In [1]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?clie
nt_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.
com&redirect_uri=urn%3Aietf%3Awg%3Aoauth%3A2.0%3Aoob&scope=email%20https%3
A%2F%2Fwww.googleapis.com%2Fauth%2Fdocs.test%20https%3A%2F%2Fwww.googleapi
s.com%2Fauth%2Fdrive%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdrive.phot
os.readonly%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fpeopleapi.readonly&
response_type=code

Enter your authorization code:
··········
Mounted at /content/drive
```

In [2]:
```
!ls
```

```
drive  sample_data
```

Import libs

In [3]:
```
from __future__ import division
from __future__ import print_function
from __future__ import absolute_import
import random
import pprint
import sys
import time
import numpy as np
from optparse import OptionParser
import pickle
import math
import cv2
import copy
from matplotlib import pyplot as plt
import tensorflow as tf
import pandas as pd
import os

from sklearn.metrics import average_precision_score

from keras import backend as K
from keras.optimizers import Adam, SGD, RMSprop
from keras.layers import Flatten, Dense, Input, Conv2D, MaxPooling2D, Dropo
ut
from keras.layers import GlobalAveragePooling2D, GlobalMaxPooling2D, TimeDi
stributed
from keras.engine.topology import get_source_inputs
from keras.utils import layer_utils
from keras.utils.data_utils import get_file
from keras.objectives import categorical_crossentropy

from keras.models import Model
from keras.utils import generic_utils
from keras.engine import Layer, InputSpec
from keras import initializers, regularizers
```

```
Using TensorFlow backend.
```

Config setting

In [0]:

```
class Config:

        def __init__(self):

                # Print the process or not
                self.verbose = True

                # Name of base network
                self.network = 'vgg'

                # Setting for data augmentation
                self.use_horizontal_flips = False
                self.use_vertical_flips = False
                self.rot_90 = False

                # Anchor box scales
    # Note that if im_size is smaller, anchor_box_scales should be scaled
    # যেসকল anchor_box scales ব্যবহার করা হচ্ছে। পেপার এ [128, 256, 512] ব্যব
হার করা হয়েছে।
                self.anchor_box_scales = [64, 128, 256]

                # Anchor box ratio পেপারে 1:1, 1:2, 2:1 ব্যবহার করা হয়েছে।
                self.anchor_box_ratios = [[1, 1], [1./math.sqrt(2), 2./math
.sqrt(2)], [2./math.sqrt(2), 1./math.sqrt(2)]]

                # Size to resize the smallest side of the image
                # Original setting in paper is 600. Set to 300 in here to s
ave training time
                self.im_size = 300

                # image channel-wise mean to subtract
                self.img_channel_mean = [103.939, 116.779, 123.68]
                self.img_scaling_factor = 1.0

                # number of ROIs at once
                self.num_rois = 4

                # stride at the RPN (this depends on the network configurat
ion)
                self.rpn_stride = 16

                self.balanced_classes = False

                # scaling the stdev
                self.std_scaling = 4.0
                self.classifier_regr_std = [8.0, 8.0, 4.0, 4.0]

                # overlaps for RPN
                self.rpn_min_overlap = 0.3
                self.rpn_max_overlap = 0.7

                # overlaps for classifier ROIs
                self.classifier_min_overlap = 0.1
                self.classifier_max_overlap = 0.5

                # placeholder for the class mapping, automatically generate
d by the parser
```

```
        self.class_mapping = None

        self.model_path = None
```

Parser the data from annotation file

In [0]:

```
def get_data(input_path):
        """Parser the data from annotation file

        Args:
                input_path: annotation file path

        Returns:
                all_data: list(filepath, width, height, list(bboxes))
                classes_count: dict{key:class_name, value:count_num}
                        e.g. {'Car': 2383, 'Mobile phone': 1108, 'Person':
 3745}
                class_mapping: dict{key:class_name, value: idx}
                        e.g. {'Car': 0, 'Mobile phone': 1, 'Person': 2}
        """
        found_bg = False
        all_imgs = {}

        classes_count = {}

        class_mapping = {}

        visualise = True

        i = 1

        with open(input_path,'r') as f:

                print('Parsing annotation files')

                for line in f:

                        # Print process
                        sys.stdout.write('\r'+'idx=' + str(i))
                        i += 1

                        line_split = line.strip().split(',')

                        # Make sure the info saved in annotation file match
ing the format (path_filename, x1, y1, x2, y2, class_name)
                        # Note:
                        #       One path_filename might has several classes
 (class_name)
                        #       x1, y1, x2, y2 are the pixel value of the o
rigial image, not the ratio value
                        #       (x1, y1) top left coordinates; (x2, y2) bot
tom right coordinates
                        #   x1,y1-------------------
                        #       |
        |
                        #       |
        |
                        #       |
        |
                        #       |
        |
                        #       ---------------------x2,y2
```

```
                        (filename,x1,y1,x2,y2,class_name) = line_split

                        if class_name not in classes_count:
                                classes_count[class_name] = 1
                        else:
                                classes_count[class_name] += 1

                        if class_name not in class_mapping:
                                if class_name == 'bg' and found_bg == False
:
                                        print('Found class name with specia
l name bg. Will be treated as a background region (this is usually for hard
 negative mining).')
                                        found_bg = True
                                class_mapping[class_name] = len(class_mappi
ng)

                        if filename not in all_imgs:
                                all_imgs[filename] = {}

                                img = cv2.imread(filename)
                                (rows,cols) = img.shape[:2]
                                all_imgs[filename]['filepath'] = filename
                                all_imgs[filename]['width'] = cols
                                all_imgs[filename]['height'] = rows
                                all_imgs[filename]['bboxes'] = []
                                # if np.random.randint(0,6) > 0:
                                #       all_imgs[filename]['imageset'] = 't
rainval'
                                # else:
                                #       all_imgs[filename]['imageset'] = 't
est'

                        all_imgs[filename]['bboxes'].append({'class': class
_name, 'x1': int(x1), 'x2': int(x2), 'y1': int(y1), 'y2': int(y2)})


                all_data = []
                for key in all_imgs:
                        all_data.append(all_imgs[key])

                # make sure the bg class is last in the list
                if found_bg:
                        if class_mapping['bg'] != len(class_mapping) - 1:
                                key_to_switch = [key for key in class_mappi
ng.keys() if class_mapping[key] == len(class_mapping)-1][0]
                                val_to_switch = class_mapping['bg']
                                class_mapping['bg'] = len(class_mapping) -
1
                                class_mapping[key_to_switch] = val_to_switc
h

                return all_data, classes_count, class_mapping
```

Define ROI Pooling Convolutional Layer

In [0]:

```
class RoiPoolingConv(Layer):
    '''
        ROI Pooling Curve সাধারণত কিছু সংখ্যক feature map
        নিয়ে তাকে একটি fixed (N x N) output size এ পরিনিত করে। যাতে করে FC
 layer এ feature map ক্রপ হয়ে না যায়।
    # Arguments
        pool_size: int
            Pooling region এর size. pool_size = 7 মানে এটি 7x7 region এর এ
কটি output shape দিবে।
        num_rois: যতসংখ্যক region of interest ব্যবহার করা হবে।
    # Input shape
        list of two 4D tensors [X_img,X_roi] with shape:
        X_img:
        `(1, rows, cols, channels)`
        X_roi:
        `(1,num_rois,4)` list of rois, with ordering (x,y,w,h)
        x,y = co-ordinate এবং w = width, h = height
    # Output shape
        3D tensor with shape:
        `(1, num_rois, channels, pool_size, pool_size)`
    '''
    def __init__(self, pool_size, num_rois, **kwargs):

        self.dim_ordering = K.image_dim_ordering()
        self.pool_size = pool_size
        self.num_rois = num_rois

        super(RoiPoolingConv, self).__init__(**kwargs)

    def build(self, input_shape):
        self.nb_channels = input_shape[0][3]

    def compute_output_shape(self, input_shape):
        return None, self.num_rois, self.pool_size, self.pool_size, self.nb
_channels

    def call(self, x, mask=None):

        assert(len(x) == 2)

        #  x[0] একটি image যার shape (rows, cols, channels)
        img = x[0]

        # x[1] হচ্ছে ROI যার shape (num_rois, x , y , w ,h)
        rois = x[1]

        input_shape = K.shape(img)

        outputs = []

        for roi_idx in range(self.num_rois):
            # x, y, w, h এর ভ্যালুগুলো roi_idx অনুসারে rois থেকে নেয়া হচ্ছে।
            x = rois[0, roi_idx, 0]
            y = rois[0, roi_idx, 1]
            w = rois[0, roi_idx, 2]
            h = rois[0, roi_idx, 3]
```

```
            x = K.cast(x, 'int32')
            y = K.cast(y, 'int32')
            w = K.cast(w, 'int32')
            h = K.cast(h, 'int32')

            # Resized করা হচ্ছে image এর ROI থেকে যার output shape হবে (7x7)
            rs = tf.image.resize_images(img[:, y:y+h, x:x+w, :], (self.pool
_size, self.pool_size))
            outputs.append(rs)


        final_output = K.concatenate(outputs, axis=0)

        # এবার চ্যানেল এবং total ROI এর সাথে stack করার পরে reshape করা হ
চ্ছে।
        # eg: (1, 4, 7, 7, 3)
        final_output = K.reshape(final_output, (1, self.num_rois, self.pool
_size, self.pool_size, self.nb_channels))

        # transpose করা হচ্ছে যাতে বাকিগুলো image পাশাপাশি stack করানো যায়।
        final_output = K.permute_dimensions(final_output, (0, 1, 2, 3, 4))

        return final_output


    def get_config(self):
        config = {'pool_size': self.pool_size,
                  'num_rois': self.num_rois}
        base_config = super(RoiPoolingConv, self).get_config()
        return dict(list(base_config.items()) + list(config.items()))
```

Vgg-16 model

In [0]:

```
def get_img_output_length(width, height):
    def get_output_length(input_length):
        return input_length//16

    return get_output_length(width), get_output_length(height)

def nn_base(input_tensor=None, trainable=False):


    input_shape = (None, None, 3)

    if input_tensor is None:
        img_input = Input(shape=input_shape)
    else:
        if not K.is_keras_tensor(input_tensor):
            img_input = Input(tensor=input_tensor, shape=input_shape)
        else:
            img_input = input_tensor

    bn_axis = 3

    # Block 1
    x = Conv2D(64, (3, 3), activation='relu', padding='same', name='block1_
conv1')(img_input)
    x = Conv2D(64, (3, 3), activation='relu', padding='same', name='block1_
conv2')(x)
    x = MaxPooling2D((2, 2), strides=(2, 2), name='block1_pool')(x)

    # Block 2
    x = Conv2D(128, (3, 3), activation='relu', padding='same', name='block2
_conv1')(x)
    x = Conv2D(128, (3, 3), activation='relu', padding='same', name='block2
_conv2')(x)
    x = MaxPooling2D((2, 2), strides=(2, 2), name='block2_pool')(x)

    # Block 3
    x = Conv2D(256, (3, 3), activation='relu', padding='same', name='block3
_conv1')(x)
    x = Conv2D(256, (3, 3), activation='relu', padding='same', name='block3
_conv2')(x)
    x = Conv2D(256, (3, 3), activation='relu', padding='same', name='block3
_conv3')(x)
    x = MaxPooling2D((2, 2), strides=(2, 2), name='block3_pool')(x)

    # Block 4
    x = Conv2D(512, (3, 3), activation='relu', padding='same', name='block4
_conv1')(x)
    x = Conv2D(512, (3, 3), activation='relu', padding='same', name='block4
_conv2')(x)
    x = Conv2D(512, (3, 3), activation='relu', padding='same', name='block4
_conv3')(x)
    x = MaxPooling2D((2, 2), strides=(2, 2), name='block4_pool')(x)

    # Block 5
    x = Conv2D(512, (3, 3), activation='relu', padding='same', name='block5
_conv1')(x)
    x = Conv2D(512, (3, 3), activation='relu', padding='same', name='block5
```

```
_conv2')(x)
    x = Conv2D(512, (3, 3), activation='relu', padding='same', name='block5
_conv3')(x)
    # x = MaxPooling2D((2, 2), strides=(2, 2), name='block5_pool')(x)

    return x
```

RPN layer

In [0]:

```
def rpn_layer(base_layers, num_anchors):
    """Create a rpn layer
        Step1:  VGG layer থেকে পাওয়া feature map কে একটি 3x3 512 channels
 এর conv layer এর মধ্যে
        পাঠানো হচ্চে যেখানে keras এর args(padding= same) যাতে করে output sha
pe, feature map এর
        shape এ থাকে।
        Step2: এরপর তাকে দুটি (1,1) conv layer এ পাঠানো হয়েছে fc layer এর
 পরিবর্তে।
                classification layer: num_anchors (9টি করা হয়েছে পেপার এ) ।
 channels ২টি করা হয়াছে (0, 1) with sigmoid activation func.
                regression layer: মোট num_anchors*4 (36 পেপার এ) channels
 bboxes এর  regression বের করার জন্য
    Args:
        base_layers: vgg in here
        num_anchors: 9 in here

    Returns:
        [x_class, x_regr, base_layers]
        x_class: classification for whether it's an object
        x_regr: bboxes regression
        base_layers: vgg in here
    """
    x = Conv2D(512, (3, 3), padding='same', activation='relu', kernel_initi
alizer='normal', name='rpn_conv1')(base_layers)

    x_class = Conv2D(num_anchors, (1, 1), activation='sigmoid', kernel_init
ializer='uniform', name='rpn_out_class')(x)
    x_regr = Conv2D(num_anchors * 4, (1, 1), activation='linear', kernel_in
itializer='zero', name='rpn_out_regress')(x)

    return [x_class, x_regr, base_layers]
```

Classifier layer

In [0]:
```
def classifier_layer(base_layers, input_rois, num_rois, nb_classes = 4):
    """Create a classifier layer

    Args:
        base_layers: vgg
        input_rois: `(1,num_rois,4)` list of rois, with ordering (x,y,w,h)
        x,y = co-ordinate এবং w = width, h = height
        num_rois: একবারে কতগুলো region of interest থেকে max pooling করবে।


    Returns:
        list(out_class, out_regr)
        out_class: classifier layer output
        out_regr: regression layer output
    """

    input_shape = (num_rois,7,7,512)

    pooling_regions = 7

    # out_roi_pool.shape = (1, num_rois, channels, pool_size, pool_size)
    # RoiPoolingConv func ব্যবহার করে feature map থেকে feature pool করে
 তাকে fixed (7x7) output shape এ  পরিবর্তিত করে।
    out_roi_pool = RoiPoolingConv(pooling_regions, num_rois)([base_layers,
input_rois])

    # এরপর ফাইনাল fc layer এবং কিছু dropout করে।
    out = TimeDistributed(Flatten(name='flatten'))(out_roi_pool)
    out = TimeDistributed(Dense(4096, activation='relu', name='fc1'))(out)
    out = TimeDistributed(Dropout(0.5))(out)
    out = TimeDistributed(Dense(4096, activation='relu', name='fc2'))(out)
    out = TimeDistributed(Dropout(0.5))(out)

    # output layer ২টি হবে।
    # out_class: softmax func ব্যবহার করা হয়ে object classify করে দেখতে যে o
bject টি আছে নাকি নেই।
    # out_regr: linear actviation func. ব্যবহার করে দেখা হয় যে bboxesটি আসল
 object কে ঠিক কতটুকু boundary করতে পেরেছে।
    #           কতটুকু shift করা লাগতে পারে।
    out_class = TimeDistributed(Dense(nb_classes, activation='softmax', ker
nel_initializer='zero'), name='dense_class_{}'.format(nb_classes))(out)
    # note: no regression target for bg class
    out_regr = TimeDistributed(Dense(4 * (nb_classes-1), activation='linea
r', kernel_initializer='zero'), name='dense_regress_{}'.format(nb_classes))
(out)

    return [out_class, out_regr]
```

Calculate IoU (Intersection of Union)

In [0]:

```
# এগুলো হচ্ছে union , intersection আর Intersection over Union func. anchor
 এর জন্য
def union(au, bu, area_intersection):
        area_a = (au[2] - au[0]) * (au[3] - au[1])
        area_b = (bu[2] - bu[0]) * (bu[3] - bu[1])
        area_union = area_a + area_b - area_intersection
        return area_union


def intersection(ai, bi):
        x = max(ai[0], bi[0])
        y = max(ai[1], bi[1])
        w = min(ai[2], bi[2]) - x
        h = min(ai[3], bi[3]) - y
        if w < 0 or h < 0:
                return 0
        return w*h


def iou(a, b):
        # a and b should be (x1,y1,x2,y2)

        if a[0] >= a[2] or a[1] >= a[3] or b[0] >= b[2] or b[1] >= b[3]:
                return 0.0

        area_i = intersection(a, b)
        area_u = union(a, b, area_i)

        return float(area_i) / float(area_u + 1e-6)
```

Calculate the rpn for all anchors of all images

In [0]:

```
def calc_rpn(C, img_data, width, height, resized_width, resized_height, img
_length_calc_function):
        """(Important part!) Calculate the rpn for all anchors
                যদি feature map এর shape 38x50=1900 হয় তাহলে 9টি anchors এর
 জন্য 1900x9= 17100টি anchors পাওয়া যাবে।

        Args:
                C: config
                img_data: augmented image data
                width: original image width (e.g. 600)
                height: original image height (e.g. 800)
                resized_width: resized image width according to C.im_size
 (e.g. 300)
                resized_height: resized image height according to C.im_size
 (e.g. 400)
                img_length_calc_function: function to calculate final laye
r's feature map (of base model) size according to input image size

        Returns:
                y_rpn_cls: list(num_bboxes, y_is_box_valid + y_rpn_overlap)
                        y_is_box_valid: 0 or 1 (0 means the box is invalid,
 1 means the box is valid)
                        y_rpn_overlap: 0 or 1 (0 means the box is not an ob
ject, 1 means the box is an object)
                y_rpn_regr: list(num_bboxes, 4*y_rpn_overlap + y_rpn_regr)
                        y_rpn_regr: x1,y1,x2,y2 bunding boxes coordinates
        """
    #anchor_sizes, anchor_ratios এবং num_anchors এর ভ্যালু নেওয়া হয়েছে।
        downscale = float(C.rpn_stride)
        anchor_sizes = C.anchor_box_scales   # 128, 256, 512
        anchor_ratios = C.anchor_box_ratios  # 1:1, 1:2*sqrt(2), 2*sqrt(2):
1
        num_anchors = len(anchor_sizes) * len(anchor_ratios) # 3x3=9

        # calculate the output map size based on the network architecture
        (output_width, output_height) = img_length_calc_function(resized_wi
dth, resized_height)

        n_anchratios = len(anchor_ratios)    # 3

        # empty variable তৈরি করা হয়েছে।
        y_rpn_overlap = np.zeros((output_height, output_width, num_anchors
))
        y_is_box_valid = np.zeros((output_height, output_width, num_anchors
))
        y_rpn_regr = np.zeros((output_height, output_width, num_anchors * 4
))

        num_bboxes = len(img_data['bboxes'])

        num_anchors_for_bbox = np.zeros(num_bboxes).astype(int)
        best_anchor_for_bbox = -1*np.ones((num_bboxes, 4)).astype(int)
        best_iou_for_bbox = np.zeros(num_bboxes).astype(np.float32)
        best_x_for_bbox = np.zeros((num_bboxes, 4)).astype(int)
        best_dx_for_bbox = np.zeros((num_bboxes, 4)).astype(np.float32)

        # Dataset থেকে GT box coordinates নিয়ে  তাকে বর্তমান image এর size এ
```

```
 resize করা হয়েছে।
        gta = np.zeros((num_bboxes, 4))
        for bbox_num, bbox in enumerate(img_data['bboxes']):
                # get the GT box coordinates, and resize to account for ima
ge resizing
                gta[bbox_num, 0] = bbox['x1'] * (resized_width / float(widt
h))
                gta[bbox_num, 1] = bbox['x2'] * (resized_width / float(widt
h))
                gta[bbox_num, 2] = bbox['y1'] * (resized_height / float(hei
ght))
                gta[bbox_num, 3] = bbox['y2'] * (resized_height / float(hei
ght))

        # rpn ground truth

        for anchor_size_idx in range(len(anchor_sizes)):
                for anchor_ratio_idx in range(n_anchratios):
           # anchor_x এবং anchor_y উভয়ই anchor এর size অনুযায়ী একটি variab
le নিচ্ছে scale এবং ratio এর গুনফল এর মাধ্যমে।
                        anchor_x = anchor_sizes[anchor_size_idx] * anchor_r
atios[anchor_ratio_idx][0]
                        anchor_y = anchor_sizes[anchor_size_idx] * anchor_r
atios[anchor_ratio_idx][1]

                        for ix in range(output_width):

                                # বর্তমান anchor box এর x-cooridnates value
                                x1_anc = downscale * (ix + 0.5) - anchor_x
/ 2
                                x2_anc = downscale * (ix + 0.5) + anchor_x
/ 2

                                # anchor box যদি image boundary এর মধ্যে পরে
 তাহলে ignore করবে।
                                if x1_anc < 0 or x2_anc > resized_width:
                                        continue

                                for jy in range(output_height):

                                        # বর্তমান anchor box এর y-cooridnate
s value
                                        y1_anc = downscale * (jy + 0.5) - a
nchor_y / 2
                                        y2_anc = downscale * (jy + 0.5) + a
nchor_y / 2

