Car Detection using Mask-RCNN using pre-trained model and Open CV

Agenda:

1) Use OpenCV Selective Search to simply detect Car using VGG16 model
2) Use Mask-RCNN + COCO weight to detect car using ResNet101

Data Set:

https://www.kaggle.com/datasets/sshikamaru/car-object-detection

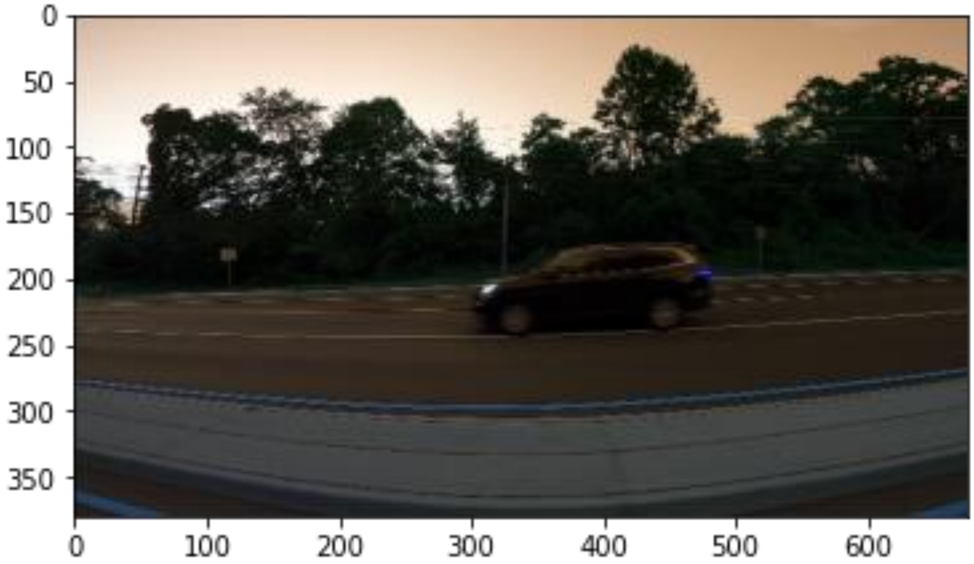
Total Number of Sample: 1001

Training set: 500 (annotated)

Validation Set: 59 (annotated)

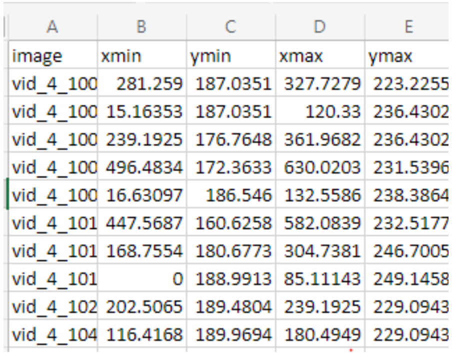
Testing Set: 175 (Unannotated)

Sample Data:

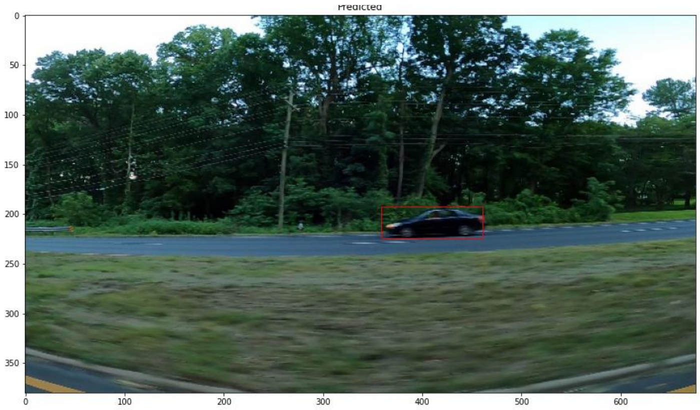


Annotated data: Data annotation done by manually

../input/car-object-detection/data/train_solution_bounding_boxes (1).csv

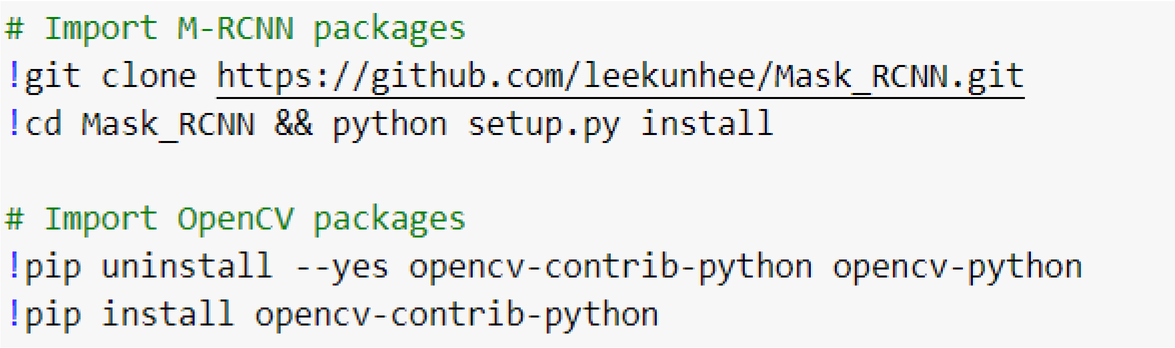


A	B	C	D	E
image	xmin	ymin	xmax	ymax
vid_4_100	281.259	187.0351	327.7279	223.2255
vid_4_100	15.16353	187.0351	120.33	236.4302
vid_4_100	239.1925	176.7648	361.9682	236.4302
vid_4_100	496.4834	172.3633	630.0203	231.5396
vid_4_100	16.63097	186.546	132.5586	238.3864
vid_4_101	447.5687	160.6258	582.0839	232.5177
vid_4_101	168.7554	180.6773	304.7381	246.7005
vid_4_101	0	188.9913	85.11143	249.1458
vid_4_102	202.5065	189.4804	239.1925	229.0943
vid_4_104	116.4168	189.9694	180.4949	229.0943

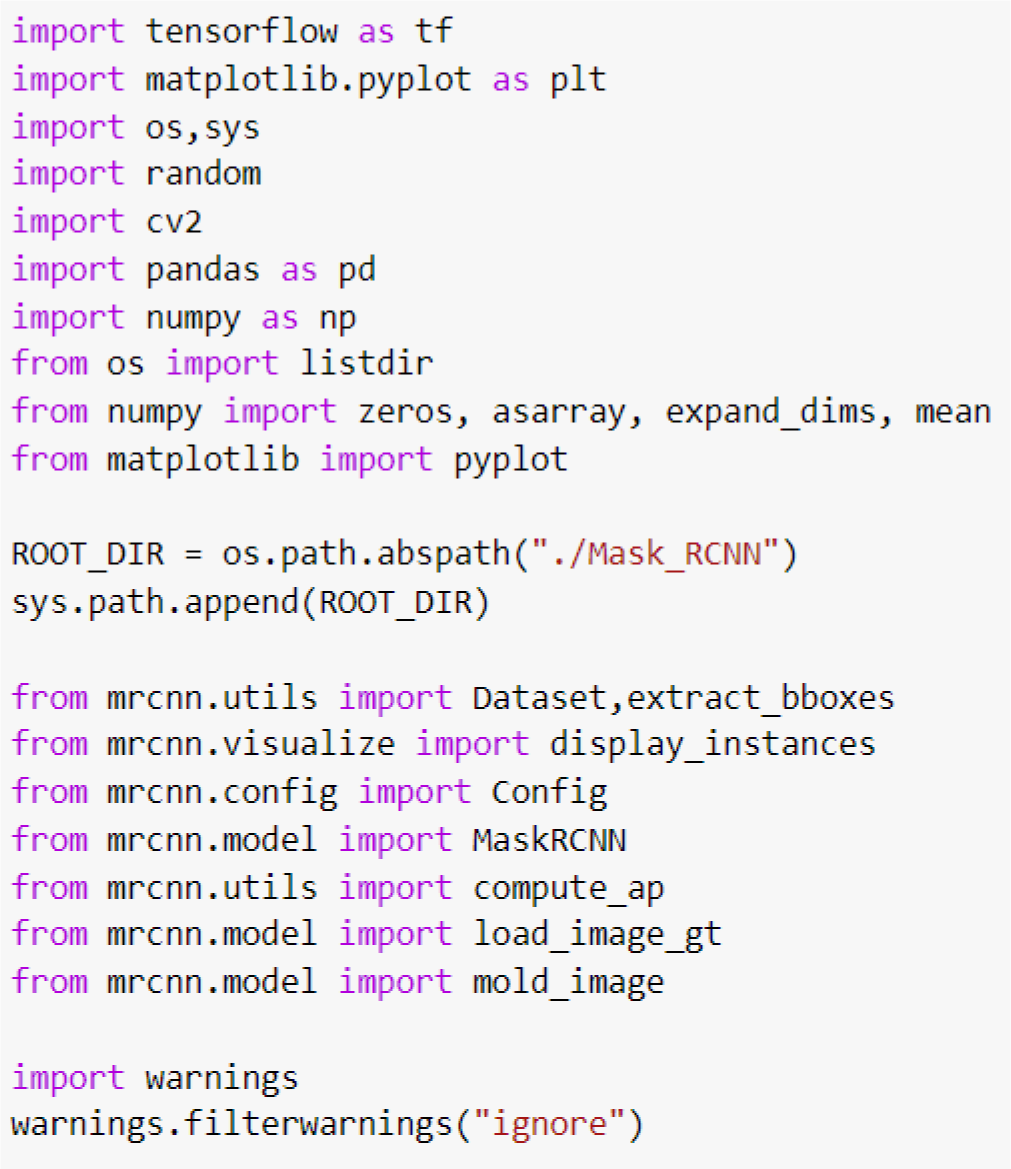


Source Code:

1. **Importing and Installing libraries**

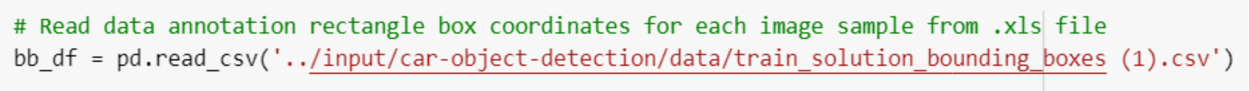


```
# Import M-RCNN packages
!git clone https://github.com/leekunhee/Mask_RCNN.git
!cd Mask_RCNN && python setup.py install

# Import OpenCV packages
!pip uninstall --yes opencv-contrib-python opencv-python
!pip install opencv-contrib-python
```

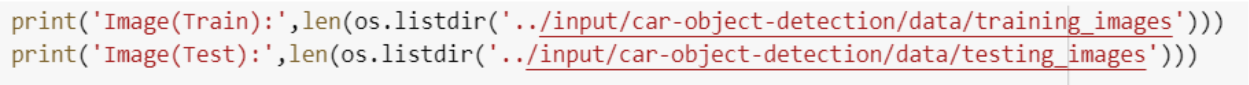


```
import tensorflow as tf
import matplotlib.pyplot as plt
import os,sys
import random
import cv2
import pandas as pd
import numpy as np
from os import listdir
from numpy import zeros, asarray, expand_dims, mean
from matplotlib import pyplot

ROOT_DIR = os.path.abspath("./Mask_RCNN")
sys.path.append(ROOT_DIR)

from mrcnn.utils import Dataset,extract_bboxes
from mrcnn.visualize import display_instances
from mrcnn.config import Config
from mrcnn.model import MaskRCNN
from mrcnn.utils import compute_ap
from mrcnn.model import load_image_gt
from mrcnn.model import mold_image

import warnings
warnings.filterwarnings("ignore")
```

