Object Detection Inference Using TF2 and TFHub

https://github.com/tensorflow/hub/blob/master/examples/colab/tf2_object_detection.ipynb

Modified by Hamilton Hitchings to compare algorithms side by side while testing with new images

Imports and Setup

```
import os
import pathlib

import matplotlib
import matplotlib.pyplot as plt

import io
import scipy.misc
import numpy as np
from six import BytesIO
from PIL import Image, ImageDraw, ImageFont
from six.moves.urllib.request import urlopen

import tensorflow as tf
import tensorflow_hub as hub

tf.get_logger().setLevel('ERROR')
```

Utilities

Run the following cell to create some utils that will be needed later:

- Helper method to load an image
- Map of Model Name to TF Hub handle
- List of tuples with Human Keypoints for the COCO 2017 dataset. This is needed for models with keyponts.

```
# https://github.com/ndrplz/google-drive-downloader
!pip install googledrivedownloader
```

```
    Requirement already satisfied: googledrivedownloader in /usr/local/lib/python3.7/dist-packages (0.4)
```

```
from google_drive_downloader import GoogleDriveDownloader as gdd
```

```
IMAGES_FOR_TEST = {
  'Bike-Race' : {'file_path' : './bike-race.jpeg', 'GoogleId' : '1E_Tzsgb0-VypJg8w8WUF_u-zvqcam3xO'},
  'Street1' : {'file_path' : './street1.jpeg', 'GoogleId' : '15ETZT6YVOvDOTDmF3ZAgkOz5HVaql3EW'},
  'Naxos_Taverna' : {'file_path' : './Naxos_Taverna.jpeg', 'GoogleId' : '1k-fMFdHH5Q0JI9ZTfMmVyBBw4mDsTYCn'},
  'Kalua-Bay' : {'file_path' : './kalua-bay.jpeg', 'GoogleId' : '1ka6giLHT7je9y4V0uKQMxqCziu972ZDY'}
}
```

```
for key in IMAGES_FOR_TEST:
  value = IMAGES_FOR_TEST[key]
  gdd.download_file_from_google_drive(file_id=value['GoogleId'], dest_path=value['file_path'])
```

```
    Downloading 1E_Tzsgb0-VypJg8w8WUF_u-zvqcam3xO into ./bike-race.jpeg... Done.
    Downloading 15ETZT6YVOvDOTDmF3ZAgkOz5HVaql3EW into ./street1.jpeg... Done.
    Downloading 1k-fMFdHH5Q0JI9ZTfMmVyBBw4mDsTYCn into ./Naxos_Taverna.jpeg... Done.
    Downloading 1ka6giLHT7je9y4V0uKQMxqCziu972ZDY into ./kalua-bay.jpeg... Done.
```

```
# Retained for reference for downloading entire image dataset
# gdd.download_file_from_google_drive(file_id='1iytA1n2z4go3uVCwE__vIKouTKyIDjEq',
#                                     dest_path='./data/mnist.zip',
#                                     unzip=True)
```

```
#                                       unzip=True)

def load_image_into_numpy_array(path):
  """Load an image from file into a numpy array.

  Puts image into numpy array to feed into tensorflow graph.
  Note that by convention we put it into a numpy array with shape
  (height, width, channels), where channels=3 for RGB.

  Args:
    path: the file path to the image

  Returns:
    uint8 numpy array with shape (img_height, img_width, 3)
  """
  image = None
  if(path.startswith('http')):
    print(path)
    response = urlopen(path)
    print("about to read")
    image_data = response.read()
    print("read")
    image_data = BytesIO(image_data)
    print(f"got image_data: {image_data}")
    image = Image.open(image_data)
    print("got image")
  else:
    print(f"reading file from disk {path}")
    image_data = tf.io.gfile.GFile(path, 'rb').read()
    print("read image data")
    image = Image.open(BytesIO(image_data))

  (im_width, im_height) = image.size
  return np.array(image.getdata()).reshape(
      (1, im_height, im_width, 3)).astype(np.uint8)

COCO17_HUMAN_POSE_KEYPOINTS = [(0, 1),
 (0, 2),
 (1, 3),
 (2, 4),
 (0, 5),
 (0, 6),
 (5, 7),
 (7, 9),
 (6, 8),
 (8, 10),
 (5, 6),
 (5, 11),
 (6, 12),
 (11, 12),
 (11, 13),
 (13, 15),
 (12, 14),
 (14, 16)]
```

Loading an image

Let's try the model on a simple image. To help with this, we provide a list of test images.