                                        # anchor box যদি image boundary এর
 মধ্যে পরে তাহলে ignore করবে।
                                        if y1_anc < 0 or y2_anc > resized_h
eight:
                                                continue

                                        # bbox_type pos নাকি neg হবে anchor
 তা নির্ধারণ করবে। শুরুতে initialization neg দেওয়া হয়েছে।
                                        bbox_type = 'neg'
```

```
                                        # best_iou_for_loc নির্ধারণ করা হয়েছে। best_iou_for_loc 0.0 সেরা হবে।
                    # কারন তখনই কেবল bbox আর anchor boundary একই হবে।
                                        best_iou_for_loc = 0.0

                                        for bbox_num in range(num_bboxes):

                                            #IOU func এর মাধ্যমে GT box আর anchor box এর IOU calculate করা হচ্ছে।
                                            curr_iou = iou([gta[bbox_num, 0], gta[bbox_num, 2], gta[bbox_num, 1], gta[bbox_num, 3]], [x1_anc, y1_anc, x2_anc, y2_anc])
                                            # regression target calculate করা হচ্ছে যে regression দরকার হবে কিনা।
                                            if curr_iou > best_iou_for_bbox[bbox_num] or curr_iou > C.rpn_max_overlap:
                                                cx = (gta[bbox_num, 0] + gta[bbox_num, 1]) / 2.0
                                                cy = (gta[bbox_num, 2] + gta[bbox_num, 3]) / 2.0
                                                cxa = (x1_anc + x2_anc)/2.0
                                                cya = (y1_anc + y2_anc)/2.0
                                # যদি থাকে তাহলে calculate করবো।
                                                # x,y are the center point of ground-truth bbox
                                                # xa,ya are the center point of anchor bbox (xa=downscale * (ix + 0.5); ya=downscale * (iy+0.5))
                                                # w,h are the width and height of ground-truth bbox
                                                # wa,ha are the width and height of anchor bboxe
                                                # tx = (x - xa) / wa
                                                # ty = (y - ya) / ha
                                                # tw = log(w / wa)
                                                # th = log(h / ha)
                                                tx = (cx - cxa) / (x2_anc - x1_anc)
                                                ty = (cy - cya) / (y2_anc - y1_anc)
                                                tw = np.log((gta[bbox_num, 1] - gta[bbox_num, 0]) / (x2_anc - x1_anc))
                                                th = np.log((gta[bbox_num, 3] - gta[bbox_num, 2]) / (y2_anc - y1_anc))

                                                if img_data['bboxes'][bbox_num]['class'] != 'bg':
                                # যদি bboxes background না হয় তবে সবগুলা GT boxes কে map করতে হবে anchor box এ।
                                # যাতে করা আমরা track করতে পারি কোন anchor box best
                                                    if curr_iou > best_iou_for_bbox[bbox_num]:
```

```
                                                                    best_anchor
_for_bbox[bbox_num] = [jy, ix, anchor_ratio_idx, anchor_size_idx]
                                                                    best_iou_fo
r_bbox[bbox_num] = curr_iou
                                                                    best_x_for_
bbox[bbox_num,:] = [x1_anc, x2_anc, y1_anc, y2_anc]
                                                                    best_dx_for
_bbox[bbox_num,:] = [tx, ty, tw, th]

                                                                # যদি IOU> 0.7 হয়
 অর্থাৎ bbox er 70% anchor box এ থাকে তাহলে bbox_type positive করে দিবো।
                                                                if curr_iou > C.rpn
_max_overlap:
                                                                    bbox_type =
 'pos'
                                                                    num_anchors
_for_bbox[bbox_num] += 1
                                                                    #যদি current
_iou_value  best_iou_value থেকে বেশি হয় তাহলে আগের regression layer থেকে
 পাওয়া value আপডেট করে দিবো।
                                                                    if curr_iou
 > best_iou_for_loc:
                                                                        bes
t_iou_for_loc = curr_iou
                                                                        bes
t_regr = (tx, ty, tw, th)

                                                                # যদি IOU value >0.3
  এবং <0.7 থাকে তবে ambiguous ধরে bbox_type neutral করে দিবো।
                                                                if C.rpn_min_overla
p < curr_iou < C.rpn_max_overlap:
                                                                    # gray zone
 between neg and pos
                                                                    if bbox_typ
e != 'pos':
                                                                        bbo
x_type = 'neutral'

                                                            # এবার pos, neutral, neg এর উপর
 ভিত্তি করে classification value আপডেট করবো।
                                                            if bbox_type == 'neg':
                                                                y_is_box_valid[jy, ix, anch
or_ratio_idx + n_anchratios * anchor_size_idx] = 1
                                                                y_rpn_overlap[jy, ix, ancho
r_ratio_idx + n_anchratios * anchor_size_idx] = 0
                                                            elif bbox_type == 'neutral':
                                                                y_is_box_valid[jy, ix, anch
or_ratio_idx + n_anchratios * anchor_size_idx] = 0
                                                                y_rpn_overlap[jy, ix, ancho
r_ratio_idx + n_anchratios * anchor_size_idx] = 0
                                                            elif bbox_type == 'pos':
                                                                y_is_box_valid[jy, ix, anch
or_ratio_idx + n_anchratios * anchor_size_idx] = 1
                                                                y_rpn_overlap[jy, ix, ancho
r_ratio_idx + n_anchratios * anchor_size_idx] = 1
                                                                start = 4 * (anchor_ratio_i
dx + n_anchratios * anchor_size_idx)
```

```
                                                y_rpn_regr[jy, ix, start:st
art+4] = best_regr

        # we ensure that every bbox has at least one positive RPN region

        for idx in range(num_anchors_for_bbox.shape[0]):
                if num_anchors_for_bbox[idx] == 0:
                        # no box with an IOU greater than zero ...
                        if best_anchor_for_bbox[idx, 0] == -1:
                                continue
                        y_is_box_valid[
                                best_anchor_for_bbox[idx,0], best_anchor_fo
r_bbox[idx,1], best_anchor_for_bbox[idx,2] + n_anchratios *
                                best_anchor_for_bbox[idx,3]] = 1
                        y_rpn_overlap[
                                best_anchor_for_bbox[idx,0], best_anchor_fo
r_bbox[idx,1], best_anchor_for_bbox[idx,2] + n_anchratios *
                                best_anchor_for_bbox[idx,3]] = 1
                        start = 4 * (best_anchor_for_bbox[idx,2] + n_anchra
tios * best_anchor_for_bbox[idx,3])
                        y_rpn_regr[
                                best_anchor_for_bbox[idx,0], best_anchor_fo
r_bbox[idx,1], start:start+4] = best_dx_for_bbox[idx, :]

        y_rpn_overlap = np.transpose(y_rpn_overlap, (2, 0, 1))
        y_rpn_overlap = np.expand_dims(y_rpn_overlap, axis=0)

        y_is_box_valid = np.transpose(y_is_box_valid, (2, 0, 1))
        y_is_box_valid = np.expand_dims(y_is_box_valid, axis=0)

        y_rpn_regr = np.transpose(y_rpn_regr, (2, 0, 1))
        y_rpn_regr = np.expand_dims(y_rpn_regr, axis=0)

        pos_locs = np.where(np.logical_and(y_rpn_overlap[0, :, :, :] == 1,
y_is_box_valid[0, :, :, :] == 1))
        neg_locs = np.where(np.logical_and(y_rpn_overlap[0, :, :, :] == 0,
y_is_box_valid[0, :, :, :] == 1))

        num_pos = len(pos_locs[0])

        # one issue is that the RPN has many more negative than positive re
gions, so we turn off some of the negative
        # regions. We also limit it to 256 regions.
        num_regions = 256

        if len(pos_locs[0]) > num_regions/2:
                val_locs = random.sample(range(len(pos_locs[0])), len(pos_l
ocs[0]) - num_regions/2)
                y_is_box_valid[0, pos_locs[0][val_locs], pos_locs[1][val_lo
cs], pos_locs[2][val_locs]] = 0
                num_pos = num_regions/2

        if len(neg_locs[0]) + num_pos > num_regions:
                val_locs = random.sample(range(len(neg_locs[0])), len(neg_l
ocs[0]) - num_pos)
                y_is_box_valid[0, neg_locs[0][val_locs], neg_locs[1][val_lo
cs], neg_locs[2][val_locs]] = 0
```

```
        y_rpn_cls = np.concatenate([y_is_box_valid, y_rpn_overlap], axis=1)
        y_rpn_regr = np.concatenate([np.repeat(y_rpn_overlap, 4, axis=1), y
_rpn_regr], axis=1)

        return np.copy(y_rpn_cls), np.copy(y_rpn_regr), num_pos
```

Get new image size and augment the image

In [0]:

```
def get_new_img_size(width, height, img_min_side=300):
        if width <= height:
                f = float(img_min_side) / width
                resized_height = int(f * height)
                resized_width = img_min_side
        else:
                f = float(img_min_side) / height
                resized_width = int(f * width)
                resized_height = img_min_side

        return resized_width, resized_height

def augment(img_data, config, augment=True):
        assert 'filepath' in img_data
        assert 'bboxes' in img_data
        assert 'width' in img_data
        assert 'height' in img_data

        img_data_aug = copy.deepcopy(img_data)

        img = cv2.imread(img_data_aug['filepath'])

        if augment:
                rows, cols = img.shape[:2]

                if config.use_horizontal_flips and np.random.randint(0, 2)
== 0:
                        img = cv2.flip(img, 1)
                        for bbox in img_data_aug['bboxes']:
                                x1 = bbox['x1']
                                x2 = bbox['x2']
                                bbox['x2'] = cols - x1
                                bbox['x1'] = cols - x2

                if config.use_vertical_flips and np.random.randint(0, 2) ==
 0:
                        img = cv2.flip(img, 0)
                        for bbox in img_data_aug['bboxes']:
                                y1 = bbox['y1']
                                y2 = bbox['y2']
                                bbox['y2'] = rows - y1
                                bbox['y1'] = rows - y2

                if config.rot_90:
                        angle = np.random.choice([0,90,180,270],1)[0]
                        if angle == 270:
                                img = np.transpose(img, (1,0,2))
                                img = cv2.flip(img, 0)
                        elif angle == 180:
                                img = cv2.flip(img, -1)
                        elif angle == 90:
                                img = np.transpose(img, (1,0,2))
                                img = cv2.flip(img, 1)
                        elif angle == 0:
                                pass

                        for bbox in img_data_aug['bboxes']:
```

```
                x1 = bbox['x1']
                x2 = bbox['x2']
                y1 = bbox['y1']
                y2 = bbox['y2']
                if angle == 270:
                        bbox['x1'] = y1
                        bbox['x2'] = y2
                        bbox['y1'] = cols - x2
                        bbox['y2'] = cols - x1
                elif angle == 180:
                        bbox['x2'] = cols - x1
                        bbox['x1'] = cols - x2
                        bbox['y2'] = rows - y1
                        bbox['y1'] = rows - y2
                elif angle == 90:
                        bbox['x1'] = rows - y2
                        bbox['x2'] = rows - y1
                        bbox['y1'] = x1
                        bbox['y2'] = x2
                elif angle == 0:
                        pass

        img_data_aug['width'] = img.shape[1]
        img_data_aug['height'] = img.shape[0]
        return img_data_aug, img
```

Generate the ground_truth anchors

In [0]:

```
def get_anchor_gt(all_img_data, C, img_length_calc_function, mode='train'):
        """ Yield the ground-truth anchors as Y (labels)

        Args:
                all_img_data: list(filepath, width, height, list(bboxes))
                C: config
                img_length_calc_function: function to calculate final laye
r's feature map (of base model) size according to input image size
                mode: 'train' or 'test'; 'train' mode need augmentation

        Returns:
                x_img: image data after resized and scaling (smallest size
 = 300px)
                Y: [y_rpn_cls, y_rpn_regr]
                img_data_aug: augmented image data (original image with aug
mentation)
                debug_img: show image for debug
                num_pos: show number of positive anchors for debug
        """
        while True:

                for img_data in all_img_data:
                        try:

                                # read in image, and optionally add augment
ation

                                if mode == 'train':
                                        img_data_aug, x_img = augment(img_d
ata, C, augment=True)
                                else:
                                        img_data_aug, x_img = augment(img_d
ata, C, augment=False)

                                (width, height) = (img_data_aug['width'], i
mg_data_aug['height'])
                                (rows, cols, _) = x_img.shape

                                assert cols == width
                                assert rows == height

                                # get image dimensions for resizing
                                (resized_width, resized_height) = get_new_i
mg_size(width, height, C.im_size)

                                # resize the image so that smalles side is
 length = 300px
                                x_img = cv2.resize(x_img, (resized_width, r
esized_height), interpolation=cv2.INTER_CUBIC)
                                debug_img = x_img.copy()

                                try:
                                        y_rpn_cls, y_rpn_regr, num_pos = ca
lc_rpn(C, img_data_aug, width, height, resized_width, resized_height, img_l
ength_calc_function)
                                except:
                                        continue
```

```
                                # Zero-center by mean pixel, and preprocess
 image

                                x_img = x_img[:,:, (2, 1, 0)]  # BGR -> RGB
                                x_img = x_img.astype(np.float32)
                                x_img[:, :, 0] -= C.img_channel_mean[0]
                                x_img[:, :, 1] -= C.img_channel_mean[1]
                                x_img[:, :, 2] -= C.img_channel_mean[2]
                                x_img /= C.img_scaling_factor

                                x_img = np.transpose(x_img, (2, 0, 1))
                                x_img = np.expand_dims(x_img, axis=0)

                                y_rpn_regr[:, y_rpn_regr.shape[1]//2:, :,
:] *= C.std_scaling

                                x_img = np.transpose(x_img, (0, 2, 3, 1))
                                y_rpn_cls = np.transpose(y_rpn_cls, (0, 2,
3, 1))
                                y_rpn_regr = np.transpose(y_rpn_regr, (0, 2
, 3, 1))

                                yield np.copy(x_img), [np.copy(y_rpn_cls),
np.copy(y_rpn_regr)], img_data_aug, debug_img, num_pos

                            except Exception as e:
                                print(e)
                                continue
```

In [0]:

```
def non_max_suppression_fast(boxes, probs, overlap_thresh=0.9, max_boxes=300):
        # code used from here: http://www.pyimagesearch.com/2015/02/16/faster-non-maximum-suppression-python/
        # if there are no boxes, return an empty list

    # Process explanation:
    #   Step 1: Sort the probs list
    #   Step 2: Find the larget prob 'Last' in the list and save it to the pick list
    #   Step 3: Calculate the IoU with 'Last' box and other boxes in the list. If the IoU is larger than overlap_threshold, delete the box from list
    #   Step 4: Repeat step 2 and step 3 until there is no item in the probs list
        if len(boxes) == 0:
                return []

        # grab the coordinates of the bounding boxes
        x1 = boxes[:, 0]
        y1 = boxes[:, 1]
        x2 = boxes[:, 2]
        y2 = boxes[:, 3]

        np.testing.assert_array_less(x1, x2)
        np.testing.assert_array_less(y1, y2)

        # if the bounding boxes integers, convert them to floats --
        # this is important since we'll be doing a bunch of divisions
        if boxes.dtype.kind == "i":
                boxes = boxes.astype("float")

        # initialize the list of picked indexes
        pick = []

        # calculate the areas
        area = (x2 - x1) * (y2 - y1)

        # sort the bounding boxes
        idxs = np.argsort(probs)

        # keep looping while some indexes still remain in the indexes
        # list
        while len(idxs) > 0:
                # grab the last index in the indexes list and add the
                # index value to the list of picked indexes
                last = len(idxs) - 1
                i = idxs[last]
                pick.append(i)

                # find the intersection

                xx1_int = np.maximum(x1[i], x1[idxs[:last]])
                yy1_int = np.maximum(y1[i], y1[idxs[:last]])
                xx2_int = np.minimum(x2[i], x2[idxs[:last]])
                yy2_int = np.minimum(y2[i], y2[idxs[:last]])

                ww_int = np.maximum(0, xx2_int - xx1_int)
```

```
                hh_int = np.maximum(0, yy2_int - yy1_int)

                area_int = ww_int * hh_int

                # find the union
                area_union = area[i] + area[idxs[:last]] - area_int

                # compute the ratio of overlap
                overlap = area_int/(area_union + 1e-6)

                # delete all indexes from the index list that have
                idxs = np.delete(idxs, np.concatenate(([last],
                        np.where(overlap > overlap_thresh)[0])))

                if len(pick) >= max_boxes:
                        break

        # return only the bounding boxes that were picked using the integer
 data type
        boxes = boxes[pick].astype("int")
        probs = probs[pick]
        return boxes, probs

def apply_regr_np(X, T):
        """Apply regression layer to all anchors in one feature map

        Args:
                X: shape=(4, 18, 25) the current anchor type for all points
 in the feature map
                T: regression layer shape=(4, 18, 25)

        Returns:
                X: regressed position and size for current anchor
        """
        try:
                x = X[0, :, :]
                y = X[1, :, :]
                w = X[2, :, :]
                h = X[3, :, :]

                tx = T[0, :, :]
                ty = T[1, :, :]
                tw = T[2, :, :]
                th = T[3, :, :]

                cx = x + w/2.
                cy = y + h/2.
                cx1 = tx * w + cx
                cy1 = ty * h + cy

                w1 = np.exp(tw.astype(np.float64)) * w
                h1 = np.exp(th.astype(np.float64)) * h
                x1 = cx1 - w1/2.
                y1 = cy1 - h1/2.

                x1 = np.round(x1)
                y1 = np.round(y1)
```

```
                w1 = np.round(w1)
                h1 = np.round(h1)
                return np.stack([x1, y1, w1, h1])
        except Exception as e:
                print(e)
                return X

def apply_regr(x, y, w, h, tx, ty, tw, th):
    # Apply regression to x, y, w and h
        try:
                cx = x + w/2.
                cy = y + h/2.
                cx1 = tx * w + cx
                cy1 = ty * h + cy
                w1 = math.exp(tw) * w
                h1 = math.exp(th) * h
                x1 = cx1 - w1/2.
                y1 = cy1 - h1/2.
                x1 = int(round(x1))
                y1 = int(round(y1))
                w1 = int(round(w1))
                h1 = int(round(h1))

                return x1, y1, w1, h1

        except ValueError:
                return x, y, w, h
        except OverflowError:
                return x, y, w, h
        except Exception as e:
                print(e)
                return x, y, w, h
```

In [0]:

```
def rpn_to_roi(rpn_layer, regr_layer, C, dim_ordering, use_regr=True, max_boxes=300,overlap_thresh=0.9):
        """Convert rpn layer to roi bboxes

        Args: (num_anchors = 9)
                rpn_layer: output layer for rpn classification
                        shape (1, feature_map.height, feature_map.width, num_anchors)
                        Might be (1, 18, 25, 9) if resized image is 400 width and 300
                regr_layer: output layer for rpn regression
                        shape (1, feature_map.height, feature_map.width, num_anchors)
                        Might be (1, 18, 25, 36) if resized image is 400 width and 300
                C: config
                use_regr: Wether to use bboxes regression in rpn
                max_boxes: max bboxes number for non-max-suppression (NMS)
                overlap_thresh: If iou in NMS is larger than this threshold, drop the box

        Returns:
                result: boxes from non-max-suppression (shape=(300, 4))
                        boxes: coordinates for bboxes (on the feature map)
        """
        regr_layer = regr_layer / C.std_scaling

        anchor_sizes = C.anchor_box_scales   # (3 in here)
        anchor_ratios = C.anchor_box_ratios  # (3 in here)

        assert rpn_layer.shape[0] == 1

        (rows, cols) = rpn_layer.shape[1:3]

        curr_layer = 0

        # A.shape = (4, feature_map.height, feature_map.width, num_anchors)

        # Might be (4, 18, 25, 9) if resized image is 400 width and 300
        # A is the coordinates for 9 anchors for every point in the feature map
        # => all 18x25x9=4050 anchors cooridnates
        A = np.zeros((4, rpn_layer.shape[1], rpn_layer.shape[2], rpn_layer.shape[3]))

        for anchor_size in anchor_sizes:
                for anchor_ratio in anchor_ratios:
                        # anchor_x = (128 * 1) / 16 = 8  => width of current anchor
                        # anchor_y = (128 * 2) / 16 = 16 => height of current anchor
                        anchor_x = (anchor_size * anchor_ratio[0])/C.rpn_stride
                        anchor_y = (anchor_size * anchor_ratio[1])/C.rpn_stride

                        # curr_layer: 0~8 (9 anchors)
```

```
                        # the Kth anchor of all position in the feature map
 (9th in total)
                        regr = regr_layer[0, :, :, 4 * curr_layer:4 * curr_
layer + 4] # shape => (18, 25, 4)
                        regr = np.transpose(regr, (2, 0, 1)) # shape => (4,
 18, 25)

                        # Create 18x25 mesh grid
                        # For every point in x, there are all the y points
 and vice versa
                        # X.shape = (18, 25)
                        # Y.shape = (18, 25)
                        X, Y = np.meshgrid(np.arange(cols),np. arange(rows
))

                        # Calculate anchor position and size for each featu
re map point
                        A[0, :, :, curr_layer] = X - anchor_x/2 # Top left
 x coordinate
                        A[1, :, :, curr_layer] = Y - anchor_y/2 # Top left
 y coordinate
                        A[2, :, :, curr_layer] = anchor_x       # width of
 current anchor
                        A[3, :, :, curr_layer] = anchor_y       # height of
 current anchor

