차량지능기초 과제1

자동차IT융합학과 20173392 김원준

Github 주소 : https://github.com/culigan3186/autonomous_vehicle_assignment1

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

1-1) BDD100K

BDD100K 는 Berkeley Deep Drive의 약자로 40초의 비디오 시퀀스, 720픽셀 해상도, 초당 30 프레임 고화질로 취득된 100,000개 비디오 시퀀스로 구성됩니다.

Video data : 10만개 비디오 데이터는 1100시간 다양한 날씨, 주행시나리오에서 테스트한 영상을 갖고 있다. GPS, IMU의 데이터 또한 포함하고 있다.

Road Object Detection : 10만개의 사진에 버스, 신호등, 표지판, 사람, 자전거, 자동차등에 2D 바운딩박스를 표시한다.

Instance Segmentation : 1만개 이상의 다양한 이미지를 높은 instance-level annotation과 pixel-level로 탐색한다.

40초의 비디오 시퀀스, 720 픽셀 해상도, 초당 30프레임의 고화질로 취득된 10만개 비디오 시퀀스 데이터셋. 운전 도구 데이터베이스에서 의미론적 세분화 및 차선 레이블링 속도를 높이려고 annotation tool을 사용함,

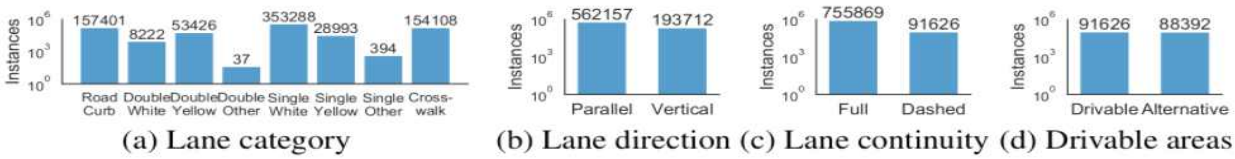
이미지에 라벨을 처리할 때 3가지 기본 레벨이 있다. (1. image-level tagging, 2. bounding box, 3. polygon annotation)

Semantic Instance Segmentation : 전체 데이터셋에서 무작위로 샘플링한 5,683개의 비디오 클립의 이미지에 대해 픽셀 레벨의 세분화된 주석을 제공. 각 픽셀에는 이미지의 객체 라벨의 instance 번호를 나타내는 해당 식별자와 라벨이 주어짐. 전체 라벨 세트는 도로 장면에서 객체의 다양성을 포착하고 각 이미지의 라벨 픽셀 수를 최대화하기 위해 선택된 40개의 객체 클래스로 구성됨.

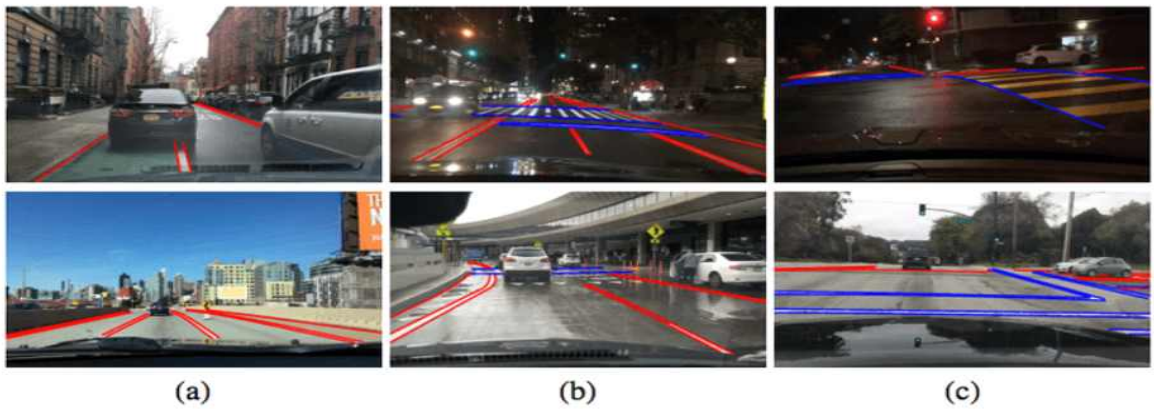
전체 데이터는 training(3,683장), validation(500장), testing(1,500장)으로 구성되어 있습니다.

BDD 데이터셋은 bike, bus, car, motor, person, rider, traffic light, traffic sing, train, truck 총 10가지 class로 구성이 되어있고 학습, 검증, 테스트 데이터셋이 나누어져 있습니다.

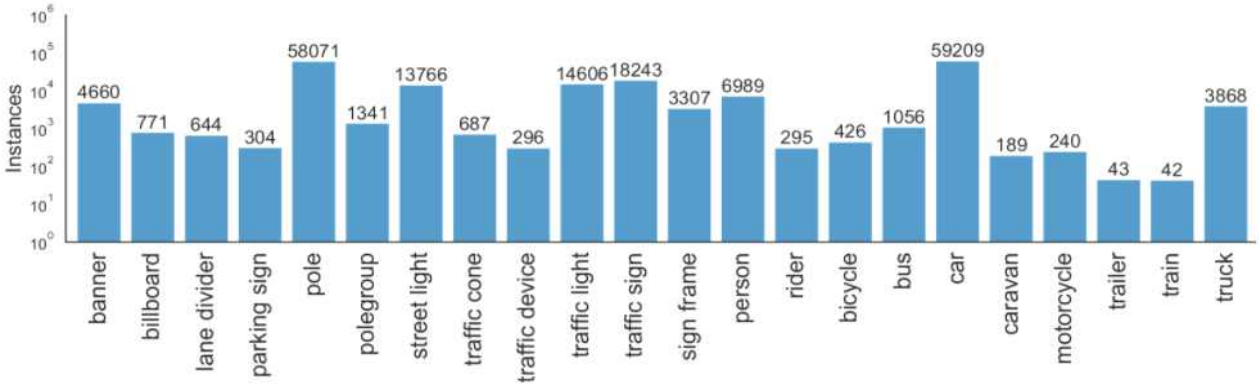
Lane

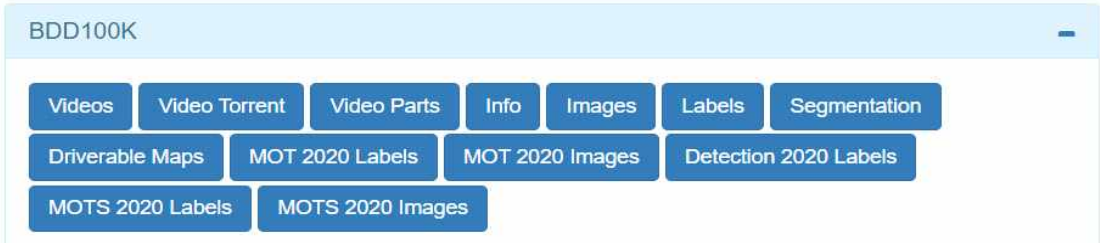


(a) Lane category (b) Lane direction (c) Lane continuity (d) Drivable areas

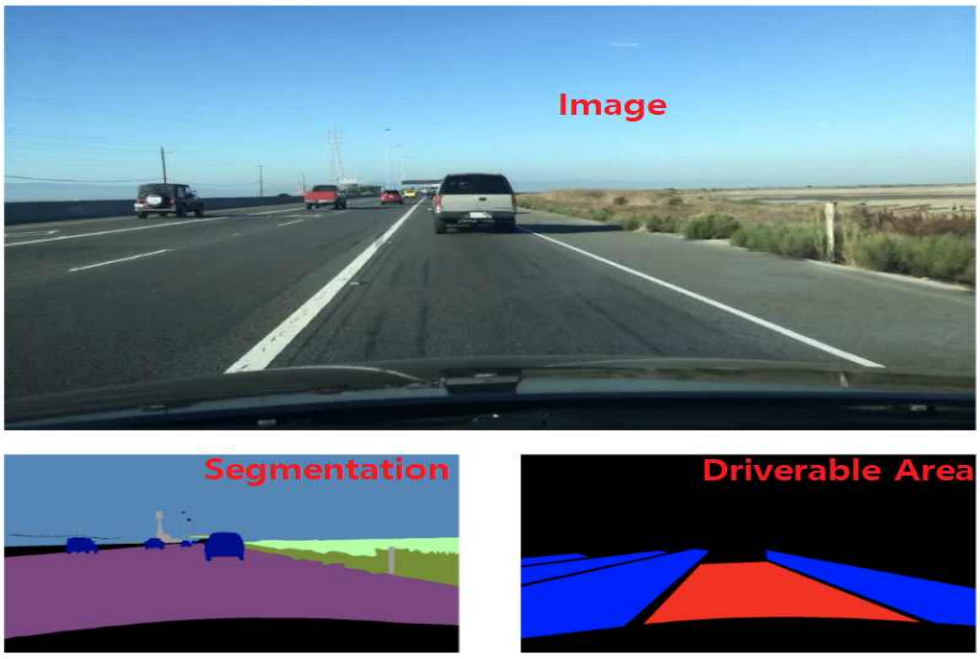


(a) (b) (c)





데이터셋에는 Videos,Video Torrent, Video Parts, Info, Images, Labels등 으로 이루어져 있습니다.
각 항목을 클릭하면 download 받을 수 있습니다.
Images, Segmentation, Driverable Maps 이렇게 세 폴더는 사진과 segmentation된 사진을 가지고 있습니다.

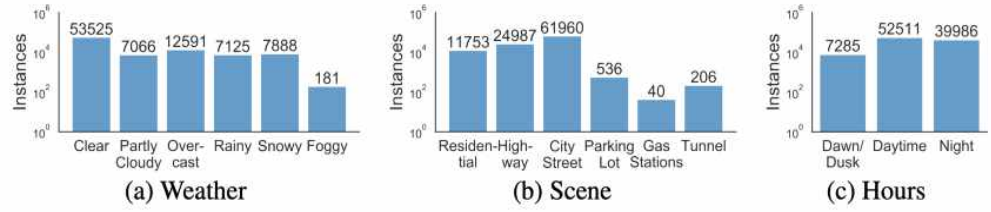


Image는 1280x720 RGB 사진이고, Segmentation은 각 객체를 픽셀단위로 표시합니다. Driverable Area (Maps)은 차선을 기준으로 주행중인 차선은 빨간색으로 표시하고 다른 차선은 파란색으로 표시합니다.
Label을 클릭하면 두 개의 json 파일을 다운받을 수 있습니다. 하나는 training 파일이고, 다른 하나는 validatation 파일입니다.

train 파일의 json을 보면 아래와 같이 train 이미지에 해당하는 환경정보를 제공합니다.


```
"name": "0000f77c-6257be58.jpg",
"attributes": {
    "weather": "clear",
    "scene": "city street",
    "timeofday": "daytime"
},
```

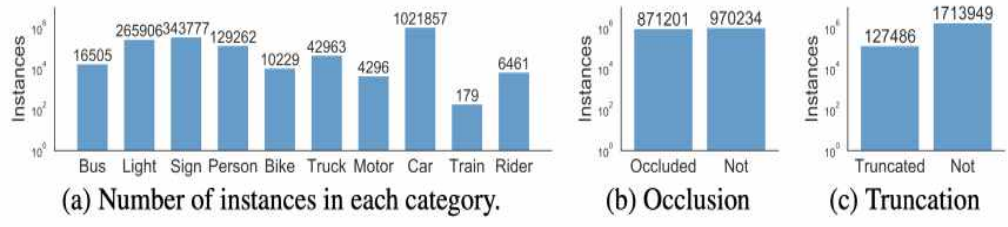

BDD100K의 데이터셋은 다양한 환경에서 골고루 비교적 균일하게 데이터를 갖고 있습니다.

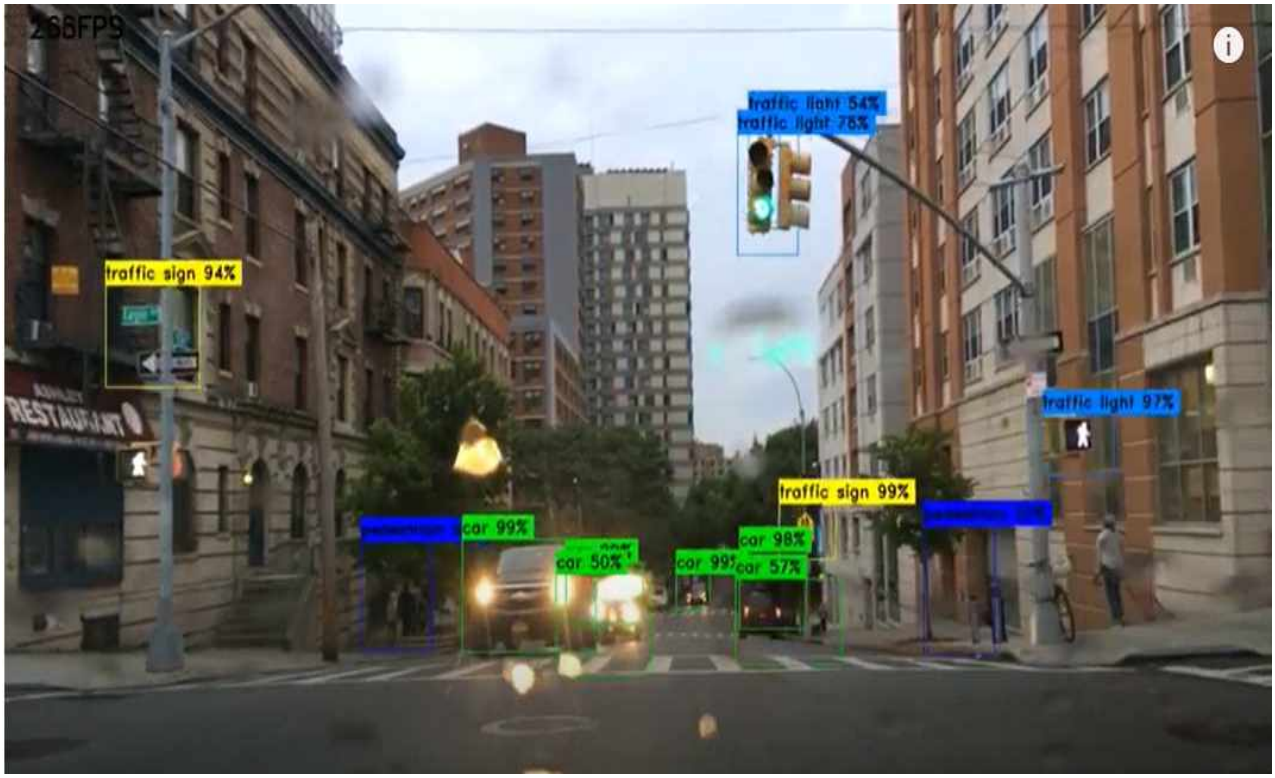


json 파일을 보면 객체 인식에 대한 정보도 제공하는데, bounding box의 정보와 그 외 정보를 갖고 잇습니다.


```
"timestamp": 10000,
"labels": [
    {
        "category": "traffic light",
        "attributes": {
            "occluded": false,
            "truncated": false,
            "trafficLightColor": "green"
        },
        "manualShape": true,
        "manualAttributes": true,
        "box2d": {
            "x1": 1125.902264,
            "y1": 133.184488,
            "x2": 1156.978645,
            "y2": 210.875445
        },
        "id": 0
    },
```


category는 10개로 구성되어 있고, 아래의 그래프와 같은 분포를 갖고 있습니다.





출처: https://www.youtube.com/watch?v=ANQczqZwaY4

1-2) CityScapes Dataset

semantic, instance-wise, dense pixel annotation된 라벨을 제공합니다. 라벨링은 8개 카테고리(평지,사람, 차량, 건축물, 객체, 자연, 하늘, void)로 그룹화된 30개의 클래스로 구성됩니다.

Class Definitions

Please click on the individual classes for details on their definitions.

Group	Classes
flat	road · sidewalk · parking+ · rail track+
human	person* · rider*
vehicle	car* · truck* · bus* · on rails* · motorcycle* · bicycle* · caravan*+ · trailer*+
construction	building · wall · fence · guard rail+ · bridge+ · tunnel+
object	pole · pole group+ · traffic sign · traffic light
nature	vegetation · terrain
sky	sky
void	ground+ · dynamic+ · static+

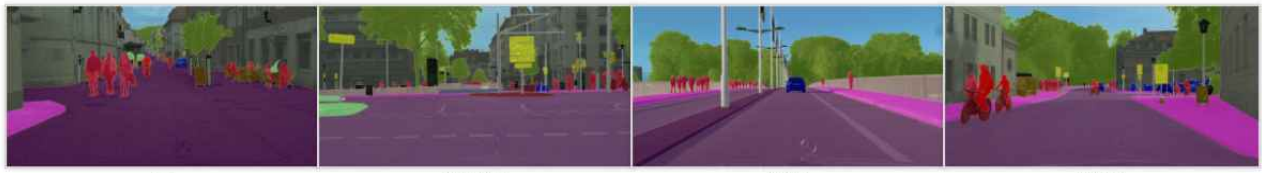
Cityscapes dataset에서는 데이터의 다양성을 위해서 계절, 밤낮, 날씨, 도시등을 다르게 했습니다. 그리고 5000개의 fine label 이미지들과 20000 coarse label 이미지들로 구성되어 있습니다.

Fine annotations

Below are examples of our high quality dense pixel annotations that we provide for a volume of 5 000 images. Overlayed colors encode semantic classes (see class definitions). Note that single instances of traffic participants are annotated individually.

Stuttgart *Zurich* *Ulm* *Tübingen*

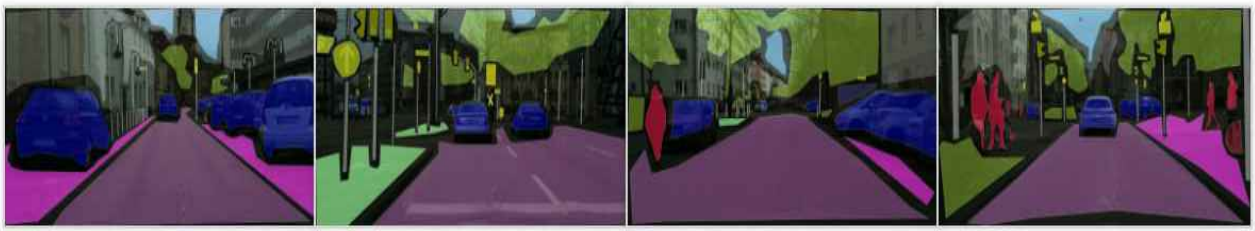
Münster *Cologne* *Bonn* *Erfurt*



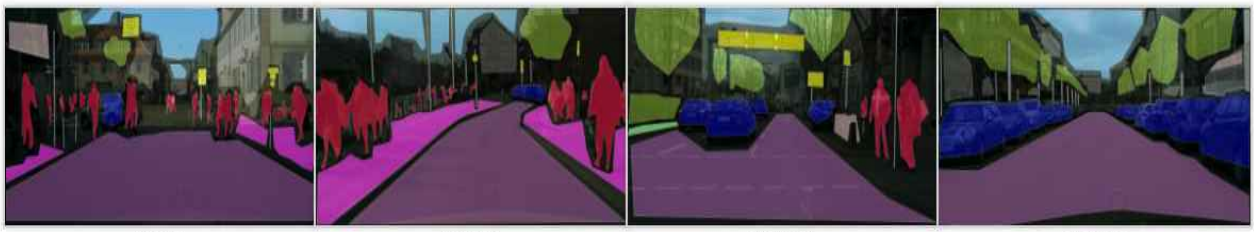
Jena *Düsseldorf* *Lindau* *Weimar*

Coarse annotations

In addition to the fine annotations, we provide coarser polygonal annotations for a set of 20 000 images in collaboration with Pallas Ludens. Again, overlayed colors encode the semantic classes (see class definitions). Note that we do not aim to annotated single instances, however, we marked polygons covering individual objects as such.