2. **Load dataset**



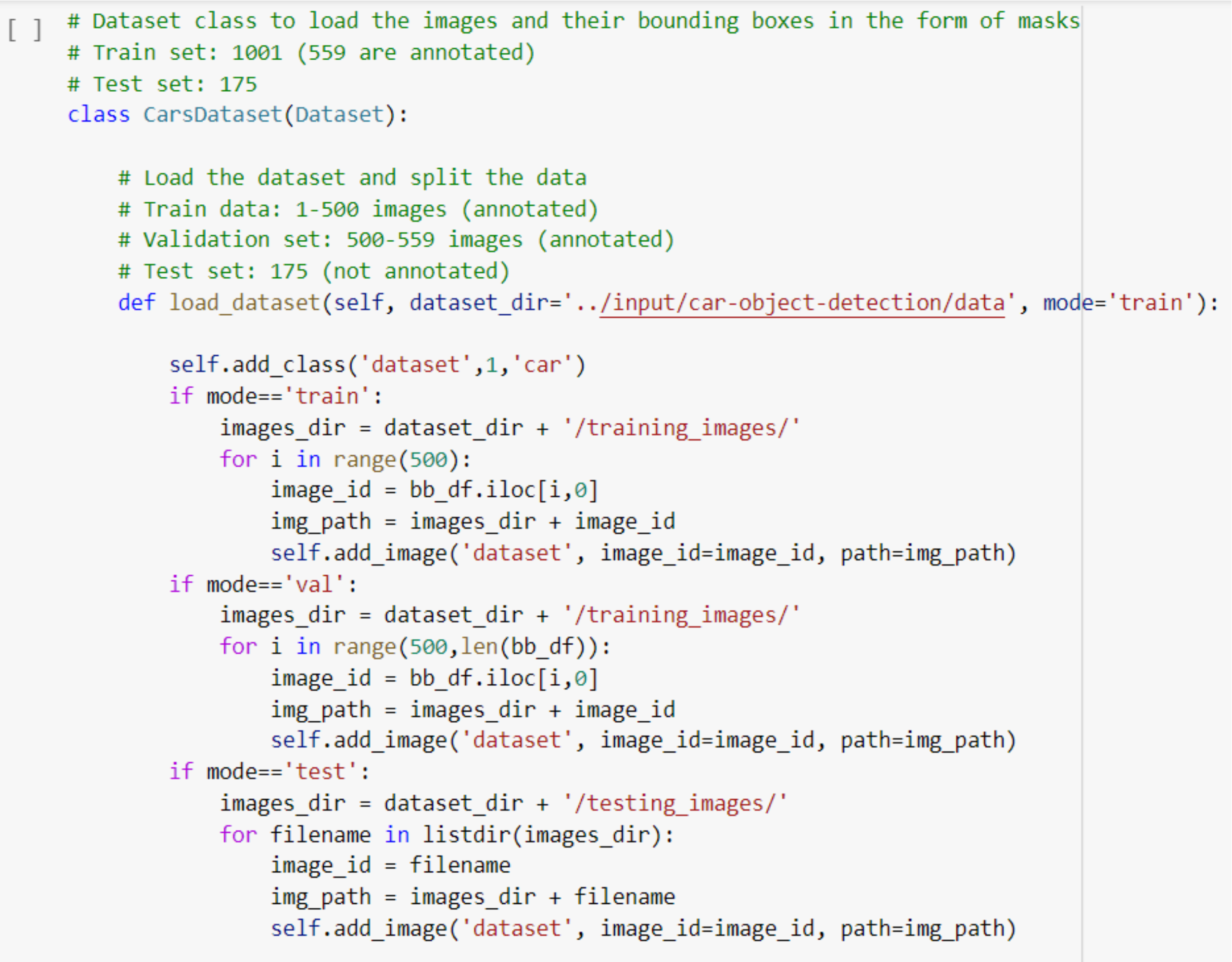
```
# Read data annotation rectangle box coordinates for each image sample from .xls file
bb_df = pd.read_csv('../input/car-object-detection/data/train_solution_bounding_boxes (1).csv')
```



```
print('Image(Train):',len(os.listdir('../input/car-object-detection/data/training_images')))
print('Image(Test):',len(os.listdir('../input/car-object-detection/data/testing_images')))
```

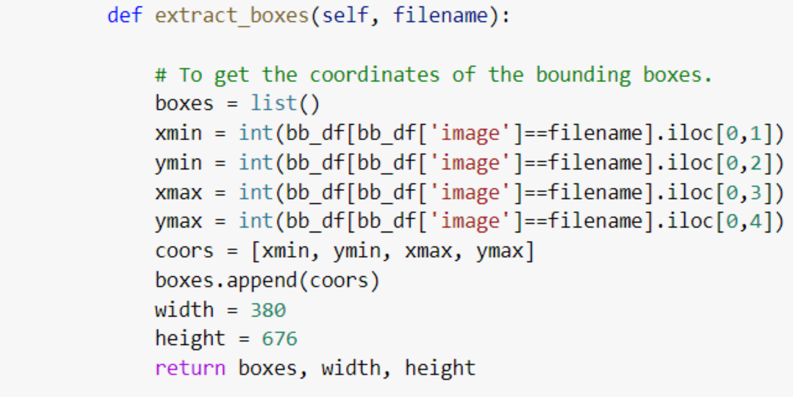
3. Preprocess the annotated data

a) Split the data into train and validation set



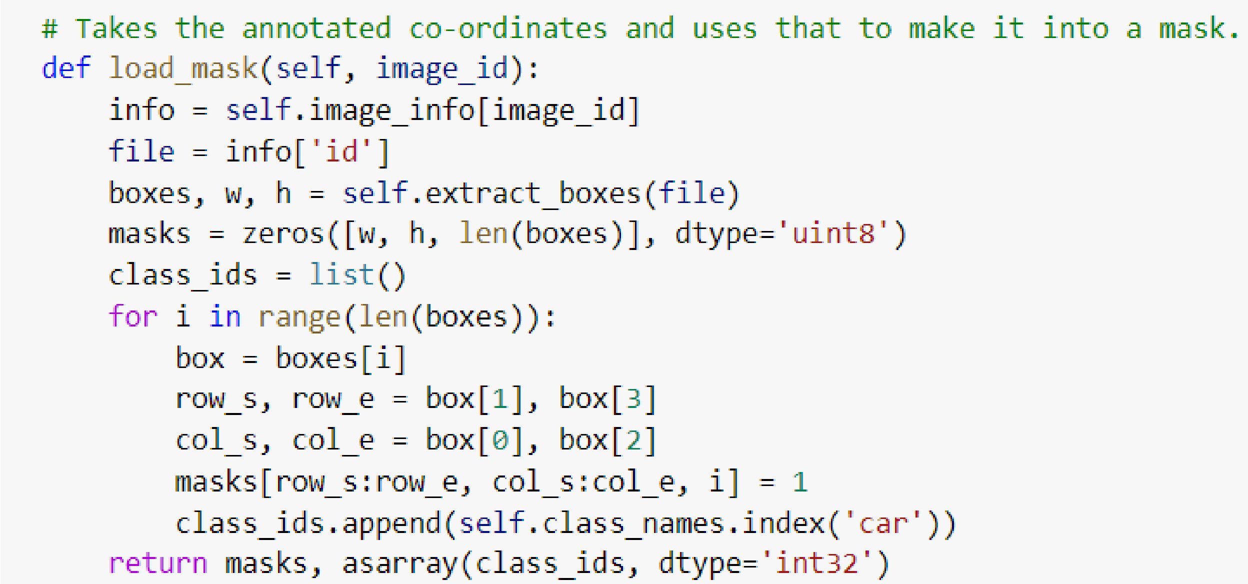
```
# Dataset class to load the images and their bounding boxes in the form of masks
# Train set: 1001 (559 are annotated)
# Test set: 175
class CarsDataset(Dataset):

    # Load the dataset and split the data
    # Train data: 1-500 images (annotated)
    # Validation set: 500-559 images (annotated)
    # Test set: 175 (not annotated)
    def load_dataset(self, dataset_dir='../input/car-object-detection/data', mode='train'):

        self.add_class('dataset',1,'car')
        if mode=='train':
            images_dir = dataset_dir + '/training_images/'
            for i in range(500):
                image_id = bb_df.iloc[i,0]
                img_path = images_dir + image_id
                self.add_image('dataset', image_id=image_id, path=img_path)
        if mode=='val':
            images_dir = dataset_dir + '/training_images/'
            for i in range(500,len(bb_df)):
                image_id = bb_df.iloc[i,0]
                img_path = images_dir + image_id
                self.add_image('dataset', image_id=image_id, path=img_path)
        if mode=='test':
            images_dir = dataset_dir + '/testing_images/'
            for filename in listdir(images_dir):
                image_id = filename
                img_path = images_dir + filename
                self.add_image('dataset', image_id=image_id, path=img_path)
```

b) Prepare Bounty box from annotated co-ordinates data



```
def extract_boxes(self, filename):

    # To get the coordinates of the bounding boxes.
    boxes = list()
    xmin = int(bb_df[bb_df['image']==filename].iloc[0,1])
    ymin = int(bb_df[bb_df['image']==filename].iloc[0,2])
    xmax = int(bb_df[bb_df['image']==filename].iloc[0,3])
    ymax = int(bb_df[bb_df['image']==filename].iloc[0,4])
    coors = [xmin, ymin, xmax, ymax]
    boxes.append(coors)
    width = 380
    height = 676
    return boxes, width, height
```

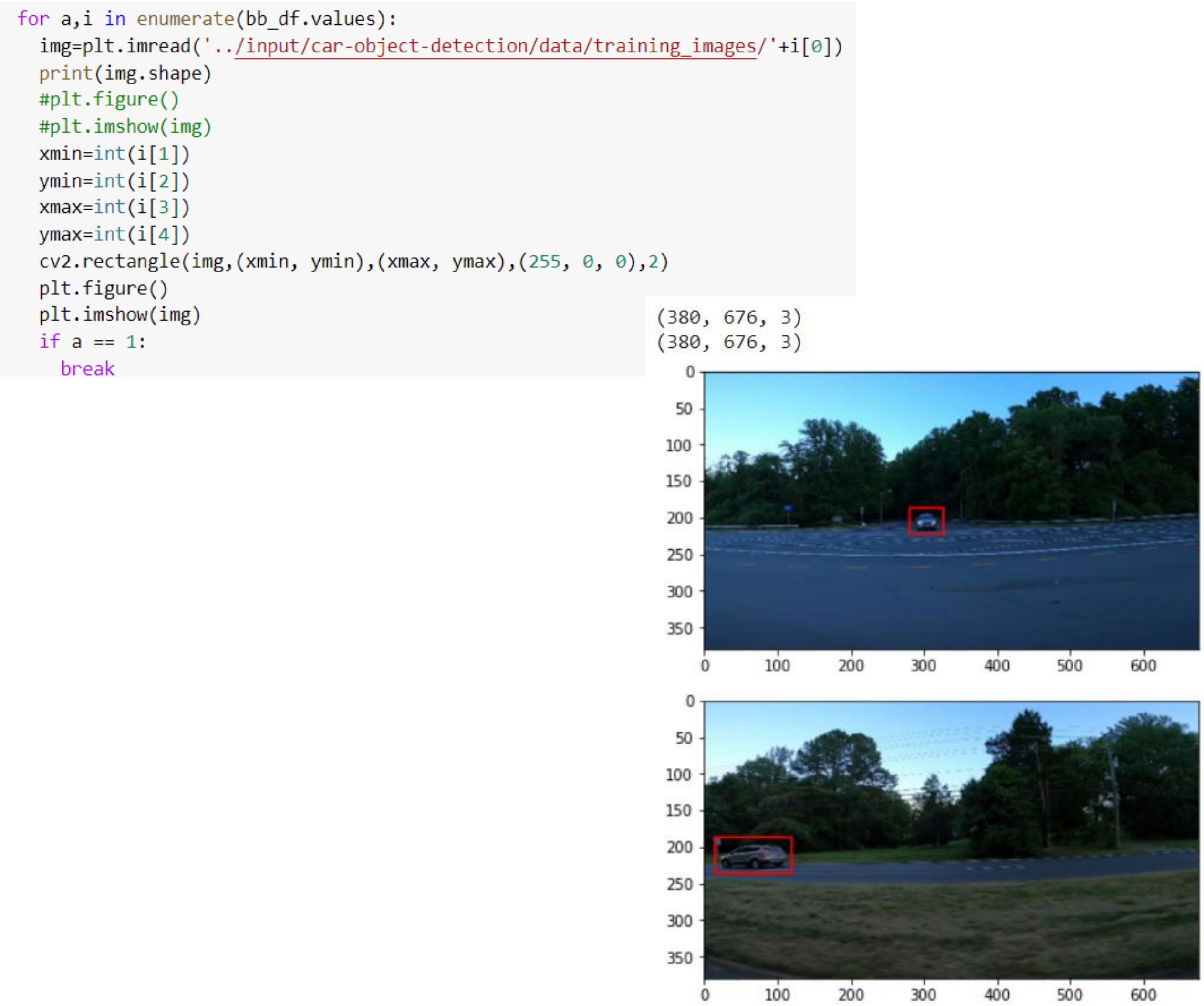
c) Prepare Mask from annotated co-ordinates data



```
# Takes the annotated co-ordinates and uses that to make it into a mask.
def load_mask(self, image_id):
    info = self.image_info[image_id]
    file = info['id']
    boxes, w, h = self.extract_boxes(file)
    masks = zeros([w, h, len(boxes)], dtype='uint8')
    class_ids = list()
    for i in range(len(boxes)):
        box = boxes[i]
        row_s, row_e = box[1], box[3]
        col_s, col_e = box[0], box[2]
        masks[row_s:row_e, col_s:col_e, i] = 1
        class_ids.append(self.class_names.index('car'))
    return masks, asarray(class_ids, dtype='int32')
```