                        # Apply regression to x, y, w and h if there is rpn
 regression layer
                        if use_regr:
                                A[:, :, :, curr_layer] = apply_regr_np(A[:,
 :, :, curr_layer], regr)

                        # Avoid width and height exceeding 1
                        A[2, :, :, curr_layer] = np.maximum(1, A[2, :, :, c
urr_layer])
                        A[3, :, :, curr_layer] = np.maximum(1, A[3, :, :, c
urr_layer])

                        # Convert (x, y , w, h) to (x1, y1, x2, y2)
                        # x1, y1 is top left coordinate
                        # x2, y2 is bottom right coordinate
                        A[2, :, :, curr_layer] += A[0, :, :, curr_layer]
                        A[3, :, :, curr_layer] += A[1, :, :, curr_layer]

                        # Avoid bboxes drawn outside the feature map
                        A[0, :, :, curr_layer] = np.maximum(0, A[0, :, :, c
urr_layer])
                        A[1, :, :, curr_layer] = np.maximum(0, A[1, :, :, c
urr_layer])
                        A[2, :, :, curr_layer] = np.minimum(cols-1, A[2, :,
 :, curr_layer])
                        A[3, :, :, curr_layer] = np.minimum(rows-1, A[3, :,
 :, curr_layer])

                        curr_layer += 1

        all_boxes = np.reshape(A.transpose((0, 3, 1, 2)), (4, -1)).transpos
```

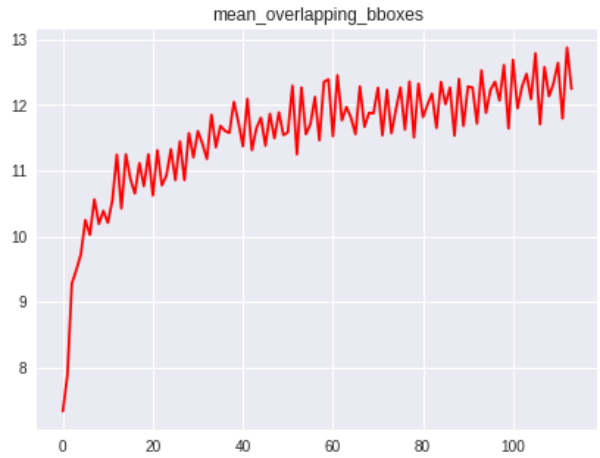
```
e((1, 0))  # shape=(4050, 4)
        all_probs = rpn_layer.transpose((0, 3, 1, 2)).reshape((-1))
           # shape=(4050,)

        x1 = all_boxes[:, 0]
        y1 = all_boxes[:, 1]
        x2 = all_boxes[:, 2]
        y2 = all_boxes[:, 3]

        # Find out the bboxes which is illegal and delete them from bboxes
 list
        idxs = np.where((x1 - x2 >= 0) | (y1 - y2 >= 0))

        all_boxes = np.delete(all_boxes, idxs, 0)
        all_probs = np.delete(all_probs, idxs, 0)

        # Apply non_max_suppression
        # Only extract the bboxes. Don't need rpn probs in the later proces
s
        result = non_max_suppression_fast(all_boxes, all_probs, overlap_thr
esh=overlap_thresh, max_boxes=max_boxes)[0]

        return result
```

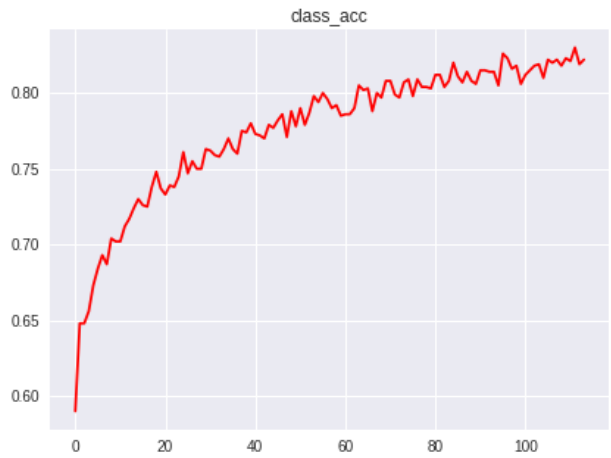
In [0]:
```
base_path = 'drive/My Drive/AI/Faster_RCNN'

test_path = 'drive/My Drive/AI/Dataset/Open Images Dataset v4 (Bounding Box
es)/person_car_phone_test_annotation.txt' # Test data (annotation file)

test_base_path = 'drive/My Drive/AI/Dataset/Open Images Dataset v4 (Boundin
g Boxes)/test' # Directory to save the test images

config_output_filename = os.path.join(base_path, 'model_vgg_config.pickle')
```

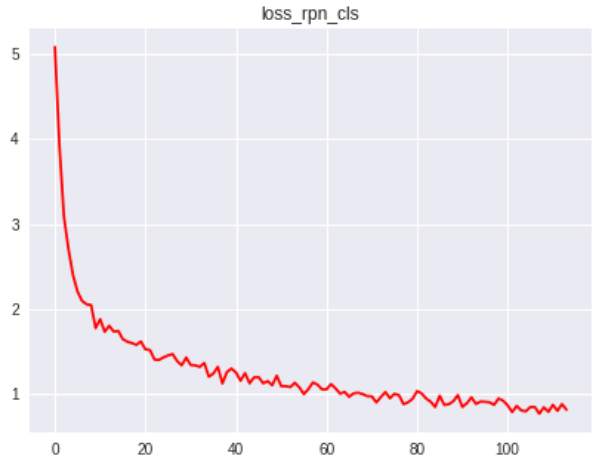
In [0]:
```
with open(config_output_filename, 'rb') as f_in:
        C = pickle.load(f_in)

# turn off any data augmentation at test time
C.use_horizontal_flips = False
C.use_vertical_flips = False
C.rot_90 = False
```

In [18]:

```
# Load the records
record_df = pd.read_csv(C.record_path)

r_epochs = len(record_df)

plt.figure(figsize=(15,5))
plt.subplot(1,2,1)
plt.plot(np.arange(0, r_epochs), record_df['mean_overlapping_bboxes'], 'r')
plt.title('mean_overlapping_bboxes')

plt.subplot(1,2,2)
plt.plot(np.arange(0, r_epochs), record_df['class_acc'], 'r')
plt.title('class_acc')

plt.show()

plt.figure(figsize=(15,5))

plt.subplot(1,2,1)
plt.plot(np.arange(0, r_epochs), record_df['loss_rpn_cls'], 'r')
plt.title('loss_rpn_cls')

plt.subplot(1,2,2)
plt.plot(np.arange(0, r_epochs), record_df['loss_rpn_regr'], 'r')
plt.title('loss_rpn_regr')
plt.show()
plt.figure(figsize=(15,5))
plt.subplot(1,2,1)
plt.plot(np.arange(0, r_epochs), record_df['loss_class_cls'], 'r')
plt.title('loss_class_cls')

plt.subplot(1,2,2)
plt.plot(np.arange(0, r_epochs), record_df['loss_class_regr'], 'r')
plt.title('loss_class_regr')
plt.show()
plt.figure(figsize=(15,5))
plt.subplot(1,2,1)
plt.plot(np.arange(0, r_epochs), record_df['curr_loss'], 'r')
plt.title('total_loss')

plt.subplot(1,2,2)
plt.plot(np.arange(0, r_epochs), record_df['elapsed_time'], 'r')
plt.title('elapsed_time')

plt.show()
```


mean_overlapping_bboxes


class_acc

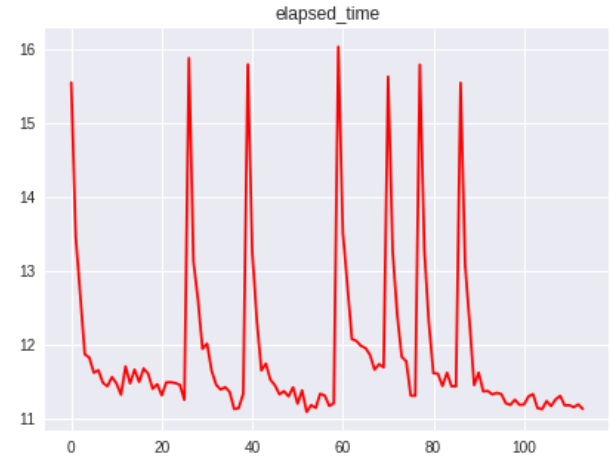

loss_rpn_cls


loss_rpn_regr


loss_class_cls


loss_class_regr


total_loss


elapsed_time

Test

In [0]:

```
def format_img_size(img, C):
        """ formats the image size based on config """
        img_min_side = float(C.im_size)
        (height,width,_) = img.shape

        if width <= height:
                ratio = img_min_side/width
                new_height = int(ratio * height)
                new_width = int(img_min_side)
        else:
                ratio = img_min_side/height
                new_width = int(ratio * width)
                new_height = int(img_min_side)
        img = cv2.resize(img, (new_width, new_height), interpolation=cv2.IN
TER_CUBIC)
        return img, ratio

def format_img_channels(img, C):
        """ formats the image channels based on config """
        img = img[:, :, (2, 1, 0)]
        img = img.astype(np.float32)
        img[:, :, 0] -= C.img_channel_mean[0]
        img[:, :, 1] -= C.img_channel_mean[1]
        img[:, :, 2] -= C.img_channel_mean[2]
        img /= C.img_scaling_factor
        img = np.transpose(img, (2, 0, 1))
        img = np.expand_dims(img, axis=0)
        return img

def format_img(img, C):
        """ formats an image for model prediction based on config """
        img, ratio = format_img_size(img, C)
        img = format_img_channels(img, C)
        return img, ratio

# Method to transform the coordinates of the bounding box to its original s
ize
def get_real_coordinates(ratio, x1, y1, x2, y2):

        real_x1 = int(round(x1 // ratio))
        real_y1 = int(round(y1 // ratio))
        real_x2 = int(round(x2 // ratio))
        real_y2 = int(round(y2 // ratio))

        return (real_x1, real_y1, real_x2 ,real_y2)
```

In [20]:
```
num_features = 512

input_shape_img = (None, None, 3)
input_shape_features = (None, None, num_features)

img_input = Input(shape=input_shape_img)
roi_input = Input(shape=(C.num_rois, 4))
feature_map_input = Input(shape=input_shape_features)

# define the base network (VGG here, can be Resnet50, Inception, etc)
shared_layers = nn_base(img_input, trainable=True)

# define the RPN, built on the base layers
num_anchors = len(C.anchor_box_scales) * len(C.anchor_box_ratios)
rpn_layers = rpn_layer(shared_layers, num_anchors)

classifier = classifier_layer(feature_map_input, roi_input, C.num_rois, nb_
classes=len(C.class_mapping))

model_rpn = Model(img_input, rpn_layers)
model_classifier_only = Model([feature_map_input, roi_input], classifier)

model_classifier = Model([feature_map_input, roi_input], classifier)

print('Loading weights from {}'.format(C.model_path))
model_rpn.load_weights(C.model_path, by_name=True)
model_classifier.load_weights(C.model_path, by_name=True)

model_rpn.compile(optimizer='sgd', loss='mse')
model_classifier.compile(optimizer='sgd', loss='mse')
```

```
Loading weights from drive/My Drive/AI/Faster_RCNN/model/model_frcnn_vgg.hd
f5
```

In [21]:
```
# Switch key value for class mapping
class_mapping = C.class_mapping
class_mapping = {v: k for k, v in class_mapping.items()}
print(class_mapping)
class_to_color = {class_mapping[v]: np.random.randint(0, 255, 3) for v in c
lass_mapping}
```

```
{0: 'Person', 1: 'Car', 2: 'Mobile phone', 3: 'bg'}
```

In [0]:
```
test_imgs = os.listdir(test_base_path)

imgs_path = []
for i in range(12):
        idx = np.random.randint(len(test_imgs))
        imgs_path.append(test_imgs[idx])

all_imgs = []

classes = {}
```

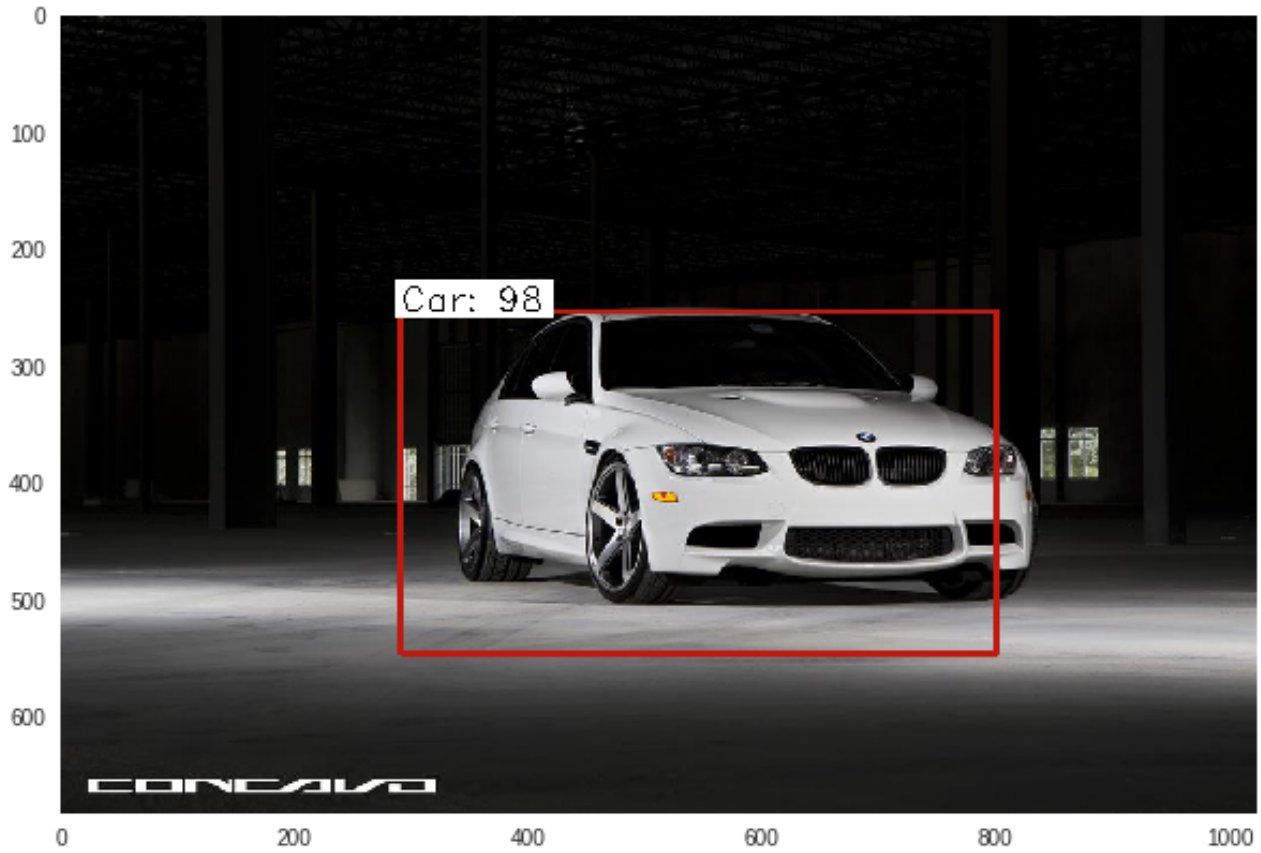
In [37]:

```
# If the box classification value is less than this, we ignore this box
bbox_threshold = 0.7

for idx, img_name in enumerate(imgs_path):
    if not img_name.lower().endswith(('.bmp', '.jpeg', '.jpg', '.png', '.ti
f', '.tiff')):
        continue
    print(img_name)
    st = time.time()
    filepath = os.path.join(test_base_path, img_name)

    img = cv2.imread(filepath)

    X, ratio = format_img(img, C)

    X = np.transpose(X, (0, 2, 3, 1))

    # get output layer Y1, Y2 from the RPN and the feature maps F
    # Y1: y_rpn_cls
    # Y2: y_rpn_regr
    [Y1, Y2, F] = model_rpn.predict(X)

    # Get bboxes by applying NMS
    # R.shape = (300, 4)
    R = rpn_to_roi(Y1, Y2, C, K.image_dim_ordering(), overlap_thresh=0.7)

    # convert from (x1,y1,x2,y2) to (x,y,w,h)
    R[:, 2] -= R[:, 0]
    R[:, 3] -= R[:, 1]

    # apply the spatial pyramid pooling to the proposed regions
    bboxes = {}
    probs = {}

    for jk in range(R.shape[0]//C.num_rois + 1):
        ROIs = np.expand_dims(R[C.num_rois*jk:C.num_rois*(jk+1), :], axis=0
)
        if ROIs.shape[1] == 0:
            break

        if jk == R.shape[0]//C.num_rois:
            #pad R
            curr_shape = ROIs.shape
            target_shape = (curr_shape[0],C.num_rois,curr_shape[2])
            ROIs_padded = np.zeros(target_shape).astype(ROIs.dtype)
            ROIs_padded[:, :curr_shape[1], :] = ROIs
            ROIs_padded[0, curr_shape[1]:, :] = ROIs[0, 0, :]
            ROIs = ROIs_padded

        [P_cls, P_regr] = model_classifier_only.predict([F, ROIs])

        # Calculate bboxes coordinates on resized image
        for ii in range(P_cls.shape[1]):
            # Ignore 'bg' class
            if np.max(P_cls[0, ii, :]) < bbox_threshold or np.argmax(P_cls[
0, ii, :]) == (P_cls.shape[2] - 1):
                continue
```

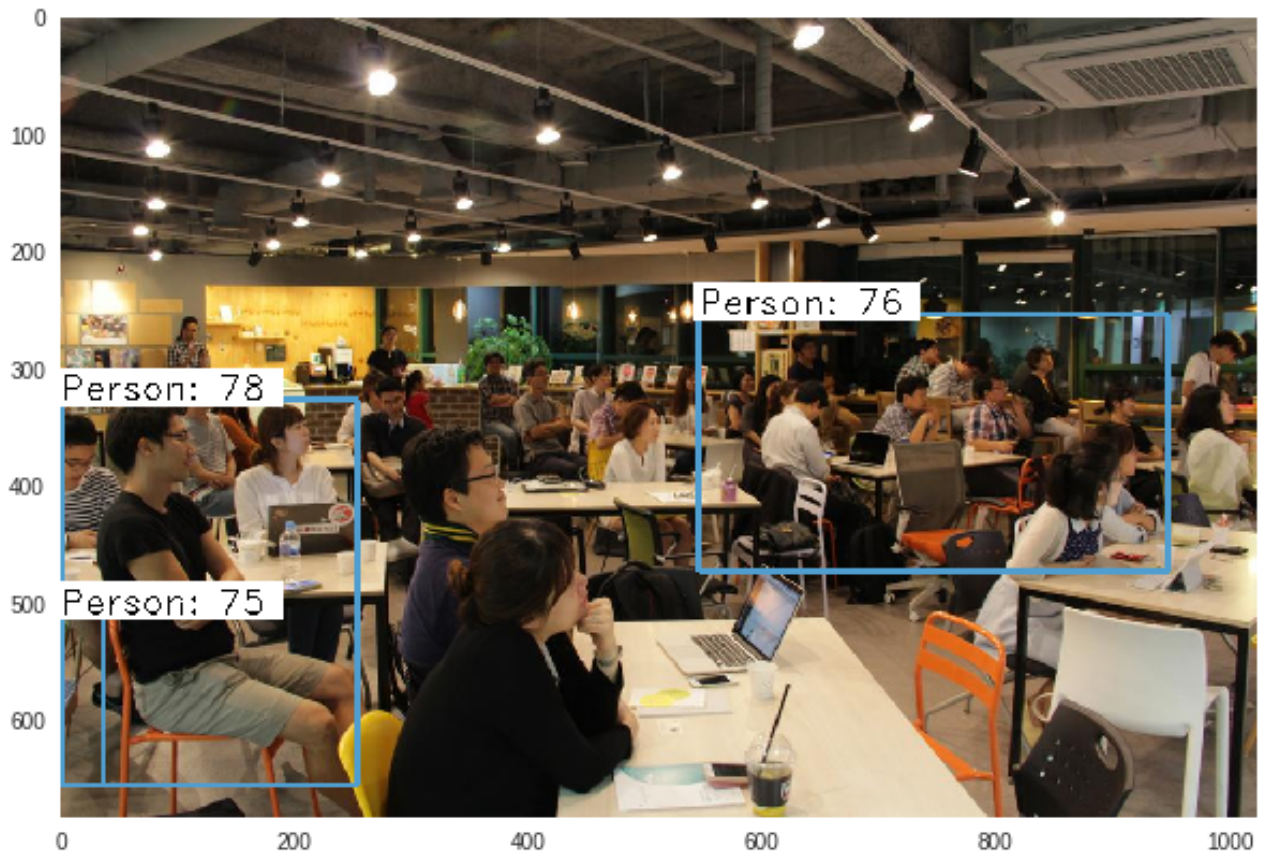
```
            cls_name = class_mapping[np.argmax(P_cls[0, ii, :])]

            if cls_name not in bboxes:
                bboxes[cls_name] = []
                probs[cls_name] = []