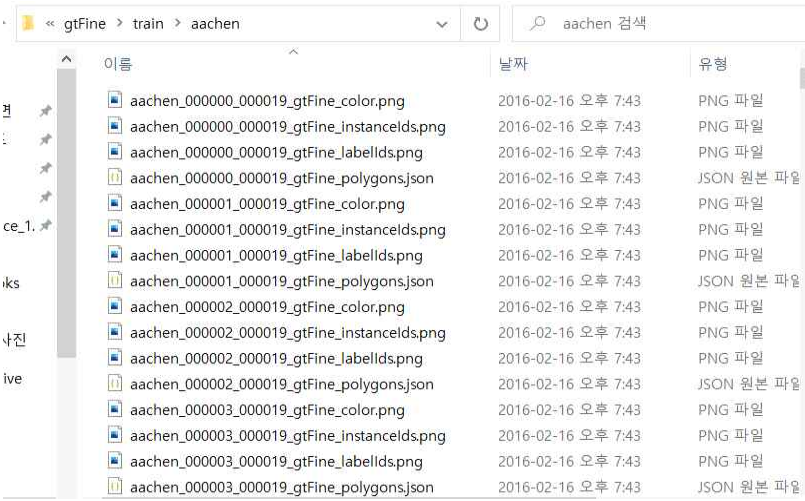
Saarbrücken *Saarbrücken* *Nuremberg* *Nuremberg*



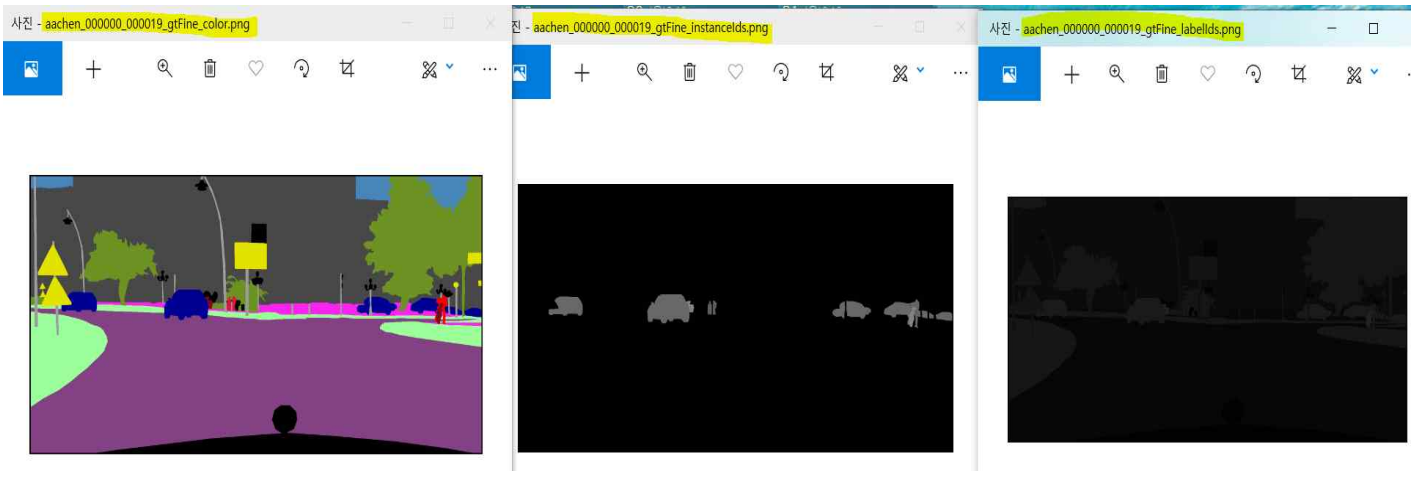
Erlangen *Bamberg* *Dortmund* *Dortmund*

아래 사진은 Cityscapes 데이터셋 중 일부를 트리의 형태로 보여줍니다. gtFine은 Fine annotation을 의미하며, train과 val 폴더로 구성됩니다. label 주석은 json 형태로 이루어져있습니다.
leftImg8bit는 왼쪽 사진의 8bit LDR format을 가진 파일로 이루어진 train, val 폴더를 갖고 있습니다.

```
$ tree -d
.
├── gtFine
│   ├── train
│   │   ├── aachen
│   │   ├── bochum
│   │   └── bremen
│   └── val
│       └── frankfurt
└── leftImg8bit
    ├── train
    │   ├── aachen
    │   ├── bochum
    │   └── bremen
    └── val
        └── frankfurt
```



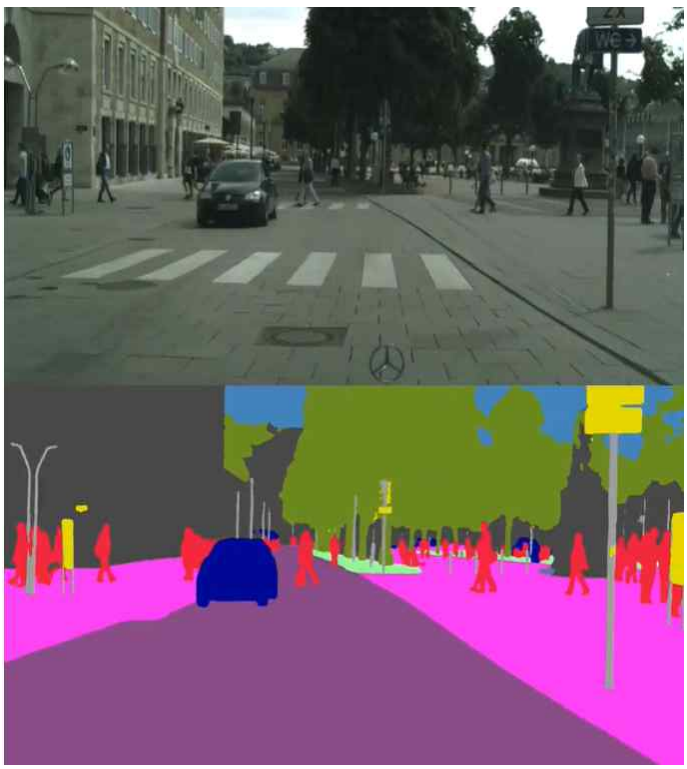
gtFine파일의 train/aachen에 들어가보면 color,instanceIds,labelIds 이미지 파일이 있고, json 파일이 있습니다. 각 이미지 파일을 열어보면 다음과 같습니다.



그리고 json 파일을 열면 다음과 같습니다. json 파일에서는 해당 이미지의 높이와 너비 그리고 인지한 객체의 이름을 label로 보여줍니다. segment 정보를 polygon으로 나타냅니다.

```
{
    "imgHeight": 1024,
    "imgWidth": 2048,
    "objects": [
        {
            "label": "road",
            "polygon": [
                [
                    0,
                    769
                ],
                [
                    290,
                    574
                ],
                [
                    93,
                    528
                ],
                [
                    0,
                    524
                ],
                [
                    0,
                    448
                ],
                [
                    0,
                    448
                ],
                [
                    210,
                    453
                ],
                [
                    511,
```


실제 데이터셋을 이용한 사례



https://www.youtube.com/watch?v=QrB7Np_8GXY

1-3) Waymo Open dataset

Waymo Open Dataset은 2019년 8월에 처음 출시되었습니다. 이 데이터셋은 고해상도 센서 데이터와 1,950개의 세그먼트를 label된 데이터로 구성되어 있습니다. 2021년 3월에 motion dataset을 포함했습니다. motion dataset은 object trajectories와 103,354개의 segment에 대응되는 3D Map으로 구성되어 있습니다.

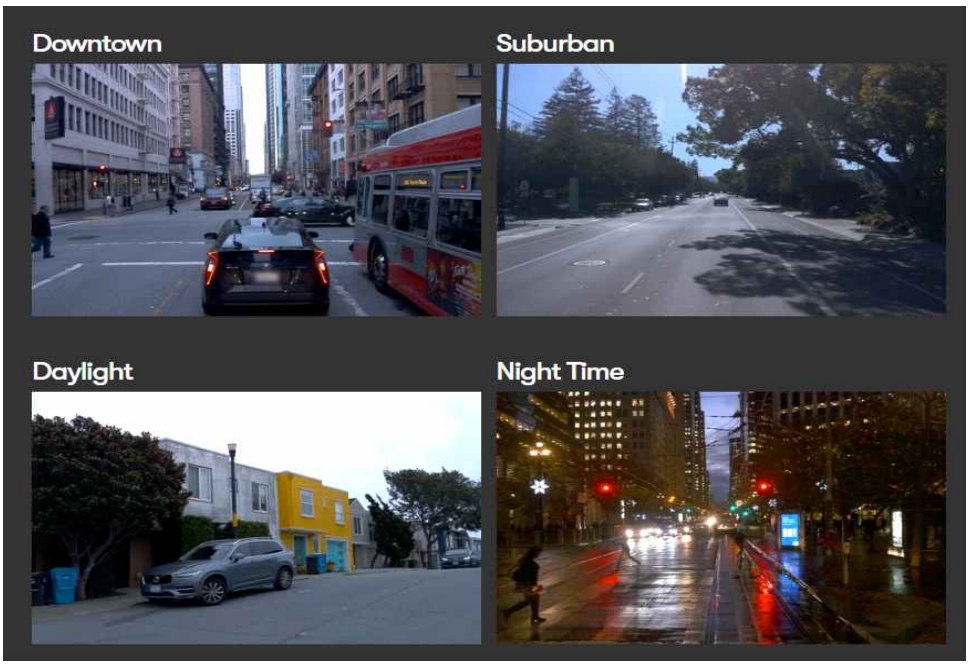
Motion Dataset

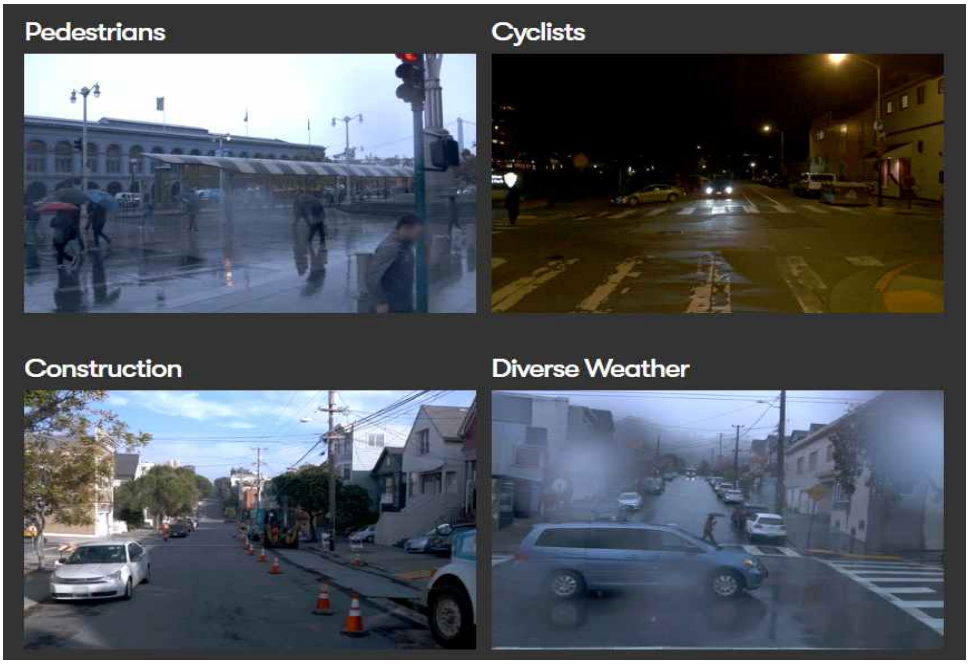
103,354, 20초, 10Hz 세그먼트 (프레임수가 2천만을 넘습니다.)
데이터 영상은 574시간입니다. Object Data는 1,080만 개의 objects로 이루어져 있으며, 각각 tracking ID가 있습니다. Object의 class는 3개로 이루어져 있습니다. Vehicles(차량), Pedestrains(보행자), Cyclists (자전거 타는사람)
3D bounding boxes를 각 객체마다 표시합니다. 각 세그먼트마다 3D map data가 있습니다.
Map data의 위치는 San Francisco, Phoenix, Mountain View, Los Angeles, Detroit, and Seattle로 구성되어 있습니다.
관련 코드는 https://github.com/waymo-research/waymo-open-dataset 주소에서 확인할 수 있습니다.

Perception Dataset (released Aug 2019, last updated March 2020)

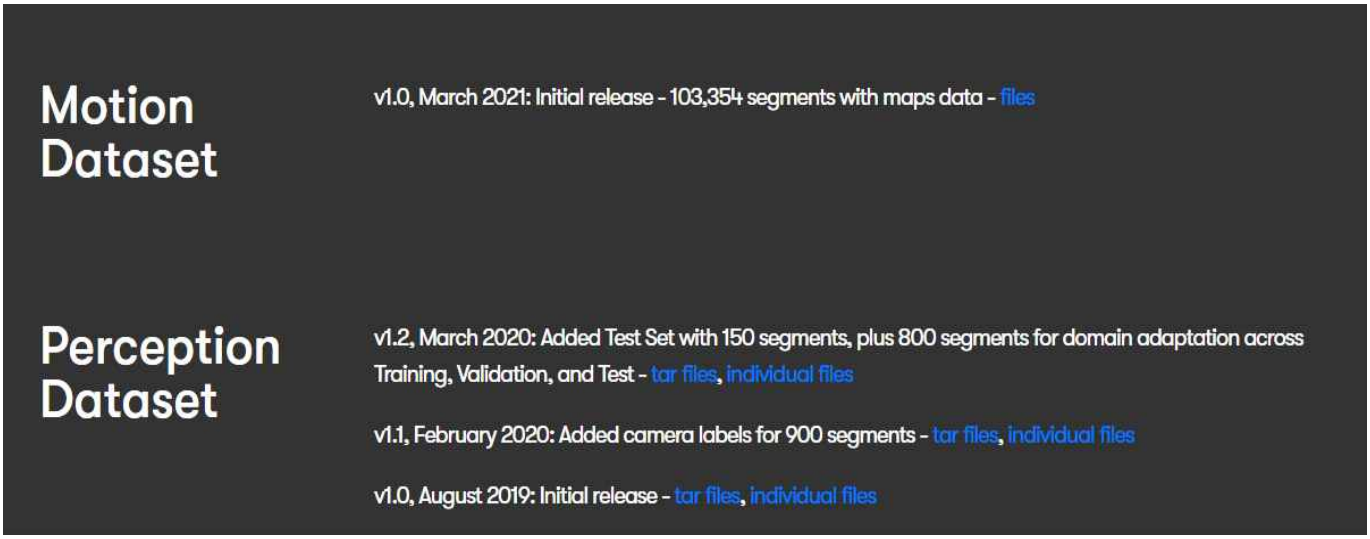
1,950 segments of 20s each, collected at 10Hz (390,000 frames) in diverse geographies and conditions. 구성은 Sensor data, Labeled data로 이루어져 있습니다.
Sensor data는 lidar와 camera 데이터를 포함하며, sensor calibration과 vehicle pose 또한 포함합니다.
Labeled data는 4개의 class로 object를 나누었습니다. Class는 Vehicles(차량), Pedestrains(보행자), Cyclists(자전거 타는 사람), Signs(표지판)로 이루어져 있습니다. 2D,3D 바운딩박스를 tracking ID와 함께 데이터를 제공합니다.

그리고 Downtown, Suburban, Daylight, Night Time, Pedestrians, Cylists, Construction, Diverse Weather 등 다양한 조건에 해당하는 데이터를 제공합니다.





Waymo Open Dataset 홈페이지 상단에 있는 Download를 클릭해 데이터셋을 확인했습니다.



이 중 Perception Dataset에서 최신 버전인 v1.2의 individual files를 확인해보았습니다.
domain_adaptation, testing, training, validation 총 4개의 폴더로 구성이 되어있었고, domain_adaptation 폴더 안에는 또 testing, training, validation이라는 폴더가 존재했습니다.
각각의 dataset의 역할은 training은 모델을 학습하기 위한 dataset이고, validation은 학습이 이미 완료된 모델을 검증하기 위한 dataset입니다. testing은 학습과 검증이 완료된 모델의 성능을 평가하기 위한 dataset입니다.
각 폴더에는 tfrecord라는 확장자를 가진 파일이 여러 개 있습니다. tfrecord 파일은 텐서플로우의 학습 데이터를 저장하기 위한 바이너리 데이터 포맷으로, 구글의 Protocol Buffer 포맷 데이터를 파일에 Serialize해서 저장합니다. CSV 파일에서 숫자나 텍스트를 읽는건 크게 지장이 없지만, 이미지 데이터를 읽을 경우 메타데이터와 라벨은 별도의 파일에 저장되어 있어서 데이터를 읽을 때 별도로 읽어 복잡해집니다.
tfreord를 사용하게 되면 데이터성능과 개발의 편의성을 높일 수 있는 장점이 있습니다.

출처: https://www.youtube.com/watch?v=ASqBKIrS6co

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

2-1) Udacity-CarND-Vehicle-Detection-and-Tracking

출처: https://github.com/harveenchadha/Udacity-CarND-Vehicle-Detection-and-Tracking

Udacity-CarND-Vehicle-Detection-and-Tracking 오픈소스는 차량을 인지하고, 차량이 있는 영역을 표시해줍니다.