4. **Visualize mask**

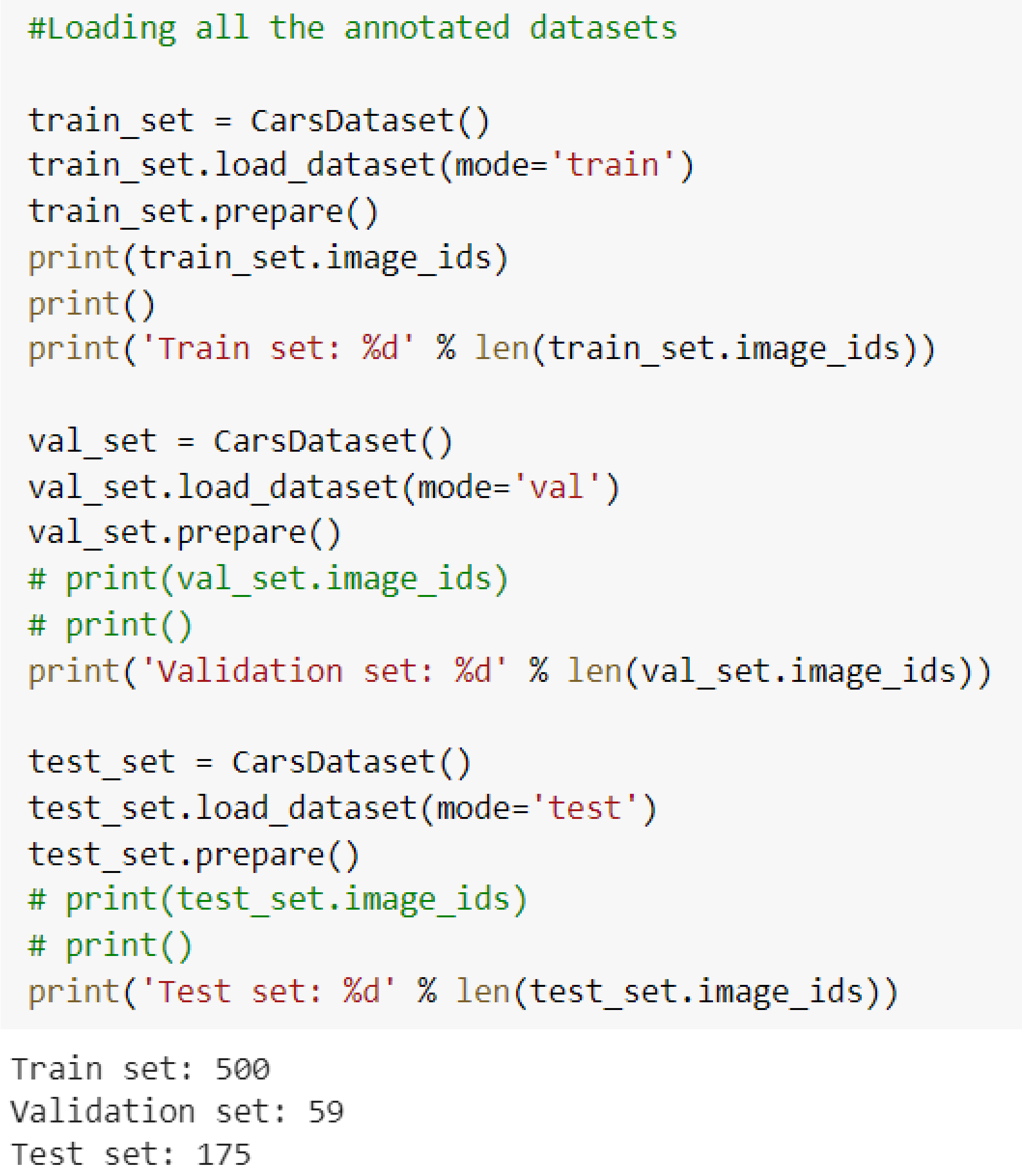
a) **Visualize using Open CV**



```
for a,i in enumerate(bb_df.values):
  img=plt.imread('../input/car-object-detection/data/training_images/'+i[0])
  print(img.shape)
  #plt.figure()
  #plt.imshow(img)
  xmin=int(i[1])
  ymin=int(i[2])
  xmax=int(i[3])
  ymax=int(i[4])
  cv2.rectangle(img,(xmin, ymin),(xmax, ymax),(255, 0, 0),2)
  plt.figure()
  plt.imshow(img)
  if a == 1:
    break
```

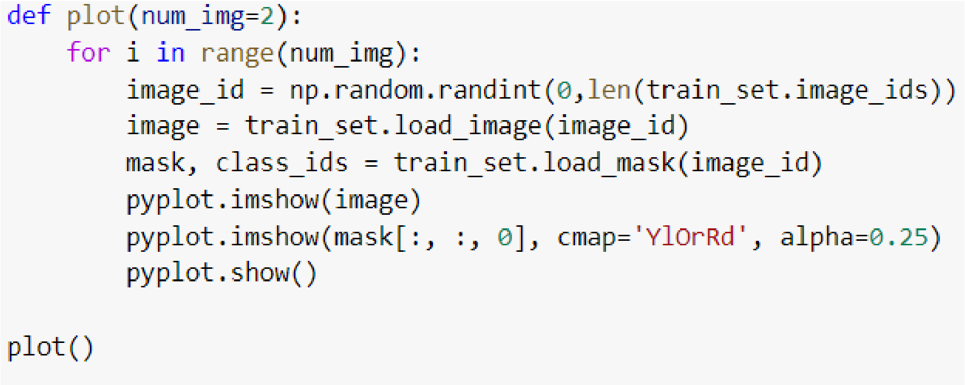
```
(380, 676, 3)
(380, 676, 3)
```

b) Visualize using Mask-R CNN package

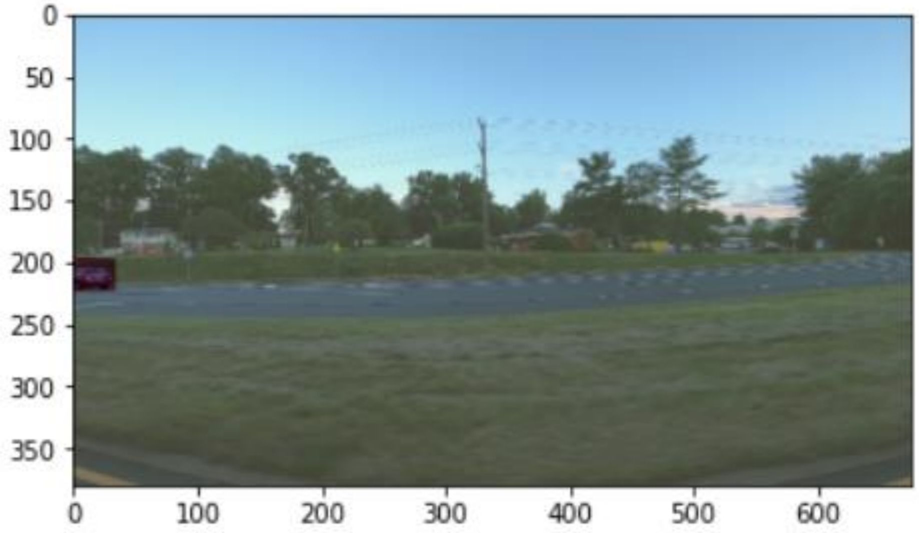


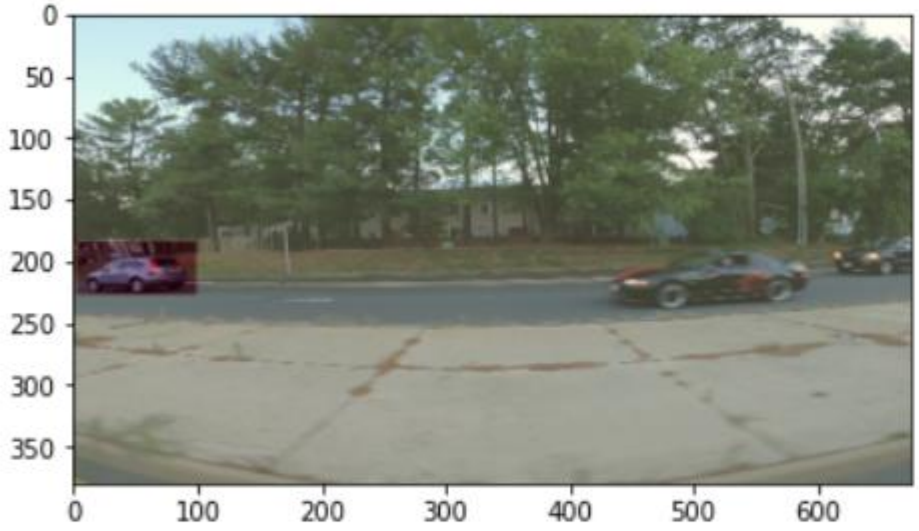
```
#Loading all the annotated datasets

train_set = CarsDataset()
train_set.load_dataset(mode='train')
train_set.prepare()
print(train_set.image_ids)
print()
print('Train set: %d' % len(train_set.image_ids))

val_set = CarsDataset()
val_set.load_dataset(mode='val')
val_set.prepare()
# print(val_set.image_ids)
# print()
print('Validation set: %d' % len(val_set.image_ids))

test_set = CarsDataset()
test_set.load_dataset(mode='test')
test_set.prepare()
# print(test_set.image_ids)
# print()
print('Test set: %d' % len(test_set.image_ids))
```

```
Train set: 500
Validation set: 59
Test set: 175
```