Here are some simple things to try out if you are curious:

- Try running inference on your own images, just upload them to colab and load the same way it's done in the cell below.
- Modify some of the input images and see if detection still works. Some simple things to try out here include flipping the image horizontally, or converting to grayscale (note that we still expect the input image to have 3 channels).

Be careful: when using images with an alpha channel, the model expect 3 channels images and the alpha will count as a 4th.

```
def load_image(image_path):
  flip_image_horizontally = False #@param {type:"boolean"}
  convert_image_to_grayscale = False #@param {type:"boolean"}

  #image_path = IMAGES_FOR_TEST[selected_image]
```

flip_image_horizontally: ☐

convert_image_to_grayscale: ☐

```
  #image_path = IMAGES_FOR_TEST[selected_image]
  image_np = load_image_into_numpy_array(image_path)

  # Flip horizontally
  if(flip_image_horizontally):
    image_np[0] = np.fliplr(image_np[0]).copy()

  # Convert image to grayscale
  if(convert_image_to_grayscale):
    image_np[0] = np.tile(
      np.mean(image_np[0], 2, keepdims=True), (1, 1, 3)).astype(np.uint8)
  return image_np


for key in IMAGES_FOR_TEST:
  print(key)
  value = IMAGES_FOR_TEST[key]
  file_path = value['file_path']
  value['image_np'] = load_image(file_path)
  IMAGES_FOR_TEST[key] = value
```

```
Bike-Race
reading file from disk ./bike-race.jpeg
read image data
Street1
reading file from disk ./street1.jpeg
read image data
Naxos_Taverna
reading file from disk ./Naxos_Taverna.jpeg
read image data
Kalua-Bay
reading file from disk ./kalua-bay.jpeg
read image data
```

Select Model

See the models speed and accuracy (mAP - mean Average Precision) at:

https://tfhub.dev/tensorflow/collections/object_detection/1

```
ALL_MODELS = {
'CenterNet HourGlass104 512x512' : 'https://tfhub.dev/tensorflow/centernet/hourglass_512x512/1',
'CenterNet HourGlass104 Keypoints 512x512' : 'https://tfhub.dev/tensorflow/centernet/hourglass_512x512_kpts/1',
'CenterNet HourGlass104 1024x1024' : 'https://tfhub.dev/tensorflow/centernet/hourglass_1024x1024/1',
'CenterNet HourGlass104 Keypoints 1024x1024' : 'https://tfhub.dev/tensorflow/centernet/hourglass_1024x1024_kpts/1',
'CenterNet Resnet50 V1 FPN 512x512' : 'https://tfhub.dev/tensorflow/centernet/resnet50v1_fpn_512x512/1',
'CenterNet Resnet50 V1 FPN Keypoints 512x512' : 'https://tfhub.dev/tensorflow/centernet/resnet50v1_fpn_512x512_kpts/1',
'CenterNet Resnet101 V1 FPN 512x512' : 'https://tfhub.dev/tensorflow/centernet/resnet101v1_fpn_512x512/1',
'CenterNet Resnet50 V2 512x512' : 'https://tfhub.dev/tensorflow/centernet/resnet50v2_512x512/1',
'CenterNet Resnet50 V2 Keypoints 512x512' : 'https://tfhub.dev/tensorflow/centernet/resnet50v2_512x512_kpts/1',
'EfficientDet D0 512x512' : 'https://tfhub.dev/tensorflow/efficientdet/d0/1',
'EfficientDet D1 640x640' : 'https://tfhub.dev/tensorflow/efficientdet/d1/1',
'EfficientDet D2 768x768' : 'https://tfhub.dev/tensorflow/efficientdet/d2/1',
'EfficientDet D3 896x896' : 'https://tfhub.dev/tensorflow/efficientdet/d3/1',
'EfficientDet D4 1024x1024' : 'https://tfhub.dev/tensorflow/efficientdet/d4/1',
'EfficientDet D5 1280x1280' : 'https://tfhub.dev/tensorflow/efficientdet/d5/1',
'EfficientDet D6 1280x1280' : 'https://tfhub.dev/tensorflow/efficientdet/d6/1',
'EfficientDet D7 1536x1536' : 'https://tfhub.dev/tensorflow/efficientdet/d7/1',
'SSD MobileNet v2 320x320' : 'https://tfhub.dev/tensorflow/ssd_mobilenet_v2/2',
'SSD MobileNet V1 FPN 640x640' : 'https://tfhub.dev/tensorflow/ssd_mobilenet_v1/fpn_640x640/1',
'SSD MobileNet V2 FPNLite 320x320' : 'https://tfhub.dev/tensorflow/ssd_mobilenet_v2/fpnlite_320x320/1',
'SSD MobileNet V2 FPNLite 640x640' : 'https://tfhub.dev/tensorflow/ssd_mobilenet_v2/fpnlite_640x640/1',
'SSD ResNet50 V1 FPN 640x640 (RetinaNet50)' : 'https://tfhub.dev/tensorflow/retinanet/resnet50_v1_fpn_640x640/1',
'SSD ResNet50 V1 FPN 1024x1024 (RetinaNet50)' : 'https://tfhub.dev/tensorflow/retinanet/resnet50_v1_fpn_1024x1024/1',
'SSD ResNet101 V1 FPN 640x640 (RetinaNet101)' : 'https://tfhub.dev/tensorflow/retinanet/resnet101_v1_fpn_640x640/1',
'SSD ResNet101 V1 FPN 1024x1024 (RetinaNet101)' : 'https://tfhub.dev/tensorflow/retinanet/resnet101_v1_fpn_1024x1024/1',
'SSD ResNet152 V1 FPN 640x640 (RetinaNet152)' : 'https://tfhub.dev/tensorflow/retinanet/resnet152_v1_fpn_640x640/1',
'SSD ResNet152 V1 FPN 1024x1024 (RetinaNet152)' : 'https://tfhub.dev/tensorflow/retinanet/resnet152_v1_fpn_1024x1024/1',
'Faster R-CNN ResNet50 V1 640x640' : 'https://tfhub.dev/tensorflow/faster_rcnn/resnet50_v1_640x640/1',
'Faster R-CNN ResNet50 V1 1024x1024' : 'https://tfhub.dev/tensorflow/faster_rcnn/resnet50_v1_1024x1024/1',
'Faster R-CNN ResNet50 V1 800x1333' : 'https://tfhub.dev/tensorflow/faster_rcnn/resnet50_v1_800x1333/1',
'Faster R-CNN ResNet101 V1 640x640' : 'https://tfhub.dev/tensorflow/faster_rcnn/resnet101_v1_640x640/1',
'Faster R-CNN ResNet101 V1 1024x1024' : 'https://tfhub.dev/tensorflow/faster_rcnn/resnet101_v1_1024x1024/1',
'Faster R-CNN ResNet101 V1 800x1333' : 'https://tfhub.dev/tensorflow/faster_rcnn/resnet101_v1_800x1333/1',
```

```
'Faster R-CNN ResNet152 V1 640x640' : 'https://tfhub.dev/tensorflow/faster_rcnn/resnet152_v1_640x640/1',
'Faster R-CNN ResNet152 V1 1024x1024' : 'https://tfhub.dev/tensorflow/faster_rcnn/resnet152_v1_1024x1024/1',
'Faster R-CNN ResNet152 V1 800x1333' : 'https://tfhub.dev/tensorflow/faster_rcnn/resnet152_v1_800x1333/1',
'Faster R-CNN Inception ResNet V2 640x640' : 'https://tfhub.dev/tensorflow/faster_rcnn/inception_resnet_v2_640x640/1',
'Faster R-CNN Inception ResNet V2 1024x1024' : 'https://tfhub.dev/tensorflow/faster_rcnn/inception_resnet_v2_1024x1024/1',
'Mask R-CNN Inception ResNet V2 1024x1024' : 'https://tfhub.dev/tensorflow/mask_rcnn/inception_resnet_v2_1024x1024/1'
}
```