<코드 전체 구조>

1. 이미지 읽기 – vehicle, non-vehicle Dataset 파일경로에 접근해 이미지 개수와 shape을 확인.
2. 2.색 검출과 HOG – RGB를 YUV로 변환하고, vehicle과 non-vehicle의 YUV 히스토그램을 확인. hog 함수를 이용하여 hog_feature과 hog_image를 구함.
3. Feature data 생성 – Dataset으로부터 Feature data를 생성함.
4. 데이터 전처리 – X_train, X_test, Y_train, Y_test 로 데이터를 분리하고, 데이터의 스케일을 조정.
5. 정의 및 학습 – LinearSVC를 활용하고, 정확도를 측정합니다.
6. 슬라이딩 윈도우 – 차량에 해당하는 부분에 박스를 그려줍니다.
7. Heatmap 적용 – heatmap에서 박스를 표시합니다.
8. window 개수 업데이트 – 최근 15개의 프레임만 저장할 수 있도록 업데이트합니다.
9. Pipeline 정의 – 앞에서 소개한 함수와 코드를 조합하여 연결된 동작과정을 나타냅니다.
10. Pipeline의 테스트 결과를 이미지로 확인
11. Pipeline의 테스트 결과를 동영상으로 확인

In [1]:
```
#코드 실행에 필요한 라이브러리를 import를 사용해 가져옵니다.
import glob
import cv2 as cv2
import numpy as np   # as를 이용해 numpy 모듈을 np로 지정해 전부 타이핑 하는 수고를 줄입니다.
import matplotlib.pyplot as plt
import random
from skimage.feature import hog # skimage.feature 모듈에서 hog만 가져옵니다.
```

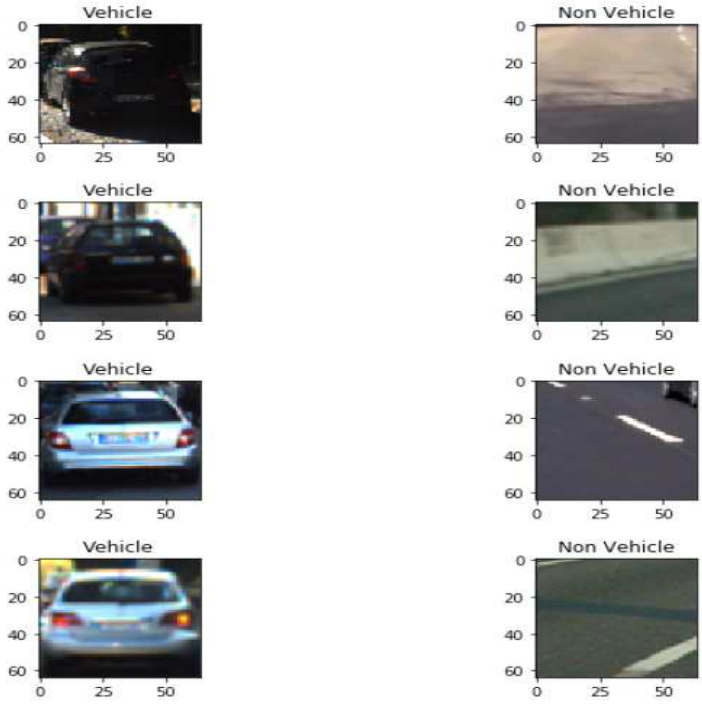
In [2]:
```
#reading image paths with glob
# glob을 이용해 이미지 파일의 경로명에 접근해 아래의 코드에서는 vehicles/vehicles/*/에서 확장자가 png인 파일만 추출합니다.
vehicle_image_arr = glob.glob('./vehicles/vehicles/*/*.png')

# read images and append to list
# images를 읽고 vehicle_images_original 리스트에 추가합니다.
vehicle_images_original=[]
for imagePath in vehicle_image_arr:
    readImage=cv2.imread(imagePath)
    rgbImage = cv2.cvtColor(readImage, cv2.COLOR_BGR2RGB)
    vehicle_images_original.append(rgbImage)

# image를 다 읽고, 리스트에 저장한 후 끝났다는 것을 알려주기 위해 print를 이용해 메시지를 출력합니다.
print('Reading of Vehicle Images Done')

#glob을 이용해 파일의 경로에 접근해 non-vehicles에 해당하는 이미지 파일들을 non_vehicle_image_arr에 저장합니다.
non_vehicle_image_arr = glob.glob('./non-vehicles/non-vehicles/*/*.png')

# images를 읽고, non_vehiloe_images_original 리스트에 추가합니다.
non_vehicle_images_original=[]
for imagePath in non_vehicle_image_arr:
    readImage=cv2.imread(imagePath)
    rgbImage = cv2.cvtColor(readImage, cv2.COLOR_BGR2RGB)
    non_vehicle_images_original.append(rgbImage)

print("Reading of Non Vehicle Images Done")

# vehicle_image_arr, non_vehicle_images_original 의 개수를 출력해 위의 코드가 제대로 실행되었는지 확인합니다.
print("No of Vehicle Images Loaded -"+ str(len(vehicle_image_arr)))
print("No of Non-Vehicle Images Loaded -"+ str(len(non_vehicle_images_original)))
```

Vehicle, Non-Vehicle의 이미지를 제대로 읽어왔는지 테스트한 결과

```
Reading of Vehicle Images Done
Reading of Non Vehicle Images Done
No of Vehicle Images Loaded -8792
No of Non-Vehicle Images Loaded -8968
```

In [3]:
```
# Visualizing the Vehicle and Non Vehicle Images

f, axes = plt.subplots(4,2, figsize=(10,10))
plt.subplots_adjust(hspace=0.5) # 서브 플롯간의 간격을 조절하기 위해 subplots_adjust() 함수를 이용함.
# hspace: The height of the padding between subplots, as a fraction of the average Axes height.

for index in range(4):
    vehicle=random.randint(0, len(vehicle_images_original)-1)
    # vehicle에  0부터 (vehicle_images_original의 개수)-1 중에서 랜덤으로 숫자 하나를 고른다.
    non_vehicle=random.randint(0, len(non_vehicle_images_original)-1)
    # vehicle에  0부터 (non_vehicle_images_original의 개수)-1 중에서 랜덤으로 숫자 하나를 고른다.
    axes[index,0].imshow(vehicle_images_original[vehicle])
    # [index,0] 에 vehicle_images_original중 vehicle 인덱스에 해당하는 이미지를 보여준다.
    #예를들어 vehicle=43010이라면 vehicle_images_original[4301]에 해당하는 이미지 보여줌
    axes[index,0].set_title("Vehicle") # 이미지에 해당하는 title을 Vehicle로 설정
    axes[index,1].imshow(non_vehicle_images_original[non_vehicle])
    axes[index,1].set_title("Non Vehicle")
print("Shape of Vehicle Image" +  str(vehicle_images_original[vehicle].shape))
# vehicle image의 shape를 출력해줍니다. 즉, (높이,폭,채널)의 순서로 나타난다.
print("Shape of Non Vehicle Image" +  str(non_vehicle_images_original[non_vehicle].shape))
# non_vehicle image의 shape를 출력해줍니다. 즉, (높이,폭,채널)의 순서로 나타난다.
```

```
Shape of Vehicle Image(64, 64, 3)
Shape of Non Vehicle Image(64, 64, 3)
```

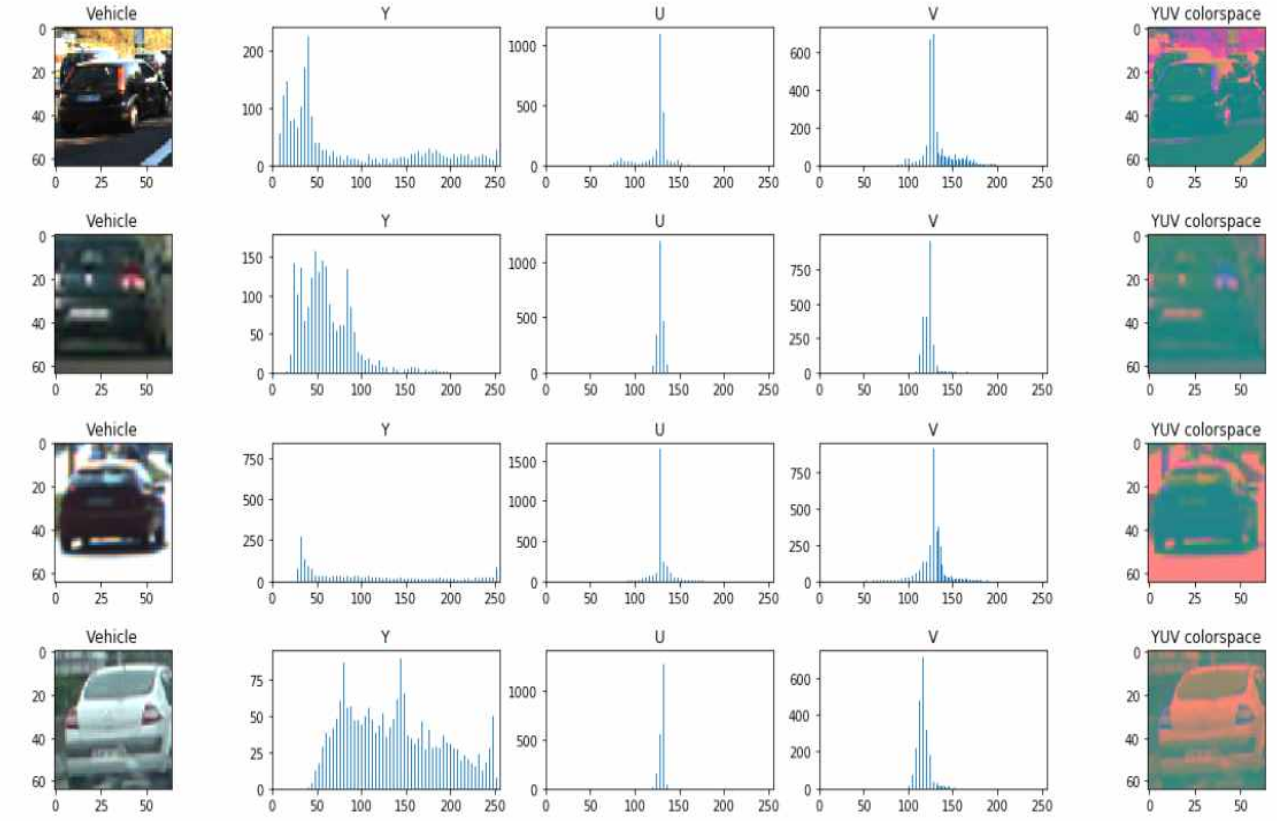


In [4]:
```
### Extract Color Space

#Histogram을 만듭니다.
# bins는 도수분포 구간을 나타냅니다.
def ExtractColorHistogram(image, nbins=32, bins_range=(0,255), resize=None):
    if(resize !=None):
        image= cv2.resize(image, resize)
    zero_channel= np.histogram(image[:,:,0], bins=nbins, range=bins_range)
    # image[:,:,0]은 0번 채널에 있는 모든 image를 histogram으로 표시.
    first_channel= np.histogram(image[:,:,1], bins=nbins, range=bins_range) # 1번 채널
    second_channel= np.histogram(image[:,:,2], bins=nbins, range=bins_range) # 2번 채널
    return zero_channel,first_channel, second_channel   # 순서대로 0,1,2 채널에 np.histogram의 결과를 반환합니다.

#Find Center of the bin edges
# bin edges의 중앙을 찾습니다.
def FindBinCenter(histogram_channel):
    bin_edges = histogram_channel[1]
    bin_centers = (bin_edges[1:]  + bin_edges[0:len(bin_edges)-1])/2   # bin_edges의 중앙을 계산합니다.
    return bin_centers

#Extracting Color Features from bin lengths
# concatenate 함수를 사용하여 zero_chnanel, first_channel, second_channel 배열을 합칩니다.
def ExtractColorFeatures(zero_channel, first_channel, second_channel):
    return np.concatenate((zero_channel[0], first_channel[0], second_channel[0]))
```

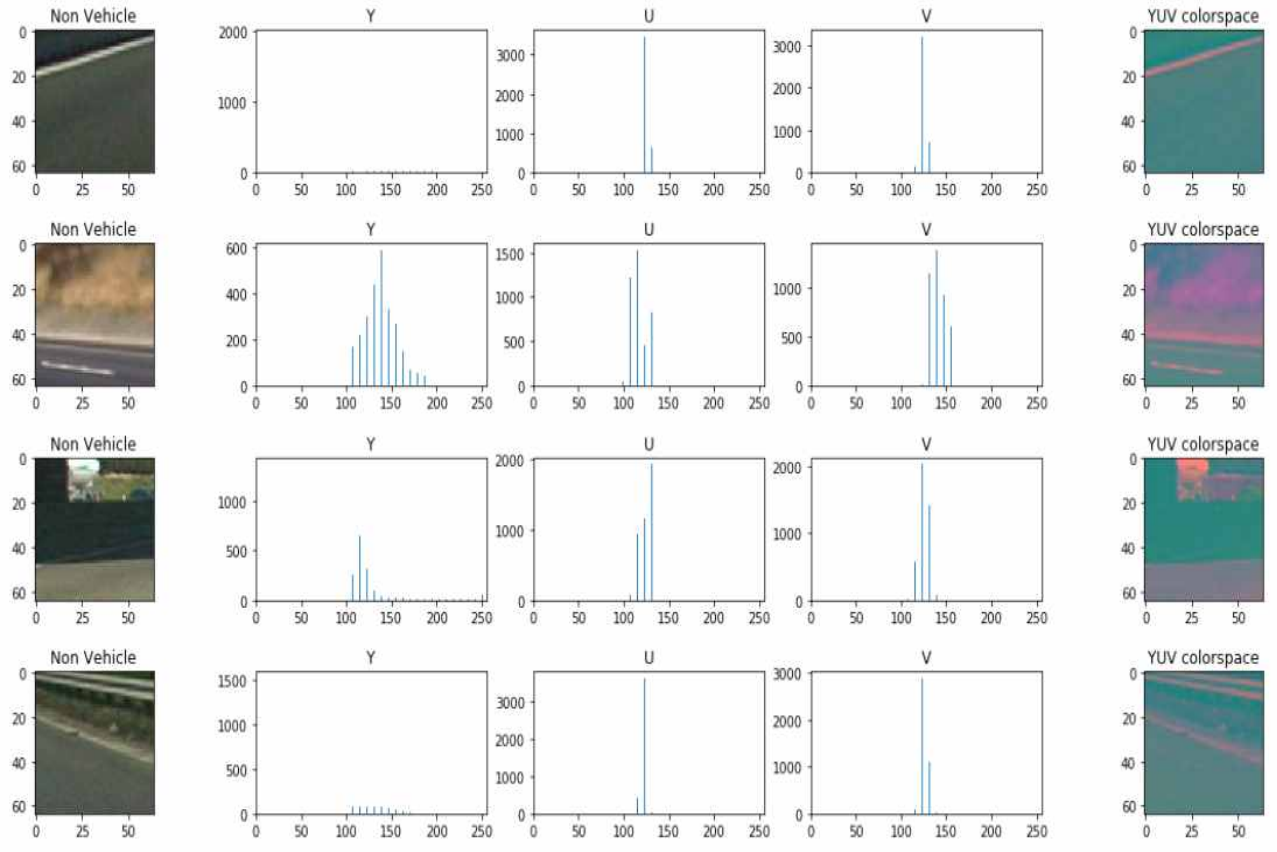
In [5]:
```
# Checking Color Features for Vehicles
# vehicles에 색상 요소를 확인합니다.

f, axes= plt.subplots(4,5, figsize=(20,10))  # 4행 5열에 해당하는 subplots을 만듭니다.
f.subplots_adjust(hspace=0.5)

for index in range(4):

    vehicle=random.randint(0, len(vehicle_images_original)-1)
    non_vehicle=random.randint(0, len(non_vehicle_images_original)-1)

    coloredImage= cv2.cvtColor(vehicle_images_original[vehicle],cv2.COLOR_RGB2YUV)   # RGB형태의 image를 YUV형태로 변환.
    r,g,b = ExtractColorHistogram(coloredImage,128)    # 0,1,2채널에 해당하는 히스토그램을 각각 r,g,b에 저장합니다.

    center= FindBinCenter(r)   # r 채널에서 center를 찾습니다.
    axes[index,0].imshow(vehicle_images_original[vehicle])  # random으로 구한 vehicle 인덱스에 해당하는 사진을 보여줍니다.
    axes[index,0].set_title("Vehicle")
    axes[index,1].set_xlim(0,256)   # x축의 한계를 0부터 256으로 지정합니다. 아래 그래프에 0~256까지만 x축에 있는것을 확인.
    axes[index,1].bar(center,r[0]) # bar는 막대그래프로 나타냄을 의미. 위에서 구한 center와 r[0]을 매개변수로 가짐
    axes[index,1].set_title("Y")   # Y를 타이틀로 합니다.
    axes[index,2].set_xlim(0,256)
    axes[index,2].bar(center,g[0])   # g는 coloredimage의 1번 채널
    axes[index,2].set_title("U")   # U를 타이틀로 합니다.
    axes[index,3].set_xlim(0,256)
    axes[index,3].bar(center,b[0])  # b는 coloredimage의 2번 채널
    axes[index,3].set_title("V")    # V를 타이틀로함.
    axes[index,4].imshow(coloredImage)     # coloredimage를 보여줍니다. 4열에 해당하는 맨 오른쪽 이미지에 해당하는 사진들입니다.
    axes[index,4].set_title("YUV colorspace")  # YUV colorspace를 타이틀로 합니다.

features = ExtractColorFeatures(r,g,b)   # features에 r,g,b 배열을 하나로 합칩니다.
print("No of features are "+ str(len(features))) # features의 개수를 출력합니다.
```

feature의 개수를 출력하고, vehicle의 YUV 히스토그램을 그립니다.

```
No of features are 384
```



In [6]:
```
# Checking Color Features for Non Vehicles
# Non Vehicles의 색상 요소를 확인합니다.

f, axes= plt.subplots(4,5, figsize=(20,10))  # 4행 5열에 해당하는 subplots을 만듭니다.
f.subplots_adjust(hspace=0.5)

for index in range(4):
    non_vehicle=random.randint(0, len(non_vehicle_images_original)-1)
    coloredImage= cv2.cvtColor(non_vehicle_images_original[non_vehicle],cv2.COLOR_RGB2YUV)  # RGB형태의 image를 YUV형태로 변환.
    r,g,b = ExtractColorHistogram(coloredImage)  # 0,1,2채널에 해당하는 히스토그램을 각각 r,g,b에 저장합니다.

    # Vehicle과 같은 방식으로 subplots에 나타냅니다.
    center= FindBinCenter(r)
    axes[index,0].imshow(non_vehicle_images_original[non_vehicle])
    axes[index,0].set_title("Non Vehicle")
    axes[index,1].set_xlim(0,256)
    axes[index,1].bar(center,r[0])
    axes[index,1].set_title("Y")
    axes[index,2].set_xlim(0,256)
    axes[index,2].bar(center,g[0])
    axes[index,2].set_title("U")
    axes[index,3].set_xlim(0,256)
    axes[index,3].bar(center,b[0])
    axes[index,3].set_title("V")
    axes[index,4].imshow(coloredImage)
    axes[index,4].set_title("YUV colorspace")
```



In [7]:
```
#Resizing Image to extract features, so as to reduce the feature vector size
# cv2.resize 함수를 이용하여 image를 size 변수에 해당하는 값으로 image size를 조절합니다.
# ravel() 함수는 다차원 배열을 1차원으로 바꾸어줍니다.
def SpatialBinningFeatures(image,size):
    image= cv2.resize(image,size)
    return image.ravel()
```

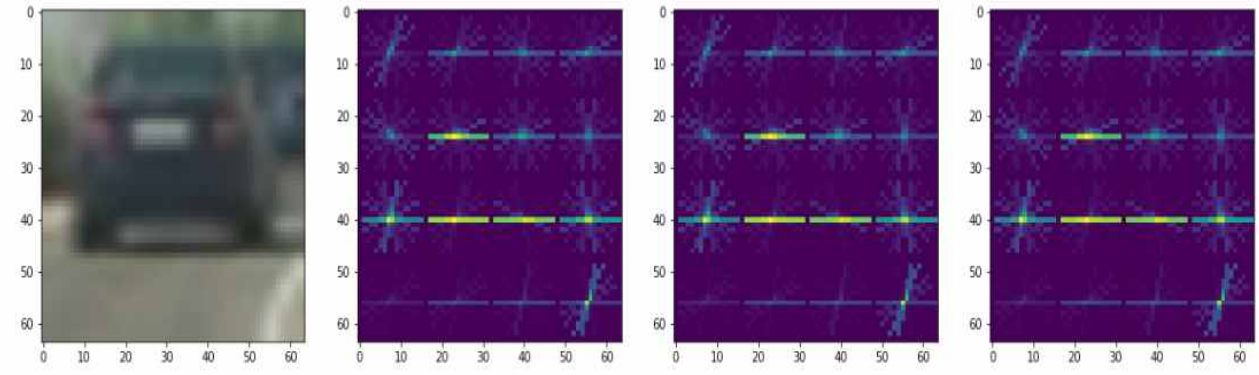
In [8]:
```
#testing the spatial binning
# 위의 SpatialBinningFeatures 함수가 제대로 작동하는지 확인해봅니다.
featureList=SpatialBinningFeatures(vehicle_images_original[1],(16,16)) # vehicle_images_original[1] 이미지를 (16,16) size로 변환
print("No of features before spatial binning",len(vehicle_images_original[1].ravel())) # 이미지를 1차원으로 변환 후 개수 출력
print("No of features after spatial binning",len(featureList)) # featureList의 개수 출력
```

```
No of features before spatial binning 12288
No of features after spatial binning 768
```