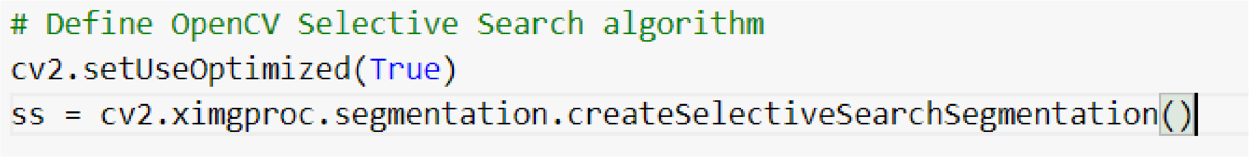


```
def plot(num_img=2):
    for i in range(num_img):
        image_id = np.random.randint(0,len(train_set.image_ids))
        image = train_set.load_image(image_id)
        mask, class_ids = train_set.load_mask(image_id)
        pyplot.imshow(image)
        pyplot.imshow(mask[:, :, 0], cmap='YlOrRd', alpha=0.25)
        pyplot.show()

plot()
```



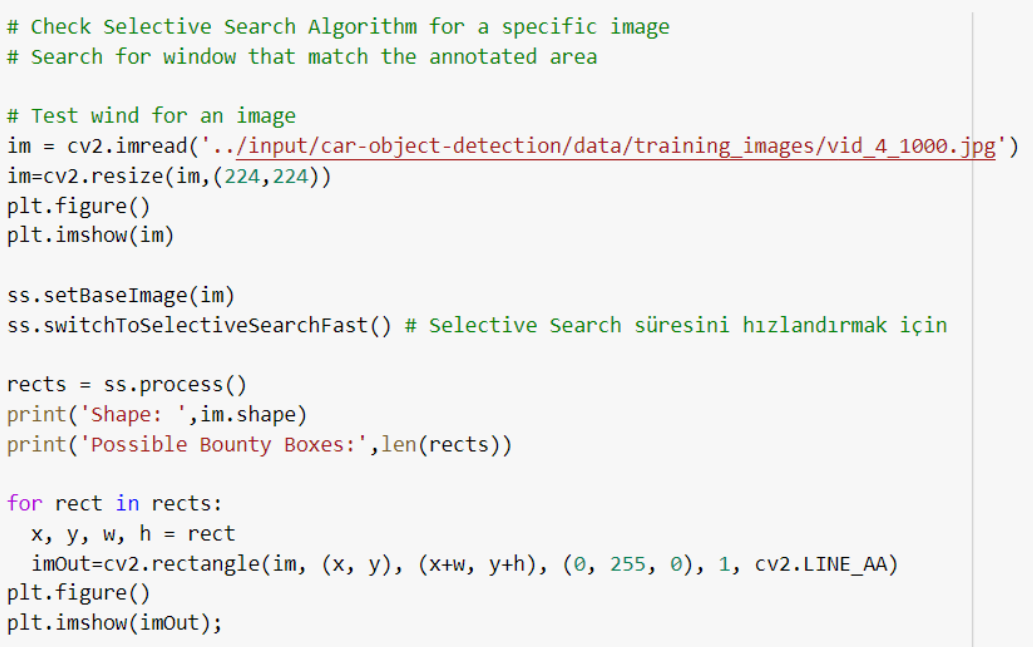


5. **Prepare Mask to fed in Deep Learning models**
   a) **Using Segmentation and Selective Search for Feature Extraction using OpenCV**



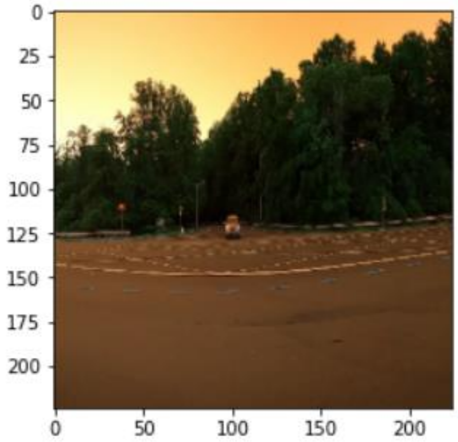
```
# Define OpenCV Selective Search algorithm
cv2.setUseOptimized(True)
ss = cv2.ximgproc.segmentation.createSelectiveSearchSegmentation()
```

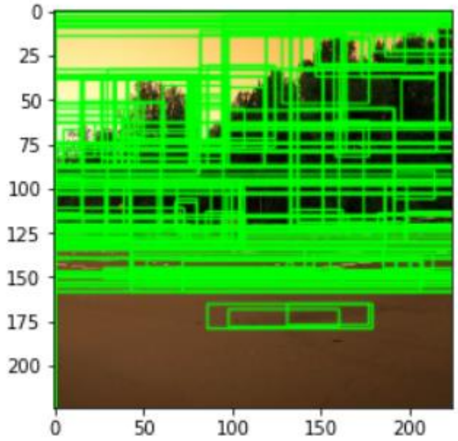
Test the Masking Search algorithm



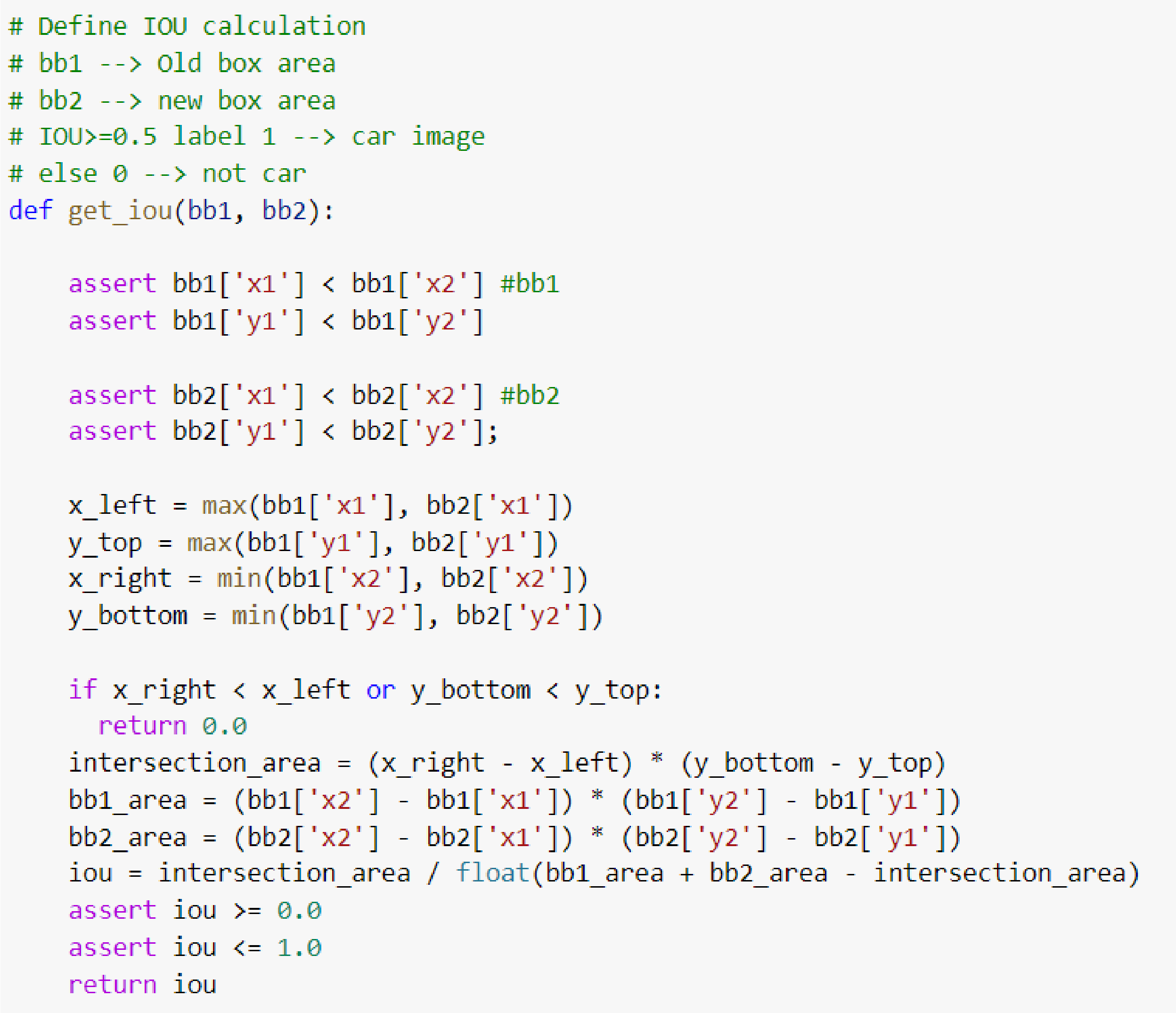
```
# Check Selective Search Algorithm for a specific image
# Search for window that match the annotated area

# Test wind for an image
im = cv2.imread('../input/car-object-detection/data/training_images/vid_4_1000.jpg')
im=cv2.resize(im,(224,224))
plt.figure()
plt.imshow(im)

ss.setBaseImage(im)
ss.switchToSelectiveSearchFast() # Selective Search süresini hızlandırmak için

rects = ss.process()
print('Shape: ',im.shape)
print('Possible Bounty Boxes:',len(rects))

for rect in rects:
  x, y, w, h = rect
  imOut=cv2.rectangle(im, (x, y), (x+w, y+h), (0, 255, 0), 1, cv2.LINE_AA)
plt.figure()
plt.imshow(imOut);
```

```
Shape:  (224, 224, 3)
Possible Bounty Boxes: 213
```



