동차의 조건이 맞는다면 주변 환경 인지와 관련된 학습을 위한 강력한 데이터가 된다고 생각한다.



annotation 데이터는 공식 홈페이지를 통해 예제를 바로 확인해볼 수 있다. DeepDrive 와의 차이점이라면 annotation box 가 3d 이고 차량에 대해 더 자세한 분류가 존재한다는 것이다. 사물 인지 측면에서 2d box 와 3d box 를 비교했을 때 어떤 장단점이 있을지 궁금하다.

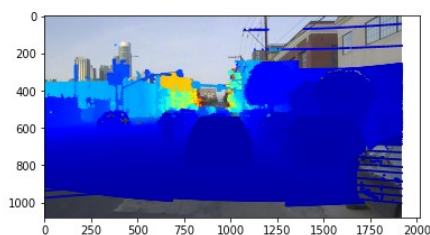
Pandaset 에서 제공하는 lidar 데이터는 공식 github 에서 제공하는 튜토리얼을 통해 어떻게 적용하는지 학습할 수 있다.[8]

#### 4. Show projected points on image colored by distances.

```
In [4]: import matplotlib.cm as cm
import numpy as np

# image after projection
plt.imshow(ori_image)
distances = np.sqrt(np.sum(np.square(camera_points_3d), axis=-1))
colors = cm.jet(distances / np.max(distances))
plt.gca().scatter(projected_points2d[:, 0], projected_points2d[:, 1], color=colors, s=1)
```

Out [4]: <matplotlib.collections.PathCollection at 0x144de3050>



jupyter notebook 을 통해 semantic segmentation, lidar 데이터 플로팅을 하는 예제를 실습할 수 있다. 그중에 인상적인 것은 사진에 보이는 것처럼 lidar 데이터를 대응 이미지에 투영한 결과를 볼 수 있다는 것이다. 거리가 가까울수록 파란색으로 색이 칠해진다. Lidar 데이터가 이미지에 투영되는 것을 보면서 자율주행에 여러 개의 카메라가 필요한 이유는 lidar 데이터와의 연계를 통해 자동차에게 보다 더 강력한 인지 능력을 부여하기 위함이라는 생각이 들었다. 다만 차량지능기초 강의에서 lidar 는 기후에 영향을 크게 받는다고 했는데 lidar 가 잘 작동하지 않을 때는 어떤 조치를 취해야 하는지 궁금하다.

여기까지 Pandaset 에서 찾을 수 있는 주요 특징을 정리해보았다. 다양한 종류의 데이터로 마련된 Pandaset 은 자율주행을 하기 위한 장비를 갖춘 자동차에 대해 확실한 학습 기반을 마련할 수 있는 data set 이라고 볼 수 있다.

NuScenes

nuscenes.org 에서 제공하는 data set 인 NuScenes 는 공식 홈페이지에 제공하는 데이터를 보여주고 있다.

## About nuScenes

The nuScenes dataset is a large-scale autonomous driving dataset with 3d object annotations. It features:

- Full sensor suite (1x LIDAR, 5x RADAR, 6x camera, IMU, GPS)
- 1000 scenes of 20s each
- 1,400,000 camera images
- 390,000 lidar sweeps
- Two diverse cities: Boston and Singapore
- Left versus right hand traffic
- Detailed map information
- 1.4M 3D bounding boxes manually annotated for 23 object classes
- Attributes such as visibility, activity and pose
- **New:** 1.1B lidar points manually annotated for 32 classes
- **New:** Explore nuScenes on [SiaSearch](#)
- Free to use for non-commercial use
- For a commercial license contact [nuScenes@motional.com](mailto:nuScenes@motional.com)

nuScenes 는 레이더 데이터도 포함하고 있어 지금까지 살펴본 데이터 중 가장 다채로운 종류를 제공하고 있다.

Mini ▾

Subset of trainval, 10 scenes, used to explore the data without downloading the whole dataset.


























↓ Metadata and sensor file blobs [\[US, Asia\]](#)

3.88 GB (4167696325 Bytes)

md5: 791dd9ced556cfa1b425682f177b5d9b

공식 홈페이지를 통해 data set 을 내려받을 수 있는데 이미지만 해도 용량이 180GB 를 넘어 상당히 크다. 그래서 소규모로 살펴볼 수 있도록 3.8GB 상당의 mini set 을 제공하고 있다.