In [9]:
```
# General method to extract the HOG of the image
# HOG 방법을 사용해 이미지를 추출합니다.
# skimage.feature에 있는 hog 함수를 사용합니다. 맨 위에 from ~ import를 사용함.
def GetFeaturesFromHog(image,orient,cellsPerBlock,pixelsPerCell, visualise= False, feature_vector_flag=True):
    if(visualise==True): # visualize가 True일 때 실행되는 조건문
        hog_features, hog_image = hog(image, orientations=orient,
                        pixels_per_cell=(pixelsPerCell, pixelsPerCell),
                        cells_per_block=(cellsPerBlock, cellsPerBlock),
                        visualise=True, feature_vector=feature_vector_flag)
        # 각각의 매개변수는 hog를 이용하여 features로 바꾸기 위해 사용자가 지정하는 값입니다.
        #pixels_per_cell은 하나의 cell당 픽셀수를 나타낸다.
        #cells_per_block은 하나의 block당 cell 수를 나타냅니다.
        #hog_featrue, hog_image에 hog 함수를 이용하여 나온 값을 각각 저장합니다.
        return hog_features, hog_image

    else:         # visualize가 False일 때 실행되는 조건문
        hog_features = hog(image, orientations=orient,
                        pixels_per_cell=(pixelsPerCell, pixelsPerCell),
                        cells_per_block=(cellsPerBlock, cellsPerBlock),
                        visualise=False, feature_vector=feature_vector_flag)
        return hog_features    # hog_features를 구합니다.
```

In [10]:
```
# 위에서 만든  GetFeaturesFromHog 가 제대로 작동하는지 확인하기 위한 코드

image=vehicle_images_original[1]
image= cv2.cvtColor(image, cv2.COLOR_RGB2YUV)
image_channel_0=image[:,:,0]
image_channel_1=image[:,:,0]
image_channel_2=image[:,:,0]

#차례대로 image_channel , orinetation = 9, cellsPerBlock=2, pixelsPerCell=16, visualize= True, feature_vector_flag는 True로 지정해주었다.
feature_0,hog_img_0=GetFeaturesFromHog(image_channel_0,9,2,16,visualise=True,feature_vector_flag=True)
feature_1,hog_img_1=GetFeaturesFromHog(image_channel_1,9,2,16,visualise=True,feature_vector_flag=True)
feature_2,hog_img_2=GetFeaturesFromHog(image_channel_2,9,2,16,visualise=True,feature_vector_flag=True)

f, axes= plt.subplots(1,4,figsize=(20,10)) # 1행 4열의 subplots을 만든다.
axes[0].imshow(vehicle_images_original[1])
axes[1].imshow(hog_img_0)
axes[2].imshow(hog_img_1)
axes[3].imshow(hog_img_2)
# 각각hog_img 채널에 해당하는 그림 보여줍니다.

print("Feature Vector Length Returned is ",len(feature_0))    # Feature Vector의 길이를 출력해줍니다.
print("No of features that can be extracted from image ",len(hog_img_0.ravel())) # 1차원으로 변환했을 때 features의 개수를 출력합니다.
```

```
Feature Vector Length Returned is  324
No of features that can be extracted from image  4096
```



In [11]:
```
#Convert Image Color Space. Note the colorspace parameter is like cv2.COLOR_RGB2YUV
# image의 Color Space를 변환합니다. 예를들어, cv2.COLOR_RGB2YUV처럼 RGB를 YUV color space로 변환하는 것과 같습니다.
def ConvertImageColorspace(image, colorspace):
    return cv2.cvtColor(image, colorspace)
```

In [12]:
```
# Method to extract the features based on the choices as available in step 2
# featureList에 각 채널에 해당하는 feature를 합친 배열을 추가합니다.

def ExtractFeatures(images,orientation,cellsPerBlock,pixelsPerCell, convertColorspace=False):
    featureList=[]
    imageList=[]
    for image in images:
        if(convertColorspace==True):   #convertColorspace가 True일 때 RGB를 YUV Color Space로 변경해줍니다.
            image= cv2.cvtColor(image, cv2.COLOR_RGB2YUV)
        local_features_1=GetFeaturesFromHog(image[:,:,0],orientation,cellsPerBlock,pixelsPerCell, False, True)
        # visualize=False이고, 각각의 feature를 hog방법을 이용하여 구합니다.
        local_features_2=GetFeaturesFromHog(image[:,:,1],orientation,cellsPerBlock,pixelsPerCell, False, True)
        local_features_3=GetFeaturesFromHog(image[:,:,2],orientation,cellsPerBlock,pixelsPerCell, False, True)
        x=np.hstack((local_features_1,local_features_2,local_features_3)) # 배열을 왼쪽에서 오른쪽으로 붙입니다.
        featureList.append(x) # featureList에 x를 추가해줍니다.
    return featureList
```

-

In [13]:
```
# 데이터셋으로 부터 Features를 출력하기 위한 Parameter
orientations=9
cellsPerBlock=2
pixelsPerBlock=16
convertColorSpace=True
vehicleFeatures= ExtractFeatures(vehicle_images_original,orientations,cellsPerBlock,pixelsPerBlock, convertColorSpace)
nonVehicleFeatures= ExtractFeatures(non_vehicle_images_original,orientations,cellsPerBlock,pixelsPerBlock, convertColorSpace)
```

In [14]:
```
featuresList= np.vstack([vehicleFeatures, nonVehicleFeatures])
# featureList에 vehicleFeatures와 nonVehicleFeatures를 vstack을 이용하여 세로로 결합한 값을 저장합니다.
print("Shape of features list is ", featuresList.shape) # featureList의 shape을 출력합니다.
labelList= np.concatenate([np.ones(len(vehicleFeatures)), np.zeros(len(nonVehicleFeatures))])
# labelList에 vehicleFeatures에 해당하는 개수만큼 np.ones() 함수를 이용하여 1로 채워진 배열을 생성함.
# nonVehicleFeatures에 해당하는 개수만큼 np.zeros() 함수를 이용하여 0으로 채워진 배열을 생성합니다
print("Shape of label list is ", labelList.shape) # label list의 shape를 출력합니다.
```

```
Shape of features list is  (17760, 972)
Shape of label list is  (17760,)
```

Step 4- Data Preprocessing

Step 4.1 - Splitting Data into Training and Test Set

In [15]:
```
# train test split of data
# data에서 train과 test를 분리합니다.
from sklearn.model_selection import train_test_split

X_train,  X_test,Y_train, Y_test = train_test_split(featuresList, labelList, test_size=0.2, shuffle=True)
#test_size =0.2라는 것은 전체 데이터 셋에서 20%를 랜덤하게 테스트로 사용하는 것을 의미합니다.
# shuffle은 데이터를 분리하기 전에 데이터를 섞을 것인지 설정하는 것입니다.
```

Step 4.2 - Normalization and Scaling of Data

In [16]:
```
# normalization and scaling
# data를 변환하는 코드

from sklearn.preprocessing import StandardScaler

scaler= StandardScaler() # StandardScaler()는 평균을 제거하고, 데이터를 단위분산으로 조정합니다.
#|이상치가 있는 경우 데이터의 확산은 매우 달라지는게 단점입니다.
scaler.fit(X_train) # 데이터 변환을 학습. fit() train 데이터에만 적용합니다.
X_train_scaled= scaler.transform(X_train) # 실제 데이터의 스케일을 조정, transform함수는 train 데이터와 test 데이터에 적용합니다.
X_test_scaled= scaler.transform(X_test)
```

Step 5- Define and Train a classifier

In [17]:
```
%%time

# Train a Linear SVM classifer
# SVM은 클래스를 구분하는 분류 문제에서 각 클래스를 잘 구분하는 선을 그어주는 방식입니다.
# SVM 방식 중 LinearSVC는 scikit-learn에서 지원하는 방식이면, 표준적으로 구현된 svm입니다.
from sklearn.svm import LinearSVC
classifier1= LinearSVC()
classifier1.fit(X_train,Y_train)
print("Accuracy of SVC is  ", classifier1.score(X_test,Y_test) ) # SVC의 정확도를 출력합니다.
```

```
Accuracy of SVC is   0.9870495495495496
Wall time: 2.14 s
```

In [18]:
```
# function to draw sliding Windows
# box를 그려주는 함수

import matplotlib.image as mpimg

def draw_boxes(img, bboxes, color=(0, 0, 255), thick=6):
    # Make a copy of the image
    # np.copy 함수를 이용하여 이미지,배열을 복사합니다.
    imcopy = np.copy(img)
    # Iterate through the bounding boxes

    for bbox in bboxes:
        r=random.randint(0,255)
        g=random.randint(0,255)
        b=random.randint(0,255)
        # r,g,b에 해당하는 값을 랜덤으로 설정해줍니다.
        color=(r, g, b)
        # rectangle 함수를 이용하여 사각형(바운딩박스)을 그립니다.
        # bbox[0]은 시작점 좌표(좌측상단), bbox[1]은 종료좀 좌표(우측 하단)입니다.
        cv2.rectangle(imcopy, bbox[0], bbox[1], color, thick)

    # 바운딩 박스가 그려진 이미지를 반환합니다.
    return imcopy
```

In [19]:
```
# function to find the windows on which we are going to run the classifier

# 슬라이딩 윈도우를 구합니다.
def slide_window(img, x_start_stop=[None, None], y_start_stop=[None, None],
                 xy_window=(64, 64), xy_overlap=(0.9, 0.9)):

    # x,y start_stop[0]이 None 일 경우 0을 대입하고,   x,y start_stop[1]이 None일 경우 img.shape에 해당하는 값을 넣어줌
    if x_start_stop[0] == None:
        x_start_stop[0]=0
    if x_start_stop[1] == None:
        x_start_stop[1]=img.shape[1]
    if y_start_stop[0] ==  None:
        y_start_stop[0]= 0
    if y_start_stop[1] ==  None:
        y_start_stop[1]=img.shape[0]


    window_list = []
    image_width_x= x_start_stop[1] - x_start_stop[0]   # x의 너비를 구함
    image_width_y= y_start_stop[1] - y_start_stop[0]   # y의 너비를 구함.

    windows_x = np.int( 1 + (image_width_x - xy_window[0])/(xy_window[0] * xy_overlap[0]))
    windows_y = np.int( 1 + (image_width_y - xy_window[1])/(xy_window[1] * xy_overlap[1]))

    modified_window_size= xy_window
    for i in range(0,windows_y):
        y_start = y_start_stop[0] + np.int( i * modified_window_size[1] * xy_overlap[1])
        for j in range(0,windows_x):
            x_start = x_start_stop[0] + np.int( j * modified_window_size[0] * xy_overlap[0])

            x1 = np.int( x_start +  modified_window_size[0])
            y1= np.int( y_start + modified_window_size[1])
            window_list.append(((x_start,y_start),(x1,y1)))
    return window_list
```

In [20]:
```
# function that returns the refined Windows
# From Refined Windows we mean that the windows where the classifier predicts the output to be a car
# 자동차에 해당하는 부분에 predict값이 1이면 refinedWindows에 추가하는 함수


def DrawCars(image,windows, converColorspace=False):
    refinedWindows=[]
    for window in windows:

        start= window[0]
        end= window[1]
        clippedImage=image[start[1]:end[1], start[0]:end[0]]

        if(clippedImage.shape[1] == clippedImage.shape[0] and clippedImage.shape[1]!=0):
            # clippedImage.shape의 [0],[1]이 같고, clippedImage[1]에 해당하는 값이 0이 아닐때 조건문이 실행됩니다.

            clippedImage=cv2.resize(clippedImage, (64,64))  # clippedImage를 (64,64)로 변환합니다.

            f1=ExtractFeatures([clippedImage], 9 , 2 , 16,converColorspace)
            # f1에 ExtractFeatures를 통해 featureList를 생성하고 f1에 대입합니다.

            predictedOutput=classifier1.predict([f1[0]])   # predict를 통해 예측값을 얻습니다.
            if(predictedOutput==1):     # 1일 경우 refinedWindow 리스트에 window 원소를 추가합니다.
                refinedWindows.append(window)

    return refinedWindows    # refinedWindows를 반환합니다.
```

In [21]:
```
# trying out SubSampling using HOG but not able to go through as feature size is not the same.

def DrawCarsOptimised(image, image1, image2,windows, converColorspace=False):
    refinedWindows=[]
    for window in windows:

        start= window[0]
        end= window[1]
        clippedImage=image[start[1]:end[1], start[0]:end[0]]
        clippedImage1=image1[start[1]:end[1], start[0]:end[0]]
        clippedImage2=image2[start[1]:end[1], start[0]:end[0]]

        if(clippedImage.shape[1] == clippedImage.shape[0] and clippedImage.shape[1]!=0):
            # clippedImage는 ravel()함수를 이용하여 나타냄
            clippedImage=cv2.resize(clippedImage, (64,64)).ravel()
            clippedImage1=cv2.resize(clippedImage1, (64,64)).ravel()
            clippedImage2=cv2.resize(clippedImage2, (64,64)).ravel()

            #f1=ExtractFeatures([clippedImage], 9 , 2 , 16,converColorspace)
            f1= np.hstack((clippedImage,clippedImage1,clippedImage2))   # clippedImage들을 세로로 합칩니다.
            f1=scaler.transform(f1.reshape(1,-1))
            print(f1.shape)  # f1의 shape를 출력합니다.
            predictedOutput=classifier1.predict([f1[0]])
            if(predictedOutput==1):
                refinedWindows.append(window)

    return refinedWindows
```

In [22]:
```
#testing our functions of slide_window and draw window. Defining here dummy windows
# 위에서 정의한 함수들이 제대로 동작하는지 확인하는 코드

image = mpimg.imread('test3.jpg')

# slide_window 함수를 이용하여 windows를 그립니다.
windows1 = slide_window(image, x_start_stop=[0, 1280], y_start_stop=[400,464],
                    xy_window=(64,64), xy_overlap=(0.15, 0.15))
windows4 = slide_window(image, x_start_stop=[0, 1280], y_start_stop=[400,480],
                    xy_window=(80,80), xy_overlap=(0.2, 0.2))
windows2 = slide_window(image, x_start_stop=[0, 1280], y_start_stop=[400,612],
                    xy_window=(96,96), xy_overlap=(0.3, 0.3))
windows3 = slide_window(image, x_start_stop=[0, 1280], y_start_stop=[400,660],
                    xy_window=(128,128), xy_overlap=(0.5, 0.5))


windows = windows1 + windows2 +  windows3 + windows4
print("Total No of windows are ",len(windows))   # windows의 개수를 출력합니다.
refinedWindows=DrawCars(image,windows, True)

f,axes= plt.subplots(2,1, figsize=(30,15))  # 2행 1열의 subplots를 그립니다.

window_img = draw_boxes(image, windows)    # windows가 그려진 이미지를 window_img에 넣습니다.

axes[0].imshow(window_img)    # window_img를 첫번째 행에 보여줍니다.
axes[0].set_title("Window Coverage")
# 모든 window를 다 보여줍니다.

window_img = draw_boxes(image, refinedWindows)
# 위에 코드의 window_img와 다르게 windows대신 refinedWindows를 두번째 매개변수에 넣었습니다.
#차량에 해당하는 부분만 box가 그려지는 것을 확인할 수 있습니다.
axes[1].set_title("Test Image with Refined Sliding Windows")
axes[1].imshow(window_img)  # refinedWindows 함수를 적용한 window_img를 두번째 행에 보여줍니다.
```

```
Total No of windows are  470
```

Out[22]: <matplotlib.image.AxesImage at 0x1a63372a408>

In [23]:
```
# function to increase the pixel by one inside each box
# heatmap을 업데이트 해주는 함수

def add_heat(heatmap, bbox_list):
    # Iterate through list of bboxes
    for box in bbox_list:
        # Add += 1 for all pixels inside each bbox
        # Assuming each "box" takes the form ((x1, y1), (x2, y2))
        heatmap[box[0][1]:box[1][1], box[0][0]:box[1][0]] += 1

    # Return updated heatmap
    return heatmap
```

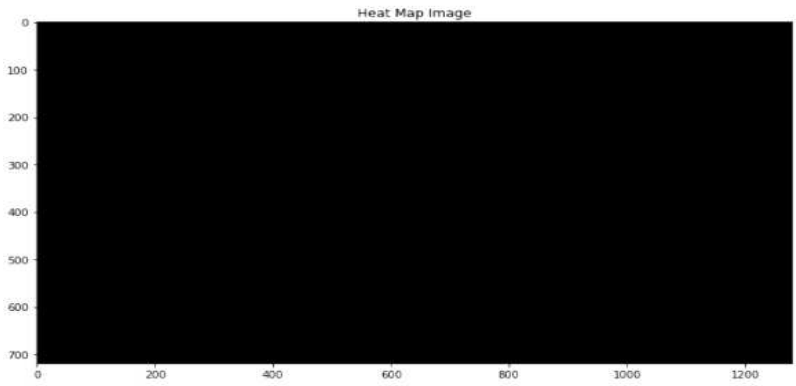
In [24]:
```
# applying a threshold value to the image to filter out low pixel cells
# threshold보다 작은 heatmap 인덱스는 0으로 초기화합니다.
# threshold를 적용한 heatmap

def apply_threshold(heatmap, threshold):
    # Zero out pixels below the threshold
    heatmap[heatmap <= threshold] = 0
    # Return thresholded map
    return heatmap
```

In [25]:
```
# find pixels with each car number and draw the final bounding boxes
#  차량을 발견하면, 바운딩 박스를 그리는 코드입니다.

from scipy.ndimage.measurements import label
def draw_labeled_bboxes(img, labels):
    # Iterate through all detected cars
    for car_number in range(1, labels[1]+1):
        # Find pixels with each car_number label value
        nonzero = (labels[0] == car_number).nonzero()
        # Identify x and y values of those pixels
        nonzeroy = np.array(nonzero[0])
        nonzerox = np.array(nonzero[1])
        # Define a bounding box based on min/max x and y
        bbox = ((np.min(nonzerox), np.min(nonzeroy)), (np.max(nonzerox), np.max(nonzeroy)))
        # Draw the box on the image
        cv2.rectangle(img, bbox[0], bbox[1], (0,0,255), 6) # (0,0,255)
    # Return the image
    return img
```

In [26]:
```
#testing our heat function
# 위에서 정의한 함수들의 테스트 해보는 코드

heat = np.zeros_like(image[:,:,0]).astype(np.float)

heat = add_heat(heat,refinedWindows)

# Apply threshold to help remove false positives
heat = apply_threshold(heat,3)

# Visualize the heatmap when displaying
heatmap = np.clip(heat, 0, 255)

heat_image=heatmap

# Find final boxes from heatmap using label function
labels = label(heatmap)   # label()함수를 통해 heatmap의 boxes를 저장합니다.
print(" Number of Cars found - ",labels[1])    # 찾은 자동차의 개수를 출력합니다.
draw_img = draw_labeled_bboxes(np.copy(image), labels)

f,axes= plt.subplots(2,1, figsize=(30,15))
axes[0].imshow(heat_image,cmap='gray')   # cmap을 이용하여 gray즉, 하나의 채널만 가져옵니다.
axes[0].set_title("Heat Map Image")
axes[1].imshow(draw_img)                 # draw_img에는 앞에서 laeled_bboxes를 통해 그려진 이미지를 보여줍니다.
axes[1].set_title("Final Image after applying Heat Map")
```

```
 Number of Cars found -  0
```

Out[26]: Text(0.5, 1.0, 'Final Image after applying Heat Map')