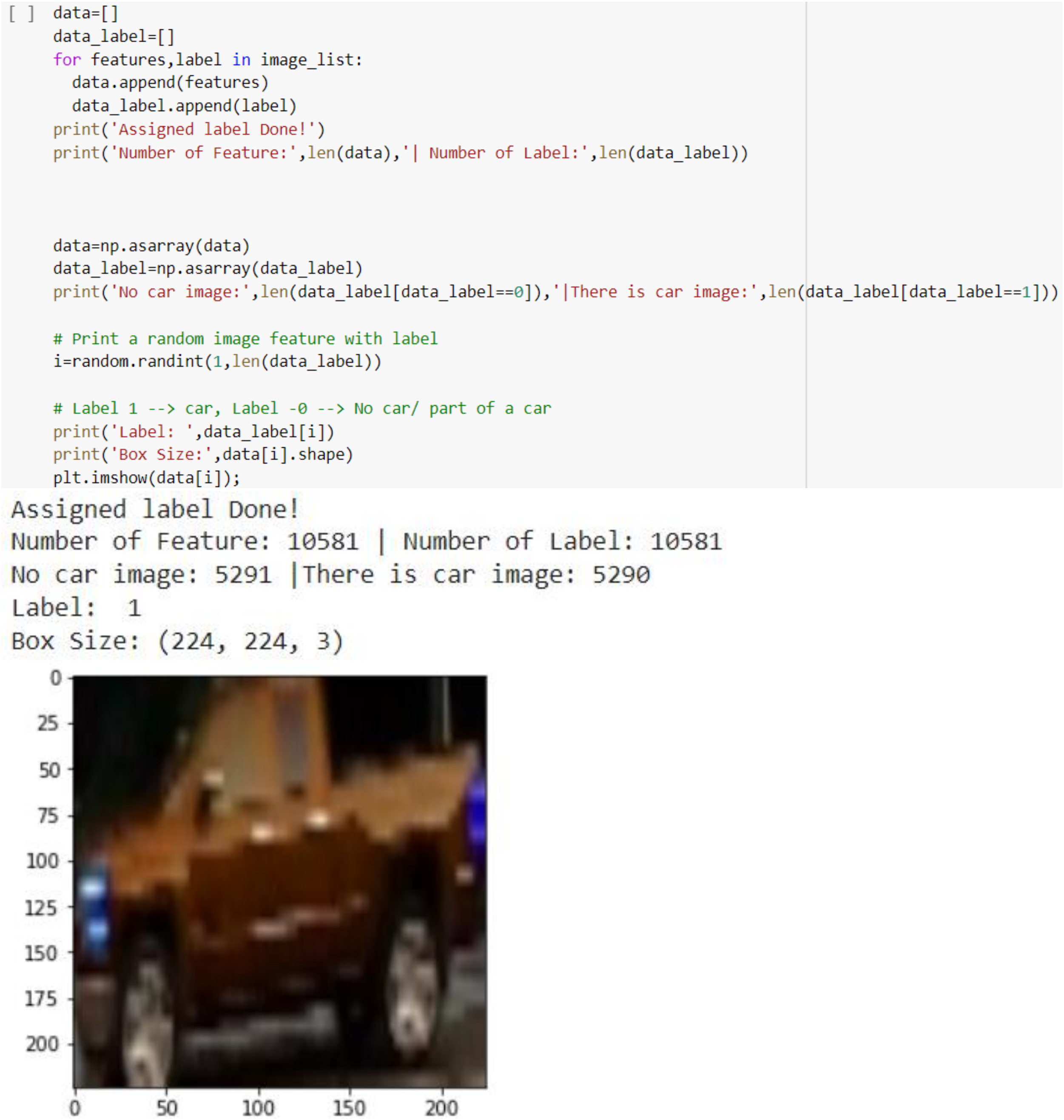


Prepare masking accuracy using IOU



```
# Define IOU calculation
# bb1 --> Old box area
# bb2 --> new box area
# IOU>=0.5 label 1 --> car image
# else 0 --> not car
def get_iou(bb1, bb2):

    assert bb1['x1'] < bb1['x2'] #bb1
    assert bb1['y1'] < bb1['y2']

    assert bb2['x1'] < bb2['x2'] #bb2
    assert bb2['y1'] < bb2['y2'];

    x_left = max(bb1['x1'], bb2['x1'])
    y_top = max(bb1['y1'], bb2['y1'])
    x_right = min(bb1['x2'], bb2['x2'])
    y_bottom = min(bb1['y2'], bb2['y2'])

    if x_right < x_left or y_bottom < y_top:
      return 0.0
    intersection_area = (x_right - x_left) * (y_bottom - y_top)
    bb1_area = (bb1['x2'] - bb1['x1']) * (bb1['y2'] - bb1['y1'])
    bb2_area = (bb2['x2'] - bb2['x1']) * (bb2['y2'] - bb2['y1'])
    iou = intersection_area / float(bb1_area + bb2_area - intersection_area)
    assert iou >= 0.0
    assert iou <= 1.0
    return iou
```

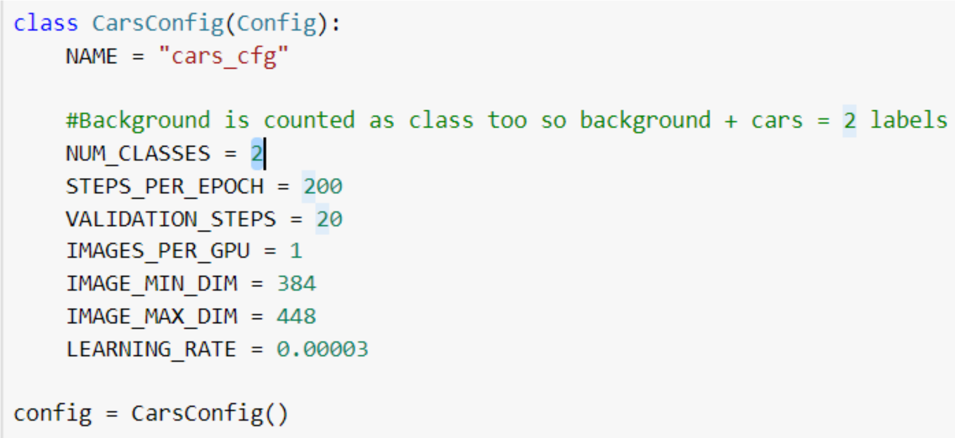
Assign label for correctly masked image



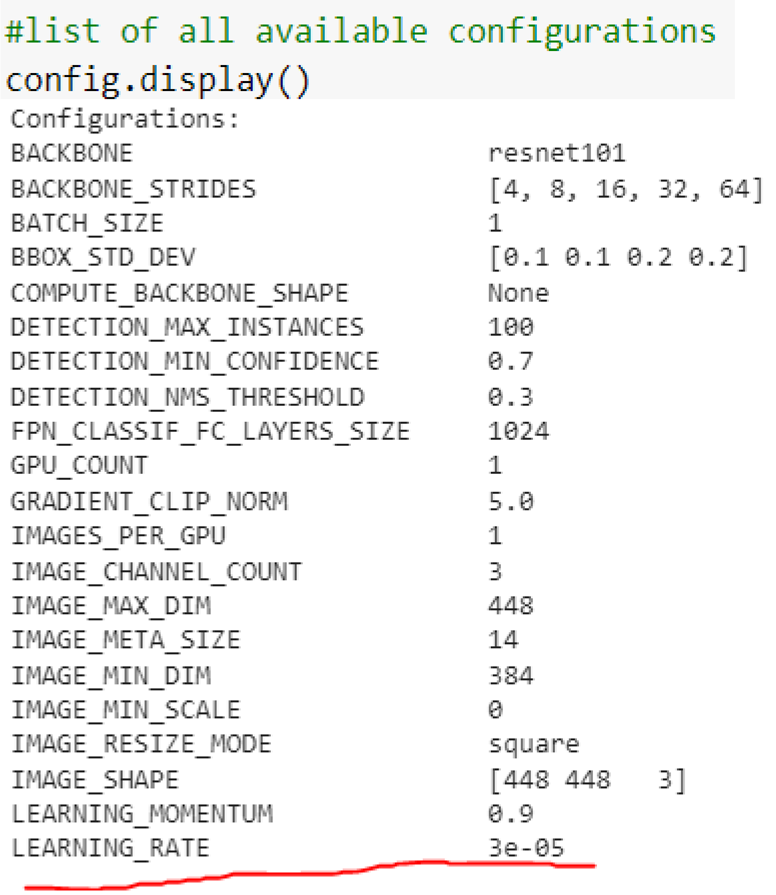
```
[ ] data=[]
    data_label=[]
    for features,label in image_list:
      data.append(features)
      data_label.append(label)
    print('Assigned label Done!')
    print('Number of Feature:',len(data),'| Number of Label:',len(data_label))



    data=np.asarray(data)
    data_label=np.asarray(data_label)
    print('No car image:',len(data_label[data_label==0]),'|There is car image:',len(data_label[data_label==1]))

    # Print a random image feature with label
    i=random.randint(1,len(data_label))

    # Label 1 --> car, Label -0 --> No car/ part of a car
    print('Label: ',data_label[i])
    print('Box Size:',data[i].shape)
    plt.imshow(data[i]);
```

```
Assigned label Done!
Number of Feature: 10581 | Number of Label: 10581
No car image: 5291 |There is car image: 5290
Label:  1
Box Size: (224, 224, 3)
```

b) Configuration setting for Mask-RCNN and ResNet101 Pretrained Configuration



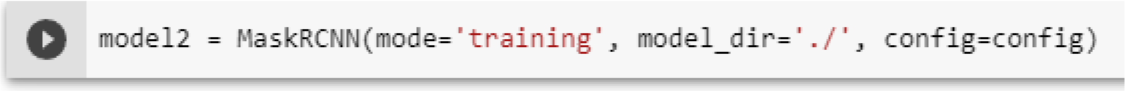
```
class CarsConfig(Config):
    NAME = "cars_cfg"

    #Background is counted as class too so background + cars = 2 labels
    NUM_CLASSES = 2
    STEPS_PER_EPOCH = 200
    VALIDATION_STEPS = 20
    IMAGES_PER_GPU = 1
    IMAGE_MIN_DIM = 384
    IMAGE_MAX_DIM = 448
    LEARNING_RATE = 0.00003

config = CarsConfig()
```



```
#list of all available configurations
config.display()
```

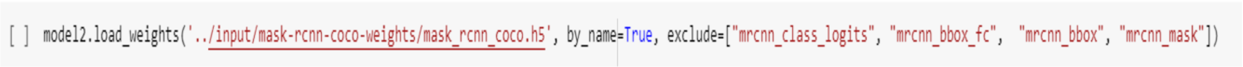
```
Configurations:
BACKBONE                       resnet101
BACKBONE_STRIDES               [4, 8, 16, 32, 64]
BATCH_SIZE                     1
BBOX_STD_DEV                   [0.1 0.1 0.2 0.2]
COMPUTE_BACKBONE_SHAPE         None
DETECTION_MAX_INSTANCES        100
DETECTION_MIN_CONFIDENCE       0.7
DETECTION_NMS_THRESHOLD        0.3
FPN_CLASSIF_FC_LAYERS_SIZE     1024
GPU_COUNT                      1
GRADIENT_CLIP_NORM             5.0
IMAGES_PER_GPU                 1
IMAGE_CHANNEL_COUNT            3
IMAGE_MAX_DIM                  448
IMAGE_META_SIZE                14
IMAGE_MIN_DIM                  384
IMAGE_MIN_SCALE                0
IMAGE_RESIZE_MODE              square
IMAGE_SHAPE                    [448 448   3]
LEARNING_MOMENTUM              0.9
LEARNING_RATE                  3e-05
```

Apply Mask-RCNN packages



```
model2 = MaskRCNN(mode='training', model_dir='./', config=config)
```

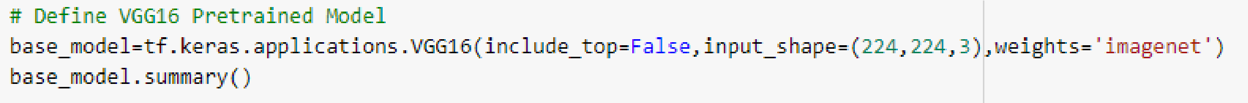
Load COCO weights



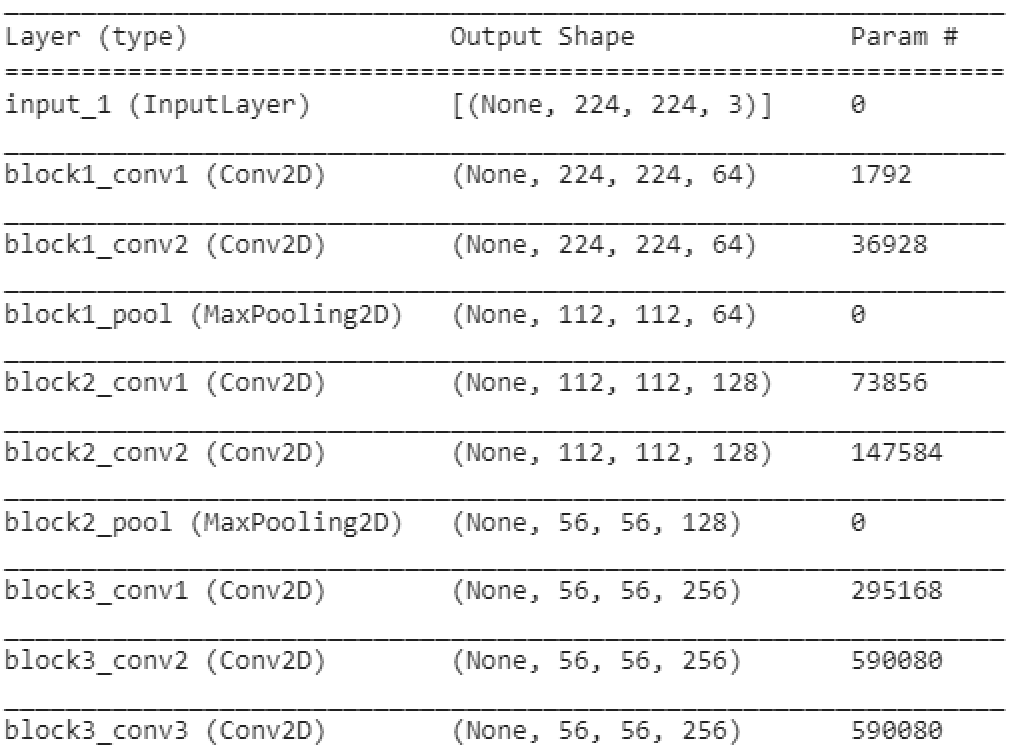
```
[ ] model2.load_weights('../input/mask-rcnn-coco-weights/mask_rcnn_coco.h5', by_name=True, exclude=["mrcnn_class_logits", "mrcnn_bbox_fc",  "mrcnn_bbox", "mrcnn_mask"])
```