 CAM_BACK	2019-03-18 오전 3:19	파일 폴더	
 CAM_BACK_LEFT	2019-03-18 오전 3:20	파일 폴더	
 CAM_BACK_RIGHT	2019-03-18 오전 3:19	파일 폴더	
 CAM_FRONT	2019-03-18 오전 3:19	파일 폴더	
 CAM_FRONT_LEFT	2019-03-18 오전 3:20	파일 폴더	
 CAM_FRONT_RIGHT	2019-03-18 오전 3:19	파일 폴더	
 LIDAR_TOP	2019-03-18 오전 3:19	파일 폴더	
 RADAR_BACK_LEFT	2019-03-18 오전 3:19	파일 폴더	
 RADAR_BACK_RIGHT	2019-03-18 오전 3:19	파일 폴더	
 RADAR_FRONT	2019-03-18 오전 3:19	파일 폴더	
 RADAR_FRONT_LEFT	2019-03-18 오전 3:19	파일 폴더	
 RADAR_FRONT_RIGHT	2021-04-03 오전 3:49	파일 폴더	
 attribute.json	2019-03-23 오전 5:12	JSON 파일	2KB
 calibrated_sensor.json	2019-03-23 오전 5:12	JSON 파일	38KB
 category.json	2019-03-23 오전 5:12	JSON 파일	5KB
 ego_pose.json	2019-03-23 오전 5:12	JSON 파일	7,475KB
 instance.json	2019-03-23 오전 5:12	JSON 파일	222KB
 log.json	2019-03-23 오전 5:12	JSON 파일	2KB
 map.json	2019-03-23 오전 5:12	JSON 파일	4KB
 sample.json	2019-03-23 오전 5:12	JSON 파일	86KB
 sample_annotation.json	2019-03-23 오전 5:12	JSON 파일	9,090KB
 sample_data.json	2019-03-23 오전 5:12	JSON 파일	15,587KB
 scene.json	2019-03-23 오전 5:12	JSON 파일	4KB
 sensor.json	2019-03-23 오전 5:12	JSON 파일	2KB
 visibility.json	2019-03-23 오전 5:12	JSON 파일	1KB

지금까지 알아본 data set 중 가장 다양한 장비를 탑재하고 있어 다양한 feature 가 존재하는 것을 볼 수 있다.

이 데이터를 어떻게 다루는지에 대해 tutorial 이 제공되고 colab 을 통해 실습해볼 수 있다.

## ▼ Google Colab (optional)

 Open in Colab

If you are running this notebook in Google Colab, you can uncomment the cell below and run it; everything will be set up nicely for you. Otherwise, manually set up everything.

```
!mkdir -p /data/sets/nuscenes # Make the directory to store the nuScenes dataset in.

!wget https://www.nuscenes.org/data/v1.0-mini.tgz # Download the nuScenes mini split.

!tar -xf v1.0-mini.tgz -C /data/sets/nuscenes # Uncompress the nuScenes mini split.

!pip install nuscenes-devkit &> /dev/null # Install nuScenes.

--2021-04-03 03:56:10-- https://www.nuscenes.org/data/v1.0-mini.tgz
Resolving www.nuscenes.org (www.nuscenes.org)... 99.84.176.31, 99.84.176.53, 99.84.176.62, ...
Connecting to www.nuscenes.org (www.nuscenes.org)|99.84.176.31|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 4167696325 (3.9G) [application/x-tar]
Saving to: 'v1.0-mini.tgz'

v1.0-mini.tgz      100%[=====>]  3.88G  49.4MB/s   in 85s

2021-04-03 03:57:35 (47.0 MB/s) - 'v1.0-mini.tgz' saved [4167696325/4167696325]
```

OK, that was easy. Let's try something harder. Let's look at the `sample_annotation` table.

```
[17] nusc.sample_annotation[0]
```

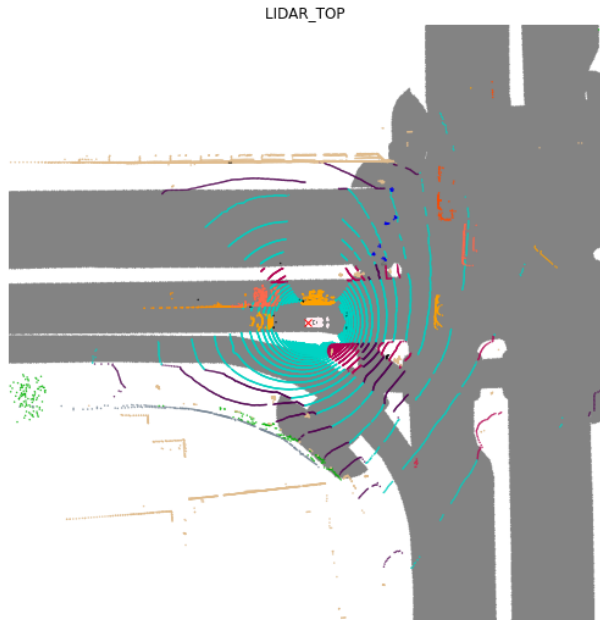
```
{'attribute_tokens': ['4d8821270b4a47e3a8a300cbec48188e'],
 'category_name': 'human.pedestrian.adult',
 'instance_token': '6dd2cbf4c24b4caeb625035869bca7b5',
 'next': '1e8e35d365a441a18dd5503a0ee1c208',
 'num_lidar_pts': 5,
 'num_radar_pts': 0,
 'prev': 'a1721876c0944cdd92ebc3c75d55d693',
 'rotation': [0.9831098797903927, 0.0, 0.0, -0.18301629506281616],
 'sample_token': 'cd21dbfc3bd749c7b10a5c42562e0c42',
 'size': [0.621, 0.669, 1.642],
 'token': '70aecbe9b64f4722ab3c230391a3beb8',
 'translation': [373.214, 1130.48, 1.25],
 'visibility_token': '4'}
```

This also has a `token` field (they all do). In addition, it has several fields of the format `[a-z]*_token`, e.g. `instance_token`. These are foreign keys in database terminology, meaning they point to another table. Using `nusc.get()` we can grab any of these in constant time. For example, let's look at the visibility record.