In [27]:
```
# Defining a class to store the refined frames found from the last 15 frames
# 클래스에 최근 15개의 frame으로부터 찾은 refined frame 들을 저장합니다.

class KeepTrack():
    def __init__(self):     # 생성자 함수에서는 비어있는 refinedWindows 리스트를 만듭니다.
        self.refinedWindows = []

    def AddWindows(self, refinedWindow):
        self.refinedWindows.append(refinedWindow)
        frameHistory=15   # 최근 프레임 중 15개만 본다고 지정해줍니다.
        if len(self.refinedWindows) > frameHistory:   # frameHistory보다 refinedWindows의 크기가 클경우 refinedWindows를 재설정합니다.
            self.refinedWindows = self.refinedWindows[len(self.refinedWindows)-frameHistory:]
```

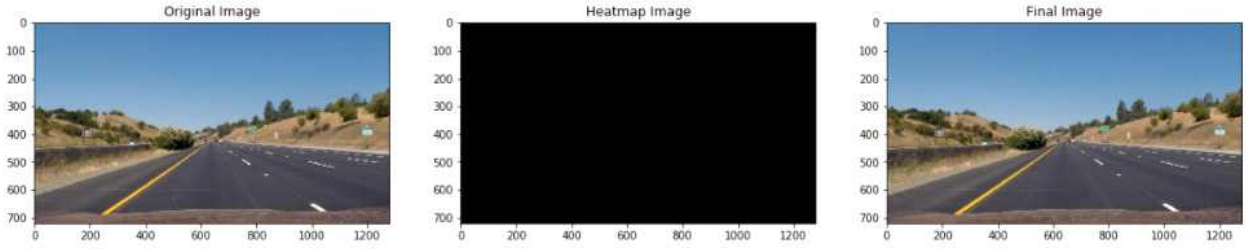
In [28]:
```
#defining the Parameters required for the pipeline to run
# pipeline이 동작하기 위해 필요한 파라미터들을 정의하는 부분입니다.

orientation=9 # No of orientations of HOG
cellsPerBlock=2 # No of cells per block
pixelsPerCell=16 # No of pixels per cell
xy_window=(64, 64) # window Size
xy_overlap=(0.15, 0.15) # Window Overlap. Please note this is different as provided by Udacity. Overlap of 0.15 means my windows are 85%
x_start_stop=[0, image.shape[1]] # X Coordinates to start and stop search
y_start_stop=[400, 660] # Y Coordinates to start and stop search

# Window 1- Size - 64x64 , Overlap-85%
windows_normal = slide_window(image, x_start_stop, [400,464],
                    xy_window, xy_overlap)

# Window 2- Size - 80x80 , Overlap-80%
xy_window_1_25= (80,80)
xy_window_1_25_overlap=(0.2, 0.2)
windows_1_25 = slide_window(image, x_start_stop, [400,480],
                    xy_window_1_25, xy_window_1_25_overlap)

# Window 3- Size - 96x96 , Overlap-70%
xy_window_1_5= (96,96)
xy_window_1_5_overlap=(0.3, 0.3)
windows_1_5 = slide_window(image, x_start_stop, [400,612],
                    xy_window_1_5, xy_window_1_5_overlap)

# Window 4- Size - 128x128 , Overlap-50%
xy_window_twice_overlap=(0.5, 0.5)
xy_window_twice = (128,128)
windows_twice = slide_window(image, x_start_stop, [400,660],
                    xy_window_twice, xy_window_twice_overlap)

# Total Windows - 470
windows= windows_normal +  windows_1_5  + windows_twice +windows_1_25
print("No of Windows are ",len(windows))
```

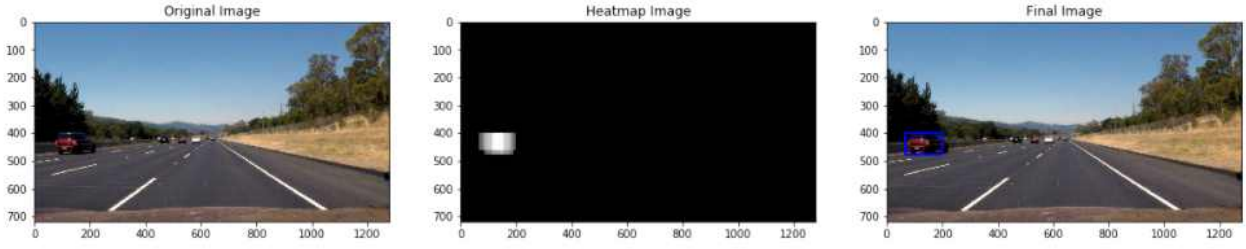
```
No of Windows are  470
```

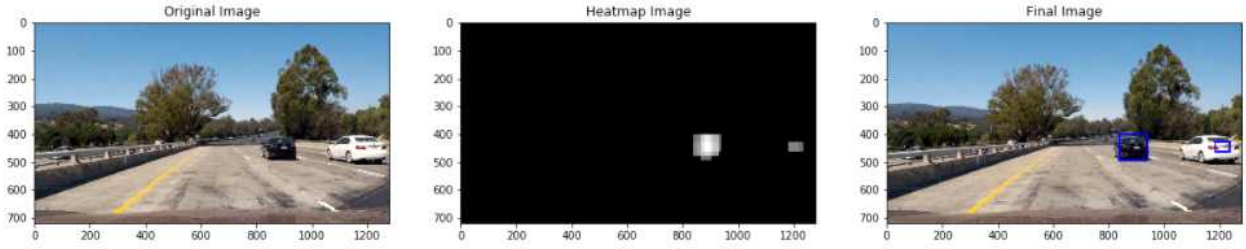
In [29]:
```
# Defining a pipeline for Video Frame Processing
# Note here the track of last 15 frames is kept
# 파이프라인에 해당하는 코드입니다.

def Pipeline(image):
#     features,hog_image=GetFeaturesFromHog(image[:,:,0],orientation,cellsPerBlock,pixelsPerCell, visualise= True, feature_vector_flag=F
#     features1,hog_image1=GetFeaturesFromHog(image[:,:,1],orientation,cellsPerBlock,pixelsPerCell, visualise= True, feature_vector_flag
#     features2,hog_image2=GetFeaturesFromHog(image[:,:,2],orientation,cellsPerBlock,pixelsPerCell, visualise= True, feature_vector_flag
#     refinedWindows=DrawCarsOptimised(hog_image,hog_image1,hog_image2,windows, True)

#     image=find_cars(image, 400, 528, 1,  orientation, pixelsPerCell, cellsPerBlock)
#     image=find_cars(image, 400, 560, 1.25,  orientation, pixelsPerCell, cellsPerBlock)
#     image=find_cars(image, 400, 588, 1.5,  orientation, pixelsPerCell, cellsPerBlock)
#     image=find_cars(image, 400, 660, 2,  orientation, pixelsPerCell, cellsPerBlock)
    rand= random.randint(0,1) # rand에 0부터 1사이에 해당하는 숫자를 랜덤으로 생성해 저장합니다.
    if(rand<0.4):  # rand가 0.4보다 작은 값일 경우 실행합니다.
        refinedWindows=keepTrack.refinedWindows[:-1]
    else:   #rand가 0.4보다 크거나 같을 경우 실행합니다.
        refinedWindows=DrawCars(image,windows, True)
        # DrawCars() 함수를 이용해 refinedWindows에 차량을 인지하고 refinedWindows리스트에 추가함.
         if len(refinedWindows) > 0:
            keepTrack.AddWindows(refinedWindows)

    #refinedWindows=DrawCars(image,windows, True)
#     if len(refinedWindows) > 0:
#         keepTrack.AddWindows(refinedWindows)

    heat = np.zeros_like(image[:,:,0]).astype(np.float)

    for refinedWindow in keepTrack.refinedWindows:
        heat = add_heat(heat, refinedWindow)



    heatmap = apply_threshold(heat, 25 + len(keepTrack.refinedWindows)//2)

    labels = label(heatmap)
    draw_img = draw_labeled_bboxes(np.copy(image), labels)
    return draw_img
```

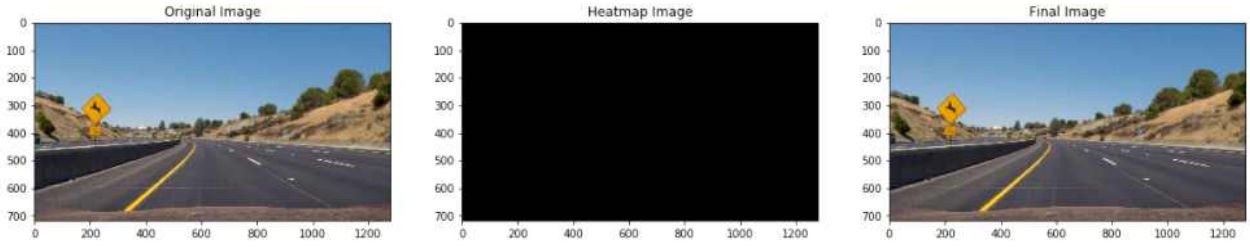
In [30]:
```
# Defining a different pipeline to process the images as we do not want to keep track of previous frames here
# PipelineImage 함수에서는 draw_img와 heatmap을 반환해줍니다.

def PipelineImage(image):

    refinedWindows=DrawCars(image,windows, True)
    heat = np.zeros_like(image[:,:,0]).astype(np.float)
    # np.zeros_like를 이용하여 image[:,:,0]와 행과 열의 개수가 같고, 원소들이 0으로 채워진 배열을 반환합니다.
    heat = add_heat(heat,refinedWindows)
    #heat에 refinedWindows에 해당하는 원소들을 추가해 heat를 업데이트 합니다.

    heatmap = np.clip(heat, 0, 255) # np.clip(heat,0,255) 는 0보다 작은 값은 0으로 255보다 큰 값은 255로 바꿔주는 함수입니다.
    heatmap = apply_threshold(heat, 4)
    labels = label(heatmap)
    draw_img = draw_labeled_bboxes(np.copy(image), labels) # draw_img는 draw_labeled_bboxes를 이용해 box를 그린 이미지를 저장합니다.
    return draw_img,heatmap
```

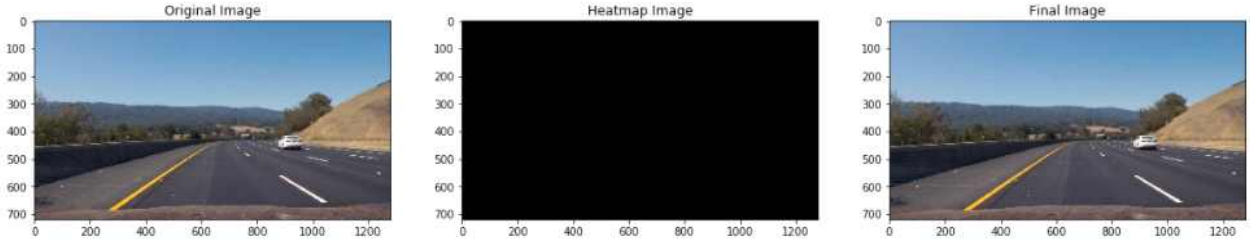
In [31]:
```

# glob을 이용해 이미지 파일의 경로명에 접근해 아래의 코드에서는 ./test_images에서 확장자가 /*.jpg인 파일만 추출합니다.
test_images= glob.glob("./test_images/*.jpg")
f, axes= plt.subplots(8,3, figsize=(20,40))

for index,image in enumerate(test_images):
    image = cv2.imread(image)
    # original , heatmap, final image를 순서대로 보여줍니다.
    # heatmap에서는 차량으로 검출되는 부분에 회색으로 표시가되고, final image에서는 사각형으로 표시되는 것을 볼 수 있습니다.
    image = cv2.cvtColor(image,cv2.COLOR_BGR2RGB)
    finalPic,heatmap = PipelineImage(image)
    axes[index,0].imshow(image)
    axes[index,0].set_title("Original Image")
    axes[index,1].imshow(heatmap,cmap='gray')
    axes[index,1].set_title("Heatmap Image")
    axes[index,2].imshow(finalPic)
    axes[index,2].set_title("Final Image")
```

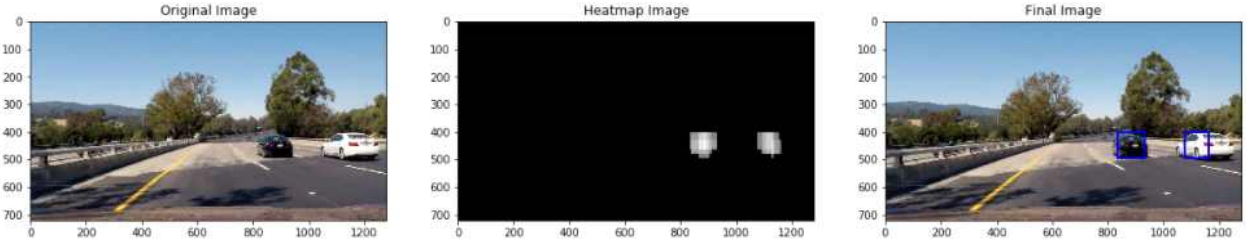

Original Image
Heatmap Image
Final Image

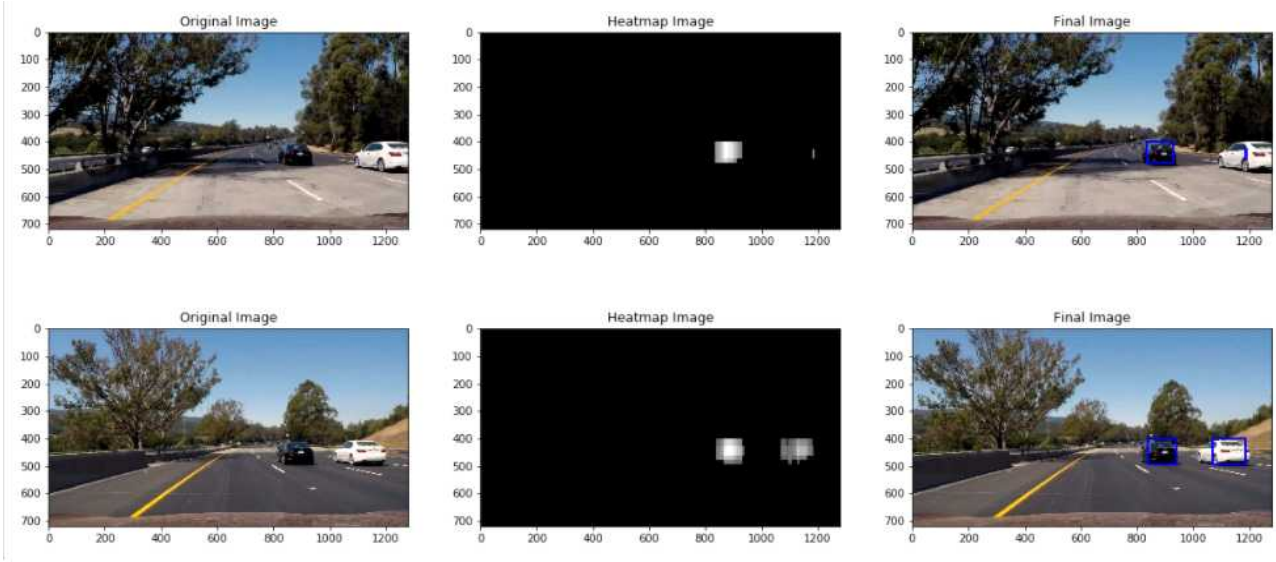

Original Image
Heatmap Image
Final Image


Original Image
Heatmap Image
Final Image


Original Image
Heatmap Image
Final Image


Original Image
Heatmap Image
Final Image


Original Image
Heatmap Image
Final Image



In [32]:
```
# 원본영상을 이용해 차량에 해당하는 부분을 표시하는 영상을 만듭니다.

keepTrack = KeepTrack()
import moviepy
from moviepy.editor import VideoFileClip
video_output1 = 'full_video_threshold_20_with_frame_skipping_my.mp4' # 결과물 영상
video_input1 = VideoFileClip('project_video.mp4')  # 원본 영상
processed_video = video_input1.fl_image(Pipeline)
%time processed_video.write_videofile(video_output1, audio=False)  # vide_output1에 processed_video를 이용해 변형된 영상을 저장한다.
video_input1.reader.close()
video_input1.audio.reader.close_proc()
```

```
Moviepy - Building video full_video_threshold_20_with_frame_skipping_my.mp4.
Moviepy - Writing video full_video_threshold_20_with_frame_skipping_my.mp4

Moviepy - Done !
Moviepy - video ready full_video_threshold_20_with_frame_skipping_my.mp4
Wall time: 14min 6s
```

데이터셋은 non-vehicles과 vehicles이 있는데, 차량이 아닌 것과 차량을 구분해 학습시키기 위해서 두 개의 데이터셋이 있습니다. vehicles/vehicles 에는 GTI_Far, GTI_Left, GTI_MiddleClose, GTI_Right, KITTI_extracted 라는 폴더가 있고, 각 폴더에는 vehicles 이미지들이 들어가 있습니다. 각 이미지에 접근하기 위해 코드에서 첫 번째 cell에 해당하는 부분에서 vehicle_image_arr = glob.glob('./vehicles/vehicles/*/*.png') 코드에서 vehicles 안에 있는 (.png) 파일에 접근합니다.

사진뿐만 아니라 영상에도 적용할 수 있도록 project_video.mp4 파일이 있어서 적용해볼 수 있습니다.

2-2) Copilot-driving asistance

https://github.com/visualbuffer/copilot

자율주행에 기본으로 들어가는 기술인 차선인지와 차량인식을 동시에 진행한다는 점에서 관심있게 보았습니다. 차선인식은 Opencv 함수를 기반으로 진행하고, 차량인식은 Yolo v3 오픈소스를 이용하여 코드가 작성되었습니다.

