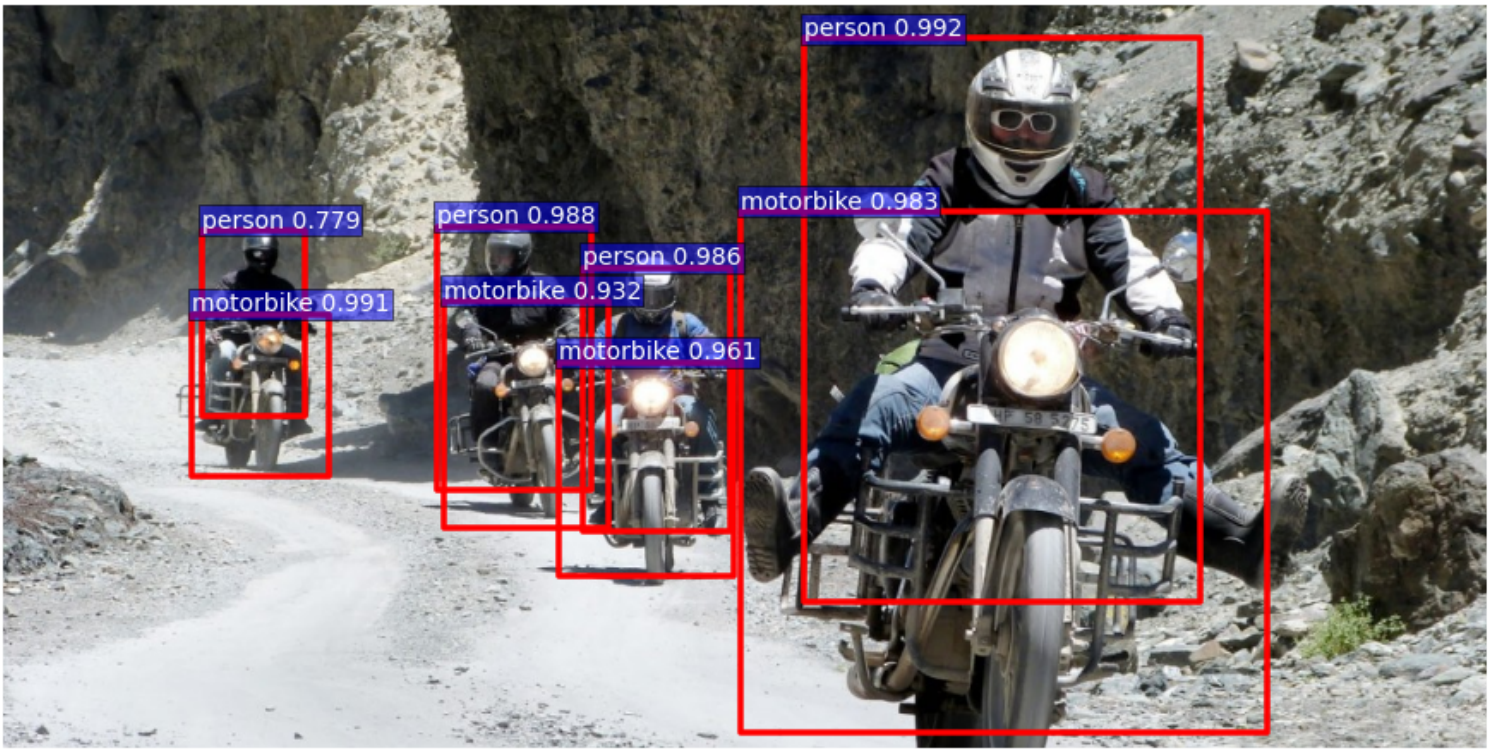
Object Detection Using Tenserflow



1. Object detection is one of the important fields in data science which is widely used in modern world.
2. In this project we are using tenserflow and in this we are using our pre-trained model which Single Shot MultiBox Detector (ssd) model.
3. Our Model Name is "ssd_mobilenet_v1_coco".
4. We are using simple tenserflow model for this project and tenserflow is a part of Deep learning.