## Render the lidarseg labels in the bird's eye view of a pointcloud

In the original nuScenes devkit, you would pass a sample data token into `render_sample_data` to render a bird's eye view of the pointcloud. However, the points would be colored according to the distance from the ego vehicle. Now with the extended nuScenes devkit, all you need to do is set `show_lidarseg=True` to visualize the class labels of the pointcloud.

```
In [7]: sample_data_token = my_sample['data']['LIDAR_TOP']
nusc.render_sample_data(sample_data_token,
                        with_anns=False,
                        show_lidarseg=True)
```

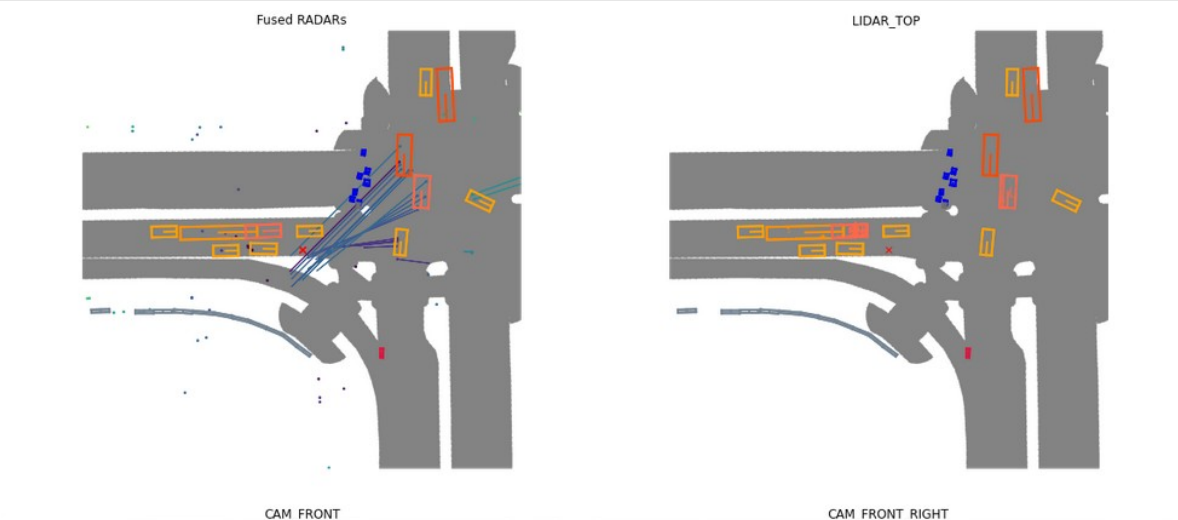


But what if you wanted to focus on only certain classes? Given the statistics of the pointcloud printed out previously, let's say you are only interested in trucks and trailers. You could see the class indices belonging to those classes from the statistics and then pass an array of those indices into `filter_lidarseg_labels` like so:

## Render sample (i.e. lidar, radar and all cameras)

Of course, like in the original nuScenes devkit, you can render all the sensors at once with `render_sample`. In this extended nuScenes devkit, you can set `show_lidarseg=True` to see the lidarseg labels. Similar to the above methods, you can use `filter_lidarseg_labels` to display only the classes you wish to see.

```
In [11]: nusc.render_sample(my_sample['token'],
                        show_lidarseg=True,
                        filter_lidarseg_labels=[22, 23])
```



특히 센서를 이용한 segmentation 예제를 통해 꼭 이미지 외의 장비로도 segmentation 이 가능하다는 것을 볼 수 있어 인상 깊었다. 예제 실행 결과에 의하면 이미지로는 보이지 않는 영역에서도 물체를 인지할 수가 있다. 이처럼 센서는 자율주행차량이 주변 상황을 파악하기 위한 강력한 데이터가 됨을 알 수 있었다.