(1) 차선인식

Fig -2 Obtaining Images from gray-scale images

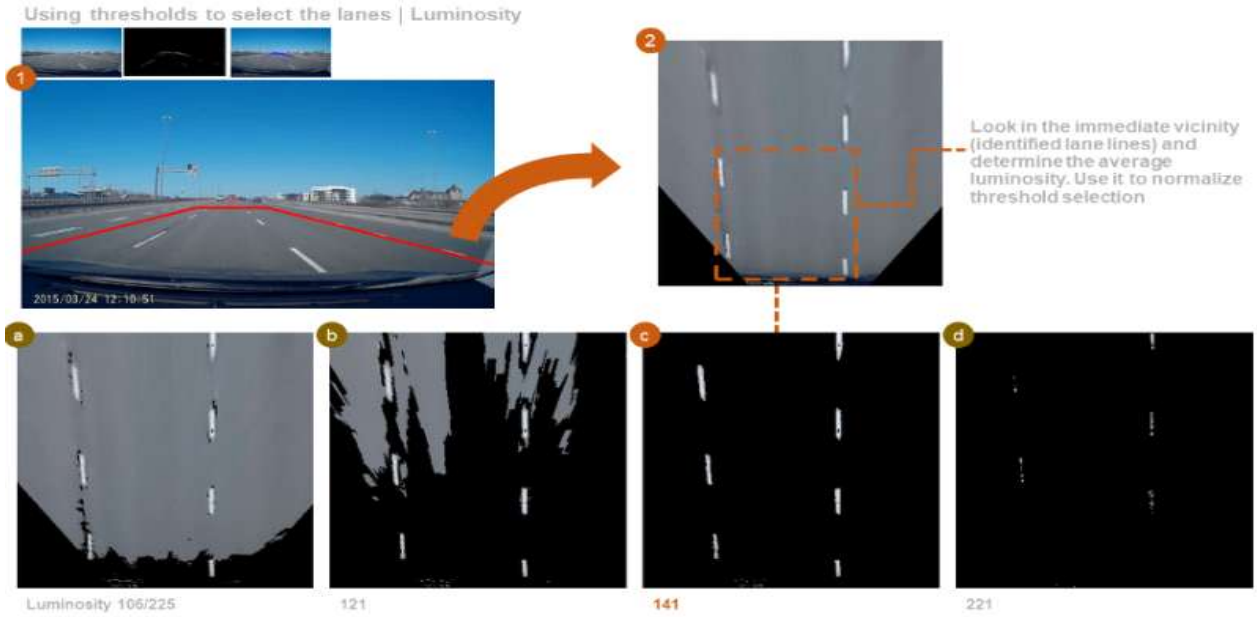
Gray scale 이미지에 Canny 필터를 적용하여 차선에 해당하는 부분만을 검출합니다. Hough Transform을 이용하여 후보선을 결정합니다.

```
330     def calc_perspective(self,  lane_start=[0.25,0.75]):
331         roi = np.zeros((self.img_dimensions[0], self.img_dimensions[1]), dtype=np.uint8) # 720 , 1280
332         roi_points = np.array([[0, self.img_dimensions[0]*7//9],
333                 [0, self.img_dimensions[0]],
334                 [self.img_dimensions[1], self.img_dimensions[0]],
335                 [self.img_dimensions[1], self.img_dimensions[0]*7//9],
336                 [self.img_dimensions[1]*45//99,self.img_dimensions[0]//2],
337                 [self.img_dimensions[1]*45//99,self.img_dimensions[0]//2]], dtype=np.int32)
338         cv2.fillPoly(roi, [roi_points], 1)
339         self.lane_roi = np.zeros((self.img_dimensions[0], self.img_dimensions[1]), dtype=np.uint8)  # lane의 roi 영역을 초기화합니다.
340         Lhs = np.zeros((2,2), dtype= np.float32)
341         Rhs = np.zeros((2,1), dtype= np.float32)
342         grey = cv2.cvtColor(self.image, cv2.COLOR_BGR2GRAY) # BGR 이미지를 GRAY 로 변환합니다.
343         mn_hsl = np.median(grey)
344         edges = cv2.Canny(grey, int(mn_hsl), int(mn_hsl*.3))     # canny 함수를 적용하여 엣지를 검출합니다.
345         #HoughLinesP 함수를 사용하면 선의 시작점과 끝점을 Return 해줘서 이용하기 편리합니다.
346         lines = cv2.HoughLinesP(edges*roi,rho =self.img_dimensions[0]//20,\
347                 theta = 2* np.pi/180,\
348                 threshold = self.img_dimensions[0]//80,\
349                 minLineLength = self.img_dimensions[0]//3,\
350                 maxLineGap = self.img_dimensions[0]//15)
351
```

해당 코드- lane_detection.py

(2)Perspective transform (Creating a mask out of the perspective image)

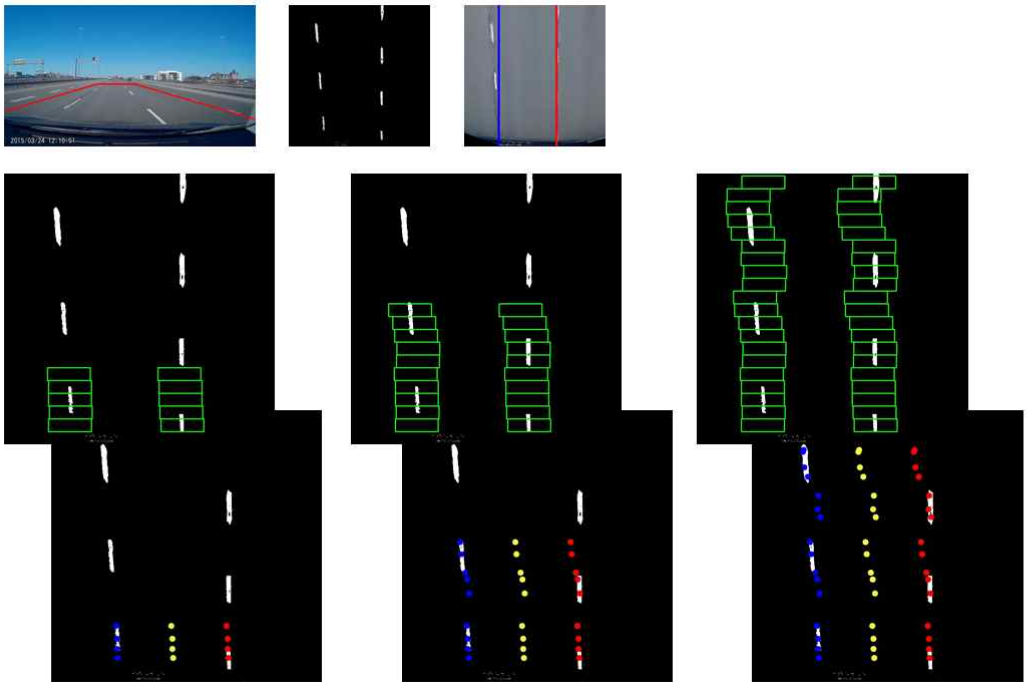
차선정보를 추출할 수 있는 Mask를 만듭니다. cv2.getPerspectiveTransform(src_points, dst_points) 함수를 이용하여 인자로 받아오는 4개의 점을 변환합니다. cv2.warpPerspective를 이용해서 변환된 이미지를 저장합니다.




```python
def on_line(p1, p2, ycoord):
    return [p1[0]+ (p2[0]-p1[0])/float(p2[1]-p1[1])*(ycoord-p1[1]), ycoord]

p1 = [self.vanishing_point[0] - self.WRAPPED_WIDTH/2, top]
p2 = [self.vanishing_point[0] + self.WRAPPED_WIDTH/2, top]
p3 = on_line(p2,self.vanishing_point, bottom)
p4 = on_line(p1,self.vanishing_point, bottom)
src_points = np.array([p1,p2,p3,p4], dtype=np.float32)
dst_points = np.array([[0, 0], [self.UNWARPED_SIZE[0], 0],
                       [self.UNWARPED_SIZE[0], self.UNWARPED_SIZE[1]],
                       [0, self.UNWARPED_SIZE[1]]], dtype=np.float32)
self.trans_mat = cv2.getPerspectiveTransform(src_points, dst_points)
self.inv_trans_mat = cv2.getPerspectiveTransform(dst_points,src_points)
min_wid = 1000
img = cv2.warpPerspective(self.image, self.trans_mat, self.UNWARPED_SIZE)
x1 = int(self.UNWARPED_SIZE[0]*lane_start[0])
x2 = int(self.UNWARPED_SIZE[0]*lane_start[1])
self.lane.leftx.append(x1)
self.lane.rightx.append(x2)
mask = self.compute_mask(img)
# x = np.linspace(0,mask.shape[0]-1,mask.shape[0])
span = self.UNWARPED_SIZE[0]//5
x1 = x1-span + self.detect_lane_start(mask[:,x1-span :x1+span])
x2 = x2-span + self.detect_lane_start(mask[:,x2-span :x2+span])
self.lane.leftx.append(x1)
self.lane.rightx.append(x2)
```

해당 코드 – lane_detection.py

(3) Sweeping windows

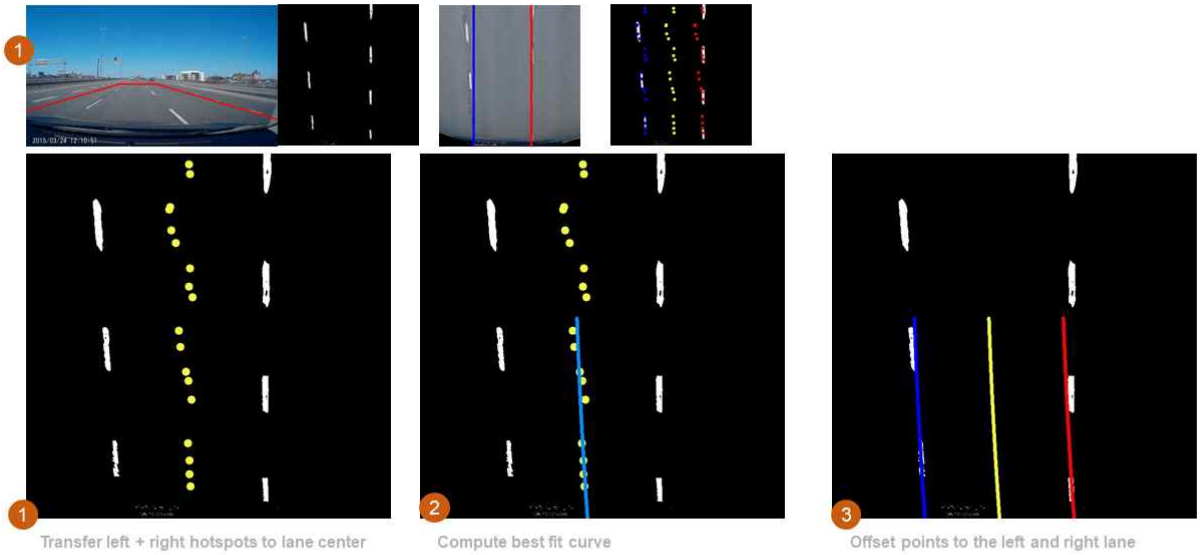


차선을 인식한 후 차선을 포함하는 픽셀을 추출할 필요가 있습니다. 그래서 차량을 기준으로 왼쪽과 오른쪽 차선에 사각형을 그리면서 해당 픽셀에 대한 위치를 갖고 다음 차선을 추측합니다.

```
631     def compute_lane_lines(self, img):
632         self.lane.left_windows = []
633         self.lane.right_windows = []
634         undst_img  = cv2.bitwise_and(img, img, mask =  self.lane_roi )
635         pp_img = cv2.warpPerspective(undst_img, self.trans_mat, (self.UNWARPED_SIZE[1],self.UNWARPED_SIZE[0]))
636         warped_img = self.compute_mask(pp_img)
637 
638         x1_av =  int(np.average(self.lane.leftx))
639         x2_av  = int(np.average(self.lane.rightx))
640         self.lane.width= min(max(int(x2_av - x1_av), self.UNWARPED_SIZE[0]//3),self.UNWARPED_SIZE[0]//2)
641         x1 = min(max(x1_av,self.margin), self.UNWARPED_SIZE[0]-self.lane.width)
642         x2 = max( min(x2_av,self.UNWARPED_SIZE[0]-self.margin-1), self.lane.width)
643         leftx_current = x1-self.margin + self.detect_lane_start(warped_img[:,x1-self.margin :x1+self.margin])
644         rightx_current = x2-self.margin + self.detect_lane_start(warped_img[:,x2-self.margin :x2+self.margin])
645         nonzero = warped_img.nonzero()
646         nonzeroy = np.array(nonzero[0])
647         nonzerox = np.array(nonzero[1])
648 
649         centerx_current = (x2_av - x1_av) //2
650         pointx = []
651         pointy=[]
652         center_idx = []
653         curve_compute =  0
654         self.max_gap =  0
655         gap  = 0
656         for window in self.windows_range:
657             win_y_low = warped_img.shape[0] - (window + 1)* self.window_height
658             win_y_high = warped_img.shape[0] - window * self.window_height
659             win_xleft_low = leftx_current - self.margin
660             win_xleft_high = leftx_current + self.margin
661             win_xright_low = rightx_current - self.margin
662             win_xright_high = rightx_current + self.margin
663             self.lane.left_windows.append([(win_xleft_low,win_y_low),(win_xleft_high,win_y_high)])
```

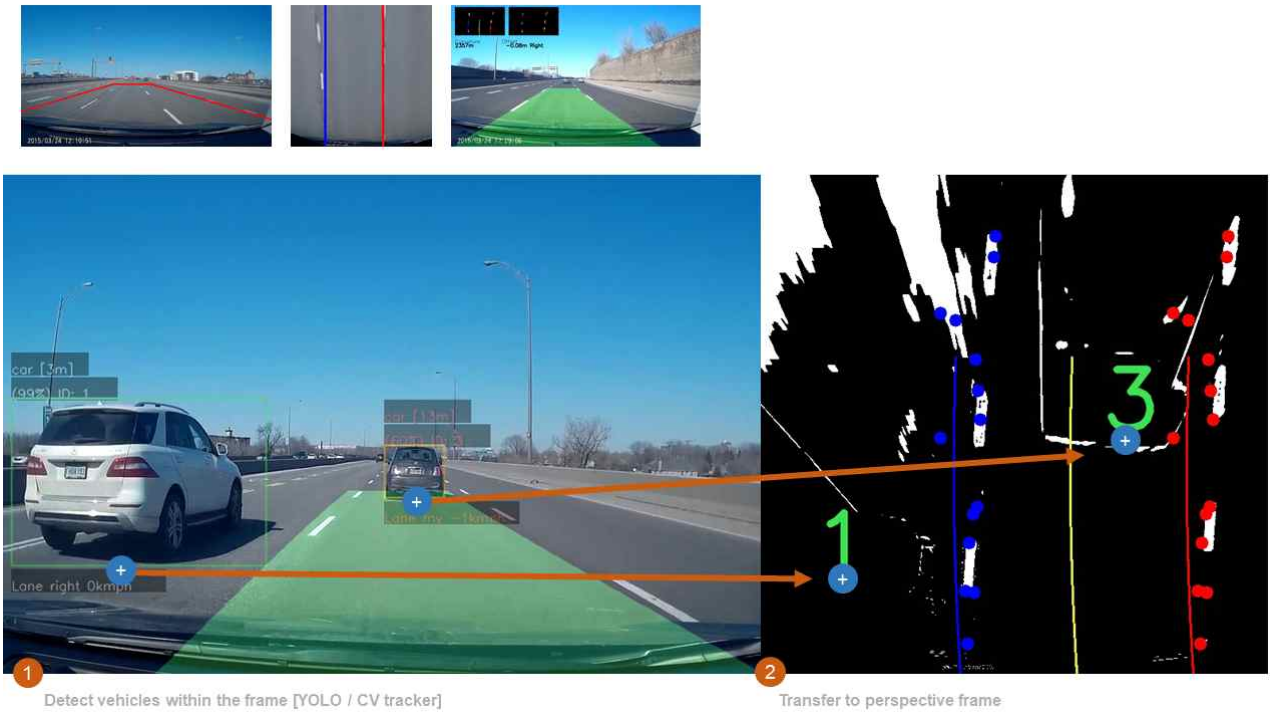
해당 코드 – lane_detection.py 의 compute-_lane_lines 함수

(5) 최적의 차선을 찾고, 오프셋을 검출합니다.



왼쪽과 오른쪽 차선의 좌표를 가지고 차선의 중점을 잡습니다. 그리고 왼쪽, 중점, 오른쪽 차선의 좌표를 갖고 곡선으로 바꾸어줍니다. 또한 최근 프레임의 곡선으로 이전 프레임의 곡선과 비교하며 추정값을 정확히 합니다.

(7) 차량감지 YOLO



차량을 YOLO로 인식하고 조감도에서 차선과 함께 차량이 위치하는 곳에 번호를 나타냅니다. 여기서는 YOLO v3를 이용해서 차량을 인식하고, 바운딩 박스를 그렸습니다.

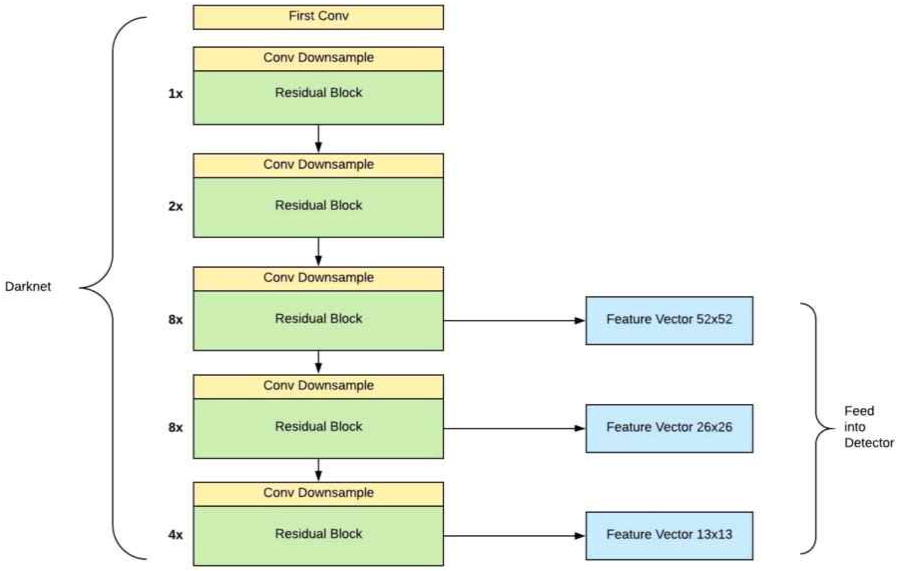
```python
        if self.count% self._yp == 0 :
            boxes= self.yolo.make_predictions(image,obstructions = obstructions,plot=True)
            self.tracker2object(boxes)
            image  =  cv2.cvtColor(np.array(image), cv2.COLOR_RGB2BGR)
            n_obs =  len(self.obstacles)
            for i in range(n_obs):
                tracker = cv2.TrackerKCF_create()#
                # tracker = cv2.TrackerMIL_create()#  # Note: Try comparing KCF with MIL
                box = self.obstacles[i]
                bbox = (box.xmin, box.ymin, box.xmax-box.xmin, box.ymax-box.ymin)
                # print(bbox)
                success = tracker.init(image, bbox )
                if success :
                    self.obstacles[i].tracker=tracker
```

fram.py에서 yolo.make_predictions()를 통해 차량의 바운딩 박스를 그립니다.