Dependencies

Dependencies Tensorflow Object Detection API depends on the following libraries:

Protobuf 3.0.0 Python-tk Pillow 1.0 lxml tf Slim (which is included in the "tensorflow/models/research/" checkout) Jupyter notebook Matplotlib Tensorflow Cython contextlib2 cocoapi

Imports

In [3]:
```
import numpy as np
import os
import six.moves.urllib as urllib
import sys
import tarfile
import tensorflow as tf
import zipfile

from collections import defaultdict
from io import StringIO
from matplotlib import pyplot as plt
from PIL import Image
```

```
C:\Users\RAVI KRISHNA\Anaconda3\lib\site-packages\h5py\__init__.py:36: FutureWarnin
g: Conversion of the second argument of issubdtype from `float` to `np.floating` is
deprecated. In future, it will be treated as `np.float64 == np.dtype(float).type`.
  from ._conv import register_converters as _register_converters
```

Env setup

In [4]:
```
# This is needed to display the images.
%matplotlib inline

# This is needed since the notebook is stored in the object_detection folder.
sys.path.append("..")
```

Object detection imports

Here are the imports from the object detection module.

In [3]:
```
from utils import label_map_util

from utils import visualization_utils as vis_util
```

Model preparation

Variables

Any model exported using the `export_inference_graph.py` tool can be loaded here simply by changing `PATH_TO_CKPT` to point to a new .pb file.

By default we use an "SSD with Mobilenet" model here. See the [detection model zoo (https://github.com/tensorflow/models/blob/master/object_detection/g3doc/detection_model_zoo.md)](https://github.com/tensorflow/models/blob/master/object_detection/g3doc/detection_model_zoo.md) for a list of other models that can be run out-of-the-box with varying speeds and accuracies.

In [4]:
```
#model initialisation

MODEL_NAME = 'ssd_mobilenet_v1_coco_11_06_2017'
MODEL_FILE = MODEL_NAME + '.tar.gz' # file extension
DOWNLOAD_BASE = 'http://download.tensorflow.org/models/object_detection/' # Our downl
oad link using tensrflow API

# Path to frozen detection graph. This is the actual model that is used for the objec
t detection.
PATH_TO_CKPT = MODEL_NAME + '/frozen_inference_graph.pb'

# List of the strings that is used to add correct label for each box.
PATH_TO_LABELS = os.path.join('data', 'mscoco_label_map.pbtxt')

NUM_CLASSES = 90
```

Download Model

In [5]:
```
#here we are downloading our pre define model

opener = urllib.request.URLopener()
opener.retrieve(DOWNLOAD_BASE + MODEL_FILE, MODEL_FILE)
tar_file = tarfile.open(MODEL_FILE)
for file in tar_file.getmembers():
  file_name = os.path.basename(file.name)
  if 'frozen_inference_graph.pb' in file_name:
    tar_file.extract(file, os.getcwd())
```

Load a (frozen) Tensorflow model into memory.

In [6]:
```
detection_graph = tf.Graph()
with detection_graph.as_default():
  od_graph_def = tf.GraphDef()
  with tf.gfile.GFile(PATH_TO_CKPT, 'rb') as fid:
    serialized_graph = fid.read()
    od_graph_def.ParseFromString(serialized_graph)
    tf.import_graph_def(od_graph_def, name='')
```

Loading label map

Label maps map indices to category names, so that when our convolution network predicts `5`, we know that this corresponds to `airplane`. Here we use internal utility functions, but anything that returns a dictionary mapping integers to appropriate string labels would be fine

In [7]:
```
label_map = label_map_util.load_labelmap(PATH_TO_LABELS)
categories = label_map_util.convert_label_map_to_categories(label_map, max_num_classe
s=NUM_CLASSES, use_display_name=True)
category_index = label_map_util.create_category_index(categories)
```

Helper code

In [8]:
```
#method to load the image
def load_image_into_numpy_array(image):
  (im_width, im_height) = image.size
  return np.array(image.getdata()).reshape(
      (im_height, im_width, 3)).astype(np.uint8)
```