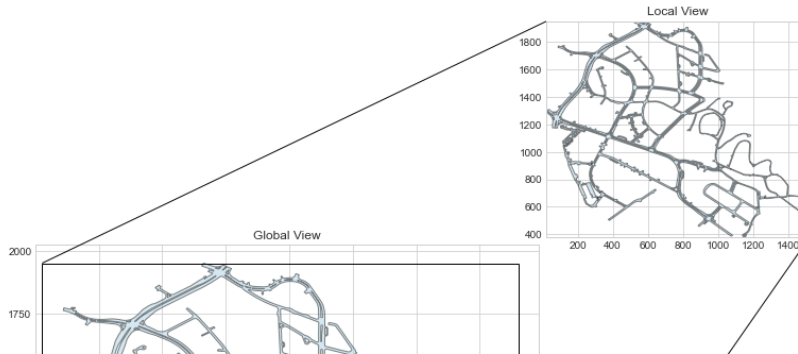
### a. Drivable Area

Drivable area is defined as the area where the car can drive, without consideration for driving directions or legal restrictions. This is the only layer in which the record can be represented by more than one geometric entity. *Note: On some machines this polygon renders incorrectly as a filled black rectangle.*

```
In [37]: sample_drivable_area = nusc_map.drivable_area[0]
sample_drivable_area

Out[37]: {'token': 'c3e29556-b711-4581-9970-b66166fb907d',
'polygon_tokens': ['fff7b0c9-leaf-4988-afe3-e4e4607f85e3',
'd235013d-2a07-4181-9862-c666b49a79b4',
'0bbf311c-405d-433b-a097-7d9c292a9b87',
'b4dfb634-2721-42d9-aa5d-0f8ec9a2fa31',
'c4b4c925-6ddb-4e4b-a4ca-609e1ca626c2',
'a60970c7-86cd-4169-ae9a-b9b51e4ec950',
'1209379e-bc10-4d65-9fb1-0ee938032130']}

In [38]: fig, ax = nusc_map.render_record('drivable_area', sample_drivable_area['token'], other_layers=[])
```



또한, drivable area 같은 데이터는 주행 데이터가 기록된 장소에 해당하는 지도에 플로팅을 할 수 있는데 전체적인 상황을 쉽게 볼 수 있어 사용자에게 매우 편리하다.

여기까지 nuScenes에 대해 살펴보았다. nuScenes는 지금까지 살펴본 set 중 가장 좋은 set 이라고 생각한다. 친절하고 자세한 tutorial을 통해 데이터를 사람이 파악하기 쉬운 형태로 제공하며 각 센서가 자율주행에서 가지는 의미를 잘 이해할 수 있었다.

2. 자율주행 인지에 관련된 2종 이상의 Open Source 조사, 정리  
작성자는 다음 오픈소스를 선정하였다.

- cvlib
- Detectron2

cvlib

cvlib은 파이썬에서 설치하고 사용할 수 있는 YOLO 모델 기반 detection 라이브러리이다. YOLO는 You Only Look Once의 약자로 단 한 번의 관찰로 사물을 인지한다는 뜻이다. 인터넷 블로그에서 YOLO의 원리를 3단계로 설명하고 있다.[9]

1. 이미지 크기를 바꾼다.
2. 이미지에 대해 1×1 convolution을 수행한다.
3. 모델을 거쳐 나온 확률값 중 threshold 기준으로 잘라 결과를 반환한다.

YOLO는 빠른 속도를 가지고 있지만, 정확도가 낮다고 한다.[10]

cvlib에서는 YOLOv4 버전이 적용되어 메소드로 편리하게 이용할 수 있다. 이런 라이브러리를 통해 1번에서 알아보았던 object detection을 수행할 수 있다. 라이브러리 사용을 위해서는 OpenCV와 텐서플로를 사전에 설치할 것을 요구하고 있다. cvlib은 크게 3개의 기능을 이용할 수 있다고 공식 repository에서 설명하고 있다.[11] 알아보도록 하자.

## Face detection

사람의 안면을 bounding box 로 검출하는 메소드를 제공한다.

### Example :

```
import cvlib as cv
faces, confidences = cv.detect_face(image)
```

Seriously, that's all it takes to do face detection with `cvlib`. Underneath it is using OpenCV's `dnn` module with a pre-trained caffemodel to detect faces.

To enable GPU

```
faces, confidences = cv.detect_face(image, enable_gpu=True)
```

Checkout `face_detection.py` in `examples` directory for the complete code.

여기서 faces 는 bounding box 를 구성하는 좌표, confidences 는 detection 결과의 신뢰도이다. 쉽게 말하면 모델이 인식한 객체가 실제로 그 객체일 확률을 뜻한다. 이렇게 얻은 값은 opencv 라이브러리를 통해 이미지로 볼 수 있다.

## Gender detection

### Gender detection

Once face is detected, it can be passed on to `detect_gender()` function to recognize gender. It will return the labels (man, woman) and associated probabilities.

### Example

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

cvlib 은 단순한 안면 인식에 더 나아가 성별도 인지할 수 있다.



## Object detection

### Object detection

Detecting common objects in the scene is enabled through a single function call `detect_common_objects()`. It will return the bounding box co-ordinates, corresponding labels and confidence scores for the detected objects in the image.

#### Example :

```
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)
```

Underneath it uses [YOLOv4](#) model trained on [COCO dataset](#) capable of detecting 80 common objects in context.

80 가지 사물에 대해 detection 을 수행하는 메소드를 제공한다.

cvlib 은 인공지능과 관련된 응용 프로그램을 개발할 때 직접 학습을 수행할 모델을 설계할 필요가 없고 메소드로 캡슐화 된 라이브러리를 이용하여 큰 비용을 들이지 않고도 개발을 할 수 있다.

## Detectron2

Detectron2(이하 디텍트론)은 페이스북에서 개발한 오픈소스 딥러닝 프레임워크이다. 인터넷 블로그에서 디텍트론의 유용성에 대해 다음과 같이 설명하고 있다.