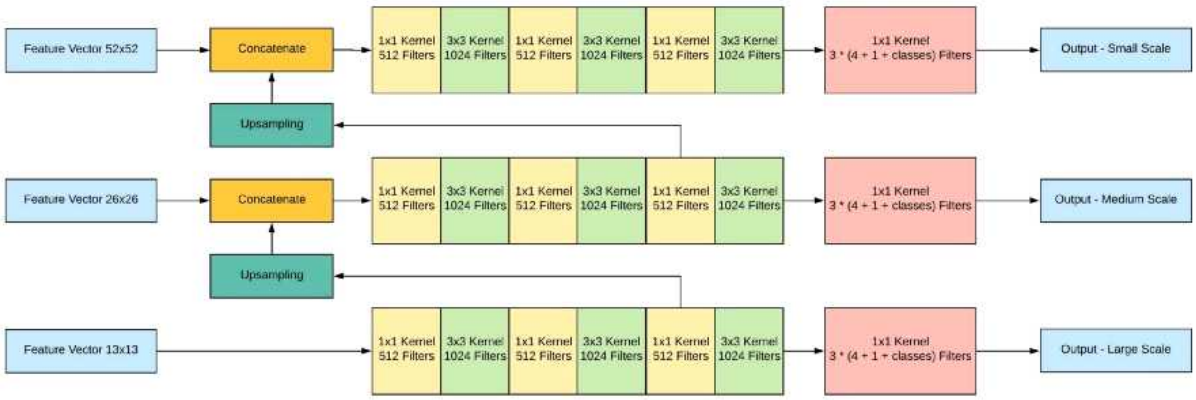
yolo.make_predictions()는 yolo_model.py에 존재하며 바운딩 박스 후보를 예측하고, 이미지에 그리는 것까지 해줍니다. 또한, NMS(Non maximum supression) 알고리즘을 통해 겹치는 바운딩 박스를 제거하고, 가장 객체와 가까운 영역에 해당하는 박스만 남깁니다.

```python
    def make_predictions(self,  image = None, img_path= None,plot =False , save_path=None, obstructions =  None, ar_th=0.0004):
        if img_path:
            image = cv2.imread(img_path)
        else:
            img_path = "./images/detection/detection.jpg"
        image_h, image_w, _ = image.shape
        image_ar =  image_w*image_h
        new_image = self.preprocess_input(image)
        obstructions = obstructions if obstructions else [0,1,2]
        yolos = self.yolov3.predict(new_image)
        boxes = []
        save_path = save_path if save_path else "./images/detection/"

        for i in range(len(yolos)):
            boxes += decode_netout(yolos[i][0], self.anchors[i], self.obj_thresh, self.nms_thresh, self.net_h, self.net_w, obstructions)
            # decode_netout에서는 바운딩 박스 후보와 라벨(클래스)를 예측합니다.

        correct_yolo_boxes(boxes, image_h, image_w, self.net_h, self.net_w,ar_th=ar_th)
        # correct_yolo_boxes에서는 decode_netout에서 그렸던 박스들을 이미지의 원래크기에 맞혀 바꿔주는 함수입니다.
        do_nms(boxes, self.nms_thresh)   # 여러 바운딩 박스들이 겹치기 때문에 겹치는 박스들 중 최대값을 가진 것을 제외하고 제거합니다.
        # print(len(boxes))
        if plot :
            draw_boxes(image, boxes, np.asarray(self.labels)[obstructions], self.obj_thresh)  # 박스를 이미지에 그려주는 함수입니다.
            # print(save_path + img_path.split("/")[-1])
            cv2.imwrite(save_path + img_path.split("/")[-1], (image).astype('uint8'))
        return boxes
```

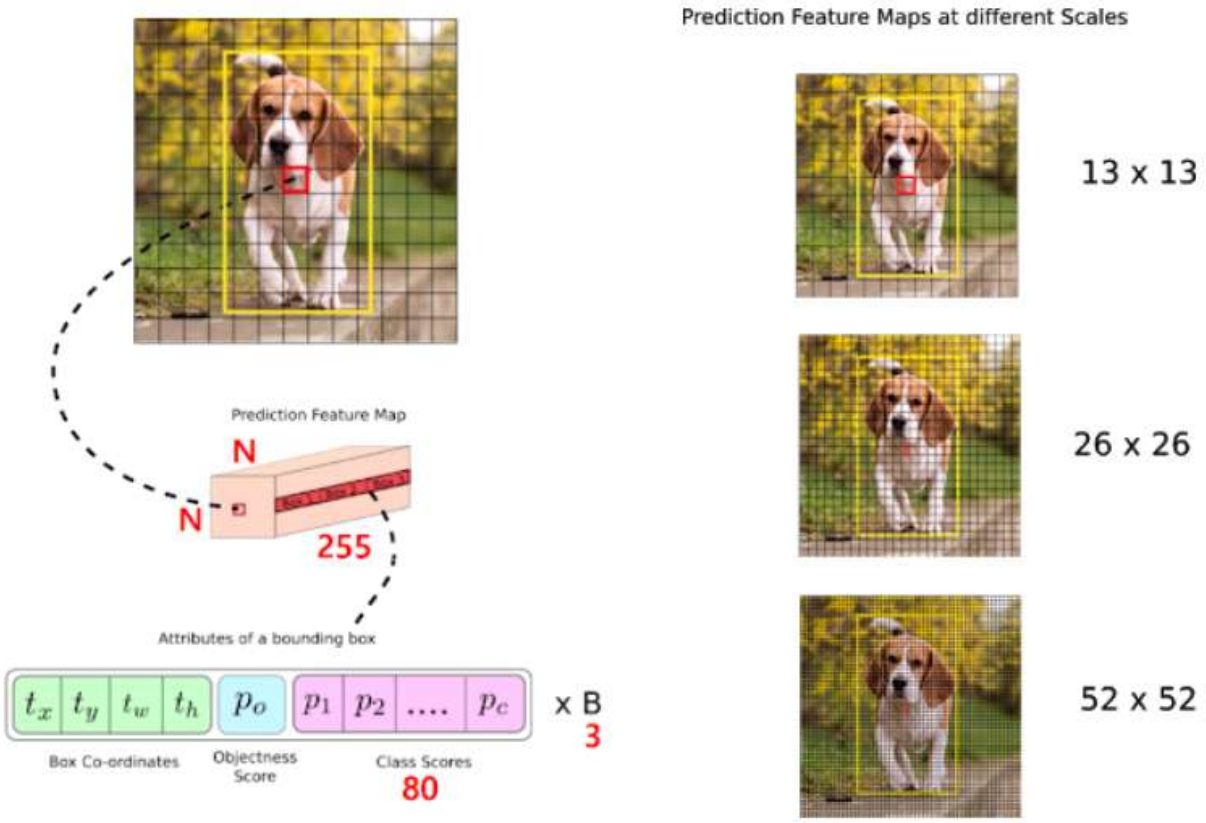
Yolo v3에 대해서 설명하면 Object Detection에 관련 오픈소스 중 하나로, 이전에 Yolo v1, v2가 나왔었고, 이후 논문을 통해 yolo v3가 발표되었습니다. 다른 yolo 버전과 다른 점은 3개의 Feature Map Output에서 각각 3개의 서로 다른 크기와 scale을 가진 anchor box로 detection합니다. 또한 Softmax 방식이 아니라 Sigmoid 기반의 logistic classifier로 개별 Object의 Multi labels 예측이 가능합니다.



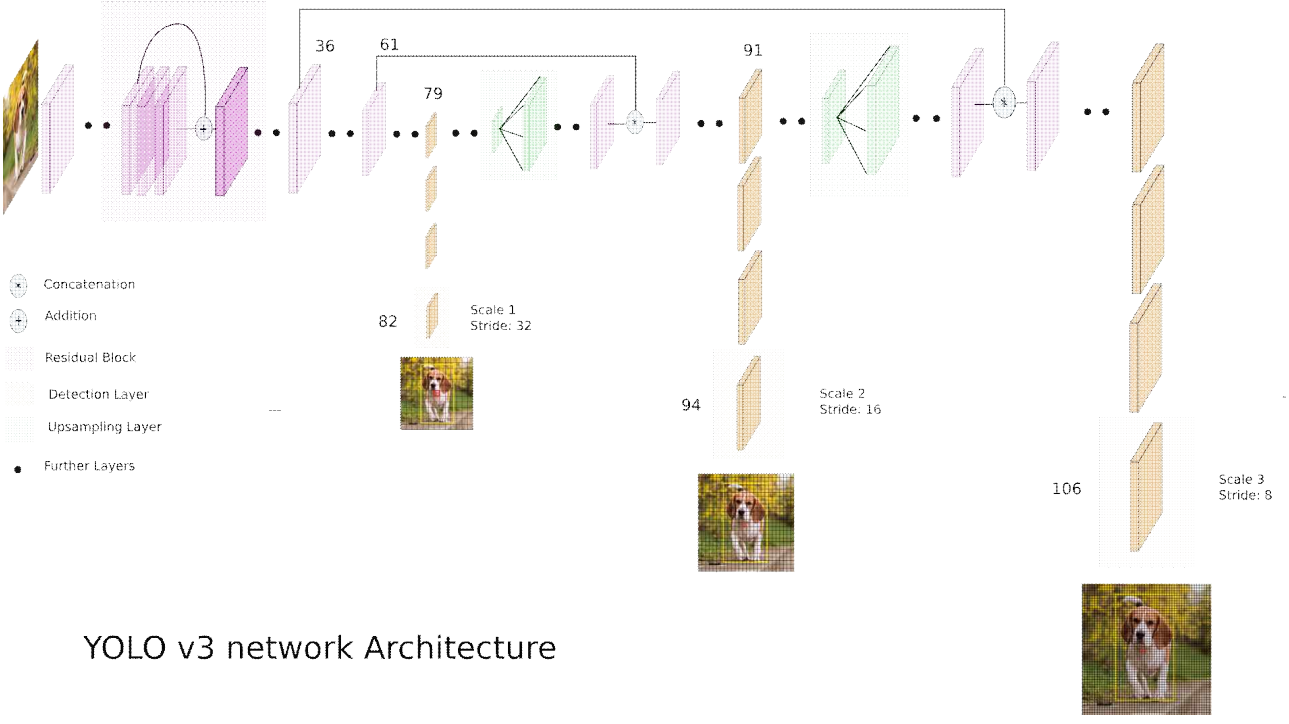
416x416 크기의 이미지를 네트워크에 입력 후 크기가 52x52, 26x26, 13x13의 크기를 가진 layer에서 feature map을 추출합니다.



그 다음 가장 높은 level, 즉 해상도가 가장 낮은 feature map을 1x1, 3x3 conv layer로 구성된 작은 FCN(Fully Convolutional Network)에 입력합니다. 이후 FCN의 output channel이 512가 되는 지점에서 feature map을 추출한 뒤 2배로 업샘플링을 수행합니다. 이후 바로 아래 level에 있는 feature map과 concatenate해줍니다. 이후 merged feature map을 FCN에 입력합니다. 이 과정을 다음 level에 있는 feature map에도 똑같이 수행합니다. 이를 통해 3개의 scale을 가진 feature map을 얻을 수 있습니다.



이 때 각 scale의 feature map의 output channel 수가 [3 x (4 + 1 + 80)](=255)이 되도록 마지막 1x1 conv layer의 channel 수를 조정합니다. 여기서 3은 grid cell당 예측하는 anchor box의 수를, 4는 bounding box offset, 1은 objectness score, 80은 COCO 데이터셋을 사용했을 때의 class 수입니다. 즉, 최종적으로 52x52(x255), 26x26(x255), 13x13(x255) 크기의 feature map을 얻을 수 있습니다.



YOLO v3 network Architecture

위의 사진은 yolo v3 네트워크 구조로 아래의 코드에서는 yolov3 model을 만들기 위한 코드입니다. Layer 106까지 있으며, layer 82,94,106번은 output layer로 Object Detect 결과를 추출합니다.

```python
def make_yolov3_model():
    input_image = Input(shape=(None, None, 3))

    # Layer  0 => 4
    x = _conv_block(input_image, [{'filter': 32, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 0},
                                  {'filter': 64, 'kernel': 3, 'stride': 2, 'bnorm': True, 'leaky': True, 'layer_idx': 1},
                                  {'filter': 32, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 2},
                                  {'filter': 64, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 3}])

    # Layer  5 => 8
    x = _conv_block(x, [{'filter': 128, 'kernel': 3, 'stride': 2, 'bnorm': True, 'leaky': True, 'layer_idx': 5},
                        {'filter':  64, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 6},
                        {'filter': 128, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 7}])

    # Layer  9 => 11
    x = _conv_block(x, [{'filter':  64, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 9},
                        {'filter': 128, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 10}])

    # Layer 12 => 15
    x = _conv_block(x, [{'filter': 256, 'kernel': 3, 'stride': 2, 'bnorm': True, 'leaky': True, 'layer_idx': 12},
                        {'filter': 128, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 13},
                        {'filter': 256, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 14}])

    # Layer 16 => 36
    for i in range(7):
        x = _conv_block(x, [{'filter': 128, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 16+i*3},
                            {'filter': 256, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 17+i*3}])

    skip_36 = x

    # Layer 37 => 40
```

```python
# Layer 80 => 82
pred_yolo_1 = _conv_block(x, [{'filter': 1024, 'kernel': 3, 'stride': 1, 'bnorm': True,  'leaky': True,  'layer_idx': 80
                              {'filter': (3*(5+nb_class)), 'kernel': 1, 'stride': 1, 'bnorm': False, 'leaky': False, 'layer_i
loss_yolo_1 = YoloLayer(anchors[12:],
                        [1*num for num in max_grid],
                        batch_size,
                        warmup_batches,
                        ignore_thresh,
                        grid_scales[0],
                        obj_scale,
                        noobj_scale,
                        xywh_scale,
                        class_scale)([input_image, pred_yolo_1, true_yolo_1, true_boxes])
```

해당 코드는 yolo_model.py

github에서 해당 git의 URL을 git clone해서 파일을 다운받으면, 나오는 폴더구조입니다.

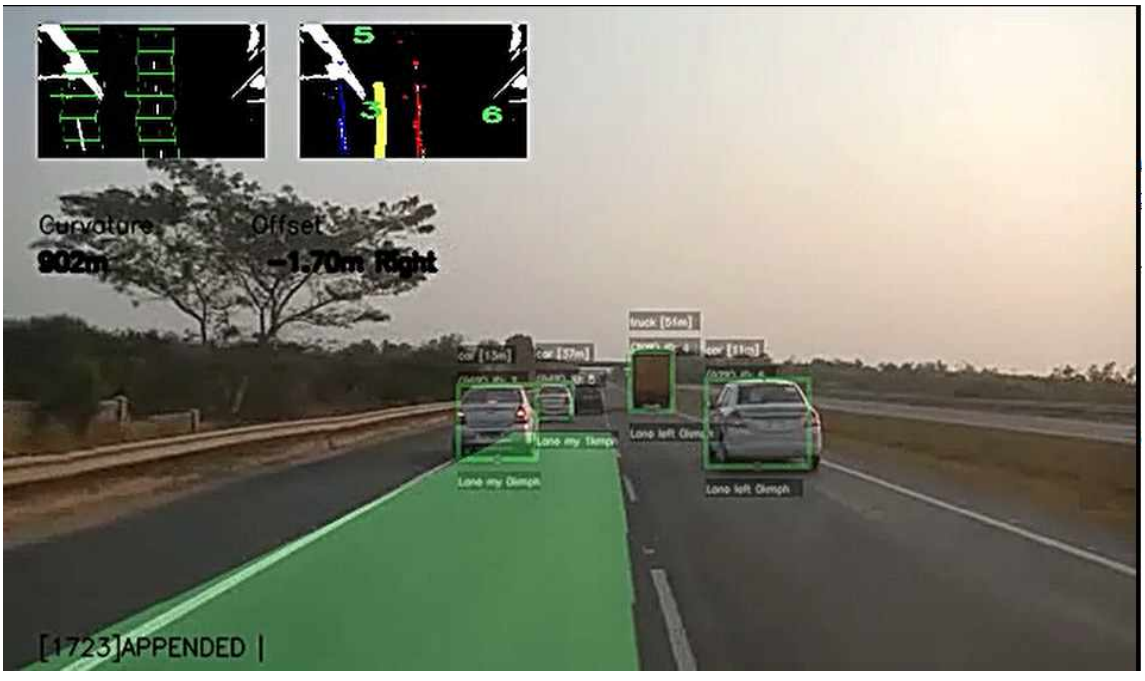
.git	2021-04-05 오전 4:40	파일 폴더	
.vscode	2021-04-05 오전 4:40	파일 폴더	
font	2021-04-05 오전 4:40	파일 폴더	
images	2021-04-05 오전 4:40	파일 폴더	
model_data	2021-04-05 오전 4:40	파일 폴더	
notebooks	2021-04-06 오전 2:14	파일 폴더	
utils	2021-04-05 오전 4:40	파일 폴더	
zoo	2021-04-05 오전 4:40	파일 폴더	
.gitignore	2021-04-05 오전 4:40	Git Ignore 원본 파...	2KB
__init__.py	2021-04-05 오전 4:40	Python 원본 파일	0KB
callbacks.py	2021-04-05 오전 4:40	Python 원본 파일	4KB
camera.py	2021-04-05 오전 4:40	Python 원본 파일	4KB
config.json	2021-04-05 오전 4:40	JSON 원본 파일	2KB
darknet53.cfg	2021-04-05 오전 4:40	CFG 파일	6KB
evaluate.py	2021-04-05 오전 4:40	Python 원본 파일	3KB
frame.py	2021-04-05 오전 4:40	Python 원본 파일	12KB
gen_anchors.py	2021-04-05 오전 4:40	Python 원본 파일	4KB
generator.py	2021-04-05 오전 4:40	Python 원본 파일	10KB
lane_detection.py	2021-04-05 오전 4:40	Python 원본 파일	32KB
LICENSE	2021-04-05 오전 4:40	파일	2KB
predict.py	2021-04-05 오전 4:40	Python 원본 파일	6KB
profile.dat	2021-04-05 오전 4:40	DAT 파일	8KB
profile.txt	2021-04-05 오전 4:40	텍스트 문서	1,370KB
README.md	2021-04-05 오전 4:40	Markdown 원본 ...	4KB
temp_file	2021-04-05 오전 4:40	파일	202KB
train.py	2021-04-05 오전 4:40	Python 원본 파일	11KB
voc.py	2021-04-05 오전 4:40	Python 원본 파일	3KB
yolo.py	2021-04-05 오전 4:40	Python 원본 파일	19KB
yolo_model.py	2021-04-05 오전 4:40	Python 원본 파일	21KB

Colab 환경에서 copilot.ipynb 파일을 실행하여 확인했습니다.

```
from frame import FRAME

file_path =  "./videos/nice_road.mp4"# <== Upload appropriate file
video_out = "videos/output11.mov"
frame =  FRAME(
    ego_vehicle_offset = .0,                    # offset for self vehicle in frame
    yellow_lower = np.uint8([ 20, 50,   110]),     # LOWER YELLOW HLS THRESHOLD
    yellow_upper = np.uint8([35, 255, 255]),       # UPER YELLOW HLS THRESHOLD
    white_lower = np.uint8([ 0, 140,   0]),        # LOWER WHITE THRESHOLD
    white_upper = np.uint8([180, 255, 100]),       # UPPER WHITE THRESHOLD
    lum_factor = 110,                              # NORMALIZING LUM FACTOR
    max_gap_th = 0.45,                             # MAX GAP THRESHOLD
    YOLO_PERIOD = 2,                            # YOLO PERIOD
    lane_start=[0.2,0.5] ,                      # LANE INITIATION
    verbose = 3)                                # VERBOSITY
frame.process_video(file_path, 2,₩
       video_out = video_out,pers_frame_time =398,₩
       t0  =398 , t1 =698)#None)
```

```
GOT EXEPTION TO PROCES THE IMAGE 3103
 69%|██████▉   | 6221/8991 [02:54<01:11, 38.78it/s]
LANE CHANGE TO LEFT

LANE CHANGE TO LEFT
 71%|███████   | 6346/8991 [02:58<01:07, 38.91it/s]
LANE CHANGE TO RIGHT
 72%|███████▏  | 6505/8991 [03:01<00:58, 42.19it/s]
LANE CHANGE TO RIGHT
```

FRAME 클래스를 이용하여 객체를 만들고, process_video 함수를 이용하여 차량과 차선을 인지한 동영상을 만듭니다. ego_vehicle_offset은 자신의 차량이 프레임의 몇 퍼센트를 차지하면 가장자리 부분을 잘라낼지 파라미터를 설정해주는 부분입니다. yellow_lower는 HLS의 낮은 임계값을 설정하는 부분입니다. 노란색 차선의 경우 RGB 보다는 HLS가 제대로 검출이되기 때문에 HLS를 사용하는데, HLS에서 노란차선이 검출이 제대로 되지 않을 경우 HLS 값을 낮춰 검출이 잘 되도록합니다. yellow_upper는 HLS의 높은 임계값을 설정하는 부분입니다. white_upper와 white_lower는 앞에 노란차선과 마찬가지로 흰색 차선에 적용할 임계값을 설정해주는 부분입니다. lum_factor는 광도를 정규화하게 사용되는 요인으로 화면에서 밝기와 관련이 있는 파라미터입니다. max_gap_th는 top-view 높이의 최대 차선감지 비율 임계값을 정해주는 값입니다. YOLO_PERIOD는 yolo가 감지된 후 시간을 설정하며, 처리해야 할 fps가 감소하면 객체 검출성능이 증가하는 설정값입니다. lane_start는 top-view에서 보았을 때, 차선의 시작이 어디일지 미리 추측한 값을 넣어줍니다. Verbose는 1일때는 많이 적게 표시하고, 2일때는 적게표시하고, 3일때는 모두 표시하는 것을 의미합니다. 즉, verbose값에 따라 화면에 표시되는 객체나 차선에 대한 정보의 양이 다릅니다.