Detection

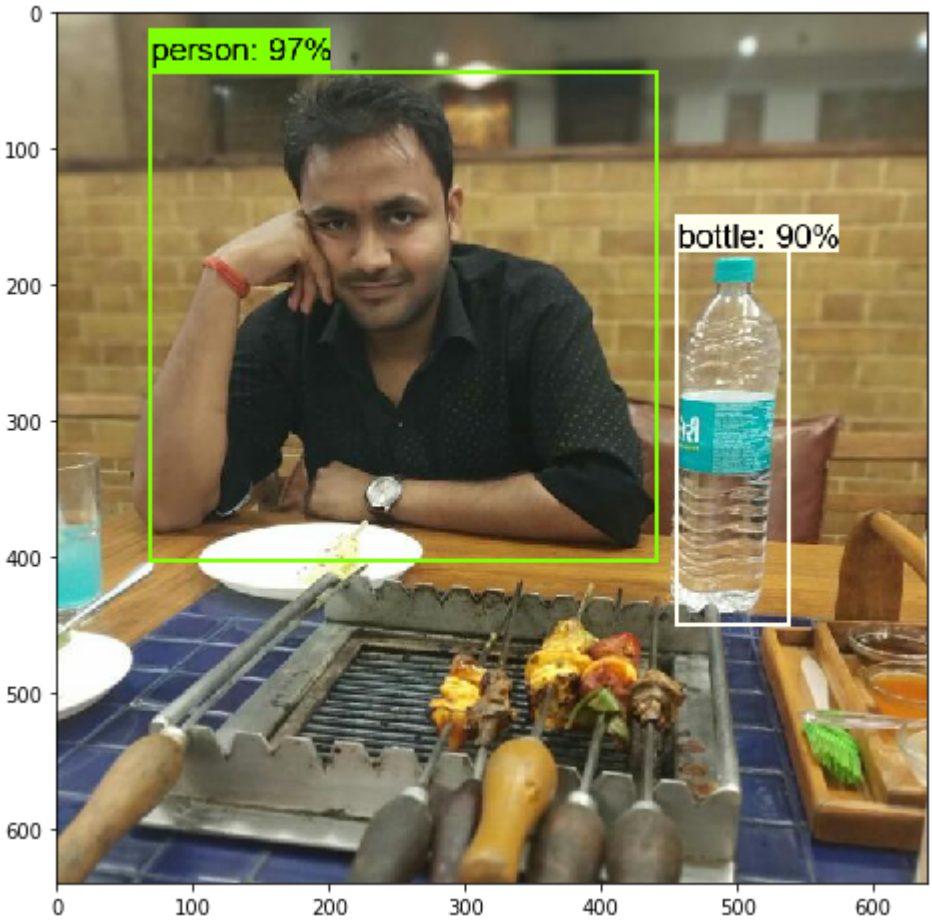
In [15]:
```
# For the sake of simplicity we will use only 4 images:
# image1.jpg
# image2.jpg
# image3.jpg
# image4.jpg
# If you want to test the code with your images, just add path to the images to the T
EST_IMAGE_PATHS.

PATH_TO_TEST_IMAGES_DIR = 'test_images'
TEST_IMAGE_PATHS = [ os.path.join(PATH_TO_TEST_IMAGES_DIR, 'image{}.jpg'.format(i)) f
or i in range(1, 5) ]

# Size, in inches, of the output images.
IMAGE_SIZE = (12, 8)
```