            (x, y, w, h) = ROIs[0, ii, :]

            cls_num = np.argmax(P_cls[0, ii, :])
            try:
                (tx, ty, tw, th) = P_regr[0, ii, 4*cls_num:4*(cls_num+1)]
                tx /= C.classifier_regr_std[0]
                ty /= C.classifier_regr_std[1]
                tw /= C.classifier_regr_std[2]
                th /= C.classifier_regr_std[3]
                x, y, w, h = apply_regr(x, y, w, h, tx, ty, tw, th)
            except:
                pass
            bboxes[cls_name].append([C.rpn_stride*x, C.rpn_stride*y, C.rpn_
stride*(x+w), C.rpn_stride*(y+h)])
            probs[cls_name].append(np.max(P_cls[0, ii, :]))

    all_dets = []

    for key in bboxes:
        bbox = np.array(bboxes[key])

        new_boxes, new_probs = non_max_suppression_fast(bbox, np.array(prob
s[key]), overlap_thresh=0.2)
        for jk in range(new_boxes.shape[0]):
            (x1, y1, x2, y2) = new_boxes[jk,:]

            # Calculate real coordinates on original image
            (real_x1, real_y1, real_x2, real_y2) = get_real_coordinates(rat
io, x1, y1, x2, y2)

            cv2.rectangle(img,(real_x1, real_y1), (real_x2, real_y2), (int(
class_to_color[key][0]), int(class_to_color[key][1]), int(class_to_color[ke
y][2])),4)

            textLabel = '{}: {}'.format(key,int(100*new_probs[jk]))
            all_dets.append((key,100*new_probs[jk]))

            (retval,baseLine) = cv2.getTextSize(textLabel,cv2.FONT_HERSHEY_
COMPLEX,1,1)
            textOrg = (real_x1, real_y1-0)

            cv2.rectangle(img, (textOrg[0] - 5, textOrg[1]+baseLine - 5), (
textOrg[0]+retval[0] + 5, textOrg[1]-retval[1] - 5), (0, 0, 0), 1)
            cv2.rectangle(img, (textOrg[0] - 5,textOrg[1]+baseLine - 5), (t
extOrg[0]+retval[0] + 5, textOrg[1]-retval[1] - 5), (255, 255, 255), -1)
            cv2.putText(img, textLabel, textOrg, cv2.FONT_HERSHEY_DUPLEX, 1
, (0, 0, 0), 1)

    print('Elapsed time = {}'.format(time.time() - st))
    print(all_dets)
```

```
    plt.figure(figsize=(10,10))
    plt.grid()
    plt.imshow(cv2.cvtColor(img,cv2.COLOR_BGR2RGB))
    plt.show()
```

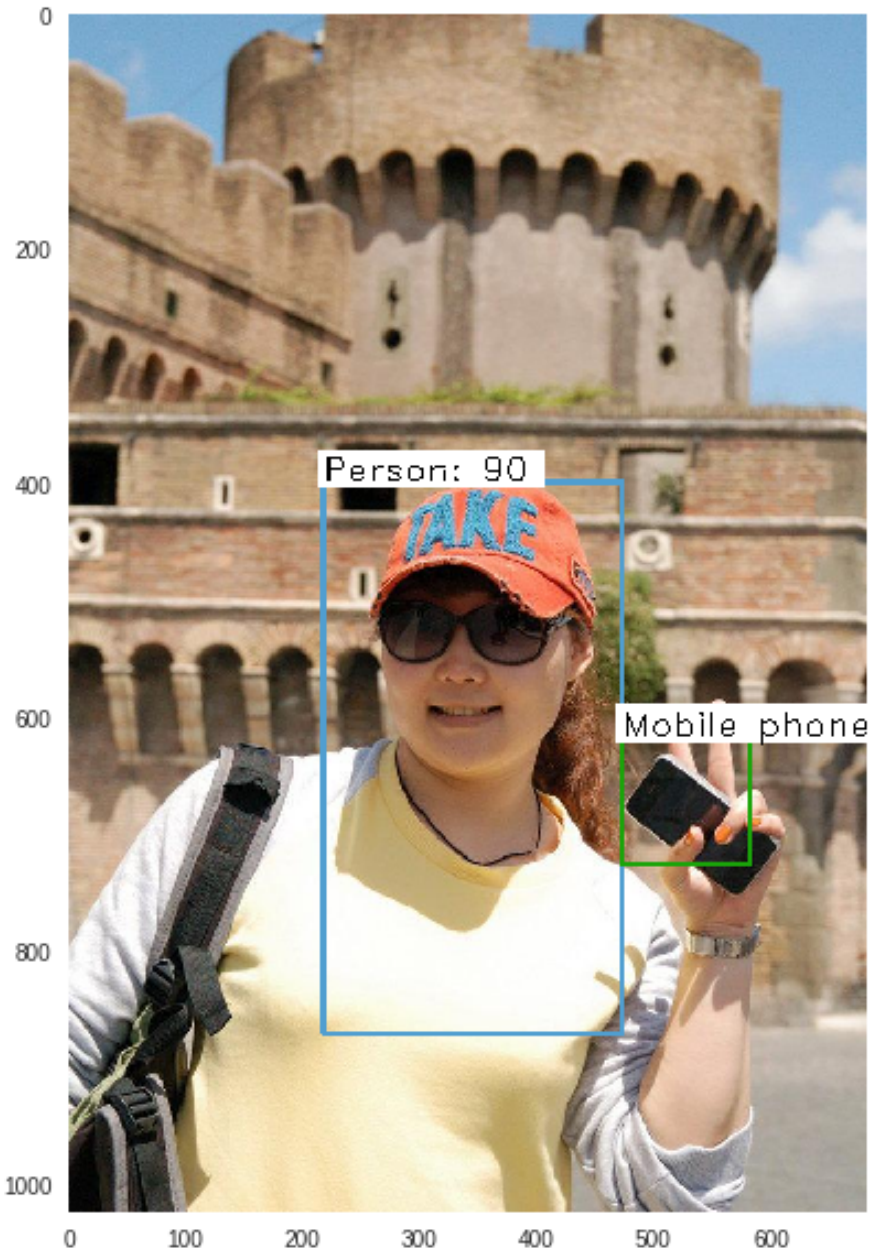
```
4dbfd180b2c1b8b1.jpg
Elapsed time = 1.081662893295288
[('Car', 98.87887835502625)]
```



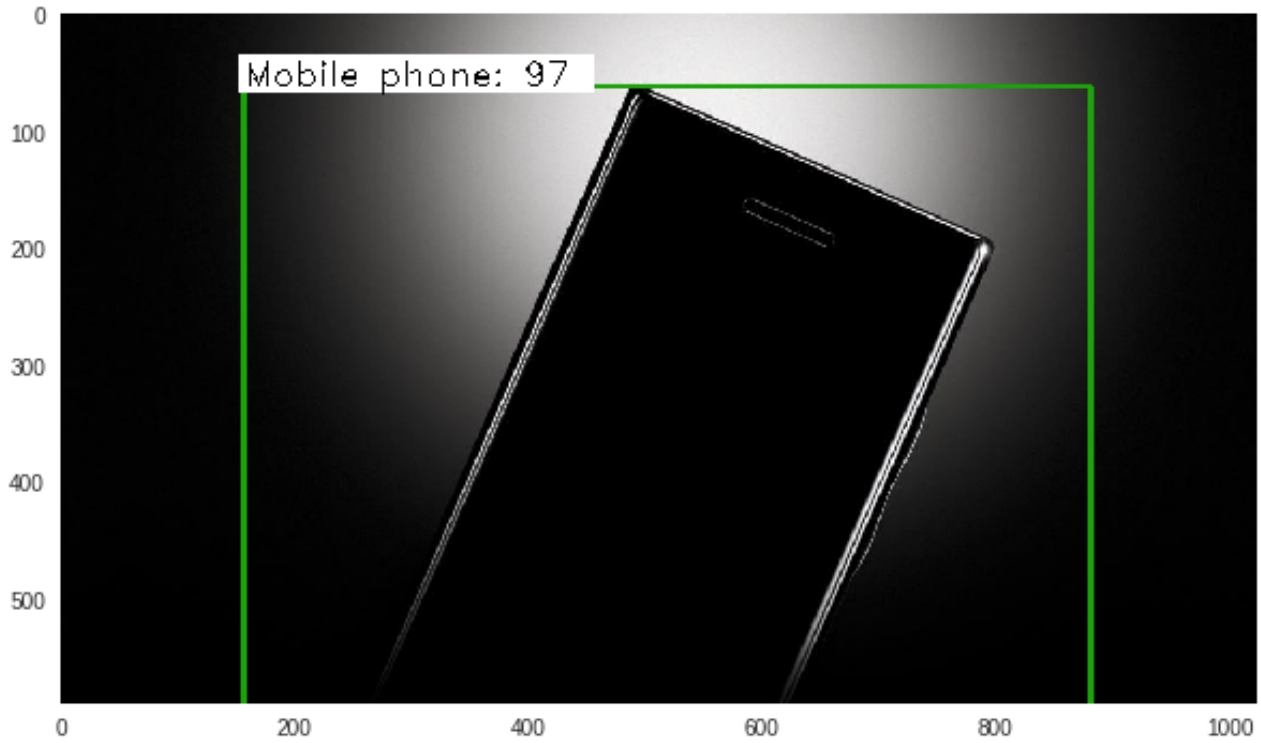

```
4ecc5942eacf7ffa.jpg
Elapsed time = 1.034881353378296
[('Person', 78.0624270439148), ('Person', 76.20276212692261), ('Person', 7
5.85426568984985)]
```



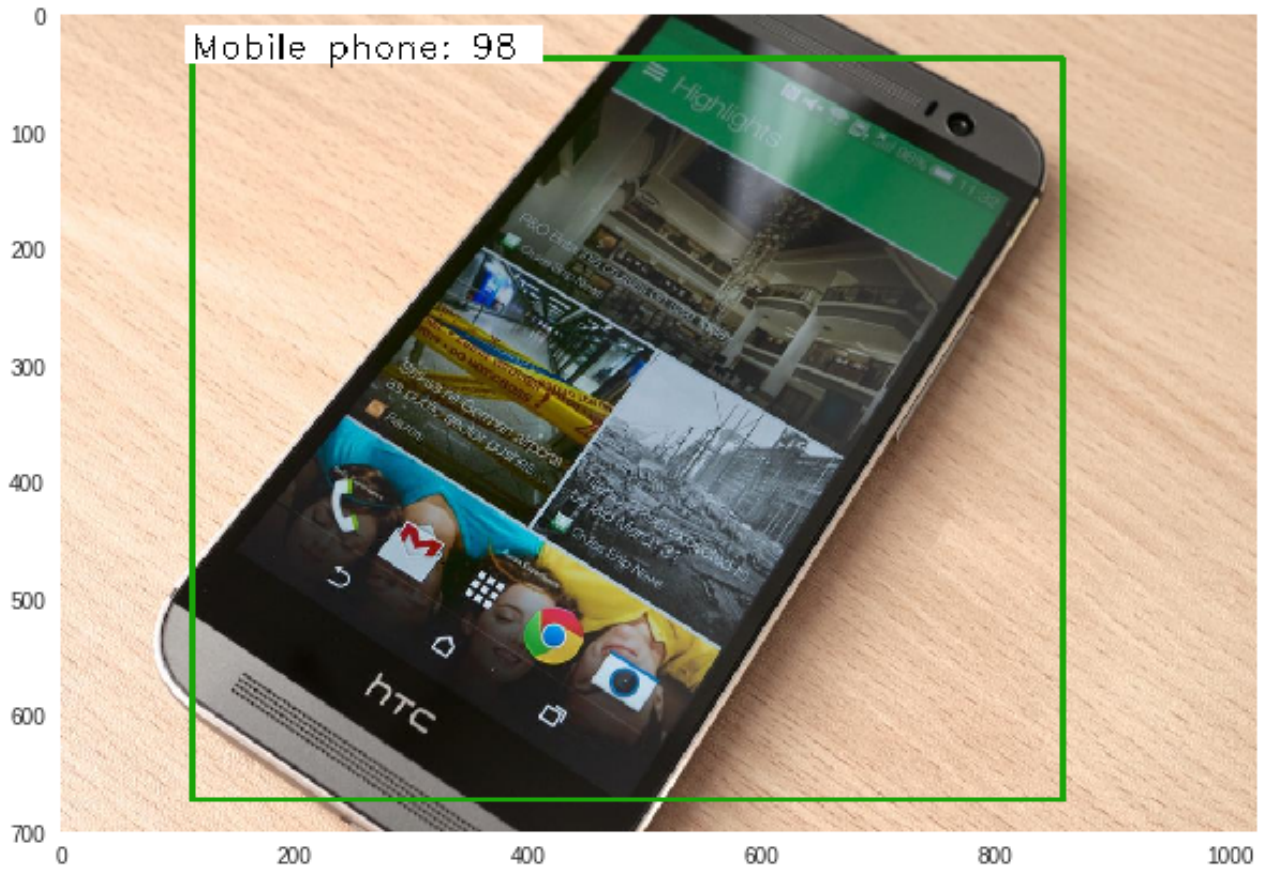
1b207b953d6c129f.jpg
Elapsed time = 1.520465612411499
[('Person', 90.48575758934021), ('Mobile phone', 96.47467732429504)]



013407956a56bed0.jpg
Elapsed time = 2.2138750553131104
[('Mobile phone', 97.9332447052002)]

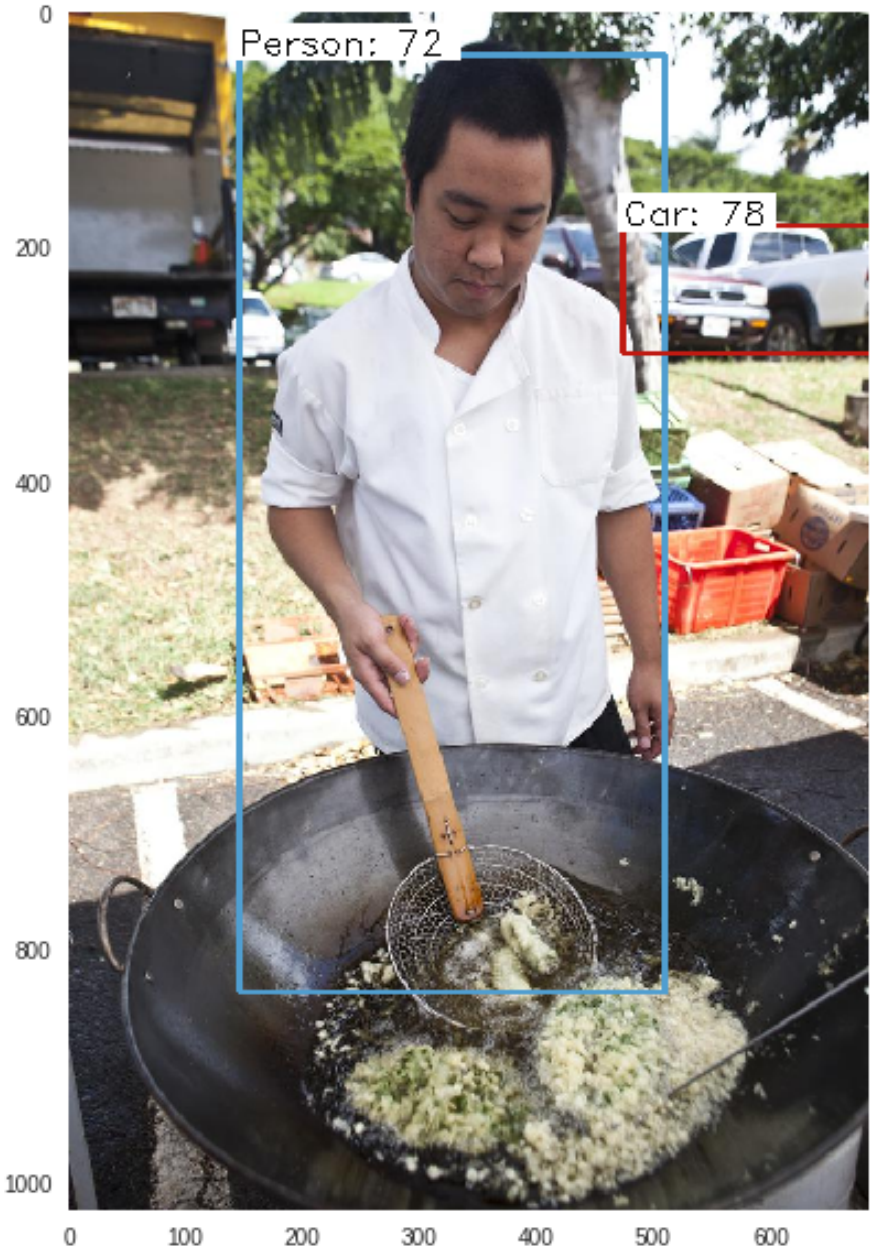


0a692491b1c2f1ce.jpg
Elapsed time = 0.6385619640350342
[('Mobile phone', 98.92338514328003)]

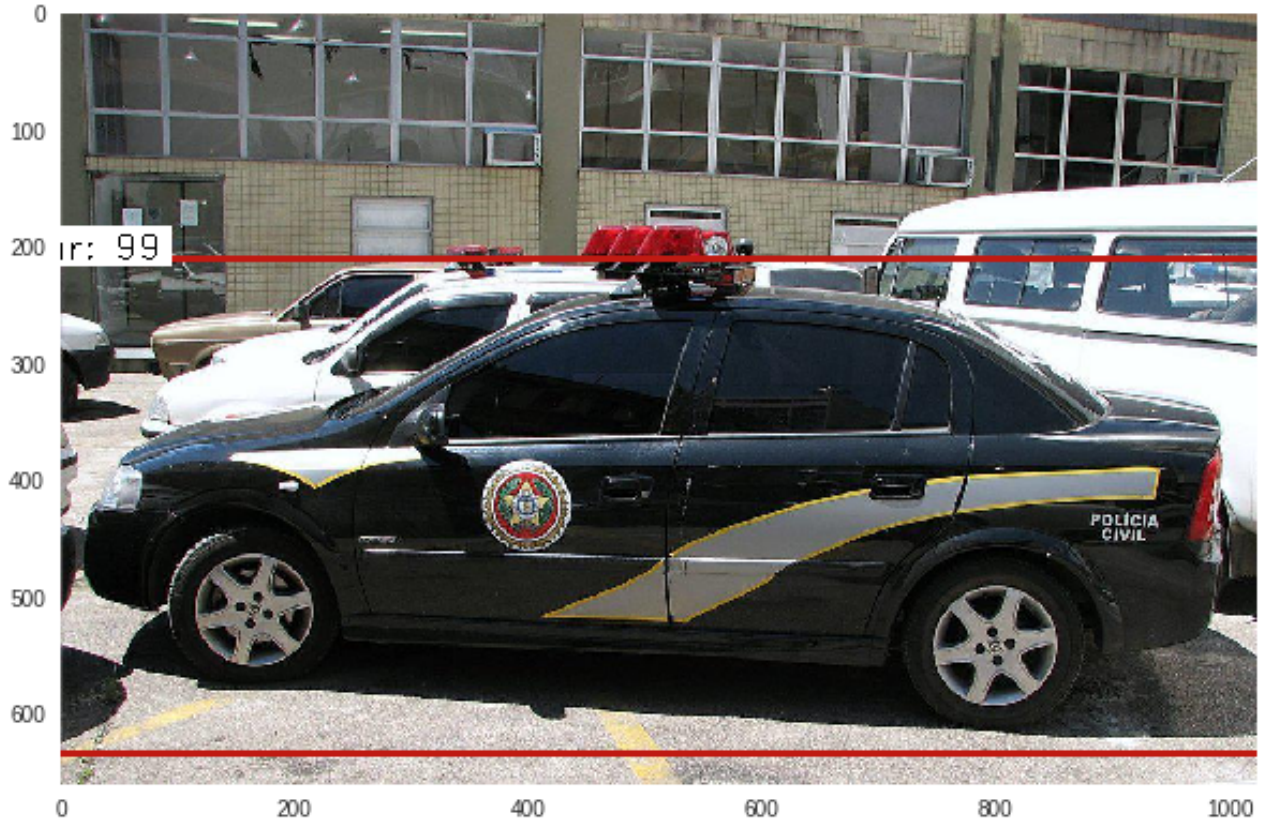


7d2d1d83020757d8.jpg
Elapsed time = 0.695296049118042
[('Car', 81.31412267684937), ('Person', 76.92756652832031)]