Visualization tools

To visualize the images with the proper detected boxes, keypoints and segmentation, we will use the TensorFlow Object Detection API. To install it we will clone the repo.

```
# Clone the tensorflow models repository
!git clone --depth 1 https://github.com/tensorflow/models
```

```
Cloning into 'models'...
remote: Enumerating objects: 2588, done.
remote: Counting objects: 100% (2588/2588), done.
remote: Compressing objects: 100% (2135/2135), done.
remote: Total 2588 (delta 646), reused 1428 (delta 421), pack-reused 0
Receiving objects: 100% (2588/2588), 32.51 MiB | 34.50 MiB/s, done.
Resolving deltas: 100% (646/646), done.
```

Installing the Object Detection API

```
%%bash
sudo apt install -y protobuf-compiler
cd models/research/
protoc object_detection/protos/*.proto --python_out=.
cp object_detection/packages/tf2/setup.py .
python -m pip install .
```

Now we can import the dependencies we will need later

```
from object_detection.utils import label_map_util
from object_detection.utils import visualization_utils as viz_utils
from object_detection.utils import ops as utils_ops

%matplotlib inline
```

Load label map data (for plotting).

Label maps correspond index numbers 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.

We are going, for simplicity, to load from the repository that we loaded the Object Detection API code

```
PATH_TO_LABELS = './models/research/object_detection/data/mscoco_label_map.pbtxt'
category_index = label_map_util.create_category_index_from_labelmap(PATH_TO_LABELS, use_display_name=True)
```

Build a detection model and load pre-trained model weights

Here we will choose which Object Detection model we will use. Select the architecture and it will be loaded automatically. If you want to change the model to try other architectures later, just change the next cell and execute following ones.

Tip: if you want to read more details about the selected model, you can follow the link (model handle) and read additional documentation on TF Hub. After you select a model, we will print the handle to make it easier.

```
#@title Model 1 Selection { display-mode: "form", run: "auto" }Model 1 Selection
model_display_name1 = 'Mask R-CNN Inception ResNet V2 1024x1024' # @param ['Cen·
model_handle1 = ALL_MODELS[model_display_name1]
```

```
#@title Model 1 Selection { display-mode: "form", run: "auto" }
model_display_name1 = 'Mask R-CNN Inception ResNet V2 1024x1024' # @param ['CenterNet HourGlass104 Keypoints 512x512', 'CenterNet HourGlass104 Keypoints 1024x1024', 'CenterNet Resnet50 V1 FPN Keypoints 512x512', 'CenterNet Resnet101 V1 FPN Keypoints 512x512', 'CenterNet HourGlass104 512x512', 'CenterNet HourGlass104 1024x1024', 'CenterNet Resnet50 V2 512x512', 'CenterNet Resnet50 V1 FPN 512x512', 'CenterNet Resnet101 V1 FPN 512x512', 'CenterNet Resnet50 V1 FPN 640x640', 'EfficientDet D0 512x512', 'EfficientDet D1 640x640', 'EfficientDet D2 768x768', 'EfficientDet D3 896x896', 'EfficientDet D4 1024x1024', 'EfficientDet D5 1280x1280', 'EfficientDet D6 1280x1280', 'EfficientDet D7 1536x1536', 'SSD MobileNet v2 320x320', 'SSD MobileNet V1 FPN 640x640', 'SSD MobileNet V2 FPNLite 320x320', 'SSD MobileNet V2 FPNLite 640x640', 'SSD ResNet50 V1 FPN 640x640 (RetinaNet50)', 'SSD ResNet50 V1 FPN 1024x1024 (RetinaNet50)', 'SSD ResNet101 V1 FPN 640x640 (RetinaNet101)', 'SSD ResNet101 V1 FPN 1024x1024 (RetinaNet101)', 'SSD ResNet152 V1 FPN 640x640 (RetinaNet152)', 'SSD ResNet152 V1 FPN 1024x1024 (RetinaNet152)', 'Faster R-CNN ResNet50 V1 640x640', 'Faster R-CNN ResNet50 V1 1024x1024', 'Faster R-CNN ResNet50 V1 800x1333', 'Faster R-CNN ResNet101 V1 640x640', 'Faster R-CNN ResNet101 V1 1024x1024', 'Faster R-CNN ResNet101 V1 800x1333', 'Faster R-CNN ResNet152 V1 640x640', 'Faster R-CNN ResNet152 V1 1024x1024', 'Faster R-CNN ResNet152 V1 800x1333', 'Faster R-CNN Inception ResNet V2 640x640', 'Faster R-CNN Inception ResNet V2 1024x1024', 'Mask R-CNN Inception ResNet V2 1024x1024']
model_handle1 = ALL_MODELS[model_display_name1]

print('Selected model 1:'+ model_display_name1)
print('Model Handle at TensorFlow Hub 1: {}'.format(model_handle1))
```

```
Selected model 1:Mask R-CNN Inception ResNet V2 1024x1024
Model Handle at TensorFlow Hub 1: https://tfhub.dev/tensorflow/mask_rcnn/inception_resnet_v2_1024x1024/1
```