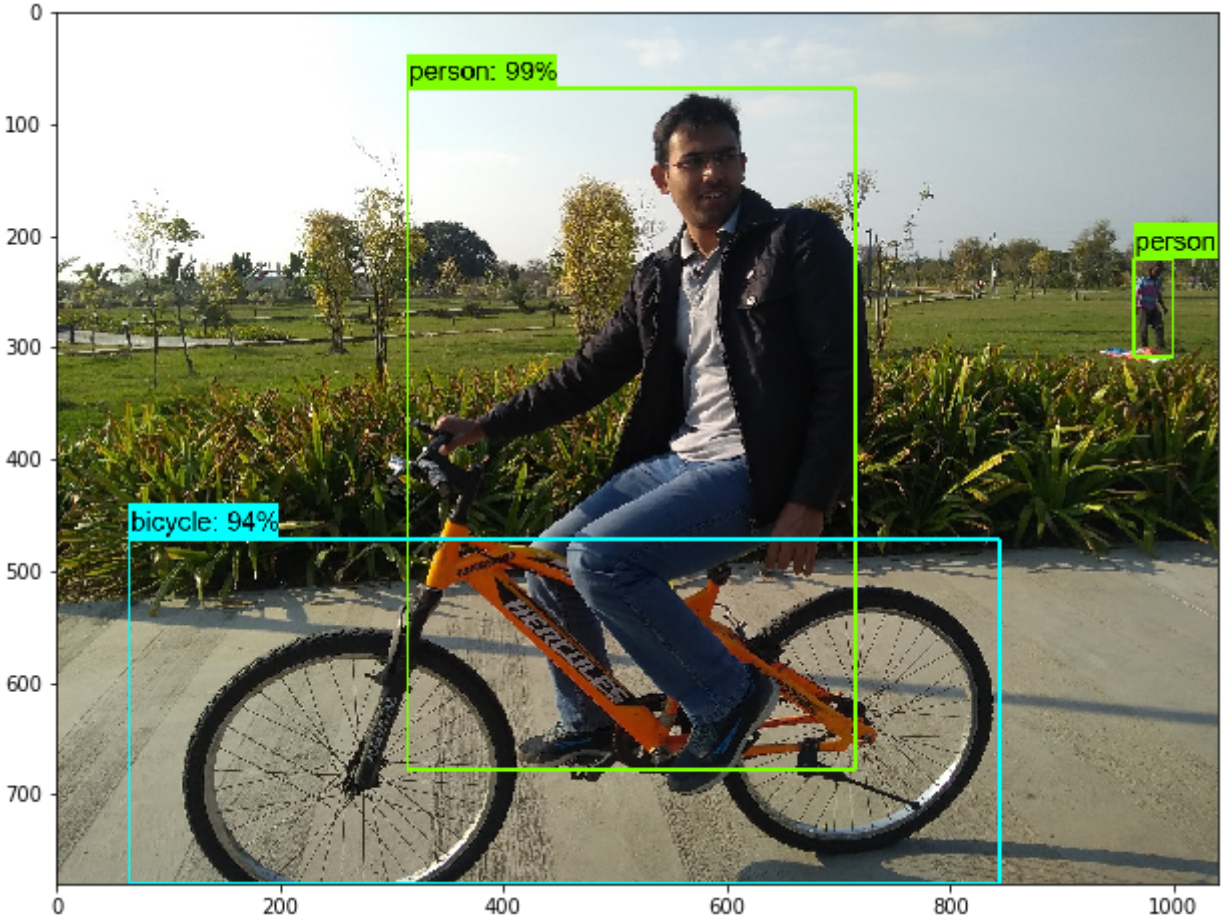
In [16]:
```
TEST_IMAGE_PATHS
```

Out[16]:
```
['test_images\\image1.jpg',
 'test_images\\image2.jpg',
 'test_images\\image3.jpg',
 'test_images\\image4.jpg']
```