*"딥러닝 모델 연구를 흔히 아이(연구자)가 블록(레이어)를 안정적으로 쌓아 올리는 과정에 비유합니다. 그런데 아이가 밑바닥부터 블록을 쌓아 모델을 완성하는 작업은 굉장히 어렵습니다. 그렇기에 완성된 블록(detectron2)를 이용해 개발을 하면 굉장히 편리하게 개발을 할 수 있습니다. 또 Github 에 올라온 코드 중에는 잘못된 구현이나, 최적화가 안된 코드가 많습니다. Detectron2 는 FAIR 에서 체계적으로 최적화시킨 모델이며, benchmark 상에서도 다른 오픈소스보다 좋은 성능을 보였습니다."*

*Faster R-CNN 계열 모델 뿐만 아니라, FAIR 에서 최근 연구하는 논문 구현체를 Detectron2 에 업데이트 하고 있습니다."*[12]

인용을 정리하면 디텍트론은 페이스북 인공지능 연구소인 FAIR 에서 제공하는 정확하고 신뢰할 수 있으며, 최적화를 거친 고성능 구현체를 빠르게 업데이트한다는 것이다.

ppwwyyxx and facebook-github-bot add mmdet wrapper tests ...			8bbc983 2 days ago	🕒 1,078 commits
📁 .circleci	update pytorch version			29 days ago
📁 .github	update docs			7 days ago
📁 configs	fix num_classes in keypoint rcnn test			2 months ago
📁 datasets	update docs			3 months ago
📁 demo	Fixed small issue with color-format conversion			2 months ago
📁 detectron2	use configurable for RRPN and TTA mapper			2 days ago
📁 dev	v0.4 release			22 days ago
📁 docker	correct docker command			26 days ago
📁 docs	update docs			7 days ago
📁 projects	Add BitMask converter for CSE output			3 days ago
📁 tests	add mmdet wrapper tests			2 days ago
📁 tools	wrapper of mmdet backbones			9 days ago
📄 .clang-format	Cut FOR_EACH_KV			10 months ago
📄 .flake8	support relative imports in config file			2 days ago
📄 .gitignore	update docs			3 months ago
📄 GETTING_STARTED.md	update docs			3 months ago
📄 INSTALL.md	update docs			7 days ago
📄 LICENSE	update copyright header			5 months ago
📄 MODEL_ZOO.md	Add PointRend R101 and X101 results to model zoo			5 months ago
📄 README.md	update docs			3 months ago
📄 setup.cfg	support relative imports in config file			2 days ago
📄 setup.py	update docs			7 days ago

공식 repository 에 들어가면 디텍트론의 디렉터리 구조를 살펴볼 수 있다. 디텍트론은 training loop 를 구현하지 않고 engine 이라는 형태로 학습을 수행한다고 한다. 추상화를 통해 개발자는 모델 개발에 집중할 수 있게 된다. 그리고 그 engine 이 바로 detectron2 폴더에 들어있다. 그리고 tools 디렉터리에는 train\_net.py 와 plain\_train\_net.py 가 존재하는데 이 두 파이썬 코드를 실행하여 학습을 실행한다고 한다. 두 코드의 차이는 다음과 같다고 한다. *“plain\_train\_net.py* 파일은 추상화가 덜 되있어 구조 파악이 쉽지만, SGD-momentum 을 이용한 학습만 지원하고 나머지 기능들은 지원하지 않는다고 적혀있습니다.

*train\_net.py* 파일은 training loop 가 추상화가 되있어 실제 training loop 가 어떻게 되는지 파악이 힘듭니다. Engine 이라는 것을 사용해 학습을 하며, learning rate warmup 과 같은 기능들을 사용해 학습을 진행합니다. 즉 이 engine 을 이용하면 사용자는 학습과정은 신경쓰지 않고 오로지 모델과 loss 만 신경쓰면 되는 구조입니다.[12]

디텍트론도 코랩을 통해 실행할 수 있는 튜토리얼을 제공한다.

```
[ ] # We can use `Visualizer` to draw the predictions on the image.
v = Visualizer(im[:, :, :-1], MetadataCatalog.get(cfg.DATASETS.TRAIN[0]), scale=1.2)
out = v.draw_instance_predictions(outputs["instances"].to("cpu"))
cv2.imshow(out.get_image()[:, :, :-1])
```



사진에 대해 Semantic segmentation 을 수행한 모습이다.

▼ Train!