6. Define Deep Learning Models

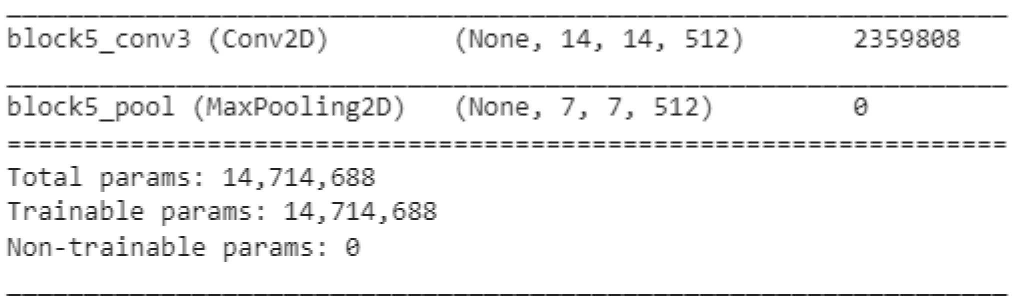
a) For Open CV approach: Define VGG + ImageNet weights



```
# Define VGG16 Pretrained Model
base_model=tf.keras.applications.VGG16(include_top=False,input_shape=(224,224,3),weights='imagenet')
base_model.summary()
```

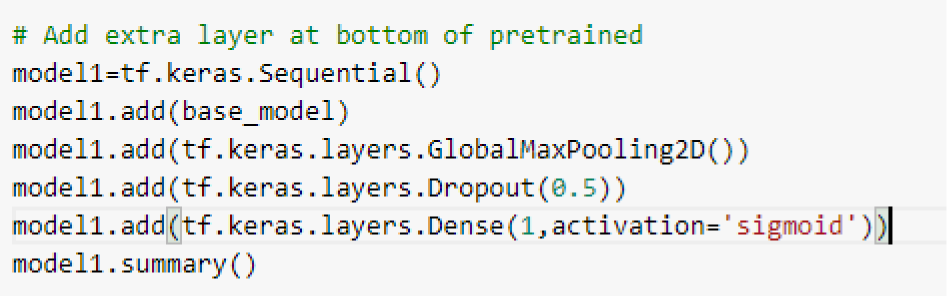


```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 224, 224, 3)]     0

_________________________________________________________________
block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792

_________________________________________________________________
block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928

_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0

_________________________________________________________________
block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856

_________________________________________________________________
block2_conv2 (Conv2D)        (None, 112, 112, 128)     147584

_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 56, 56, 128)       0

_________________________________________________________________
block3_conv1 (Conv2D)        (None, 56, 56, 256)       295168

_________________________________________________________________
block3_conv2 (Conv2D)        (None, 56, 56, 256)       590080

_________________________________________________________________
block3_conv3 (Conv2D)        (None, 56, 56, 256)       590080
```

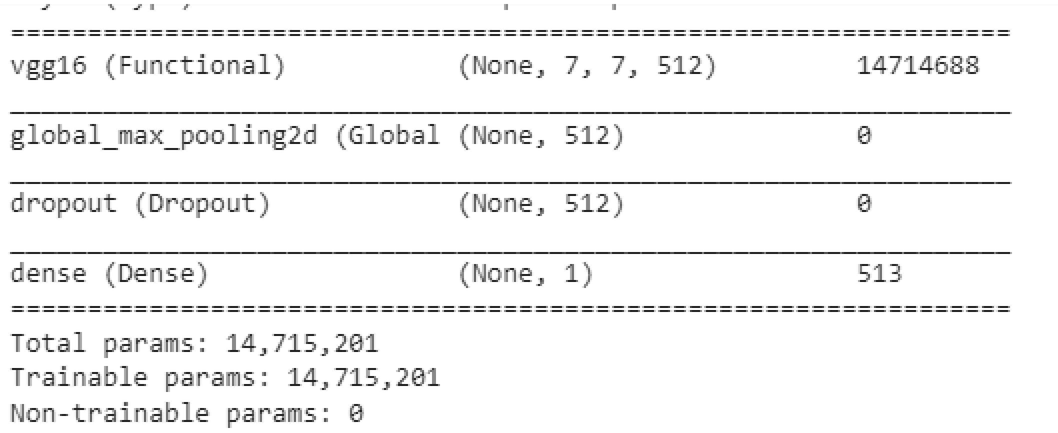


```
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808

_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0
=================================================================
Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0

_________________________________________________________________
```

Define Extra layer for pretrained model

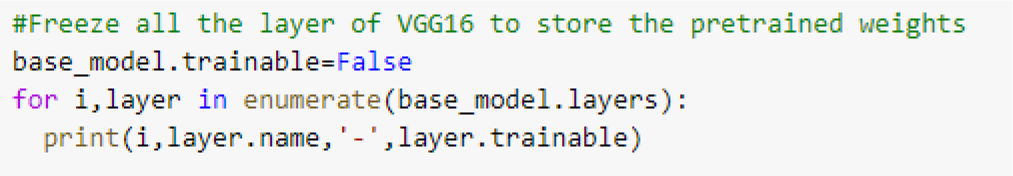


```
# Add extra layer at bottom of pretrained
model1=tf.keras.Sequential()
model1.add(base_model)
model1.add(tf.keras.layers.GlobalMaxPooling2D())
model1.add(tf.keras.layers.Dropout(0.5))
model1.add(tf.keras.layers.Dense(1,activation='sigmoid'))
model1.summary()
```

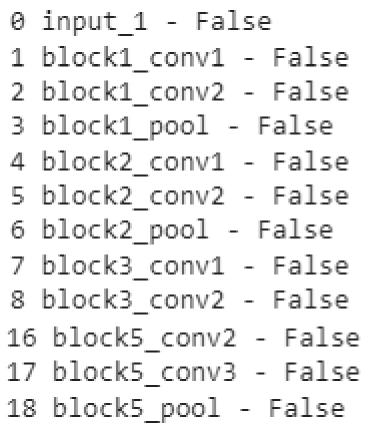


```
=================================================================
vgg16 (Functional)           (None, 7, 7, 512)         14714688
_________________________________________________________________
global_max_pooling2d (Global (None, 512)               0
_________________________________________________________________
dropout (Dropout)            (None, 512)               0
_________________________________________________________________
dense (Dense)                (None, 1)                 513
=================================================================
Total params: 14,715,201
Trainable params: 14,715,201
Non-trainable params: 0
```

Freeze the base layer

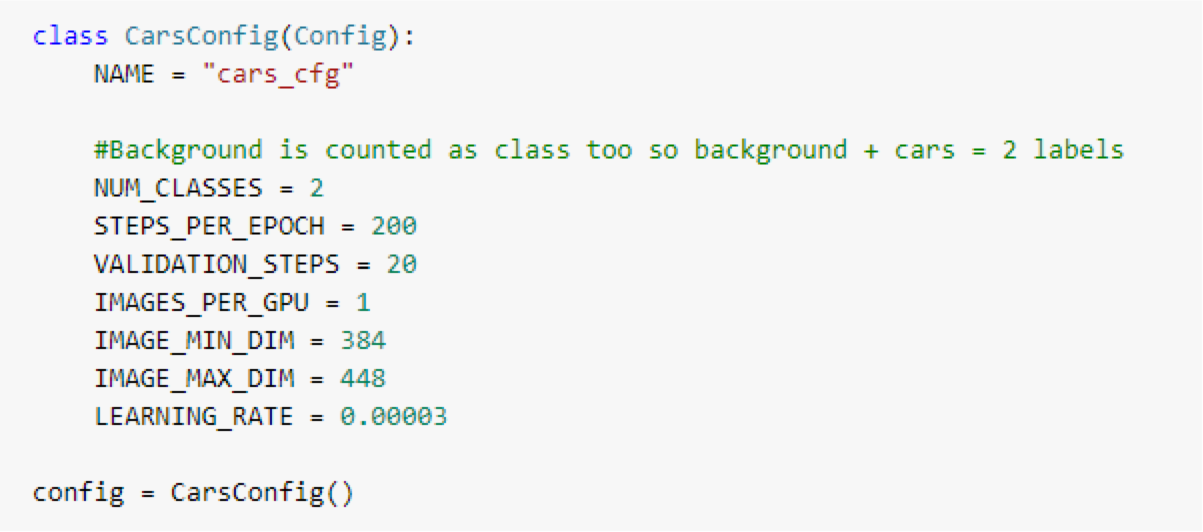


```
#Freeze all the layer of VGG16 to store the pretrained weights
base_model.trainable=False
for i,layer in enumerate(base_model.layers):
  print(i,layer.name,'-',layer.trainable)
```

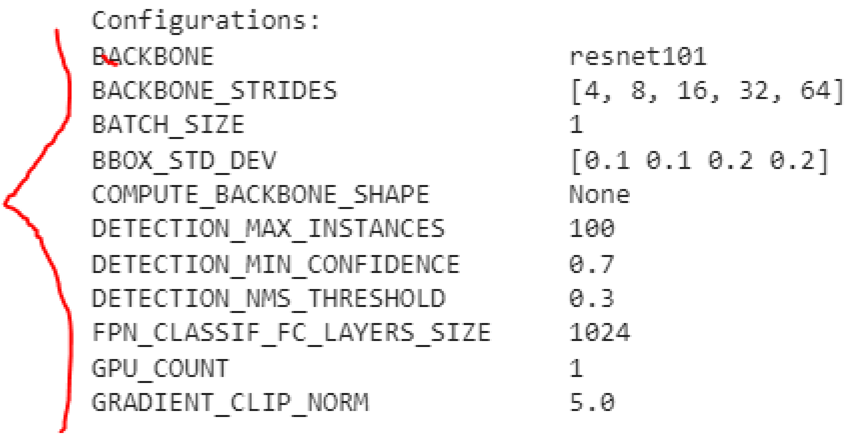


```
0 input_1 - False
1 block1_conv1 - False
2 block1_conv2 - False
3 block1_pool - False
4 block2_conv1 - False
5 block2_conv2 - False
6 block2_pool - False
7 block3_conv1 - False
8 block3_conv2 - False
16 block5_conv2 - False
17 block5_conv3 - False
18 block5_pool - False
```

b) Define ResNet 101+ COCO weights for Mask-RCNN



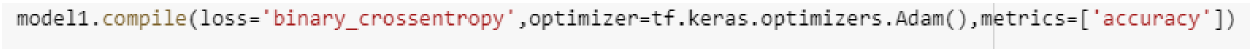
```
class CarsConfig(Config):
    NAME = "cars_cfg"

    #Background is counted as class too so background + cars = 2 labels
    NUM_CLASSES = 2
    STEPS_PER_EPOCH = 200
    VALIDATION_STEPS = 20
    IMAGES_PER_GPU = 1
    IMAGE_MIN_DIM = 384
    IMAGE_MAX_DIM = 448
    LEARNING_RATE = 0.00003

config = CarsConfig()
```