Model 1 Selection

model_display_name1: Mask R-CNN Inception ResNet V2 1024x1

```
#@title Model 2 Selection { display-mode: "form", run: "auto" }
model_display_name2 = 'EfficientDet D7 1536x1536' # @param ['CenterNet HourGlass104 Keypoints 512x512', 'CenterNet HourGlass104 Keypoints 1024x1024', 'CenterNet Resnet50 V1 FPN Keypoints 512x512', 'CenterNet Resnet101 V1 FPN Keypoints 512x512', 'CenterNet HourGlass104 512x512', 'CenterNet HourGlass104 1024x1024', 'CenterNet Resnet50 V2 512x512', 'CenterNet Resnet50 V1 FPN 512x512', 'CenterNet Resnet101 V1 FPN 512x512', 'CenterNet Resnet50 V1 FPN 640x640', 'EfficientDet D0 512x512', 'EfficientDet D1 640x640', 'EfficientDet D2 768x768', 'EfficientDet D3 896x896', 'EfficientDet D4 1024x1024', 'EfficientDet D5 1280x1280', 'EfficientDet D6 1280x1280', 'EfficientDet D7 1536x1536', 'SSD MobileNet v2 320x320', 'SSD MobileNet V1 FPN 640x640', 'SSD MobileNet V2 FPNLite 320x320', 'SSD MobileNet V2 FPNLite 640x640', 'SSD ResNet50 V1 FPN 640x640 (RetinaNet50)', 'SSD ResNet50 V1 FPN 1024x1024 (RetinaNet50)', 'SSD ResNet101 V1 FPN 640x640 (RetinaNet101)', 'SSD ResNet101 V1 FPN 1024x1024 (RetinaNet101)', 'SSD ResNet152 V1 FPN 640x640 (RetinaNet152)', 'SSD ResNet152 V1 FPN 1024x1024 (RetinaNet152)', 'Faster R-CNN ResNet50 V1 640x640', 'Faster R-CNN ResNet50 V1 1024x1024', 'Faster R-CNN ResNet50 V1 800x1333', 'Faster R-CNN ResNet101 V1 640x640', 'Faster R-CNN ResNet101 V1 1024x1024', 'Faster R-CNN ResNet101 V1 800x1333', 'Faster R-CNN ResNet152 V1 640x640', 'Faster R-CNN ResNet152 V1 1024x1024', 'Faster R-CNN ResNet152 V1 800x1333', 'Faster R-CNN Inception ResNet V2 640x640', 'Faster R-CNN Inception ResNet V2 1024x1024', 'Mask R-CNN Inception ResNet V2 1024x1024']
model_handle2 = ALL_MODELS[model_display_name2]

print('Selected model 2:'+ model_display_name2)
print('Model Handle at TensorFlow Hub 2: {}'.format(model_handle2))
```

```
Selected model 2:EfficientDet D7 1536x1536
Model Handle at TensorFlow Hub 2: https://tfhub.dev/tensorflow/efficientdet/d7/1
```