```
978cd66b8ca034fc.jpg
Elapsed time = 1.2788197994232178
[('Person', 72.23764657974243), ('Car', 78.980952501297)]
```



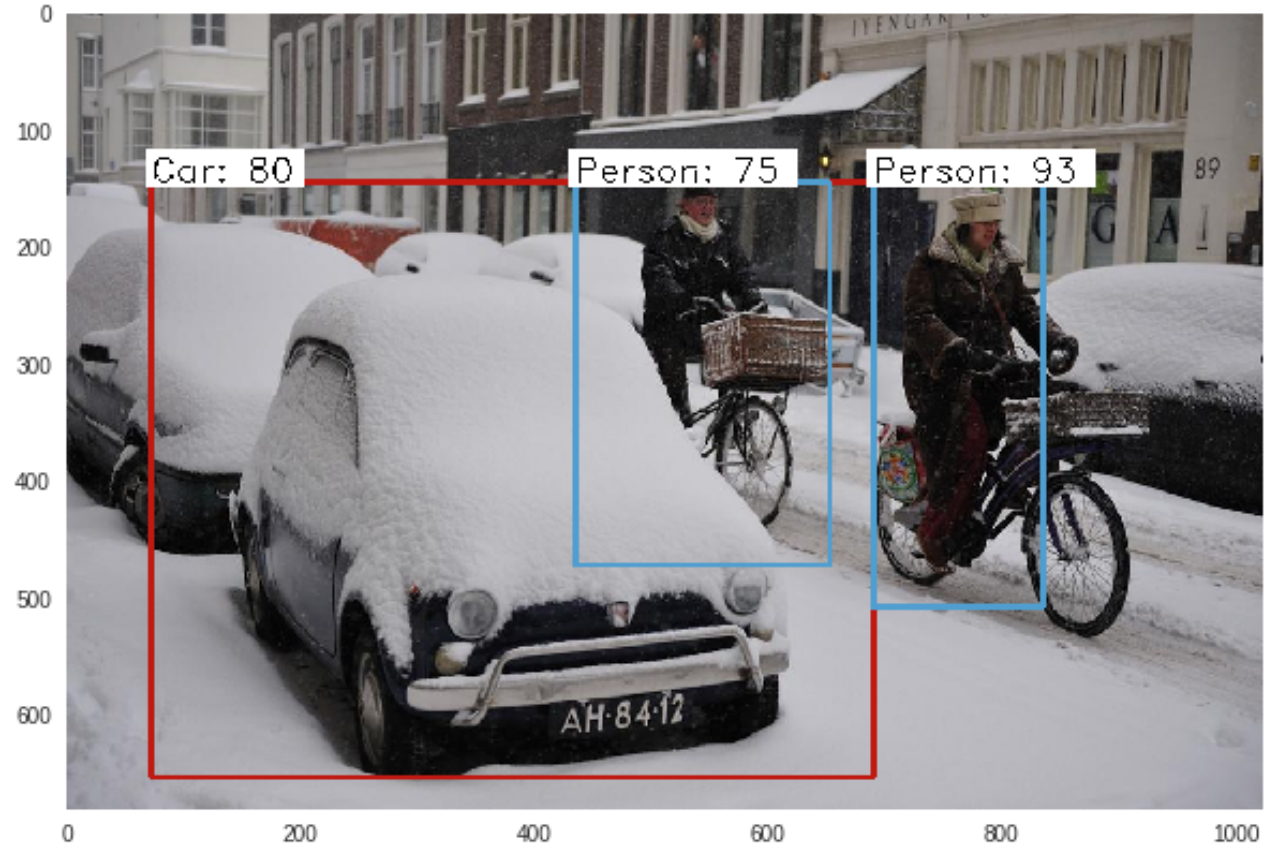

```
fc95d5a4e54023fc.jpg
Elapsed time = 2.075000762939453
[('Car', 99.30254220962524)]
```



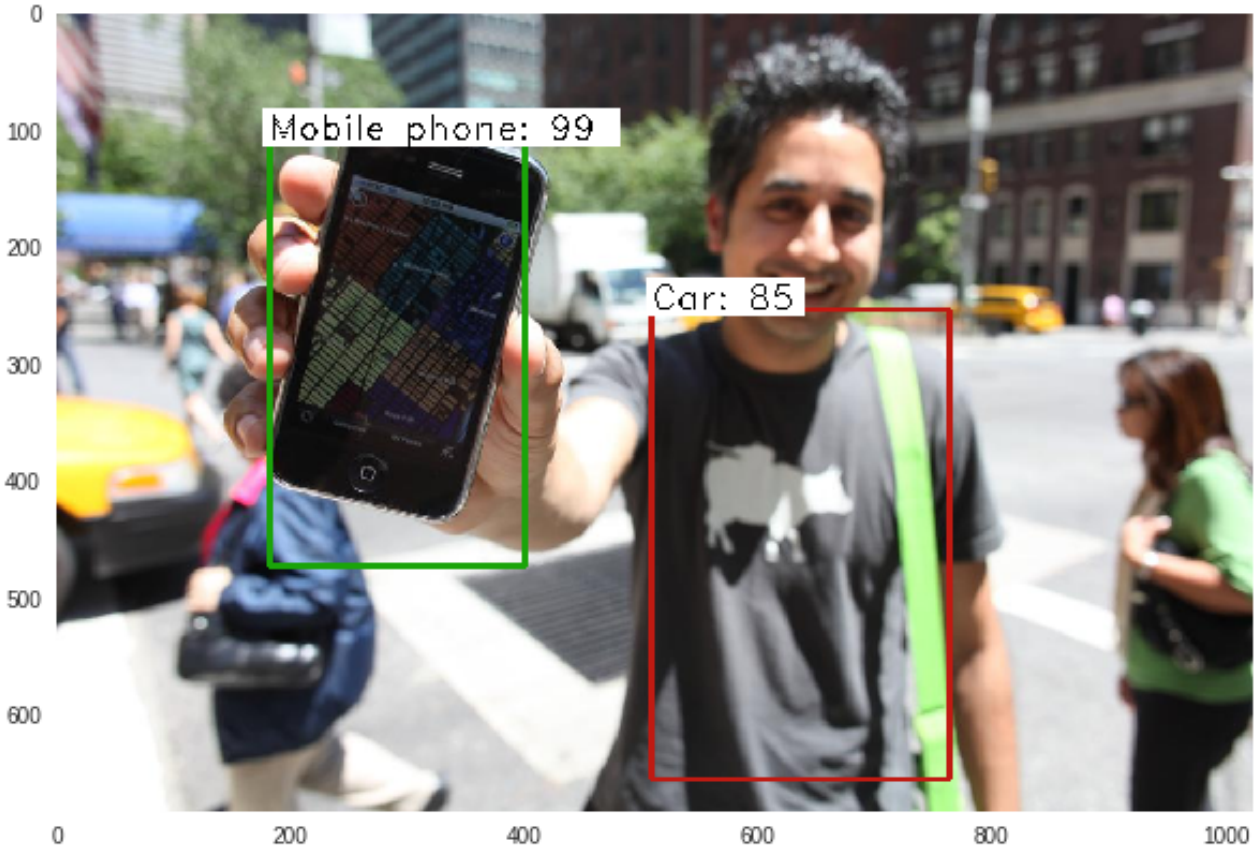
77cb02dc68f50bb6.jpg
Elapsed time = 1.2737236022949219
[('Car', 95.05159258842468), ('Car', 84.442800283432), ('Car', 80.111199617
38586), ('Car', 76.83292031288147), ('Car', 74.07441139221191)]

e4c680fc33abedf5.jpg
Elapsed time = 1.5652105808258057
[('Car', 99.73841309547424)]


```
2b80decf37e74a11.jpg
Elapsed time = 0.6664595603942871
[('Car', 80.89892268180847), ('Person', 93.7919557094574), ('Person', 75.90
610384941101)]
```




```
ad268ddd6919db46.jpg
Elapsed time = 0.9589686393737793
[('Mobile phone', 99.92089867591858), ('Car', 85.92617511749268)]
```


Mobile phone: 99
Car: 85
0
100
200
300
400
500
600
0
200
400
600
800
1000

Measure mAP

In [0]:

```
def get_map(pred, gt, f):
        T = {}
        P = {}
        fx, fy = f

        for bbox in gt:
                bbox['bbox_matched'] = False

        pred_probs = np.array([s['prob'] for s in pred])
        box_idx_sorted_by_prob = np.argsort(pred_probs)[::-1]

        for box_idx in box_idx_sorted_by_prob:
                pred_box = pred[box_idx]
                pred_class = pred_box['class']
                pred_x1 = pred_box['x1']
                pred_x2 = pred_box['x2']
                pred_y1 = pred_box['y1']
                pred_y2 = pred_box['y2']
                pred_prob = pred_box['prob']
                if pred_class not in P:
                        P[pred_class] = []
                        T[pred_class] = []
                P[pred_class].append(pred_prob)
                found_match = False

                for gt_box in gt:
                        gt_class = gt_box['class']
                        gt_x1 = gt_box['x1']/fx
                        gt_x2 = gt_box['x2']/fx
                        gt_y1 = gt_box['y1']/fy
                        gt_y2 = gt_box['y2']/fy
                        gt_seen = gt_box['bbox_matched']
                        if gt_class != pred_class:
                                continue
                        if gt_seen:
                                continue
                        iou_map = iou((pred_x1, pred_y1, pred_x2, pred_y2),
 (gt_x1, gt_y1, gt_x2, gt_y2))
                        if iou_map >= 0.5:
                                found_match = True
                                gt_box['bbox_matched'] = True
                                break
                        else:
                                continue

                T[pred_class].append(int(found_match))

        for gt_box in gt:
                if not gt_box['bbox_matched']:# and not gt_box['difficul
t']:
                        if gt_box['class'] not in P:
                                P[gt_box['class']] = []
                                T[gt_box['class']] = []

                        T[gt_box['class']].append(1)
                        P[gt_box['class']].append(0)
```

```
        #import pdb
        #pdb.set_trace()
        return T, P
```

In [0]:
```
def format_img_map(img, C):
        """Format image for mAP. Resize original image to C.im_size (300 in
 here)

        Args:
                img: cv2 image
                C: config

        Returns:
                img: Scaled and normalized image with expanding dimension
                fx: ratio for width scaling
                fy: ratio for height scaling
        """

        img_min_side = float(C.im_size)
        (height,width,_) = img.shape

        if width <= height:
                f = img_min_side/width
                new_height = int(f * height)
                new_width = int(img_min_side)
        else:
                f = img_min_side/height
                new_width = int(f * width)
                new_height = int(img_min_side)
        fx = width/float(new_width)
        fy = height/float(new_height)
        img = cv2.resize(img, (new_width, new_height), interpolation=cv2.IN
TER_CUBIC)
        # Change image channel from BGR to RGB
        img = img[:, :, (2, 1, 0)]
        img = img.astype(np.float32)
        img[:, :, 0] -= C.img_channel_mean[0]
        img[:, :, 1] -= C.img_channel_mean[1]
        img[:, :, 2] -= C.img_channel_mean[2]
        img /= C.img_scaling_factor
        # Change img shape from (height, width, channel) to (channel, heigh
t, width)
        img = np.transpose(img, (2, 0, 1))
        # Expand one dimension at axis 0
        # img shape becames (1, channel, height, width)
        img = np.expand_dims(img, axis=0)
        return img, fx, fy
```

In [0]:
```
print(class_mapping)
```

```
{0: 'Person', 1: 'Car', 2: 'Mobile phone', 3: 'bg'}
```

In [0]:
```
# This might takes a while to parser the data
test_imgs, _, _ = get_data(test_path)
```

```
Parsing annotation files
idx=1931
```

In [0]:

```
T = {}
P = {}
mAPs = []
for idx, img_data in enumerate(test_imgs):
    print('{}/{}'.format(idx,len(test_imgs)))
    st = time.time()
    filepath = img_data['filepath']

    img = cv2.imread(filepath)

    X, fx, fy = format_img_map(img, C)

    # Change X (img) shape from (1, channel, height, width) to (1, height,
 width, channel)
    X = np.transpose(X, (0, 2, 3, 1))

    # get the feature maps and output from the RPN
    [Y1, Y2, F] = model_rpn.predict(X)


    R = rpn_to_roi(Y1, Y2, C, K.image_dim_ordering(), overlap_thresh=0.7)

    # convert from (x1,y1,x2,y2) to (x,y,w,h)
    R[:, 2] -= R[:, 0]
    R[:, 3] -= R[:, 1]

    # apply the spatial pyramid pooling to the proposed regions
    bboxes = {}
    probs = {}

    for jk in range(R.shape[0] // C.num_rois + 1):
        ROIs = np.expand_dims(R[C.num_rois * jk:C.num_rois * (jk + 1), :],
axis=0)
        if ROIs.shape[1] == 0:
            break

        if jk == R.shape[0] // C.num_rois:
            # pad R
            curr_shape = ROIs.shape
            target_shape = (curr_shape[0], C.num_rois, curr_shape[2])
            ROIs_padded = np.zeros(target_shape).astype(ROIs.dtype)
            ROIs_padded[:, :curr_shape[1], :] = ROIs
            ROIs_padded[0, curr_shape[1]:, :] = ROIs[0, 0, :]
            ROIs = ROIs_padded

        [P_cls, P_regr] = model_classifier_only.predict([F, ROIs])

        # Calculate all classes' bboxes coordinates on resized image (300,
 400)
        # Drop 'bg' classes bboxes
        for ii in range(P_cls.shape[1]):

            # If class name is 'bg', continue
            if np.argmax(P_cls[0, ii, :]) == (P_cls.shape[2] - 1):
                continue

            # Get class name
```

```
            cls_name = class_mapping[np.argmax(P_cls[0, ii, :])]

            if cls_name not in bboxes:
                bboxes[cls_name] = []
                probs[cls_name] = []

            (x, y, w, h) = ROIs[0, ii, :]

            cls_num = np.argmax(P_cls[0, ii, :])
            try:
                (tx, ty, tw, th) = P_regr[0, ii, 4 * cls_num:4 * (cls_num + 1)]
                tx /= C.classifier_regr_std[0]
                ty /= C.classifier_regr_std[1]
                tw /= C.classifier_regr_std[2]
                th /= C.classifier_regr_std[3]
                x, y, w, h = roi_helpers.apply_regr(x, y, w, h, tx, ty, tw, th)
            except:
                pass
            bboxes[cls_name].append([16 * x, 16 * y, 16 * (x + w), 16 * (y + h)])
            probs[cls_name].append(np.max(P_cls[0, ii, :]))

    all_dets = []

    for key in bboxes:
        bbox = np.array(bboxes[key])

        # Apply non-max-suppression on final bboxes to get the output bounding boxe
        new_boxes, new_probs = non_max_suppression_fast(bbox, np.array(probs[key]), overlap_thresh=0.5)
        for jk in range(new_boxes.shape[0]):
            (x1, y1, x2, y2) = new_boxes[jk, :]
            det = {'x1': x1, 'x2': x2, 'y1': y1, 'y2': y2, 'class': key, 'prob': new_probs[jk]}
            all_dets.append(det)


    print('Elapsed time = {}'.format(time.time() - st))
    t, p = get_map(all_dets, img_data['bboxes'], (fx, fy))
    for key in t.keys():
        if key not in T:
            T[key] = []
            P[key] = []
        T[key].extend(t[key])
        P[key].extend(p[key])
    all_aps = []
    for key in T.keys():
        ap = average_precision_score(T[key], P[key])
        print('{} AP: {}'.format(key, ap))
        all_aps.append(ap)
    print('mAP = {}'.format(np.mean(np.array(all_aps))))
    mAPs.append(np.mean(np.array(all_aps)))
    #print(T)
    #print(P)
```

```python
print()
print('mean average precision:', np.mean(np.array(mAPs)))
```

```
0/599
Elapsed time = 1.9499390125274658
Mobile phone AP: 0.16666666666666666
Person AP: nan
mAP = nan
1/599

/usr/local/lib/python3.6/dist-packages/sklearn/metrics/ranking.py:444: Runt
imeWarning: invalid value encountered in true_divide
  recall = tps / tps[-1]
```

```
Elapsed time = 0.6297092437744141
Mobile phone AP: 0.16666666666666666
Person AP: 0.125
Car AP: nan
mAP = nan
2/599
Elapsed time = 0.6374988555908203
Mobile phone AP: 0.6
Person AP: 0.125
Car AP: nan
mAP = nan
3/599
Elapsed time = 0.625216007232666
Mobile phone AP: 0.6
Person AP: 0.125
Car AP: 1.0
mAP = 0.5750000000000001
4/599
Elapsed time = 0.6181929111480713
Mobile phone AP: 0.47619047619047616
Person AP: 0.1111111111111111
Car AP: 1.0
mAP = 0.5291005291005291
5/599
Elapsed time = 0.6407523155212402
Mobile phone AP: 0.47619047619047616
Person AP: 0.09375
Car AP: 1.0
mAP = 0.5233134920634921
6/599
Elapsed time = 1.4576971530914307
Mobile phone AP: 0.4583333333333333
Person AP: 0.10526315789473684
Car AP: 1.0
mAP = 0.5211988304093568
7/599
Elapsed time = 0.6355772018432617
Mobile phone AP: 0.45098039215686275
Person AP: 0.10526315789473684
Car AP: 0.43333333333333335
mAP = 0.329858961128311
8/599
Elapsed time = 1.5699424743652344
Mobile phone AP: 0.3863636363636363
Person AP: 0.08163265306122448
Car AP: 0.43333333333333335
mAP = 0.30044320758606474
9/599
Elapsed time = 0.6315431594848633
Mobile phone AP: 0.3863636363636363
Person AP: 0.10344827586206896
Car AP: 0.27142857142857146
mAP = 0.25374682788475894
10/599
Elapsed time = 0.6364405155181885
Mobile phone AP: 0.3863636363636363
Person AP: 0.09722222222222222
```

```
Car AP: 0.26497695852534564
mAP = 0.24952093903706807
11/599
Elapsed time = 0.6231091022491455
Mobile phone AP: 0.38636363636363635
Person AP: 0.11392405063291139
Car AP: 0.24789915966386555
mAP = 0.24939561555347112
12/599
Elapsed time = 0.6340842247009277
Mobile phone AP: 0.38636363636363635
Person AP: 0.12380952380952381
Car AP: 0.24285714285714288
mAP = 0.251010101010101
13/599
Elapsed time = 0.648099422454834
Mobile phone AP: 0.35384615384615387
Person AP: 0.10833333333333334
Car AP: 0.24285714285714288
mAP = 0.23501221001221004
14/599
Elapsed time = 0.6277978420257568
Mobile phone AP: 0.35384615384615387
Person AP: 0.10833333333333334
Car AP: 0.1865530303030303
mAP = 0.2162441724941725
15/599
Elapsed time = 0.8371407985687256
Mobile phone AP: 0.35384615384615387
Person AP: 0.10743801652892562
Car AP: 0.1919568822553897
mAP = 0.21774701754348977
16/599
Elapsed time = 0.624809980392456
Mobile phone AP: 0.35384615384615387
Person AP: 0.09923664122137404
Car AP: 0.17923076923076925
mAP = 0.21077118809943238
17/599
Elapsed time = 0.5801253318786621
Mobile phone AP: 0.35384615384615387
Person AP: 0.09701492537313433
Car AP: 0.17222305929623002
mAP = 0.20769471283850607
18/599
Elapsed time = 0.6218504905700684
Mobile phone AP: 0.23958333333333331
Person AP: 0.09701492537313433
Car AP: 0.17222305929623002
mAP = 0.16960710600089923
19/599
Elapsed time = 0.6318004131317139
Mobile phone AP: 0.23958333333333331
Person AP: 0.10429447852760736
Car AP: 0.17222305929623002
mAP = 0.17203362371905692
20/599
```

```
Elapsed time = 1.3984620571136475
Mobile phone AP: 0.26785714285714285
Person AP: 0.10429447852760736
Car AP: 0.17222305929623002
mAP = 0.18145822689366601
21/599
Elapsed time = 0.6239464282989502
Mobile phone AP: 0.31471631205673756
Person AP: 0.10429447852760736
Car AP: 0.17222305929623002
mAP = 0.19707794996019165
22/599
Elapsed time = 0.6305208206176758
Mobile phone AP: 0.30833333333333335
Person AP: 0.10404624277456648
Car AP: 0.17222305929623002
mAP = 0.19486754513470994
23/599
Elapsed time = 0.636115550994873
Mobile phone AP: 0.3063725490196078
Person AP: 0.10404624277456648
Car AP: 0.16589138846122684
mAP = 0.19210339341846705
24/599
Elapsed time = 1.7665777206420898
Mobile phone AP: 0.3063725490196078
Person AP: 0.09947643979057591
Car AP: 0.16589138846122684
mAP = 0.19058012575713687
25/599
Elapsed time = 0.6252090930938721
Mobile phone AP: 0.3063725490196078
Person AP: 0.0979381443298969
Car AP: 0.16085586408476432
mAP = 0.18838885247808967
26/599
Elapsed time = 0.6412127017974854
Mobile phone AP: 0.28042328042328046
Person AP: 0.0979381443298969
Car AP: 0.16085586408476432
mAP = 0.17973909627931386
27/599
Elapsed time = 0.8501982688903809
Mobile phone AP: 0.27467581998474444
Person AP: 0.10344827586206896
Car AP: 0.16085586408476432
mAP = 0.1796599866438592
28/599
Elapsed time = 1.3496074676513672
Mobile phone AP: 0.27467581998474444
Person AP: 0.09859154929577464
Car AP: 0.18675052011095702
mAP = 0.1866726297971587
29/599
Elapsed time = 1.8816485404968262
Mobile phone AP: 0.27467581998474444
Person AP: 0.09859154929577464
```