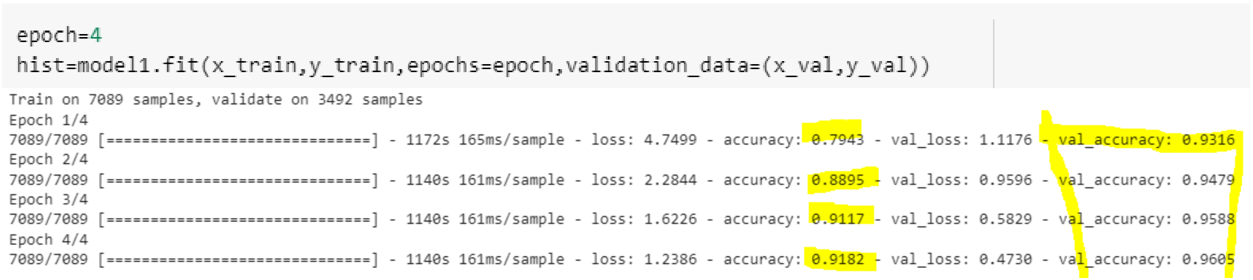
```
Configurations:
BACKBONE                       resnet101
BACKBONE_STRIDES               [4, 8, 16, 32, 64]
BATCH_SIZE                     1
BBOX_STD_DEV                   [0.1 0.1 0.2 0.2]
COMPUTE_BACKBONE_SHAPE         None
DETECTION_MAX_INSTANCES        100
DETECTION_MIN_CONFIDENCE       0.7
DETECTION_NMS_THRESHOLD        0.3
FPN_CLASSIF_FC_LAYERS_SIZE     1024
GPU_COUNT                      1
GRADIENT_CLIP_NORM             5.0
```

7. Train the model

a) VGG16 + ImageNet weights



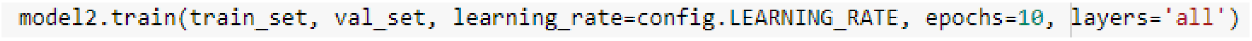
```
model1.compile(loss='binary_crossentropy',optimizer=tf.keras.optimizers.Adam(),metrics=['accuracy'])
```



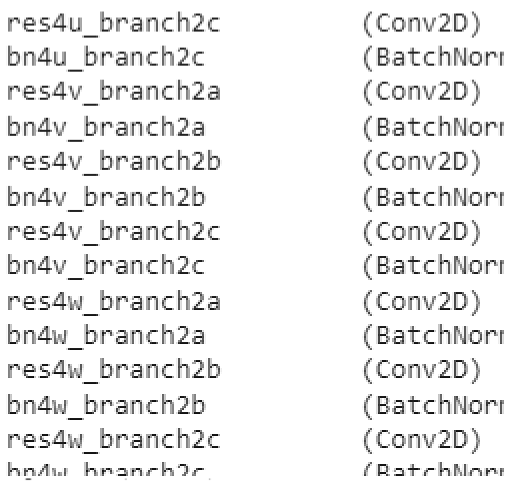
```
epoch=4
hist=model1.fit(x_train,y_train,epochs=epoch,validation_data=(x_val,y_val))
```

```
Train on 7089 samples, validate on 3492 samples
Epoch 1/4
7089/7089 [==============================] - 1172s 165ms/sample - loss: 4.7499 - accuracy: 0.7943 - val_loss: 1.1176 - val_accuracy: 0.9316
Epoch 2/4
7089/7089 [==============================] - 1140s 161ms/sample - loss: 2.2844 - accuracy: 0.8895 - val_loss: 0.9596 - val_accuracy: 0.9479
Epoch 3/4
7089/7089 [==============================] - 1140s 161ms/sample - loss: 1.6226 - accuracy: 0.9117 - val_loss: 0.5829 - val_accuracy: 0.9588
Epoch 4/4
7089/7089 [==============================] - 1140s 161ms/sample - loss: 1.2386 - accuracy: 0.9182 - val_loss: 0.4730 - val_accuracy: 0.9605
```

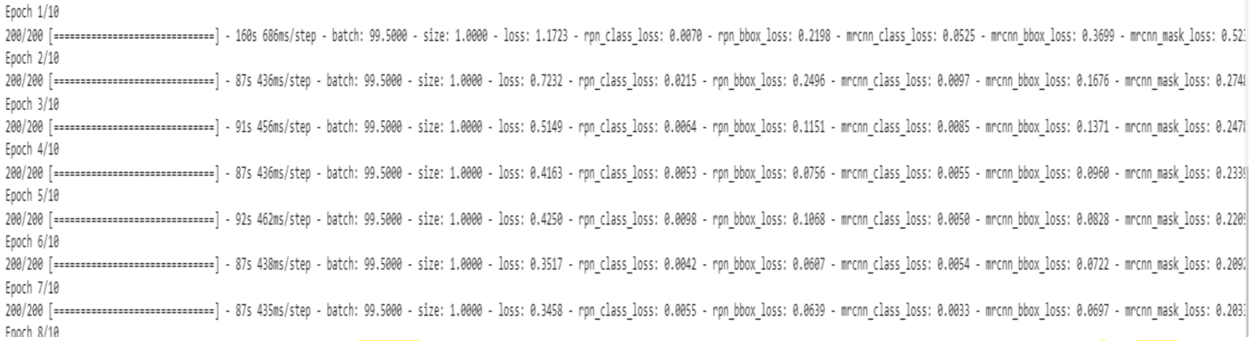
b) ResNet 101 + COCO weights



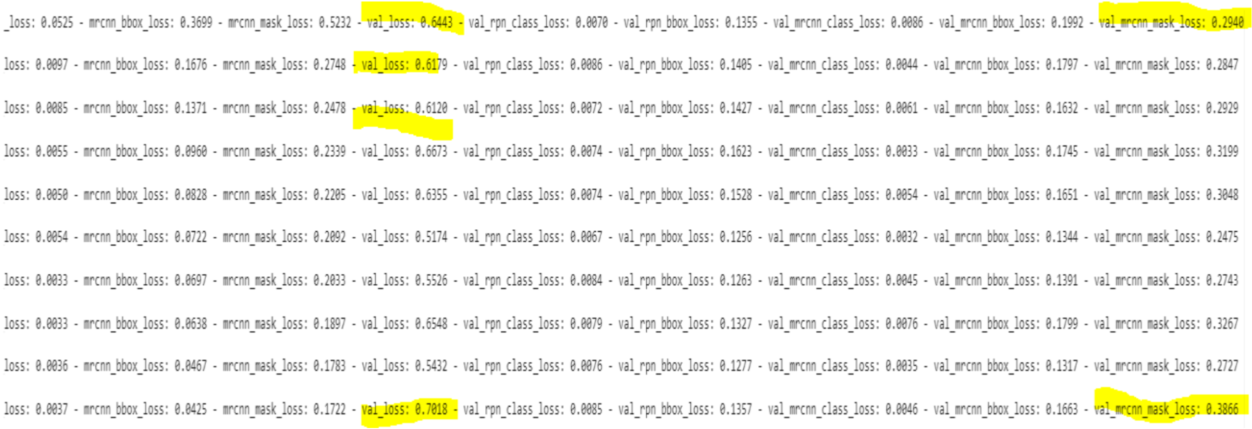
```
model2.train(train_set, val_set, learning_rate=config.LEARNING_RATE, epochs=10, layers='all')
```



```
res4u_branch2c          (Conv2D)
bn4u_branch2c           (BatchNorm
res4v_branch2a          (Conv2D)
bn4v_branch2a           (BatchNorm
res4v_branch2b          (Conv2D)
bn4v_branch2b           (BatchNorm
res4v_branch2c          (Conv2D)
bn4v_branch2c           (BatchNorm
res4w_branch2a          (Conv2D)
bn4w_branch2a           (BatchNorm
res4w_branch2b          (Conv2D)
bn4w_branch2b           (BatchNorm
res4w_branch2c          (Conv2D)
bn4w_branch2c           (BatchNorm
```