Now, let's fine-tune a COCO-pretrained R50-FPN Mask R-CNN model on the balloon dataset. It takes ~6 minutes to train 300 iterations on Colab's K80 GPU, or ~2 minutes on a P100 GPU.

```
from detectron2.engine import DefaultTrainer

cfg = get_cfg()
cfg.merge_from_file(model_zoo.get_config_file("COCO-InstanceSegmentation/mask_rcnn_R_50_FPN_3x.yaml"))
cfg.DATASETS.TRAIN = ("balloon_train",)
cfg.DATASETS.TEST = ()
cfg.DATALOADER.NUM_WORKERS = 2
cfg.MODEL.WEIGHTS = model_zoo.get_checkpoint_url("COCO-InstanceSegmentation/mask_rcnn_R_50_FPN_3x.yaml") # Let trainin
cfg.SOLVER.IMS_PER_BATCH = 2
cfg.SOLVER.BASE_LR = 0.00025 # pick a good LR
cfg.SOLVER.MAX_ITER = 300 # 300 iterations seems good enough for this toy dataset; you will need to train longer for
cfg.SOLVER.STEPS = [] # do not decay learning rate
cfg.MODEL.ROI_HEADS.BATCH_SIZE_PER_IMAGE = 128 # faster, and good enough for this toy dataset (default: 512)
cfg.MODEL.ROI_HEADS.NUM_CLASSES = 1 # only has one class (balloon). (see https://detectron2.readthedocs.io/tutorials/da
# NOTE: this config means the number of classes, but a few popular unofficial tutorials incorrect uses num_classes+1 he

os.makedirs(cfg.OUTPUT_DIR, exist_ok=True)
trainer = DefaultTrainer(cfg)
trainer.resume_or_load(resume=False)
trainer.train()
```

예제 코드를 통해 학습을 수행할 수 있다.

```
[04/03 16:09:41 d2.data.build]: Removed 0 images with no usable annotations. 61 images left.
[04/03 16:09:41 d2.data.build]: Distribution of instances among all 1 categories:
| category | #instances |
|:-----:|:-----:|
| balloon | 255        |
|         |           |

[04/03 16:09:41 d2.data.dataset_mapper]: [DatasetMapper] Augmentations used in training: [ResizeShortestEdge(short_edge
[04/03 16:09:41 d2.data.build]: Using training sampler TrainingSampler
[04/03 16:09:41 d2.data.common]: Serializing 61 elements to byte tensors and concatenating them all ...
[04/03 16:09:41 d2.data.common]: Serialized dataset takes 0.17 MiB
Skip loading parameter 'roi_heads.box_predictor.cls_score.weight' to the model due to incompatible shapes: (81, 1024)
Skip loading parameter 'roi_heads.box_predictor.cls_score.bias' to the model due to incompatible shapes: (81,) in the
Skip loading parameter 'roi_heads.box_predictor.bbox_pred.weight' to the model due to incompatible shapes: (320, 1024)
Skip loading parameter 'roi_heads.box_predictor.bbox_pred.bias' to the model due to incompatible shapes: (320,) in the
Skip loading parameter 'roi_heads.mask_head.predictor.weight' to the model due to incompatible shapes: (80, 256, 1, 1)
Skip loading parameter 'roi_heads.mask_head.predictor.bias' to the model due to incompatible shapes: (80,) in the che
[04/03 16:09:42 d2.engine.train_loop]: Starting training from iteration 0
[04/03 16:09:52 d2.utils.events]: eta: 0:02:12 iter: 19 total_loss: 2.072 loss_cls: 0.6865 loss_box_reg: 0.6507
[04/03 16:10:01 d2.utils.events]: eta: 0:02:02 iter: 39 total_loss: 2.007 loss_cls: 0.6414 loss_box_reg: 0.6578
[04/03 16:10:10 d2.utils.events]: eta: 0:01:50 iter: 59 total_loss: 1.951 loss_cls: 0.5681 loss_box_reg: 0.6574
[04/03 16:10:20 d2.utils.events]: eta: 0:01:42 iter: 79 total_loss: 1.732 loss_cls: 0.4875 loss_box_reg: 0.7051
[04/03 16:10:29 d2.utils.events]: eta: 0:01:32 iter: 99 total_loss: 1.608 loss_cls: 0.4252 loss_box_reg: 0.6402
[04/03 16:10:38 d2.utils.events]: eta: 0:01:23 iter: 119 total_loss: 1.51 loss_cls: 0.388 loss_box_reg: 0.6972
[04/03 16:10:48 d2.utils.events]: eta: 0:01:14 iter: 139 total_loss: 1.381 loss_cls: 0.3568 loss_box_reg: 0.6706
[04/03 16:10:58 d2.utils.events]: eta: 0:01:06 iter: 159 total_loss: 1.314 loss_cls: 0.3177 loss_box_reg: 0.6578
[04/03 16:11:08 d2.utils.events]: eta: 0:00:57 iter: 179 total_loss: 1.238 loss_cls: 0.2713 loss_box_reg: 0.6229
[04/03 16:11:18 d2.utils.events]: eta: 0:00:48 iter: 199 total_loss: 1.158 loss_cls: 0.2367 loss_box_reg: 0.6788
[04/03 16:11:28 d2.utils.events]: eta: 0:00:38 iter: 219 total_loss: 1.133 loss_cls: 0.2394 loss_box_reg: 0.6325
[04/03 16:11:38 d2.utils.events]: eta: 0:00:29 iter: 239 total_loss: 0.9481 loss_cls: 0.1828 loss_box_reg: 0.6104
[04/03 16:11:48 d2.utils.events]: eta: 0:00:19 iter: 259 total_loss: 0.9236 loss_cls: 0.1601 loss_box_reg: 0.5616
[04/03 16:11:58 d2.utils.events]: eta: 0:00:09 iter: 279 total_loss: 0.8822 loss_cls: 0.1512 loss_box_reg: 0.5319
[04/03 16:12:08 d2.utils.events]: eta: 0:00:00 iter: 299 total_loss: 0.7682 loss_cls: 0.1234 loss_box_reg: 0.4728
[04/03 16:12:09 d2.engine.hooks]: Overall training speed: 298 iterations in 0:02:23 (0.4817 s / it)
[04/03 16:12:09 d2.engine.hooks]: Total training time: 0:02:25 (0:00:01 on hooks)
```