```
Car AP: 0.18703908898260427
mAP = 0.1867688194210411
30/599
Elapsed time = 0.623687744140625
Mobile phone AP: 0.27467581998474444
Person AP: 0.1013215859030837
Car AP: 0.18703908898260427
mAP = 0.18767883162347745
31/599
Elapsed time = 0.6284241676330566
Mobile phone AP: 0.27467581998474444
Person AP: 0.1013215859030837
Car AP: 0.19241740715424926
mAP = 0.18947160434735913
32/599
Elapsed time = 0.6280961036682129
Mobile phone AP: 0.27467581998474444
Person AP: 0.09647940766264665
Car AP: 0.1872081048286679
mAP = 0.186121110825353
33/599
Elapsed time = 0.6306867599487305
Mobile phone AP: 0.27065390749601276
Person AP: 0.13536843451015523
Car AP: 0.18463539277253416
mAP = 0.1968859115929007
34/599
Elapsed time = 1.3802618980407715
Mobile phone AP: 0.2656641604010025
Person AP: 0.13300208563366459
Car AP: 0.18463539277253416
mAP = 0.1944338796024004
35/599
Elapsed time = 0.6391420364379883
Mobile phone AP: 0.2656641604010025
Person AP: 0.13618521510135434
Car AP: 0.18088062152292722
mAP = 0.19424333234176136
36/599
Elapsed time = 0.6301233768463135
Mobile phone AP: 0.2656641604010025
Person AP: 0.13490446335273923
Car AP: 0.17736686390532544
mAP = 0.19264516255302236
37/599
Elapsed time = 1.514383316040039
Mobile phone AP: 0.2656641604010025
Person AP: 0.13448322510822514
Car AP: 0.18423680306762205
mAP = 0.19479472952561658
38/599
Elapsed time = 0.6371490955352783
Mobile phone AP: 0.24437389770723103
Person AP: 0.13448322510822514
Car AP: 0.18423680306762205
mAP = 0.1876979752943594
39/599
```

```
Elapsed time = 1.1731345653533936
Mobile phone AP: 0.24046662841843566
Person AP: 0.13590778772453338
Car AP: 0.18423680306762205
mAP = 0.18687040640353036
40/599
Elapsed time = 0.631439208984375
Mobile phone AP: 0.23305860805860806
Person AP: 0.1334357398873528
Car AP: 0.18423680306762205
mAP = 0.18357705033786098
41/599
Elapsed time = 0.8485889434814453
Mobile phone AP: 0.22859477124183009
Person AP: 0.1334357398873528
Car AP: 0.18423680306762205
mAP = 0.18208910473226833
42/599
Elapsed time = 0.629176139831543
Mobile phone AP: 0.22859477124183009
Person AP: 0.17772324179884444
Car AP: 0.1738554977948191
mAP = 0.19339117027849786
43/599
Elapsed time = 0.6207375526428223
Mobile phone AP: 0.22859477124183009
Person AP: 0.17962626111797383
Car AP: 0.17190229649789343
mAP = 0.19337444295256578
44/599
Elapsed time = 0.6184797286987305
Mobile phone AP: 0.2230925834836312
Person AP: 0.1799028472532885
Car AP: 0.17190229649789343
mAP = 0.19163257574493772
45/599
Elapsed time = 0.6341612339019775
Mobile phone AP: 0.2230925834836312
Person AP: 0.17408724853577795
Car AP: 0.17190229649789343
mAP = 0.18969404283910085
46/599
Elapsed time = 0.6253108978271484
Mobile phone AP: 0.2230925834836312
Person AP: 0.17141527873079243
Car AP: 0.17587212895620796
mAP = 0.19012666372354384
47/599
Elapsed time = 0.5836925506591797
Mobile phone AP: 0.22525356778797145
Person AP: 0.16387052530592655
Car AP: 0.17587212895620796
mAP = 0.18833207401670196
48/599
Elapsed time = 1.4797735214233398
Mobile phone AP: 0.22038225172074727
Person AP: 0.16387052530592655
```

```
Car AP: 0.17927220724976767
mAP = 0.1878416614254805
49/599
Elapsed time = 0.6266448497772217
Mobile phone AP: 0.22038225172074727
Person AP: 0.161958438770033
Car AP: 0.17927220724976767
mAP = 0.1872042992468493
50/599
Elapsed time = 1.5676143169403076
Mobile phone AP: 0.2159010524499655
Person AP: 0.15893101464979278
Car AP: 0.17927220724976767
mAP = 0.1847014247831753
51/599
Elapsed time = 1.5210213661193848
Mobile phone AP: 0.2159010524499655
Person AP: 0.15615820045967105
Car AP: 0.17927220724976767
mAP = 0.18377715338646805
52/599
Elapsed time = 1.1908555030822754
Mobile phone AP: 0.2159010524499655
Person AP: 0.15615820045967105
Car AP: 0.17416579512986358
mAP = 0.1820750160131667
53/599
Elapsed time = 0.8283178806304932
Mobile phone AP: 0.21717771820348156
Person AP: 0.15441133978644325
Car AP: 0.17416579512986358
mAP = 0.18191828437326282
54/599
Elapsed time = 0.5857057571411133
Mobile phone AP: 0.22217801795619604
Person AP: 0.15270362396068396
Car AP: 0.17416579512986358
mAP = 0.1830158123489145
55/599
Elapsed time = 0.6079537868499756
Mobile phone AP: 0.217077620662091
Person AP: 0.1512348533244101
Car AP: 0.17416579512986358
mAP = 0.1808260897054549
56/599
Elapsed time = 0.6133551597595215
Mobile phone AP: 0.217077620662091
Person AP: 0.1512348533244101
Car AP: 0.17590727161271952
mAP = 0.18140658186640687
57/599
Elapsed time = 1.772890329360962
Mobile phone AP: 0.217077620662091
Person AP: 0.15058442679080233
Car AP: 0.1710102150143296
mAP = 0.17955742082240764
58/599
```

```
Elapsed time = 1.9392690658569336
Mobile phone AP: 0.21546092796092794
Person AP: 0.14497196685346847
Car AP: 0.1710102150143296
mAP = 0.177147703276242
59/599
Elapsed time = 1.0223965644836426
Mobile phone AP: 0.21546092796092794
Person AP: 0.1437678295515824
Car AP: 0.1710102150143296
mAP = 0.1767463241756133
60/599
Elapsed time = 0.6099638938903809
Mobile phone AP: 0.21125415000155617
Person AP: 0.1437678295515824
Car AP: 0.1710102150143296
mAP = 0.17534406485582274
61/599
Elapsed time = 1.3625209331512451
Mobile phone AP: 0.21125415000155617
Person AP: 0.14071784446123248
Car AP: 0.1710102150143296
mAP = 0.1743274031590394
62/599
Elapsed time = 0.618798017501831
Mobile phone AP: 0.21035115722549835
Person AP: 0.14071784446123248
Car AP: 0.16975634570743267
mAP = 0.17360844913138784
63/599
Elapsed time = 0.6461169719696045
Mobile phone AP: 0.20517172696282726
Person AP: 0.14406719549487593
Car AP: 0.16975634570743267
mAP = 0.17299842272171195
64/599
Elapsed time = 0.6264939308166504
Mobile phone AP: 0.18624293265357705
Person AP: 0.14324237703788578
Car AP: 0.16911664999512285
mAP = 0.1662006532288619
65/599
Elapsed time = 0.6309032440185547
Mobile phone AP: 0.18624293265357705
Person AP: 0.14935766872421952
Car AP: 0.16824262830786563
mAP = 0.16794774322855408
66/599
Elapsed time = 1.5270755290985107
Mobile phone AP: 0.18624293265357705
Person AP: 0.14973378836770293
Car AP: 0.16824262830786563
mAP = 0.16807311644304854
67/599
Elapsed time = 0.6226944923400879
Mobile phone AP: 0.19568961179190797
Person AP: 0.14973378836770293
```

```
Car AP: 0.16428471747802337
mAP = 0.16990270587921144
68/599
Elapsed time = 1.1006176471710205
Mobile phone AP: 0.19568961179190797
Person AP: 0.14973378836770293
Car AP: 0.16424286781151307
mAP = 0.16988875599037467
69/599
Elapsed time = 0.660290002822876
Mobile phone AP: 0.19568961179190797
Person AP: 0.15611805720982821
Car AP: 0.16424286781151307
mAP = 0.17201684560441644
70/599
Elapsed time = 1.2703585624694824
Mobile phone AP: 0.19568961179190797
Person AP: 0.1544751732580928
Car AP: 0.19017179119631897
mAP = 0.18011219208210658
71/599
Elapsed time = 0.6373848915100098
Mobile phone AP: 0.19314565788469867
Person AP: 0.1544751732580928
Car AP: 0.19017179119631897
mAP = 0.17926420744637014
72/599
Elapsed time = 0.6259360313415527
Mobile phone AP: 0.19314565788469867
Person AP: 0.15525807797505717
Car AP: 0.18957520367692987
mAP = 0.17932631317889522
73/599
Elapsed time = 0.6306297779083252
Mobile phone AP: 0.19314565788469867
Person AP: 0.15432035918640621
Car AP: 0.18957520367692987
mAP = 0.17901374024934494
74/599
Elapsed time = 0.6422438621520996
Mobile phone AP: 0.19314565788469867
Person AP: 0.15949375395369045
Car AP: 0.1871513365779441
mAP = 0.1799302494721111
75/599
Elapsed time = 1.9006156921386719
Mobile phone AP: 0.18688515395832467
Person AP: 0.15872694863268255
Car AP: 0.18967994347647882
mAP = 0.17843068202249535
76/599
Elapsed time = 0.6430211067199707
Mobile phone AP: 0.18922847995867764
Person AP: 0.15746490062033533
Car AP: 0.18967994347647882
mAP = 0.17879110801849726
77/599
```

```
Elapsed time = 0.6330370903015137
Mobile phone AP: 0.18922847995867764
Person AP: 0.15427644351602238
Car AP: 0.202547576743219
mAP = 0.1820175000726397
78/599
Elapsed time = 0.6239159107208252
Mobile phone AP: 0.18922847995867764
Person AP: 0.15427644351602238
Car AP: 0.19224558043050524
mAP = 0.1785835013017351
79/599
Elapsed time = 0.6404385566711426
Mobile phone AP: 0.18922847995867764
Person AP: 0.15707937139810227
Car AP: 0.18791102698366255
mAP = 0.17807295944681414
80/599
Elapsed time = 0.6343429088592529
Mobile phone AP: 0.1842820648343904
Person AP: 0.15707937139810227
Car AP: 0.18791102698366255
mAP = 0.1764241544053851
81/599
Elapsed time = 0.650383710861206
Mobile phone AP: 0.1842820648343904
Person AP: 0.15707937139810227
Car AP: 0.19204158963706472
mAP = 0.17780100862318582
82/599
Elapsed time = 0.6382215023040771
Mobile phone AP: 0.18239129526699743
Person AP: 0.15707937139810227
Car AP: 0.19204158963706472
mAP = 0.1771707521007215
83/599
Elapsed time = 0.6209616661071777
Mobile phone AP: 0.18239129526699743
Person AP: 0.16048374927929923
Car AP: 0.1910246651073923
mAP = 0.17796656988456303
84/599
Elapsed time = 0.6481554508209229
Mobile phone AP: 0.18239129526699743
Person AP: 0.17275447229419905
Car AP: 0.1910246651073923
mAP = 0.1820568108895296
85/599
Elapsed time = 1.4909043312072754
Mobile phone AP: 0.18239129526699743
Person AP: 0.16917682549872032
Car AP: 0.1910246651073923
mAP = 0.18086426195770336
86/599
Elapsed time = 1.440119743347168
Mobile phone AP: 0.17964017610000393
Person AP: 0.16781532310119576
```

```
Car AP: 0.1910246651073923
mAP = 0.17949338810286397
87/599
Elapsed time = 0.6797595024108887
Mobile phone AP: 0.18035410354732873
Person AP: 0.16781532310119576
Car AP: 0.1910246651073923
mAP = 0.17973136391863895
88/599
Elapsed time = 0.6330955028533936
Mobile phone AP: 0.17968463893227185
Person AP: 0.16779677848543348
Car AP: 0.19000572055942844
mAP = 0.17916237932571125
89/599
Elapsed time = 0.6390426158905029
Mobile phone AP: 0.17968463893227185
Person AP: 0.16736153444377283
Car AP: 0.19166332622564045
mAP = 0.17956983320056172
90/599
Elapsed time = 1.4849824905395508
Mobile phone AP: 0.1790209812033967
Person AP: 0.16692854252855874
Car AP: 0.19169371325624515
mAP = 0.1792144123294002
91/599
Elapsed time = 0.6368308067321777
Mobile phone AP: 0.18064573159504982
Person AP: 0.16628568410183361
Car AP: 0.19169371325624515
mAP = 0.17954170965104288
92/599
Elapsed time = 0.6407597064971924
Mobile phone AP: 0.18064573159504982
Person AP: 0.16565010145927683
Car AP: 0.19038479852863777
mAP = 0.17889354386098813
93/599
Elapsed time = 0.6282062530517578
Mobile phone AP: 0.17933727031578822
Person AP: 0.16481017819698826
Car AP: 0.19061557400547596
mAP = 0.1782543408394175
94/599
Elapsed time = 1.8086888790130615
Mobile phone AP: 0.17950061780647839
Person AP: 0.16481017819698826
Car AP: 0.19061557400547596
mAP = 0.17830879000298086
95/599
Elapsed time = 0.6400961875915527
Mobile phone AP: 0.17950061780647839
Person AP: 0.16401804554684363
Car AP: 0.18969038235863483
mAP = 0.17773634857065232
96/599
```

```
Elapsed time = 0.6286892890930176
Mobile phone AP: 0.17950061780647839
Person AP: 0.16385572415279087
Car AP: 0.18831062459264844
mAP = 0.17722232218397257
97/599
Elapsed time = 1.3508436679840088
Mobile phone AP: 0.17909457100633572
Person AP: 0.16385572415279087
Car AP: 0.18651599438261263
mAP = 0.17648876318057974
98/599
Elapsed time = 0.6431310176849365
Mobile phone AP: 0.17909457100633572
Person AP: 0.16071238942201185
Car AP: 0.18870557786869682
mAP = 0.1761708460990148
99/599
Elapsed time = 0.6312546730041504
Mobile phone AP: 0.17641642554432663
Person AP: 0.16071238942201185
Car AP: 0.18870557786869682
mAP = 0.17527813094501177
100/599
Elapsed time = 0.6324281692504883
Mobile phone AP: 0.17641642554432663
Person AP: 0.16056336723701195
Car AP: 0.18826731836504848
mAP = 0.17508237038212904
101/599
Elapsed time = 0.6369788646697998
Mobile phone AP: 0.17641642554432663
Person AP: 0.15910363978413647
Car AP: 0.1878366100244397
mAP = 0.17445222511763428
102/599
Elapsed time = 0.6325197219848633
Mobile phone AP: 0.1758246400241564
Person AP: 0.15910363978413647
Car AP: 0.1877219294327896
mAP = 0.17421673641369417
103/599
Elapsed time = 0.5964012145996094
Mobile phone AP: 0.1758246400241564
Person AP: 0.15873235685345857
Car AP: 0.18934893211368503
mAP = 0.17463530966376664
104/599
Elapsed time = 0.6340019702911377
Mobile phone AP: 0.17563514815226078
Person AP: 0.1587871620980544
Car AP: 0.18934893211368503
mAP = 0.1745904141213334
105/599
Elapsed time = 0.6230204105377197
Mobile phone AP: 0.172368589012016 2
Person AP: 0.1587871620980544
```

```
Car AP: 0.18934893211368503
mAP = 0.17350156107458523
106/599
Elapsed time = 0.6369125843048096
Mobile phone AP: 0.1723685890120162
Person AP: 0.15660949430828366
Car AP: 0.18993331683893952
mAP = 0.17297046671974647
107/599
Elapsed time = 1.3544726371765137
Mobile phone AP: 0.1723685890120162
Person AP: 0.15589679598388453
Car AP: 0.19100732268335513
mAP = 0.17309090255975193
108/599
Elapsed time = 1.029439926147461
Mobile phone AP: 0.17099893230073526
Person AP: 0.15519055500534096
Car AP: 0.19100732268335513
mAP = 0.17239893666314376
109/599
Elapsed time = 0.6304776668548584
Mobile phone AP: 0.1755808686826959
Person AP: 0.1537969688724256
Car AP: 0.19100732268335513
mAP = 0.1734617200794922
110/599
Elapsed time = 0.6332917213439941
Mobile phone AP: 0.1755808686826959
Person AP: 0.15388278474202116
Car AP: 0.19061926472476656
mAP = 0.17336097271649453
111/599
Elapsed time = 0.6326241493225098
Mobile phone AP: 0.1755808686826959
Person AP: 0.15286030238338114
Car AP: 0.2033456136736674
mAP = 0.17726226157991484
112/599
Elapsed time = 0.6267335414886475
Mobile phone AP: 0.1755808686826959
Person AP: 0.1518788546480227
Car AP: 0.2033456136736674
mAP = 0.17693511233479534
113/599
Elapsed time = 0.6177537441253662
Mobile phone AP: 0.17504105761656902
Person AP: 0.1518788546480227
Car AP: 0.2002474266772712
mAP = 0.1757224463139543
114/599
Elapsed time = 0.6308410167694092
Mobile phone AP: 0.17397005274826266
Person AP: 0.15052285851988578
Car AP: 0.2002474266772712
mAP = 0.17491344598180655
115/599
```

```
Elapsed time = 0.6238949298858643
Mobile phone AP: 0.17397005274826266
Person AP: 0.14934910545909708
Car AP: 0.2002474266772712
mAP = 0.17452219496154364
116/599
Elapsed time = 1.2405524253845215
Mobile phone AP: 0.17397005274826266
Person AP: 0.150825554916237
Car AP: 0.19495798285088856
mAP = 0.17325119683846277
117/599
Elapsed time = 0.6250267028808594
Mobile phone AP: 0.1734388039777813
Person AP: 0.15092360852667525
Car AP: 0.19495798285088856
mAP = 0.1731067984517817
118/599
Elapsed time = 0.5899343490600586
Mobile phone AP: 0.1734388039777813
Person AP: 0.15029348422342037
Car AP: 0.1923257942315435
mAP = 0.17201936081091507
119/599
Elapsed time = 0.6366286277770996
Mobile phone AP: 0.1734388039777813
Person AP: 0.15029348422342037
Car AP: 0.19055918995881976
mAP = 0.17143049272000713
120/599
Elapsed time = 0.6189961433410645
Mobile phone AP: 0.1734388039777813
Person AP: 0.15029348422342037
Car AP: 0.19451501743096633
mAP = 0.17274910187738934
121/599
Elapsed time = 1.395289421081543
Mobile phone AP: 0.1734388039777813
Person AP: 0.14912019588204434
Car AP: 0.19451501743096633
mAP = 0.1723580057635973
122/599
Elapsed time = 1.1379642486572266
Mobile phone AP: 0.1734388039777813
Person AP: 0.14850766584865838
Car AP: 0.20109292379465296
mAP = 0.1743464645403642
123/599
Elapsed time = 0.6344521045684814
Mobile phone AP: 0.1734388039777813
Person AP: 0.14908021748902053
Car AP: 0.20109292379465296
mAP = 0.17453731508715162
124/599
Elapsed time = 1.7985219955444336
Mobile phone AP: 0.1734388039777813
Person AP: 0.15628627599094047
```

```
Car AP: 0.20074974610120305
mAP = 0.17682494202330826
125/599
Elapsed time = 0.6388740539550781
Mobile phone AP: 0.1734388039777813
Person AP: 0.1554263233808 2013
Car AP: 0.20074974610120305
mAP = 0.17653829115326816
126/599
Elapsed time = 0.625328779220581
Mobile phone AP: 0.17377910224437043
Person AP: 0.15542632338082013
Car AP: 0.20074974610120305
mAP = 0.17665172390879788
127/599
Elapsed time = 0.6348025798797607
Mobile phone AP: 0.17221759451602675
Person AP: 0.15549136936084265
Car AP: 0.20275499702660177
mAP = 0.17682132030115705
128/599
Elapsed time = 0.6444220542907715
Mobile phone AP: 0.17221759451602675
Person AP: 0.15370173697763082
Car AP: 0.20404268732050312
mAP = 0.17665400627138692
129/599
Elapsed time = 1.7960522174835205
Mobile phone AP: 0.17221759451602675
Person AP: 0.15296948593864357
Car AP: 0.20536128803876064
mAP = 0.17684945616447698
130/599
Elapsed time = 1.2769114971160889
Mobile phone AP: 0.17068501916563567
Person AP: 0.1526785024620991
Car AP: 0.20599067944596489
mAP = 0.17645140035789986
131/599
Elapsed time = 0.8172857761383057
Mobile phone AP: 0.17068501916563567
Person AP: 0.15080523494487222
Car AP: 0.20393029963994252
mAP = 0.17514018458348346
132/599
Elapsed time = 0.6140670776367188
Mobile phone AP: 0.170183346924833
Person AP: 0.1516018764112221
Car AP: 0.20198087851944826
mAP = 0.1745887006185011
133/599
Elapsed time = 0.6266067028045654
Mobile phone AP: 0.16869115761089776
Person AP: 0.15196958437259817
Car AP: 0.20198087851944826
mAP = 0.1742138735009814
134/599
```