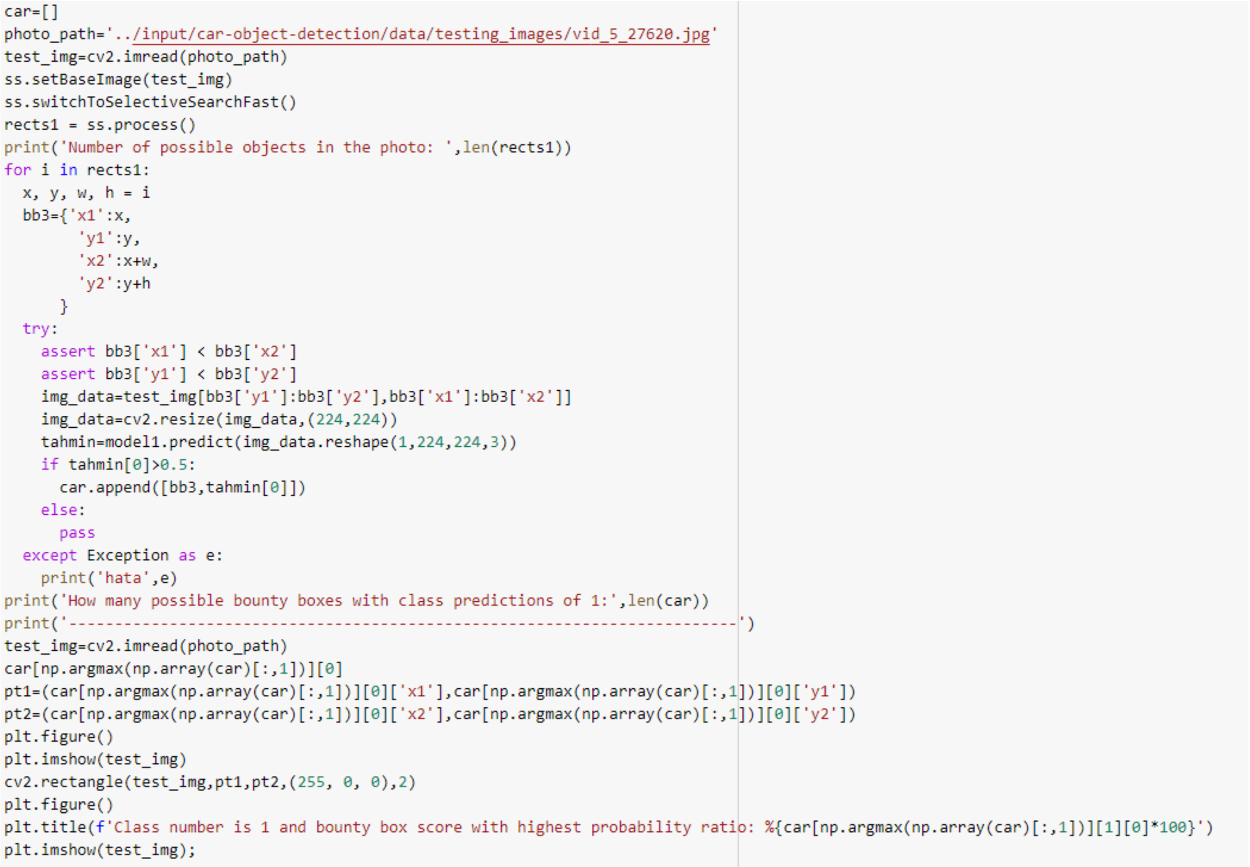
```
Epoch 1/10
200/200 [==============================] - 160s 686ms/step - batch: 99.5000 - size: 1.0000 - loss: 1.1723 - rpn_class_loss: 0.0070 - rpn_bbox_loss: 0.2198 - mrcnn_class_loss: 0.0525 - mrcnn_bbox_loss: 0.3699 - mrcnn_mask_loss: 0.523
Epoch 2/10
200/200 [==============================] - 87s 436ms/step - batch: 99.5000 - size: 1.0000 - loss: 0.7232 - rpn_class_loss: 0.0215 - rpn_bbox_loss: 0.2496 - mrcnn_class_loss: 0.0097 - mrcnn_bbox_loss: 0.1676 - mrcnn_mask_loss: 0.2748
Epoch 3/10
200/200 [==============================] - 91s 456ms/step - batch: 99.5000 - size: 1.0000 - loss: 0.5149 - rpn_class_loss: 0.0064 - rpn_bbox_loss: 0.1151 - mrcnn_class_loss: 0.0085 - mrcnn_bbox_loss: 0.1371 - mrcnn_mask_loss: 0.2478
Epoch 4/10
200/200 [==============================] - 87s 436ms/step - batch: 99.5000 - size: 1.0000 - loss: 0.4163 - rpn_class_loss: 0.0053 - rpn_bbox_loss: 0.0756 - mrcnn_class_loss: 0.0055 - mrcnn_bbox_loss: 0.0960 - mrcnn_mask_loss: 0.2339
Epoch 5/10
200/200 [==============================] - 92s 462ms/step - batch: 99.5000 - size: 1.0000 - loss: 0.4250 - rpn_class_loss: 0.0098 - rpn_bbox_loss: 0.1068 - mrcnn_class_loss: 0.0050 - mrcnn_bbox_loss: 0.0828 - mrcnn_mask_loss: 0.2205
Epoch 6/10
200/200 [==============================] - 87s 438ms/step - batch: 99.5000 - size: 1.0000 - loss: 0.3517 - rpn_class_loss: 0.0042 - rpn_bbox_loss: 0.0607 - mrcnn_class_loss: 0.0054 - mrcnn_bbox_loss: 0.0722 - mrcnn_mask_loss: 0.2092
Epoch 7/10
200/200 [==============================] - 87s 435ms/step - batch: 99.5000 - size: 1.0000 - loss: 0.3458 - rpn_class_loss: 0.0055 - rpn_bbox_loss: 0.0639 - mrcnn_class_loss: 0.0033 - mrcnn_bbox_loss: 0.0697 - mrcnn_mask_loss: 0.2033
Epoch 8/10
```



```
_loss: 0.0525 - mrcnn_bbox_loss: 0.3699 - mrcnn_mask_loss: 0.5232 - val_loss: 0.6443 - val_rpn_class_loss: 0.0070 - val_rpn_bbox_loss: 0.1355 - val_mrcnn_class_loss: 0.0086 - val_mrcnn_bbox_loss: 0.1992 - val_mrcnn_mask_loss: 0.2940

loss: 0.0097 - mrcnn_bbox_loss: 0.1676 - mrcnn_mask_loss: 0.2748 - val_loss: 0.6179 - val_rpn_class_loss: 0.0086 - val_rpn_bbox_loss: 0.1405 - val_mrcnn_class_loss: 0.0044 - val_mrcnn_bbox_loss: 0.1797 - val_mrcnn_mask_loss: 0.2847

loss: 0.0085 - mrcnn_bbox_loss: 0.1371 - mrcnn_mask_loss: 0.2478 - val_loss: 0.6120 - val_rpn_class_loss: 0.0072 - val_rpn_bbox_loss: 0.1427 - val_mrcnn_class_loss: 0.0061 - val_mrcnn_bbox_loss: 0.1632 - val_mrcnn_mask_loss: 0.2929

loss: 0.0055 - mrcnn_bbox_loss: 0.0960 - mrcnn_mask_loss: 0.2339 - val_loss: 0.6673 - val_rpn_class_loss: 0.0074 - val_rpn_bbox_loss: 0.1623 - val_mrcnn_class_loss: 0.0033 - val_mrcnn_bbox_loss: 0.1745 - val_mrcnn_mask_loss: 0.3199

loss: 0.0050 - mrcnn_bbox_loss: 0.0828 - mrcnn_mask_loss: 0.2205 - val_loss: 0.6355 - val_rpn_class_loss: 0.0074 - val_rpn_bbox_loss: 0.1528 - val_mrcnn_class_loss: 0.0054 - val_mrcnn_bbox_loss: 0.1651 - val_mrcnn_mask_loss: 0.3048

loss: 0.0054 - mrcnn_bbox_loss: 0.0722 - mrcnn_mask_loss: 0.2092 - val_loss: 0.5174 - val_rpn_class_loss: 0.0067 - val_rpn_bbox_loss: 0.1256 - val_mrcnn_class_loss: 0.0032 - val_mrcnn_bbox_loss: 0.1344 - val_mrcnn_mask_loss: 0.2475

loss: 0.0033 - mrcnn_bbox_loss: 0.0697 - mrcnn_mask_loss: 0.2033 - val_loss: 0.5526 - val_rpn_class_loss: 0.0084 - val_rpn_bbox_loss: 0.1263 - val_mrcnn_class_loss: 0.0045 - val_mrcnn_bbox_loss: 0.1391 - val_mrcnn_mask_loss: 0.2743

loss: 0.0033 - mrcnn_bbox_loss: 0.0638 - mrcnn_mask_loss: 0.1897 - val_loss: 0.6548 - val_rpn_class_loss: 0.0079 - val_rpn_bbox_loss: 0.1327 - val_mrcnn_class_loss: 0.0076 - val_mrcnn_bbox_loss: 0.1799 - val_mrcnn_mask_loss: 0.3267

loss: 0.0036 - mrcnn_bbox_loss: 0.0467 - mrcnn_mask_loss: 0.1783 - val_loss: 0.5432 - val_rpn_class_loss: 0.0076 - val_rpn_bbox_loss: 0.1277 - val_mrcnn_class_loss: 0.0035 - val_mrcnn_bbox_loss: 0.1317 - val_mrcnn_mask_loss: 0.2727

loss: 0.0037 - mrcnn_bbox_loss: 0.0425 - mrcnn_mask_loss: 0.1722 - val_loss: 0.7018 - val_rpn_class_loss: 0.0085 - val_rpn_bbox_loss: 0.1357 - val_mrcnn_class_loss: 0.0046 - val_mrcnn_bbox_loss: 0.1663 - val_mrcnn_mask_loss: 0.3866
```

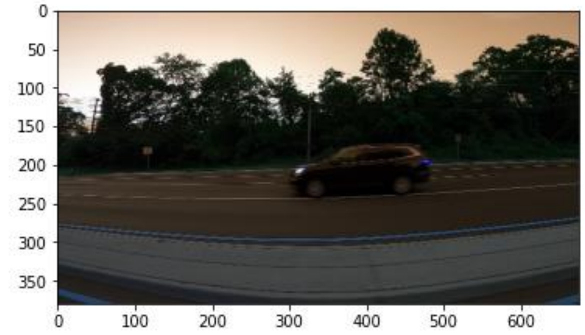
8) Evaluation

a) VGG + ImageNet: Model accuracy + IOU window

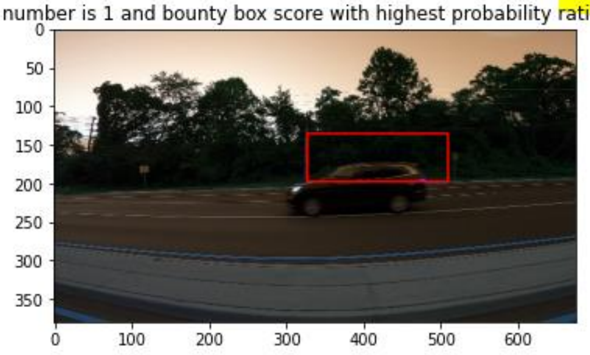


```
car=[]
photo_path='../input/car-object-detection/data/testing_images/vid_5_27620.jpg'
test_img=cv2.imread(photo_path)
ss.setBaseImage(test_img)
ss.switchToSelectiveSearchFast()
rects1 = ss.process()
print('Number of possible objects in the photo: ',len(rects1))
for i in rects1:
  x, y, w, h = i
  bb3={'x1':x,
        'y1':y,
        'x2':x+w,
        'y2':y+h
      }
  try:
    assert bb3['x1'] < bb3['x2']
    assert bb3['y1'] < bb3['y2']
    img_data=test_img[bb3['y1']:bb3['y2'],bb3['x1']:bb3['x2']]
    img_data=cv2.resize(img_data,(224,224))
    tahmin=model1.predict(img_data.reshape(1,224,224,3))
    if tahmin[0]>0.5:
      car.append([bb3,tahmin[0]])
    else:
      pass
  except Exception as e:
    print('hata',e)
print('How many possible bounty boxes with class predictions of 1:',len(car))
print('----------------------------------------------------------------------')
test_img=cv2.imread(photo_path)
car[np.argmax(np.array(car)[:,1])][0]
pt1=(car[np.argmax(np.array(car)[:,1])][0]['x1'],car[np.argmax(np.array(car)[:,1])][0]['y1'])
pt2=(car[np.argmax(np.array(car)[:,1])][0]['x2'],car[np.argmax(np.array(car)[:,1])][0]['y2'])
plt.figure()
plt.imshow(test_img)
cv2.rectangle(test_img,pt1,pt2,(255, 0, 0),2)
plt.figure()
plt.title(f'Class number is 1 and bounty box score with highest probability ratio: %{car[np.argmax(np.array(car)[:,1])][1][0]*100}')
plt.imshow(test_img);
```

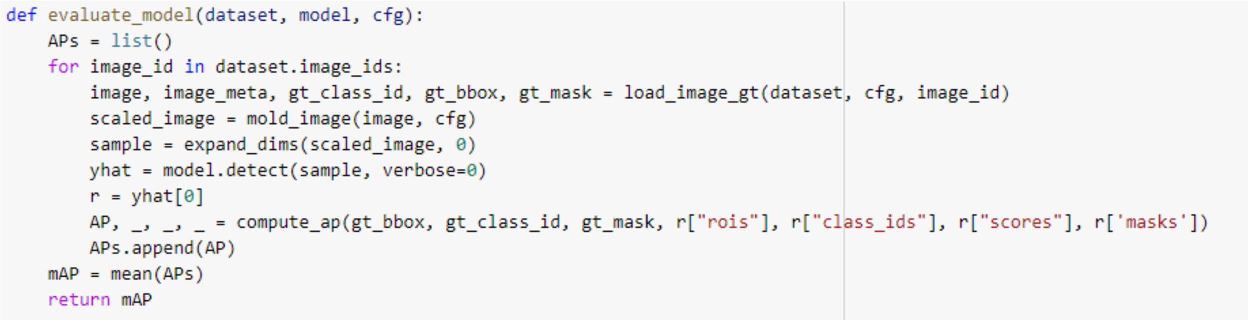
```
Number of possible objects in the photo:  2037
How many possible bounty boxes with class predictions of 1: 79
----------------------------------------------------------------------
```



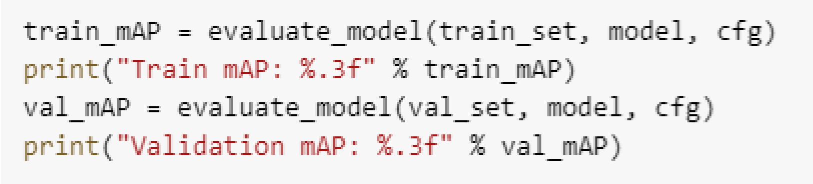
Class number is 1 and bounty box score with highest probability ratio: %100.0



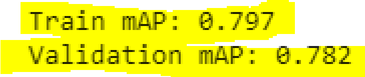
b) Mask-RCNN ResNet101 + COCO weights: MAP evaluation



```python
def evaluate_model(dataset, model, cfg):
    APs = list()
    for image_id in dataset.image_ids:
        image, image_meta, gt_class_id, gt_bbox, gt_mask = load_image_gt(dataset, cfg, image_id)
        scaled_image = mold_image(image, cfg)
        sample = expand_dims(scaled_image, 0)
        yhat = model.detect(sample, verbose=0)
        r = yhat[0]
        AP, _, _, _ = compute_ap(gt_bbox, gt_class_id, gt_mask, r["rois"], r["class_ids"], r["scores"], r['masks'])
        APs.append(AP)
    mAP = mean(APs)
    return mAP
```



```python
train_mAP = evaluate_model(train_set, model, cfg)
print("Train mAP: %.3f" % train_mAP)
val_mAP = evaluate_model(val_set, model, cfg)
print("Validation mAP: %.3f" % val_mAP)
```



```
Train mAP: 0.797
Validation mAP: 0.782
```

Compare Some Samples