iter 가 늘어날 수록 loss 가 떨어지는 것을 실시간으로 관찰할 수 있다.

여기까지 디텍트론에 대해 알아보았다. 디텍트론은 FAIR 에서 개발하고 최적화를 한 모델을 이용할 수 있는 아주 믿음직한 프레임워크이다. 개발자는 추상화된 학습 과정을 뒤로 하고 모델 개발에 역량을 집중할 수 있다.



### 3. 2 의 정리한 코드 중 하나 실행해서 결과 확인

작성자는 cvlib 을 통해 몇 장의 사진을 대상으로 object detection 을 수행하는 코드를 구현했다. 해당 코드는 작성자의 깃허브 repository 의 hw1 디렉터리에서 yolov4\_example.py 라는 이름으로 되어있다.

<https://github.com/nicotina04/kmu-autonomous2021-20171717/tree/master/hw1>

```
1  import tensorflow
2      import cv2
3      import cvlib
4  import sys
5
6
7  # Print python environment
8  def get_env():
9      print("Version of tf ->", tensorflow.__version__)
10     print("Version of opencv-python ->", cv2.__version__)
11     print("Version of python ->", sys.version_info)
12
13
14  if __name__ == "__main__":
15     """Print my environment"""
16     get_env()
17
18     """Import object_detection (YOLO)"""
19     from cvlib.object_detection import draw_bbox
20
21     try:
22         """Read an image file and predict"""
23         img_path = 'doll.jpeg'
24         img_read = cv2.imread(img_path)
25         """Detect objects"""
26         bbox, label, conf = cvlib.detect_common_objects(img_read)
27         """Print and draw the result"""
28         print(bbox, label, conf)
29         cv2.imwrite('result.jpg', draw_bbox(img_read, bbox, label, conf))
30     except AttributeError:
31         print("Couldn't find a image file. Exit the program")
32
```

cvlib 을 이용해 object detection 을 수행하는 코드이다. 코드를 실행하는 데 사용된 환경은 다음과 같다.

[python: 3.8.2, opencv-python: 4.5.1, cvlib, tensorflow: 2.4.1]

이는 코드에 있는 get\_env() 함수를 통해 확인할 수 있다. main 함수를 살펴보면 cvlib 의 object\_detection 에서 draw\_bbox 라는 메소드를 import 한 것을 볼 수 있는데 이 메소드는 object detection 의 결과를 bounding box 로 이미지에 입혀주는 함수이다. 그리고 계속해서 코드를 보면 opencv 의 imread() 를 통해 이미지를 불러온다. 그다음 2 번에서 설명한 detect\_common\_objects() 메소드를 호출하여 얻어낸 결과를 이미지로 저장한다.