```
Elapsed time = 0.6258466243743896
Mobile phone AP: 0.16869115761089776
Person AP: 0.15196958437259817
Car AP: 0.20130477760552112
mAP = 0.17398850652967235
135/599
Elapsed time = 0.6327071189880371
Mobile phone AP: 0.16946703070146082
Person AP: 0.15196958437259817
Car AP: 0.20130477760552112
mAP = 0.17424713089319335
136/599
Elapsed time = 0.6304538249969482
Mobile phone AP: 0.1658670948001743
Person AP: 0.15196958437259817
Car AP: 0.20130477760552112
mAP = 0.1730471522594312
137/599
Elapsed time = 0.6200418472290039
Mobile phone AP: 0.16679993012468455
Person AP: 0.15305360534151674
Car AP: 0.20130477760552112
mAP = 0.17371943769057416
138/599
Elapsed time = 0.6245882511138916
Mobile phone AP: 0.16793082507889817
Person AP: 0.15220644203278264
Car AP: 0.20130477760552112
mAP = 0.17381401490573398
139/599
Elapsed time = 0.6154186725616455
Mobile phone AP: 0.166630650154505
Person AP: 0.15178781062153554
Car AP: 0.20157364887316848
mAP = 0.173330703216403
140/599
Elapsed time = 0.6273915767669678
Mobile phone AP: 0.166630650154505
Person AP: 0.14999856496851274
Car AP: 0.2051417731793317
mAP = 0.17392366276744983
141/599
Elapsed time = 1.5106732845306396
Mobile phone AP: 0.16525971296524938
Person AP: 0.14999856496851274
Car AP: 0.2051417731793317
mAP = 0.1734666837043646
142/599
Elapsed time = 0.6256675720214844
Mobile phone AP: 0.16525971296524938
Person AP: 0.14905316865649648
Car AP: 0.20508849056002157
mAP = 0.1731337907272558
143/599
Elapsed time = 0.6453239917755127
Mobile phone AP: 0.16483633379542567
Person AP: 0.14817934575825403
```

```
Car AP: 0.2044663563232603
mAP = 0.17249401195897998
144/599
Elapsed time = 0.6351580619812012
Mobile phone AP: 0.16483633379542567
Person AP: 0.14726736239874347
Car AP: 0.20279177575479682
mAP = 0.17163182398298862
145/599
Elapsed time = 0.6344945430755615
Mobile phone AP: 0.16483633379542567
Person AP: 0.14595962062902612
Car AP: 0.20279177575479682
mAP = 0.17119591005974955
146/599
Elapsed time = 0.6653459072113037
Mobile phone AP: 0.16483633379542567
Person AP: 0.14595962062902612
Car AP: 0.20451038136293087
mAP = 0.1717687785957942
147/599
Elapsed time = 0.6235880851745605
Mobile phone AP: 0.16445818213476227
Person AP: 0.14595962062902612
Car AP: 0.20451038136293087
mAP = 0.17164272804223976
148/599
Elapsed time = 0.6225917339324951
Mobile phone AP: 0.164186729394846
Person AP: 0.14595962062902612
Car AP: 0.20451038136293087
mAP = 0.171552243795601
149/599
Elapsed time = 0.638146162033081
Mobile phone AP: 0.164186729394846
Person AP: 0.14518608119180232
Car AP: 0.20422207677134763
mAP = 0.17119829578599866
150/599
Elapsed time = 0.613328218460083
Mobile phone AP: 0.16450397620948554
Person AP: 0.14481392226296427
Car AP: 0.20422207677134763
mAP = 0.1711799917479325
151/599
Elapsed time = 1.3622653484344482
Mobile phone AP: 0.16450397620948554
Person AP: 0.14541956616032203
Car AP: 0.2033730860354866
mAP = 0.17109887613509808
152/599
Elapsed time = 2.1739001274108887
Mobile phone AP: 0.16514677209604617
Person AP: 0.14541956616032203
Car AP: 0.2033730860354866
mAP = 0.17131314143061827
153/599
```

```
Elapsed time = 0.6160740852355957
Mobile phone AP: 0.1643541184121743
Person AP: 0.1449250903063388
Car AP: 0.20636894691239088
mAP = 0.17188271854363468
154/599
Elapsed time = 0.6707422733306885
Mobile phone AP: 0.1643541184121743
Person AP: 0.14702421943451943
Car AP: 0.20495973435372525
mAP = 0.172112690733473
155/599
Elapsed time = 0.6290054321289062
Mobile phone AP: 0.1643541184121743
Person AP: 0.14702421943451943
Car AP: 0.20425855149583416
mAP = 0.17187896311417594
156/599
Elapsed time = 1.026209831237793
Mobile phone AP: 0.16589193005469371
Person AP: 0.14702421943451943
Car AP: 0.20425855149583416
mAP = 0.1723915669950158
157/599
Elapsed time = 0.618638277053833
Mobile phone AP: 0.16589193005469371
Person AP: 0.14617196755980685
Car AP: 0.20553835440562782
mAP = 0.17253408400670947
158/599
Elapsed time = 0.6271841526031494
Mobile phone AP: 0.16589193005469371
Person AP: 0.14612916625246147
Car AP: 0.20553835440562782
mAP = 0.172519816904261
159/599
Elapsed time = 0.6327893733978271
Mobile phone AP: 0.16507919357722453
Person AP: 0.14612916625246147
Car AP: 0.20553835440562782
mAP = 0.1722489047451046
160/599
Elapsed time = 1.2590818405151367
Mobile phone AP: 0.1636864288350334
Person AP: 0.14612916625246147
Car AP: 0.20438466456847687
mAP = 0.1714000865519906
161/599
Elapsed time = 0.6352620124816895
Mobile phone AP: 0.1636864288350334
Person AP: 0.1463077955195372
Car AP: 0.20438466456847687
mAP = 0.1714596296410158
162/599
Elapsed time = 0.6448037624359131
Mobile phone AP: 0.16440999067912287
Person AP: 0.14673473014662422
```

```
Car AP: 0.20411720929082008
mAP = 0.1717539767055224
163/599
Elapsed time = 1.8047339916229248
Mobile phone AP: 0.16440999067912287
Person AP: 0.14710291290972402
Car AP: 0.20411720929082008
mAP = 0.17187670429322233
164/599
Elapsed time = 0.6281242370605469
Mobile phone AP: 0.16465106530880882
Person AP: 0.14652542116098993
Car AP: 0.20411720929082008
mAP = 0.17176456525353964
165/599
Elapsed time = 0.8202886581420898
Mobile phone AP: 0.1642805750091529
Person AP: 0.14641096751427699
Car AP: 0.20717429754119554
mAP = 0.1726219466882085
166/599
Elapsed time = 1.8345141410827637
Mobile phone AP: 0.16404963201921802
Person AP: 0.14606867813204688
Car AP: 0.20717429754119554
mAP = 0.17243086923082016
167/599
Elapsed time = 1.2725458145141602
Mobile phone AP: 0.16504507326209444
Person AP: 0.1458413740014256
Car AP: 0.20717429754119554
mAP = 0.1726869149349052
168/599
Elapsed time = 0.6415565013885498
Mobile phone AP: 0.1629150669401519
Person AP: 0.1458413740014256
Car AP: 0.20531350851426744
mAP = 0.17135664981861498
169/599
Elapsed time = 0.6311070919036865
Mobile phone AP: 0.1629150669401519
Person AP: 0.14438108383591736
Car AP: 0.21179427901779085
mAP = 0.17303014326462005
170/599
Elapsed time = 0.637265682220459
Mobile phone AP: 0.16331784034013228
Person AP: 0.14438108383591736
Car AP: 0.21098233998387986
mAP = 0.17289375471997648
171/599
Elapsed time = 1.490403652191162
Mobile phone AP: 0.16331784034013228
Person AP: 0.14522412598541917
Car AP: 0.21098233998387986
mAP = 0.17317476876981042
172/599
```

```
Elapsed time = 0.6275155544281006
Mobile phone AP: 0.16193069717184347
Person AP: 0.14522412598541917
Car AP: 0.21012178858765534
mAP = 0.17242553724830598
173/599
Elapsed time = 0.6384525299072266
Mobile phone AP: 0.16401924805567564
Person AP: 0.14489440262580955
Car AP: 0.21012178858765534
mAP = 0.1730118130897135
174/599
Elapsed time = 0.6232569217681885
Mobile phone AP: 0.16401924805567564
Person AP: 0.14596577616847606
Car AP: 0.20959628686366794
mAP = 0.17319377036260653
175/599
Elapsed time = 0.6484320163726807
Mobile phone AP: 0.16401924805567564
Person AP: 0.15485064987353936
Car AP: 0.20907577982298417
mAP = 0.17598189258406638
176/599
Elapsed time = 0.6133208274841309
Mobile phone AP: 0.1632877034761758
Person AP: 0.15485064987353936
Car AP: 0.20907577982298417
mAP = 0.1757380443908998
177/599
Elapsed time = 0.6282830238342285
Mobile phone AP: 0.1632877034761758
Person AP: 0.15445319400631843
Car AP: 0.20907577982298417
mAP = 0.17560555910182615
178/599
Elapsed time = 0.6519532203674316
Mobile phone AP: 0.1632877034761758
Person AP: 0.16374273742687157
Car AP: 0.20856032280674394
mAP = 0.17853025456993044
179/599
Elapsed time = 0.6160964965820312
Mobile phone AP: 0.1632877034761758
Person AP: 0.16318468977844197
Car AP: 0.2116710188583571
mAP = 0.17938113737099162
180/599
Elapsed time = 1.2753472328186035
Mobile phone AP: 0.1617737594466314
Person AP: 0.16318468977844197
Car AP: 0.2116710188583571
mAP = 0.1788764893611435
181/599
Elapsed time = 0.6599199771881104
Mobile phone AP: 0.1617737594466314
Person AP: 0.16318468977844197
```

```
Car AP: 0.21022245911917134
mAP = 0.17839363611474823
182/599
Elapsed time = 0.6348116397857666
Mobile phone AP: 0.16154510098704644
Person AP: 0.16318468977844197
Car AP: 0.21022245911917134
mAP = 0.1783174166282199
183/599
Elapsed time = 0.8534741401672363
Mobile phone AP: 0.16950108165868308
Person AP: 0.16455621856147723
Car AP: 0.21022245911917134
mAP = 0.1814265864464439
184/599
Elapsed time = 0.624058723449707
Mobile phone AP: 0.16950108165868308
Person AP: 0.16373390617751318
Car AP: 0.2099675104418549
mAP = 0.18106749942601708
185/599
Elapsed time = 0.6195166110992432
Mobile phone AP: 0.16950108165868308
Person AP: 0.16373390617751318
Car AP: 0.21009469967520422
mAP = 0.18110989583713347
186/599
Elapsed time = 0.9232707023620605
Mobile phone AP: 0.16950108165868308
Person AP: 0.16286942760268228
Car AP: 0.20937116302807304
mAP = 0.1805805574298128
187/599
Elapsed time = 0.5811827182769775
Mobile phone AP: 0.16959831100692482
Person AP: 0.16286942760268228
Car AP: 0.20810932534495624
mAP = 0.18019235465152109
188/599
Elapsed time = 0.6330814361572266
Mobile phone AP: 0.16959831100692482
Person AP: 0.16114377565890292
Car AP: 0.20781821654423124
mAP = 0.17952010107001967
189/599
Elapsed time = 0.6252045631408691
Mobile phone AP: 0.16959831100692482
Person AP: 0.15965796370040683
Car AP: 0.20777104453182088
mAP = 0.17900910641305087
190/599
Elapsed time = 0.6117432117462158
Mobile phone AP: 0.16737987104749844
Person AP: 0.15965796370040683
Car AP: 0.20777104453182088
mAP = 0.17826962642657537
191/599
```

```
Elapsed time = 0.6307814121246338
Mobile phone AP: 0.16737987104749844
Person AP: 0.16105649774900005
Car AP: 0.20777104453182088
mAP = 0.17873580444277315
192/599
Elapsed time = 0.6157083511352539
Mobile phone AP: 0.16737987104749844
Person AP: 0.16055814493504322
Car AP: 0.20777104453182088
mAP = 0.17856968683812088
193/599
Elapsed time = 1.1286985874176025
Mobile phone AP: 0.16718002733855866
Person AP: 0.16055814493504322
Car AP: 0.20777104453182088
mAP = 0.17850307226847426
194/599
Elapsed time = 0.6176061630249023
Mobile phone AP: 0.16687082039253384
Person AP: 0.16035262038808185
Car AP: 0.2092653501858941
mAP = 0.1788295969888366
195/599
Elapsed time = 0.6303651332855225
Mobile phone AP: 0.16687082039253384
Person AP: 0.16014762190625176
Car AP: 0.20831266747578178
mAP = 0.17844370325818915
196/599
Elapsed time = 0.6166484355926514
Mobile phone AP: 0.16779299345014226
Person AP: 0.16014762190625176
Car AP: 0.20831266747578178
mAP = 0.17875109427739191
197/599
Elapsed time = 0.6277327537536621
Mobile phone AP: 0.16779299345014226
Person AP: 0.15984117301605616
Car AP: 0.21092228410475913
mAP = 0.17951881685698587
198/599
Elapsed time = 0.6295242309570312
Mobile phone AP: 0.16779299345014226
Person AP: 0.1594343405756152
Car AP: 0.2104656241974692
mAP = 0.17923098607440888
199/599
Elapsed time = 1.4404523372650146
Mobile phone AP: 0.16658110718435748
Person AP: 0.1586983500286712
Car AP: 0.2104656241974692
mAP = 0.17858169380349928
200/599
Elapsed time = 0.6332700252532959
Mobile phone AP: 0.16658110718435748
Person AP: 0.15799940935665562
```

```
Car AP: 0.21062845976522102
mAP = 0.17840299210207824
201/599
Elapsed time = 0.6275463104248047
Mobile phone AP: 0.16658110718435748
Person AP: 0.1575288312919943
Car AP: 0.21062845976522102
mAP = 0.17824613274719092
202/599
Elapsed time = 0.6380245685577393
Mobile phone AP: 0.1732957250112946
Person AP: 0.1575288312919943
Car AP: 0.21062845976522102
mAP = 0.1804843386895033
203/599
Elapsed time = 0.6310625076293945
Mobile phone AP: 0.1723940612378762
Person AP: 0.1575288312919943
Car AP: 0.21062845976522102
mAP = 0.18018378409836386
204/599
Elapsed time = 1.2409789562225342
Mobile phone AP: 0.17210130334091803
Person AP: 0.17074225387681558
Car AP: 0.21062845976522102
mAP = 0.18449067232765157
205/599
Elapsed time = 0.6283981800079346
Mobile phone AP: 0.17210130334091803
Person AP: 0.17074225387681558
Car AP: 0.21151874206120125
mAP = 0.1847874330929783
206/599
Elapsed time = 0.6416089534759521
Mobile phone AP: 0.17180963463318635
Person AP: 0.17107625922279499
Car AP: 0.21206512278295014
mAP = 0.18498367221297718
207/599
Elapsed time = 1.1493170261383057
Mobile phone AP: 0.17039535093026956
Person AP: 0.17116152905946586
Car AP: 0.21206512278295014
mAP = 0.18454066759089516
208/599
Elapsed time = 0.9928264617919922
Mobile phone AP: 0.17150423397087247
Person AP: 0.17165547586743213
Car AP: 0.21184216069402084
mAP = 0.18500062351077515
209/599
Elapsed time = 0.6253664493560791
Mobile phone AP: 0.1712170192592075
Person AP: 0.18858297265141416
Car AP: 0.2113960272842723
mAP = 0.19039867306496464
210/599
```

```
Elapsed time = 0.6175355911254883
Mobile phone AP: 0.1712170192592075
Person AP: 0.18858297265141416
Car AP: 0.21087619435245883
mAP = 0.19022539542102682
211/599
Elapsed time = 0.6371002197265625
Mobile phone AP: 0.17060735102854885
Person AP: 0.18806624023508617
Car AP: 0.20913380091217146
mAP = 0.18926913072526882
212/599
Elapsed time = 0.6313595771789551
Mobile phone AP: 0.17060735102854885
Person AP: 0.18752116176648917
Car AP: 0.21062661287335852
mAP = 0.18958504188946548
213/599
Elapsed time = 0.6257786750793457
Mobile phone AP: 0.16998790009069759
Person AP: 0.18697990329363498
Car AP: 0.21062661287335852
mAP = 0.18919813875256367
214/599
Elapsed time = 0.6334741115570068
Mobile phone AP: 0.16998790009069759
Person AP: 0.18641151647656257
Car AP: 0.21062661287335852
mAP = 0.18900867648020622
215/599
Elapsed time = 0.6212978363037109
Mobile phone AP: 0.16998790009069759
Person AP: 0.18651912676294166
Car AP: 0.21062661287335852
mAP = 0.1890445465756659
216/599
Elapsed time = 0.6402182579040527
Mobile phone AP: 0.16998790009069759
Person AP: 0.18529959650556693
Car AP: 0.2091352783082947
mAP = 0.18814092496818638
217/599
Elapsed time = 0.6226620674133301
Mobile phone AP: 0.16998790009069759
Person AP: 0.184987856112661
Car AP: 0.21224369029144988
mAP = 0.18907314883160284
218/599
Elapsed time = 1.4558992385864258
Mobile phone AP: 0.17101792565651622
Person AP: 0.184987856112661
Car AP: 0.21224369029144988
mAP = 0.1894164906868757
219/599
Elapsed time = 0.6215975284576416
Mobile phone AP: 0.1728583390989483
Person AP: 0.184987856112661
```

```
Car AP: 0.21203205149427112
mAP = 0.18995941556862683
220/599
Elapsed time = 0.6342976093292236
Mobile phone AP: 0.1728583390989483
Person AP: 0.18517225739068768
Car AP: 0.21203205149427112
mAP = 0.19002088266130235
221/599
Elapsed time = 1.1634256839752197
Mobile phone AP: 0.1728583390989483
Person AP: 0.18517225739068768
Car AP: 0.20994628534603949
mAP = 0.18932562727855848
222/599
Elapsed time = 0.6304795742034912
Mobile phone AP: 0.1728583390989483
Person AP: 0.18506900993054404
Car AP: 0.20763605261246382
mAP = 0.18852113388065206
223/599
Elapsed time = 0.721031665802002
Mobile phone AP: 0.17251510742645246
Person AP: 0.18506900993054404
Car AP: 0.20763605261246382
mAP = 0.18840672332315345
224/599
Elapsed time = 0.6199288368225098
Mobile phone AP: 0.17238385030288184
Person AP: 0.18500538686580187
Car AP: 0.20763605261246382
mAP = 0.18834176326038252
225/599
Elapsed time = 0.6384921073913574
Mobile phone AP: 0.170771297778032
Person AP: 0.18588812882375774
Car AP: 0.20763605261246382
mAP = 0.18809849307141788
226/599
Elapsed time = 0.6411266326904297
Mobile phone AP: 0.1725444082189738
Person AP: 0.18466303287249078
Car AP: 0.2074363396997758
mAP = 0.18821459359708012
227/599
Elapsed time = 1.2519676685333252
Mobile phone AP: 0.1725444082189738
Person AP: 0.18581365250356385
Car AP: 0.2074363396997758
mAP = 0.18859813347410448
228/599
Elapsed time = 0.635974645614624
Mobile phone AP: 0.177792970140561 6
Person AP: 0.185310403750311
Car AP: 0.2074363396997758
mAP = 0.19017990453021613
229/599
```

```
Elapsed time = 0.8371164798736572
Mobile phone AP: 0.1777929701405616
Person AP: 0.18455273174545395
Car AP: 0.2072370353436415
mAP = 0.18986091180679324
230/599
Elapsed time = 0.589754581451416
Mobile phone AP: 0.1777929701405616
Person AP: 0.18386093911800325
Car AP: 0.21104242402709655
mAP = 0.19089877776188713
231/599
Elapsed time = 0.6208305358886719
Mobile phone AP: 0.17697803158483083
Person AP: 0.1837625603540087
Car AP: 0.2104239372547736
mAP = 0.19038817639787106
232/599
Elapsed time = 0.6306309700012207
Mobile phone AP: 0.17697803158483083
Person AP: 0.18143398630207566
Car AP: 0.2104239372547736
mAP = 0.18961198504722668
233/599
Elapsed time = 0.6256532669067383
Mobile phone AP: 0.17671153719736415
Person AP: 0.18072499390678956
Car AP: 0.21181337788802
mAP = 0.1897499696640579
234/599
Elapsed time = 0.6152141094207764
Mobile phone AP: 0.17838704439958586
Person AP: 0.18030635455852168
Car AP: 0.21181337788802
mAP = 0.19016892561537582
235/599
Elapsed time = 0.7446484565734863
Mobile phone AP: 0.17812044108420205
Person AP: 0.17872157868700195
Car AP: 0.21229436905827356
mAP = 0.18971212960982586
236/599
Elapsed time = 0.6334171295166016
Mobile phone AP: 0.17812044108420205
Person AP: 0.18033521320321566
Car AP: 0.21482977034446954
mAP = 0.19109514154396243
237/599
Elapsed time = 0.6083431243896484
Mobile phone AP: 0.17732134197718685
Person AP: 0.18006052854587118
Car AP: 0.21702022846846145
mAP = 0.1914673663305065
238/599
Elapsed time = 0.6194207668304443
Mobile phone AP: 0.17732134197718685
Person AP: 0.18006052854587118
```