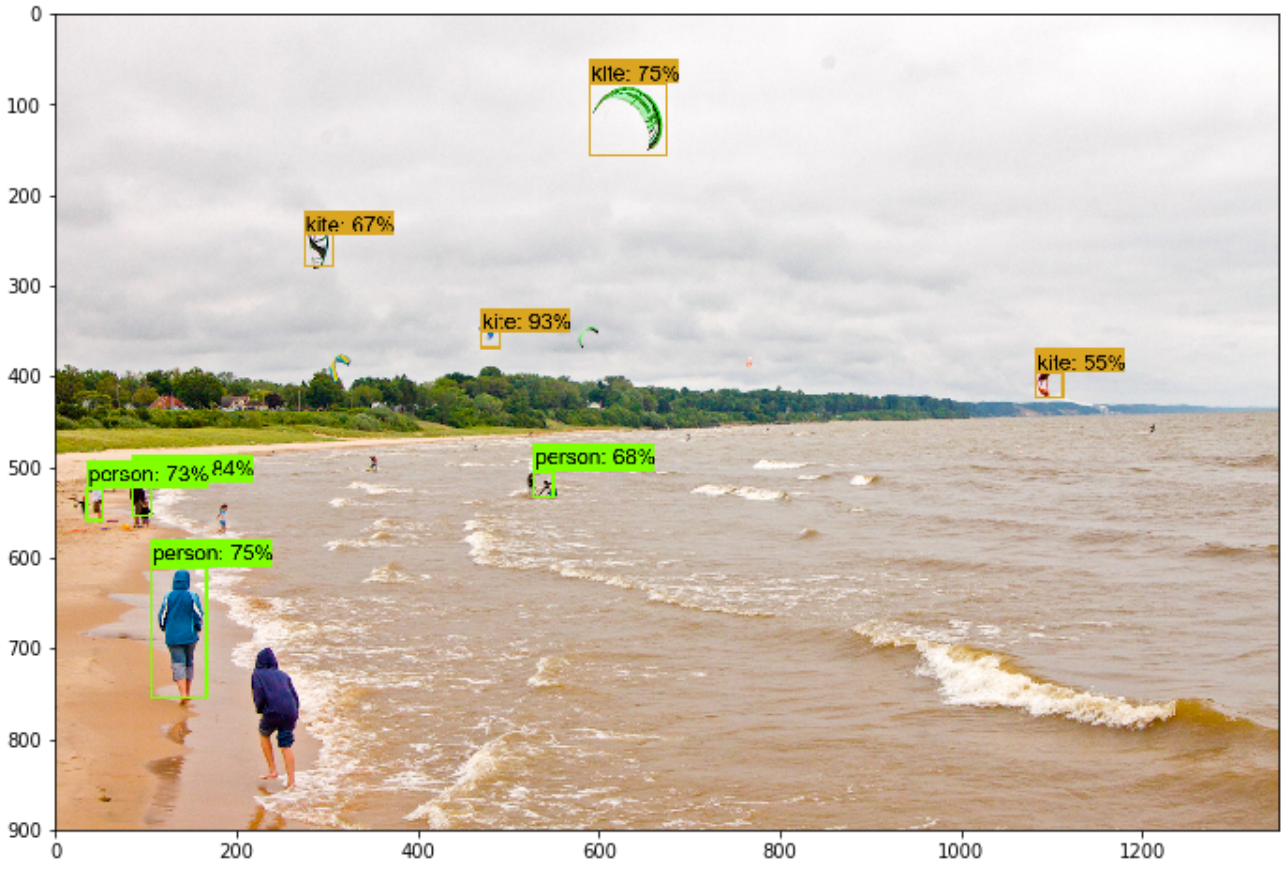
In [17]:

```
#object detection

with detection_graph.as_default():
    with tf.Session(graph=detection_graph) as sess:

        for image_path in TEST_IMAGE_PATHS:
            image = Image.open(image_path)
            # the array based representation of the image will be used later in order
to prepare the
            # result image with boxes and labels on it.
            image_np = load_image_into_numpy_array(image)
            # Expand dimensions since the model expects images to have shape: [1, Non
e, None, 3]
            image_np_expanded = np.expand_dims(image_np, axis=0)
            image_tensor = detection_graph.get_tensor_by_name('image_tensor:0')
            # Each box represents a part of the image where a particular object was d
etected.
            boxes = detection_graph.get_tensor_by_name('detection_boxes:0')
            # Each score represent how level of confidence for each of the objects.
            # Score is shown on the result image, together with the class label.
            scores = detection_graph.get_tensor_by_name('detection_scores:0')
            classes = detection_graph.get_tensor_by_name('detection_classes:0')
            num_detections = detection_graph.get_tensor_by_name('num_detections:0')
            # Actual detection.
            (boxes, scores, classes, num_detections) = sess.run(
            [boxes, scores, classes, num_detections],
            feed_dict={image_tensor: image_np_expanded})
            # Visualization of the results of a detection.
            vis_util.visualize_boxes_and_labels_on_image_array(
            image_np,
            np.squeeze(boxes),
            np.squeeze(classes).astype(np.int32),
            np.squeeze(scores),
            category_index,
            use_normalized_coordinates=True,
            line_thickness=3)

        plt.figure(figsize=IMAGE_SIZE)
        plt.imshow(image_np)
```


person: 97%
bottle: 90%
0
100
200
300
400
500
600


person: 99%
person
bicycle: 94%
0
100
200
300
400
500
600
700
0
200
400
600
800
1000


truck: 53%
person: 60%
person: 95%


kite: 75%
kite: 67%
kite: 93%
kite: 55%
person: 73%
person: 68%
person: 75%

---xxx---