3) 2)의 정리한 코드 중 하나 실행해서 결과 확인
- 구현 환경 및 구현하기 위한 코드 설명

2)에서 정리한 코드 중 2-1) Udacity-CarND-Vehicle-Detection-and-Tracking을 구현해보았습니다.

코드 실행에 사용된 컴퓨터의 사양은 프로세서: Intel i7-9750H @ 2.60GHz, Ram은 16GB 그래픽카드는 GeForce GTX 1660Ti입니다.

윈도우10에서 Anaconda3를 설치해 jupyter notebook을 사용하였고, 파이썬 버전은 3.7.4입니다.
Anaconda3는 https://www.anaconda.com/products/individual 사이트의 맨 아래쪽에 있는 Anaconda installers 중에서 windows 64-bit Graphical Installer를 설치했습니다.


```
(base) C:₩Users₩wonjune>python
Python 3.7.4 (default, Aug  9 2019, 18:34:13) [MSC v.1915 64 bit (AMD64)] :: Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

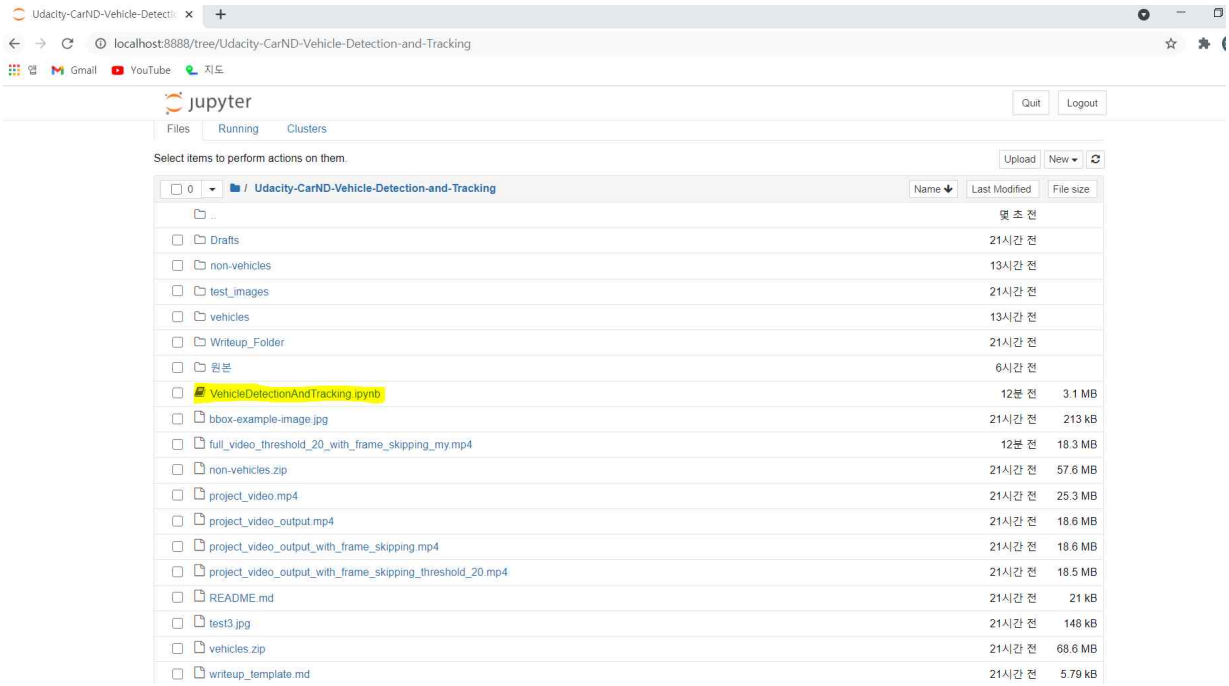
opencv 버전은 4.5.1입니다.

```
(base) C:₩Users₩wonjune>python
Python 3.7.4 (default, Aug  9 2019, 18:34:13) [MSC v.1915 64 bit (AMD64)] :: Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import cv2
>>> cv2.__version__
'4.5.1'
```

아나콘다 설치 후 pip install을 이용하여 numpy, scikit-learn, matplotlib, scipy등 필요하다고 생각되는 라이브러리를 설치했습니다.

Anaconda Prompt(Anaconda3)에서 jupyter notebook을 입력하면 jupyter notebook 화면이 나옵니다.

```
(base) C:₩Users₩wonjune₩Desktop₩VI_homework1>jupyter notebook
[I 02:11:39.049 NotebookApp] JupyterLab extension loaded from C:₩Users₩wonjune₩Anaconda3₩lib₩site-packages₩jupyterlab
[I 02:11:39.050 NotebookApp] JupyterLab application directory is C:₩Users₩wonjune₩Anaconda3₩share₩jupyter₩lab
[I 02:11:39.052 NotebookApp] Serving notebooks from local directory: C:₩Users₩wonjune₩Desktop₩VI_homework1
[I 02:11:39.052 NotebookApp] The Jupyter Notebook is running at:
[I 02:11:39.053 NotebookApp] http://localhost:8888/?token=85ad81f9160ba996f4794b660749ca838e044d9ec3cc8993
[I 02:11:39.053 NotebookApp]  or http://127.0.0.1:8888/?token=85ad81f9160ba996f4794b660749ca838e044d9ec3cc8993
[I 02:11:39.053 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 02:11:39.120 NotebookApp]

    To access the notebook, open this file in a browser:
        file:///C:/Users/wonjune/AppData/Roaming/jupyter/runtime/nbserver-25956-open.html
    Or copy and paste one of these URLs:
        http://localhost:8888/?token=85ad81f9160ba996f4794b660749ca838e044d9ec3cc8993
     or http://127.0.0.1:8888/?token=85ad81f9160ba996f4794b660749ca838e044d9ec3cc8993
```



노란색으로 표시된 ipynb 파일은 jupyter notebook 환경에서 쉽게 확인하고, 수정 및 실행이 가능합니다. 지금은 VehicleDetectionAndTracking.ipynb 왼쪽에 회색 책으로 표시가 되어있지만, 이 파일을 열게 되면 초록색으로 바뀌게 됩니다. 초록색으로 바뀌는 이유는 실행 중이라는 것을 나타냅니다.

위의 사진처럼 초록색으로 변하고, 오른쪽에 Running이라고 표시가 되는 것을 알 수 있습니다.

jupyter VehicleDetectionAndTracking Last Checkpoint: 한 시간 전 (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted Python 3

Run Code

In [1]:
```
#코드 실행에 필요한 라이브러리를 import를 사용해 가져옵니다.
import glob
import cv2 as cv2
import numpy as np   # as를 이용해 numpy 모듈을 np로 지정해 전부 타이핑 하는 수고를 줍니다.
import matplotlib.pyplot as plt
import random
from skimage.feature import hog # skimage.feature 모듈에서 hog만 가져옵니다.
```

실행순서 1번째를 나타냅니다.

하나의 Cell

Step 1 - Read Images

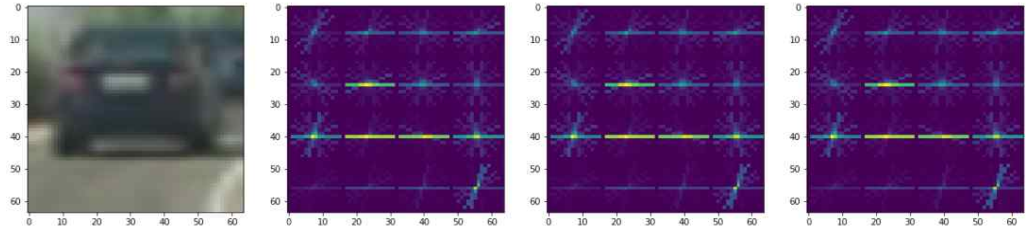
In [2]:
```
#reading image paths with glob
# glob을 이용해 이미지 파일의 경로명에 접근해 아래의 코드에서는 vehicles/vehicles/*/에서 확장자가 png인 파일만 추출합니다.
vehicle_image_arr = glob.glob('./vehicles/vehicles/*/*.png')

# read images and append to list
# images를 읽고 vehicle_images_original 리스트에 추가합니다.
vehicle_images_original=[]
for imagePath in vehicle_image_arr:
    readImage=cv2.imread(imagePath)
    rgbImage = cv2.cvtColor(readImage, cv2.COLOR_BGR2RGB)
    vehicle_images_original.append(rgbImage)

# image를 다 읽고, 리스트에 저장한 후 끝났다는 것을 알려주기 위해 print를 이용해 메시지를 출력합니다.
print('Reading of Vehicle Images Done')

#glob을 이용해 파일의 경로에 접근해 non-vehicles에 해당하는 이미지 파일들을 non_vehicle_image_arr에 저장합니다.
non_vehicle_image_arr = glob.glob('./non-vehicles/non-vehicles/*/*.png')

# images를 읽고, non_vehilce_images_original 리스트에 추가합니다.
non_vehicle_images_original=[]
for imagePath in non_vehicle_image_arr:
```

쥬피터 노트북에 들어가면 각각의 Cell로 이루어져 있습니다. Cell을 클릭하면 얇은 초록색 사각형이 그려지는 것을 알 수 있습니다. 왼쪽의 In 옆에 있는 [1], [2]는 실행순서를 의미합니다.
[1]인 경우 가장 먼저 실행되었음을 알 수 있습니다. Cell을 실행하기 위해서는 위의 메뉴 중 Cell에서 Run Cells을 클릭하면 실행됩니다. 단축키로 실행할 수도 있습니다. 셀을 클릭 후 'Ctrl + Enter' 키를 누르면 Cell이 실행이 됩니다. 그리고 Cell 실행 후 아래로 커서를 옮기고 싶을 경우 'Shift + Enter' 키를 누릅니다. 또한 Cell에서 Run ALL을 클릭하면 전부 실행이 되는 것을 알 수 있습니다.

In [10]:
```
# 위에서 만든  GetFeaturesFromHog 가 제대로 작동하는지 확인하기 위한 코드

image=vehicle_images_original[1]
image= cv2.cvtColor(image, cv2.COLOR_RGB2YUV)
image_channel_0=image[:,:,0]
image_channel_1=image[:,:,0]
image_channel_2=image[:,:,0]

#차례대로 image_channel , orinetation = 9, cellsPerBlock=2, pixelsPerCell=16, visualize= True, feature_vector_flag는 True로 지정해주었다.
feature_0,hog_img_0=GetFeaturesFromHog(image_channel_0,9,2,16,visualise=True,feature_vector_flag=True)
feature_1,hog_img_1=GetFeaturesFromHog(image_channel_1,9,2,16,visualise=True,feature_vector_flag=True)
feature_2,hog_img_2=GetFeaturesFromHog(image_channel_2,9,2,16,visualise=True,feature_vector_flag=True)

f, axes= plt.subplots(1,4,figsize=(20,10)) # 1행 4열의 subplots를 만든다.
axes[0].imshow(vehicle_images_original[1])
axes[1].imshow(hog_img_0)
axes[2].imshow(hog_img_1)
axes[3].imshow(hog_img_2)
# 각각hog_img 채널에 해당하는 그림 보여줍니다.

print("Feature Vector Length Returned is ",len(feature_0))    # Feature Vector의 길이를 출력해줍니다.
print("No of features that can be extracted from image ",len(hog_img_0.ravel())) # 1차원으로 변환했을 때 features의 개수를 출력합니다.
```

```
Feature Vector Length Returned is  324
No of features that can be extracted from image  4096
```

해당 셀에서 실행된 결과



각 셀 실행 후 결과는 해당 셀 아래에 보여줍니다.

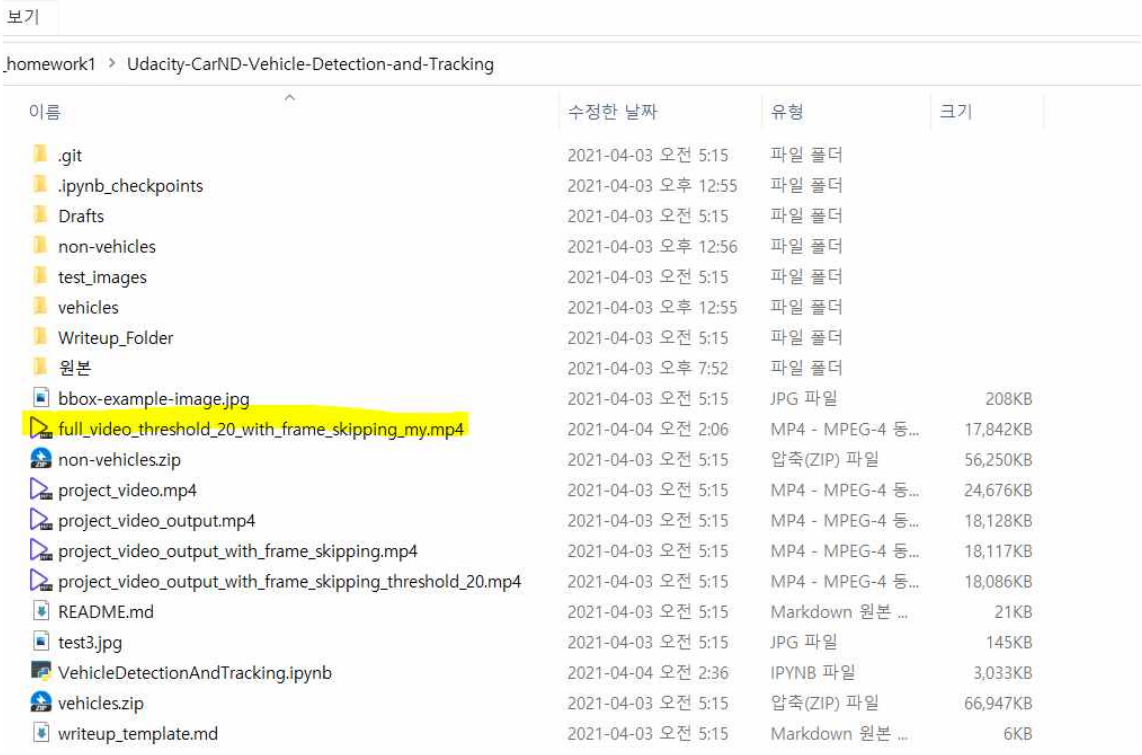
Step 11- Testing Pipeline on Test Video

In [32]:
```
# 원본영상을 이용해 차량에 해당하는 부분을 표시하는 영상을 만듭니다.

keepTrack = KeepTrack()
import moviepy
from moviepy.editor import VideoFileClip
video_output1 = 'full_video_threshold_20_with_frame_skipping_my.mp4' # 결과물 영상
video_input1 = VideoFileClip('project_video.mp4')   # 원본 영상
processed_video = video_input1.fl_image(Pipeline)
%time processed_video.write_videofile(video_output1, audio=False)  # vide_output1에
video_input1.reader.close()
video_input1.audio.reader.close_proc()
```

```
Moviepy - Building video full_video_threshold_20_with_frame_skipping_my.mp4.
Moviepy - Writing video full_video_threshold_20_with_frame_skipping_my.mp4
```

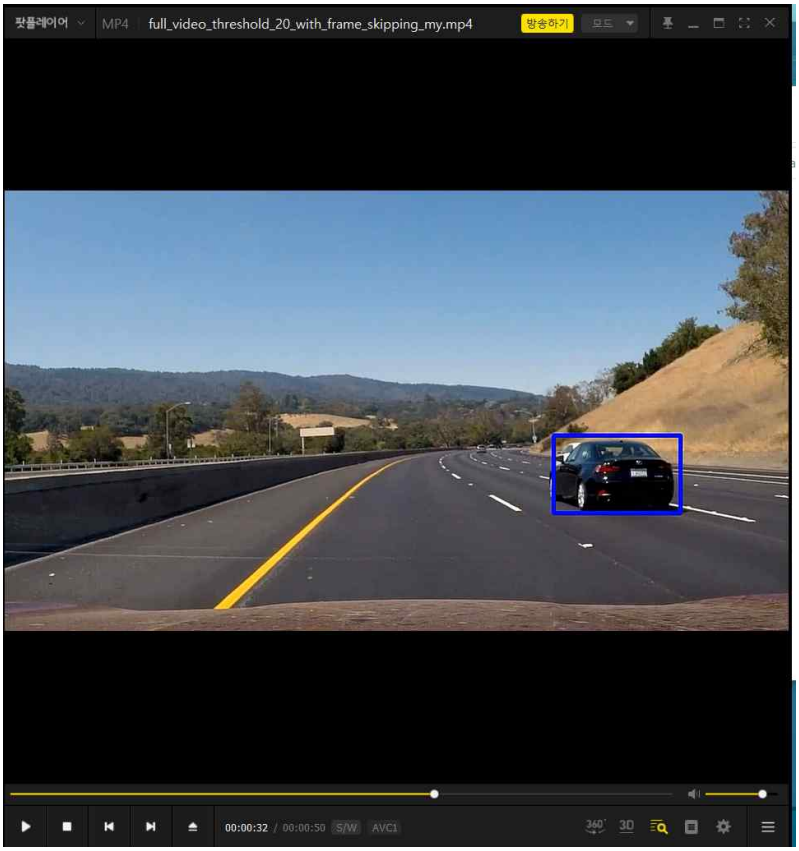
```
Moviepy - Done !
Moviepy - video ready full_video_threshold_20_with_frame_skipping_my.mp4
Wall time: 14min 6s
```

VehicleDetectionAndTracking.ipynb 파일의 마지막 부분에서 원본 영상을 가지고, 위의 소스코드를 활용해 차량을 인지한 결과영상을 만듭니다.



위와 같이 원본 영상이 있는 파일 경로에 결과 영상이 생긴 것을 확인할 수 있습니다.

full_video_threshold_20_with_frame_skipping_my.mp4 파일을 실행시켜보면 아래와 같이 차량을 인지하고 파란색 사각형을 그리는 것을 확인할 수 있었습니다. 영상은 팟플레이어를 이용해 확인했습니다.



참고문헌

-Dataset

1) BDD100K : https://bdd-data.berkeley.edu/
2) BDD100K 추가설명(korean): http://www.rex-ai.info/docs/Papers_BDD
3) BDD100K youtube : https://www.youtube.com/watch?v=ANQczqZwaY4
4) Cityscapes dataset : https://www.cityscapes-dataset.com/
5) Cityscapes github : https://github.com/mcordts/cityscapesScripts
6) Cityscapes dataset tree : https://towardsdatascience.com/training-road-scene-segmentation-on-cityscapes-with-supervisely-tensorflow-and-unet-1232314781a8
7) Cityscapes using dataset youtube : https://www.youtube.com/watch?v=QrB7Np_8GXY
8) Waymo dataset : https://waymo.com/open/
9) Waymo dataset github : https://github.com/waymo-research/waymo-open-dataset
10) Waymo various environment gif : https://waymo.com/open/about/
11) Waymo dataset video : https://www.youtube.com/watch?v=ASqBKIrS6co

-Open source

1) Udacity-CarND-Vehicle-Detection-and-Tracking의 github 주소 : https://github.com/harveenchadha/Udacity-CarND-Vehicle-Detection-and-Tracking
2) Copilot의 github 주소 : https://github.com/visualbuffer/copilot
3) Copilot article : https://towardsdatascience.com/copilot-driving-assistance-635e1a50f14
4) Yolo v3 설명 : https://wingnim.tistory.com/56
5) 아나콘다에서 케라스 설치하는 방법(가상환경설정) : https://like-edp.tistory.com/3
6) yolo v3 모델,출력 구조: http://daddynkidsmakers.blogspot.com/2020/05/yolo.html