```
Car AP: 0.2157556115789448
mAP = 0.19104582736733425
239/599
Elapsed time = 0.6292135715484619
Mobile phone AP: 0.17631159828809148
Person AP: 0.18006052854587118
Car AP: 0.2157556115789448
mAP = 0.19070924613763585
240/599
Elapsed time = 0.6270163059234619
Mobile phone AP: 0.17631159828809148
Person AP: 0.17978663823978902
Car AP: 0.21421876480557472
mAP = 0.19010566711115176
241/599
Elapsed time = 1.7768890857696533
Mobile phone AP: 0.17631159828809148
Person AP: 0.17920040293061212
Car AP: 0.2140212012775345
mAP = 0.18984440083207935
242/599
Elapsed time = 1.182424545288086
Mobile phone AP: 0.17631159828809148
Person AP: 0.1785307196430716
Car AP: 0.21382708067122208
mAP = 0.18955646620079505
243/599
Elapsed time = 0.6263754367828369
Mobile phone AP: 0.17631159828809148
Person AP: 0.17954409713234143
Car AP: 0.2133573216765042
mAP = 0.18973767236564573
244/599
Elapsed time = 0.6232409477233887
Mobile phone AP: 0.17631159828809148
Person AP: 0.17950050332525128
Car AP: 0.2133573216765042
mAP = 0.18972314109661567
245/599
Elapsed time = 0.6364455223083496
Mobile phone AP: 0.1760524744120149
Person AP: 0.17850324073262877
Car AP: 0.2133573216765042
mAP = 0.1893043456070493
246/599
Elapsed time = 0.6337966918945312
Mobile phone AP: 0.17422445390181157
Person AP: 0.17850324073262877
Car AP: 0.2133573216765042
mAP = 0.1886950054369815
247/599
Elapsed time = 0.6263587474822998
Mobile phone AP: 0.17422445390181157
Person AP: 0.17850324073262877
Car AP: 0.21600161172495647
mAP = 0.18957643545313227
248/599
```

```
Elapsed time = 0.6383152008056641
Mobile phone AP: 0.17422445390181157
Person AP: 0.17927484365473562
Car AP: 0.21580999278882482
mAP = 0.18976976344845733
249/599
Elapsed time = 2.015411615371704
Mobile phone AP: 0.17422445390181157
Person AP: 0.17874835027489694
Car AP: 0.21571614954445775
mAP = 0.1895629845737221
250/599
Elapsed time = 0.6370940208435059
Mobile phone AP: 0.17422445390181157
Person AP: 0.17874835027489694
Car AP: 0.2144057923710267
mAP = 0.18912619884924506
251/599
Elapsed time = 0.6656715869903564
Mobile phone AP: 0.17422445390181157
Person AP: 0.17731166007209379
Car AP: 0.2144057923710267
mAP = 0.18864730211497735
252/599
Elapsed time = 0.6352522373199463
Mobile phone AP: 0.17422445390181157
Person AP: 0.1764190836486535
Car AP: 0.2144057923710267
mAP = 0.18834977664049726
253/599
Elapsed time = 0.6302316188812256
Mobile phone AP: 0.17397443586088512
Person AP: 0.1764190836486535
Car AP: 0.21358897346670636
mAP = 0.18799416432541496
254/599
Elapsed time = 0.6270284652709961
Mobile phone AP: 0.17372519838759742
Person AP: 0.17622050999885117
Car AP: 0.21802191512989402
mAP = 0.1893225411721142
255/599
Elapsed time = 0.6626696586608887
Mobile phone AP: 0.17316629138709763
Person AP: 0.17622050999885117
Car AP: 0.21802191512989402
mAP = 0.18913623883861427
256/599
Elapsed time = 0.6213967800140381
Mobile phone AP: 0.17316629138709763
Person AP: 0.17529679247115326
Car AP: 0.21860352279810158
mAP = 0.18902220221878416
257/599
Elapsed time = 0.6278176307678223
Mobile phone AP: 0.17316629138709763
Person AP: 0.17504666230262986
```

```
Car AP: 0.22174965281104062
mAP = 0.18998753550025604
258/599
Elapsed time = 0.6300950050354004
Mobile phone AP: 0.1736269242822902
Person AP: 0.17504666230262986
Car AP: 0.22174965281104062
mAP = 0.19014107979865358
259/599
Elapsed time = 0.6342434883117676
Mobile phone AP: 0.17338274115854477
Person AP: 0.17480187490649357
Car AP: 0.22174965281104062
mAP = 0.18997808962535967
260/599
Elapsed time = 0.646754264831543
Mobile phone AP: 0.17264501792527828
Person AP: 0.17327129151270376
Car AP: 0.22174965281104062
mAP = 0.1892219874163409
261/599
Elapsed time = 0.6204729080200195
Mobile phone AP: 0.17264501792527828
Person AP: 0.1731107629740649
Car AP: 0.2209402547976984
mAP = 0.18889867856568054
262/599
Elapsed time = 0.639068603515625
Mobile phone AP: 0.17236899336301892
Person AP: 0.17270746577463394
Car AP: 0.2209402547976984
mAP = 0.18867223797845042
263/599
Elapsed time = 0.6160850524902344
Mobile phone AP: 0.1737994431623242
Person AP: 0.17270746577463394
Car AP: 0.2209402547976984
mAP = 0.18914905457821884
264/599
Elapsed time = 1.0343892574310303
Mobile phone AP: 0.1737994431623242
Person AP: 0.1728653096301456
Car AP: 0.2209402547976984
mAP = 0.18920166919672274
265/599
Elapsed time = 2.1110808849334717
Mobile phone AP: 0.1737994431623242
Person AP: 0.17176449030824562
Car AP: 0.22077377516534646
mAP = 0.1887792362119721
266/599
Elapsed time = 0.6378109455108643
Mobile phone AP: 0.1739553105562091
Person AP: 0.1708262385072254
Car AP: 0.22077377516534646
mAP = 0.18851844140959365
267/599
```

```
Elapsed time = 1.1866497993469238
Mobile phone AP: 0.17328830165628312
Person AP: 0.16982822910026912
Car AP: 0.22058934784266723
mAP = 0.18790195953307318
268/599
Elapsed time = 0.6606228351593018
Mobile phone AP: 0.17282557572845247
Person AP: 0.16951574450005624
Car AP: 0.22058934784266723
mAP = 0.18764355602372532
269/599
Elapsed time = 0.6614596843719482
Mobile phone AP: 0.17282557572845247
Person AP: 0.16951574450005624
Car AP: 0.2236309086449699
mAP = 0.18865740962449287
270/599
Elapsed time = 0.5873360633850098
Mobile phone AP: 0.1736822697302139
Person AP: 0.16913647647061494
Car AP: 0.2236309086449699
mAP = 0.18881655161526625
271/599
Elapsed time = 0.6341850757598877
Mobile phone AP: 0.17345165803798954
Person AP: 0.16891036375012508
Car AP: 0.22518240913567608
mAP = 0.1891814769745969
272/599
Elapsed time = 0.647850751876831
Mobile phone AP: 0.17345165803798954
Person AP: 0.176716593198515
Car AP: 0.22521478964867309
mAP = 0.1917943469617259
273/599
Elapsed time = 0.6238491535186768
Mobile phone AP: 0.17345165803798954
Person AP: 0.17633173586588607
Car AP: 0.22328099099992707
mAP = 0.19102146163460088
274/599
Elapsed time = 1.9302842617034912
Mobile phone AP: 0.17327447150225334
Person AP: 0.17633173586588607
Car AP: 0.22328099099992707
mAP = 0.19096239945602214
275/599
Elapsed time = 0.8521585464477539
Mobile phone AP: 0.1727903882311389
Person AP: 0.17633173586588607
Car AP: 0.22736631438707822
mAP = 0.19216281282803438
276/599
Elapsed time = 0.5818727016448975
Mobile phone AP: 0.1725643392699454
Person AP: 0.17809339245721544
```

```
Car AP: 0.22736631438707822
mAP = 0.19267468203807966
277/599
Elapsed time = 0.6175661087036133
Mobile phone AP: 0.17211010532727605
Person AP: 0.1780933924572154
Car AP: 0.22682456286173058
mAP = 0.192342686882074
278/599
Elapsed time = 0.622175931930542
Mobile phone AP: 0.17211010532727605
Person AP: 0.1778764888509971
Car AP: 0.22682456286173058
mAP = 0.19227038568000124
279/599
Elapsed time = 0.6265923976898193
Mobile phone AP: 0.17211010532727605
Person AP: 0.17704122803765002
Car AP: 0.22804728930458149
mAP = 0.19239954088983588
280/599
Elapsed time = 0.6342606544494629
Mobile phone AP: 0.17211010532727605
Person AP: 0.17704122803765002
Car AP: 0.22657266037343937
mAP = 0.19190799791278848
281/599
Elapsed time = 0.6381485462188721
Mobile phone AP: 0.17097799572748007
Person AP: 0.17696587243713674
Car AP: 0.22657266037343937
mAP = 0.1915055095126854
282/599
Elapsed time = 0.6343226432800293
Mobile phone AP: 0.17097799572748007
Person AP: 0.17659011328637644
Car AP: 0.22497902412607773
mAP = 0.19084904437997807
283/599
Elapsed time = 0.6313779354095459
Mobile phone AP: 0.17159946354122071
Person AP: 0.17569348309009766
Car AP: 0.22497902412607773
mAP = 0.1907573235857987
284/599
Elapsed time = 0.6200847625732422
Mobile phone AP: 0.17159946354122071
Person AP: 0.17537435690528896
Car AP: 0.22497902412607773
mAP = 0.19065094819086248
285/599
Elapsed time = 0.6233277320861816
Mobile phone AP: 0.17159946354122071
Person AP: 0.17489243522350045
Car AP: 0.22497902412607773
mAP = 0.1904903076302663
286/599
```

```
Elapsed time = 0.6304314136505127
Mobile phone AP: 0.1711279351188025
Person AP: 0.17489243522350045
Car AP: 0.22497902412607773
mAP = 0.1903331314894602
287/599
Elapsed time = 0.6201958656311035
Mobile phone AP: 0.1711279351188025
Person AP: 0.17416971529909078
Car AP: 0.22514700948833283
mAP = 0.190148219968742
288/599
Elapsed time = 0.6396536827087402
Mobile phone AP: 0.1711279351188025
Person AP: 0.17356251960318977
Car AP: 0.22514700948833283
mAP = 0.1899458214034417
289/599
Elapsed time = 0.622643232345581
Mobile phone AP: 0.17069847631807095
Person AP: 0.17356251960318977
Car AP: 0.22500071987723272
mAP = 0.18975390526616445
290/599
Elapsed time = 0.8607714176177979
Mobile phone AP: 0.17069847631807095
Person AP: 0.1793409938864245
Car AP: 0.22500071987723272
mAP = 0.19168006336057605
291/599
Elapsed time = 0.6412549018859863
Mobile phone AP: 0.16931278924630172
Person AP: 0.1793409938864245
Car AP: 0.22417222194470007
mAP = 0.19094200169247544
292/599
Elapsed time = 0.6001384258270264
Mobile phone AP: 0.16971520350137684
Person AP: 0.177833007712841
Car AP: 0.22417222194470007
mAP = 0.1905734777196393
293/599
Elapsed time = 1.0583086013793945
Mobile phone AP: 0.16971520350137684
Person AP: 0.1776202825547057
Car AP: 0.22398122818393132
mAP = 0.19043890474667127
294/599
Elapsed time = 0.6148154735565186
Mobile phone AP: 0.1685727407487528
Person AP: 0.1776202825547057
Car AP: 0.22398122818393132
mAP = 0.19005808382912992
295/599
Elapsed time = 0.6644139289855957
Mobile phone AP: 0.1674000981702299
Person AP: 0.17740769504553813
```

```
Car AP: 0.22381219505343775
mAP = 0.18953999608973526
296/599
Elapsed time = 0.6252424716949463
Mobile phone AP: 0.1674000981702299
Person AP: 0.1832443949948193
Car AP: 0.22381219505343775
mAP = 0.19148556273949566
297/599
Elapsed time = 0.639331579208374
Mobile phone AP: 0.1674000981702299
Person AP: 0.18295858979766522
Car AP: 0.22183418035969377
mAP = 0.19073095610919633
298/599
Elapsed time = 0.6242048740386963
Mobile phone AP: 0.16758794683494635
Person AP: 0.18295858979766522
Car AP: 0.22183418035969377
mAP = 0.19079357233076846
299/599
Elapsed time = 0.6278657913208008
Mobile phone AP: 0.16657552132122483
Person AP: 0.18327064575562435
Car AP: 0.22183418035969377
mAP = 0.190560115812181
300/599
Elapsed time = 0.627272367477417
Mobile phone AP: 0.16657552132122483
Person AP: 0.18289443344619524
Car AP: 0.22183418035969377
mAP = 0.19043471170903792
301/599
Elapsed time = 0.9841630458831787
Mobile phone AP: 0.16617497499529502
Person AP: 0.1856847021604339
Car AP: 0.22183418035969377
mAP = 0.19123128583847424
302/599
Elapsed time = 1.7349107265472412
Mobile phone AP: 0.16617497499529502
Person AP: 0.1856847021604339
Car AP: 0.22065852406282652
mAP = 0.19083940040618516
303/599
Elapsed time = 0.6281843185424805
Mobile phone AP: 0.16575327904899703
Person AP: 0.18540322749463659
Car AP: 0.22065852406282652
mAP = 0.19060501020215337
304/599
Elapsed time = 0.6336698532104492
Mobile phone AP: 0.16341395590387714
Person AP: 0.18540322749463659
Car AP: 0.22033453343058446
mAP = 0.18971723894303275
305/599
```

```
Elapsed time = 0.6216144561767578
Mobile phone AP: 0.16341395590387714
Person AP: 0.18533314125431263
Car AP: 0.21921123982005583
mAP = 0.1893194456594152
306/599
Elapsed time = 0.587165117263794
Mobile phone AP: 0.16341395590387714
Person AP: 0.18533314125431263
Car AP: 0.219706207580123
mAP = 0.1894844349127709
307/599
Elapsed time = 0.6302683353424072
Mobile phone AP: 0.16438850625894896
Person AP: 0.18441042599383137
Car AP: 0.219706207580123
mAP = 0.18950171327763443
308/599
Elapsed time = 0.620664119720459
Mobile phone AP: 0.16438850625894896
Person AP: 0.18557477717260437
Car AP: 0.219706207580123
mAP = 0.18988983033722548
309/599
Elapsed time = 0.6404168605804443
Mobile phone AP: 0.16438850625894896
Person AP: 0.18681482133199373
Car AP: 0.21954708068665504
mAP = 0.19025013609253258
310/599
Elapsed time = 0.6402792930603027
Mobile phone AP: 0.16438850625894896
Person AP: 0.18711516034002773
Car AP: 0.21825779020006505
mAP = 0.18992048559968058
311/599
Elapsed time = 0.6231896877288818
Mobile phone AP: 0.1647588229587175
Person AP: 0.18711516034002773
Car AP: 0.21825779020006505
mAP = 0.19004392449960342
312/599
Elapsed time = 0.6252036094665527
Mobile phone AP: 0.16465481507082677
Person AP: 0.18711516034002773
Car AP: 0.2180961966679548
mAP = 0.18995539069293643
313/599
Elapsed time = 0.6285800933837891
Mobile phone AP: 0.1642744545240163
Person AP: 0.18649969152236162
Car AP: 0.21761877554357284
mAP = 0.18946430719665028
314/599
Elapsed time = 0.6357941627502441
Mobile phone AP: 0.1642744545240163
Person AP: 0.18649969152236162
```

```
Car AP: 0.2160802106050137
mAP = 0.18895145221713053
315/599
Elapsed time = 0.6248383522033691
Mobile phone AP: 0.16419450127599547
Person AP: 0.18702622595646998
Car AP: 0.2160802106050137
mAP = 0.18910031261249305
316/599
Elapsed time = 0.6225669384002686
Mobile phone AP: 0.16398410765586305
Person AP: 0.18702622595646998
Car AP: 0.2160802106050137
mAP = 0.18903018140578223
317/599
Elapsed time = 0.6319863796234131
Mobile phone AP: 0.16306318870252096
Person AP: 0.18748316232579418
Car AP: 0.2160802106050137
mAP = 0.18887552054444293
318/599
Elapsed time = 0.618480920791626
Mobile phone AP: 0.16322873738065632
Person AP: 0.18748316232579418
Car AP: 0.2160802106050137
mAP = 0.18893070343715 47
319/599
Elapsed time = 0.879023551940918
Mobile phone AP: 0.16304487530477332
Person AP: 0.18748316232579418
Car AP: 0.21518639539578838
mAP = 0.188571477675452
320/599
Elapsed time = 0.6244618892669678
Mobile phone AP: 0.16304487530477332
Person AP: 0.18757111110640295
Car AP: 0.21518639539578838
mAP = 0.18860079393565488
321/599
Elapsed time = 0.6148555278778076
Mobile phone AP: 0.16274184646995654
Person AP: 0.18750352880878102
Car AP: 0.21518639539578838
mAP = 0.18847725689150865
322/599
Elapsed time = 1.142272710800171
Mobile phone AP: 0.16274184646995654
Person AP: 0.18743599535497726
Car AP: 0.2154320946587487
mAP = 0.18853664549456084
323/599
Elapsed time = 0.6402859687805176
Mobile phone AP: 0.16274184646995654
Person AP: 0.18669514351621017
Car AP: 0.21579107966097363
mAP = 0.18840935654904678
324/599
```

```
Elapsed time = 0.6300914287567139
Mobile phone AP: 0.1622581754494083
Person AP: 0.18669514351621017
Car AP: 0.21564092055510573
mAP = 0.1881980798402414
325/599
Elapsed time = 0.6444058418273926
Mobile phone AP: 0.16208048094851013
Person AP: 0.1868761149333292
Car AP: 0.21512334781116893
mAP = 0.18802664789766943
326/599
Elapsed time = 0.598834753036499
Mobile phone AP: 0.16208048094851013
Person AP: 0.1886156002794915
Car AP: 0.21512334781116893
mAP = 0.1886064763463902
327/599
Elapsed time = 0.6351406574249268
Mobile phone AP: 0.16208048094851013
Person AP: 0.18948286030990724
Car AP: 0.21512334781116893
mAP = 0.18889556302319543
328/599
Elapsed time = 0.6152186393737793
Mobile phone AP: 0.1618438450383707
Person AP: 0.18948286030990724
Car AP: 0.21497397758214115
mAP = 0.1887668943101397
329/599
Elapsed time = 2.00144100189209
Mobile phone AP: 0.16184583338497527
Person AP: 0.18948286030990724
Car AP: 0.21497397758214115
mAP = 0.1887675570923412
330/599
Elapsed time = 0.6383957862854004
Mobile phone AP: 0.16155258177133325
Person AP: 0.18922088286779726
Car AP: 0.21497397758214115
mAP = 0.18858248074042386
331/599
Elapsed time = 0.5948054790496826
Mobile phone AP: 0.16155258177133325
Person AP: 0.1888315806648746
Car AP: 0.21497397758214115
mAP = 0.18845271333944966
332/599
Elapsed time = 1.247004508972168
Mobile phone AP: 0.16098224463101174
Person AP: 0.18852204272922377
Car AP: 0.21497397758214115
mAP = 0.1881594216474589
333/599
Elapsed time = 0.6181333065032959
Mobile phone AP: 0.16098224463101174
Person AP: 0.18827318523253664
```

```
Car AP: 0.21497397758214115
mAP = 0.18807646914856316
334/599
Elapsed time = 0.6615011692047119
Mobile phone AP: 0.160489532953637
Person AP: 0.18827318523253664
Car AP: 0.21497397758214115
mAP = 0.1879122319227716
335/599
Elapsed time = 0.6443088054656982
Mobile phone AP: 0.16032004273399306
Person AP: 0.18810472626521219
Car AP: 0.21497397758214115
mAP = 0.18779958219378212
336/599
Elapsed time = 1.2284462451934814
Mobile phone AP: 0.16032004273399306
Person AP: 0.18804077361081803
Car AP: 0.2153683913538399
mAP = 0.18790973589955032
337/599
Elapsed time = 0.6472022533416748
Mobile phone AP: 0.16032004273399306
Person AP: 0.18804077361081803
Car AP: 0.21452525478754397
mAP = 0.1876286903774517
338/599
Elapsed time = 0.6596508026123047
Mobile phone AP: 0.16032004273399306
Person AP: 0.1873403917736263
Car AP: 0.21600571431351925
mAP = 0.18788871627371287
339/599
Elapsed time = 0.6279973983764648
Mobile phone AP: 0.16263475861762489
Person AP: 0.1873403917736263
Car AP: 0.21600571431351925
mAP = 0.18866028823492345
340/599
Elapsed time = 0.6261036396026611
Mobile phone AP: 0.16334183548970627
Person AP: 0.18727700019188928
Car AP: 0.21600571431351925
mAP = 0.1888748499983716
341/599
Elapsed time = 1.7098381519317627
Mobile phone AP: 0.16334183548970627
Person AP: 0.1868939611870438
Car AP: 0.21623887593272043
mAP = 0.1888248908698235
342/599
Elapsed time = 0.632821798324585
Mobile phone AP: 0.16321033579285757
Person AP: 0.1868939611870438
Car AP: 0.21623887593272043
mAP = 0.1887810576375406
343/599
```

```
Elapsed time = 0.6250131130218506
Mobile phone AP: 0.1627041078432481
Person AP: 0.1866309875233534
Car AP: 0.21594741899523842
mAP = 0.18842750478727996
344/599
Elapsed time = 0.6368401050567627
Mobile phone AP: 0.1627041078432481
Person AP: 0.18844853211817006
Car AP: 0.21668330309475842
mAP = 0.18927864768539218
345/599
```

In [0]:

```
mAP = [mAP for mAP in mAPs if str(mAP)!='nan']
mean_average_prec = round(np.mean(np.array(mAP)), 3)
print('After training %dk batches, the mean average precision is %0.3f'%(le
n(record_df), mean_average_prec))

# record_df.loc[len(record_df)-1, 'mAP'] = mean_average_prec
# record_df.to_csv(C.record_path, index=0)
# print('Save mAP to {}'.format(C.record_path))
```