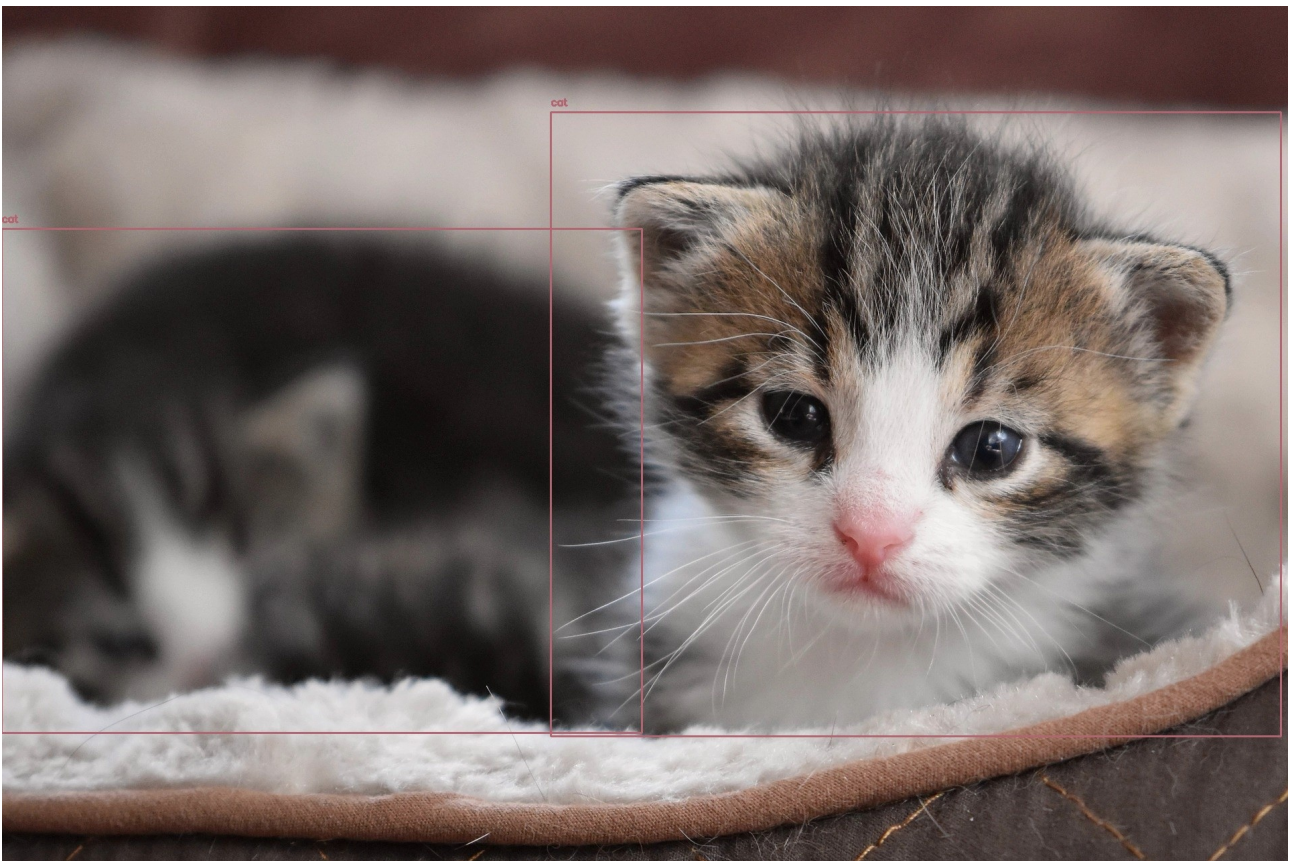
```
Downloading yolov4.cfg from https://raw.githubusercontent.com/AlexeyAB/darknet/master/cfg/yolov4.cfg
Downloading yolov4.weights from https://github.com/AlexeyAB/darknet/releases/download/darknet_yolo_v3
20% | #####
```

이 코드를 최초로 실행하면 인터넷에서 yolov4 설정을 다운로드한다.

```
/home/nicotina04/school/kmu-autonomous2021-20171717/venv/bin/python /home/nicotina04/school/kmu-a
2021-04-03 16:36:31.969121: W tensorflow/stream_executor/platform/default/dso_loader.cc:60] Could
2021-04-03 16:36:31.969133: I tensorflow/stream_executor/cuda/cudart_stub.cc:29] Ignore above cud
Version of tf -> 2.4.1
Version of opencv-python -> 4.5.1
Version of python -> sys.version_info(major=3, minor=8, micro=5, releaselevel='final', serial=0)
[[292, 98, 1691, 2055]] ['teddy bear'] [0.9425265192985535]

Process finished with exit code 0
```

다운로드를 완료하면 코드를 실행한 다음 결과 이미지를 result.jpg 로 저장하게 된다.



고양이를 판별한 결과 이미지. 작성자는 뒤에 검은 형체가 고양이인 줄도 몰랐는데 bounding box 가 잡힌 것을 보고 나서야 인식하게 되었다.



집에 있는 인형을 인식한 결과 이미지. teddy bear 로 인식이 잘 되었음을 확인할 수 있다.





표범을 giraffe(기린)으로 인식한 건 좀 아쉬웠다.



부두에 놓인 자동차들을 인식한 결과는 맨 밑에 두 대를 제외하고는 아예 부두 영역을 car 라고 인식해버렸다. 지금까지 자율주행을 위한 data set 과 open source library 를 조사했다. 그리고 그 중 cvlib 을 통해 object detection 을 한 결과를 보았다. 앞으로 강의를 통해 배울 내용이 기대된다.

#### 4. 참고 자료

- [1] Yu 외 14 명, "BDD100K: A Diverse Driving Dataset for Heterogeneous Multitask Learning", 2020.
- [2] <https://scale.com/open-datasets/pandaset>
- [3] <https://www.nuscenes.org/>
- [4] <https://github.com/ucbdrive/bdd100k/blob/master/doc/format.md>
- [5] <https://bskyvision.com/491>
- [6] [https://www.jeremyjordan.me/semantic-segmentation/#dilated\\_convolution](https://www.jeremyjordan.me/semantic-segmentation/#dilated_convolution)
- [7] <https://pytorchhair.gitbook.io/project/introduction/semantic-segmentation>
- [8] <https://github.com/scaleapi/pandaset-devkit>
- [9] <https://hohodu.tistory.com/15>
- [10] <https://curt-park.github.io/2017-03-26/yolo/>
- [11] <https://github.com/arunponnusamy/cvlib>
- [12] <https://chacha95.github.io/2020-04-28-Object-Detection5/>