model_display_name2: EfficientDet D7 1536x1536

Loading the selected model from TensorFlow Hub

Here we just need the model handle that was selected and use the Tensorflow Hub library to load it to memory.

```
print('loading model 1...')
hub_model1 = hub.load(model_handle1)
print('model 1 loaded!')
```

```
loading model 1...
model 1 loaded!
```

```
print('loading model 2...')
hub_model2 = hub.load(model_handle2)
print('model 2 loaded!')
```

Doing the inference

To do the inference we just need to call our TF Hub loaded model.

Things you can try:

- Print out `result['detection_boxes']` and try to match the box locations to the boxes in the image. Notice that coordinates are given in normalized form (i.e., in the interval [0, 1]).
- inspect other output keys present in the result. A full documentation can be seen on the models documentation page (pointing your browser to the model handle printed earlier)

```
# running inference
image_np = IMAGES_FOR_TEST['Bike-Race']['image_np']
results = hub_model1(image_np)

# different object detection models have additional results
# all of them are explained in the documentation
result = {key:value.numpy() for key,value in results.items()}
print(result.keys())
```

```
dict_keys(['detection_boxes', 'image_shape', 'rpn_objectness_predictions_with_background', 'detection_scores', 'detect
```

Visualizing the results

Here is where we will need the TensorFlow Object Detection API to show the squares from the inference step (and the keypoints when available).

the full documentation of this method can be seen here

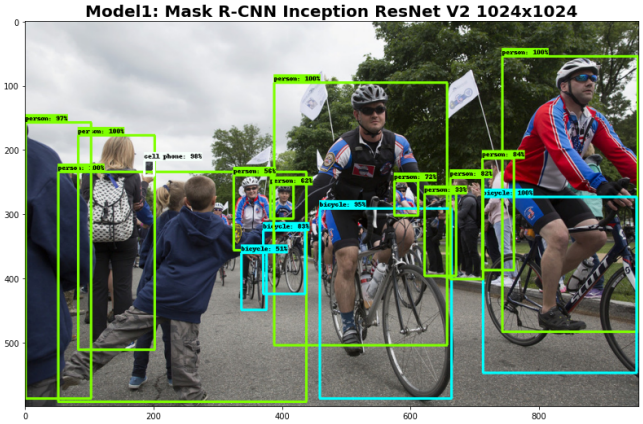
Here you can, for example, set `min_score_thresh` to other values (between 0 and 1) to allow more detections in or to filter out more detections.

```
def plot_results(image_np, result, x, y, axs):
  label_id_offset = 0
```

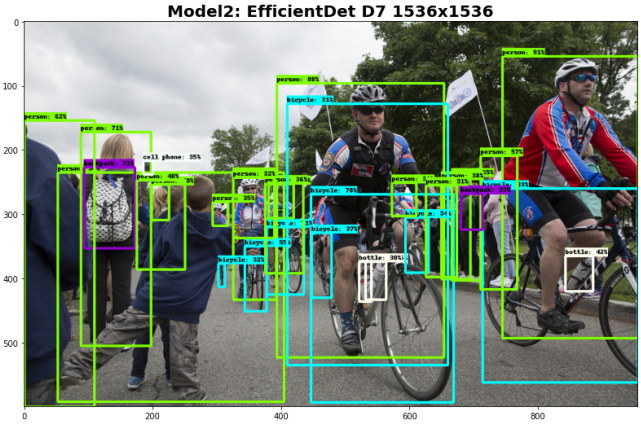
```
  image_np_with_detections = image_np.copy()

  # Use keypoints if available in detections
  keypoints, keypoint_scores = None, None
  if 'detection_keypoints' in result:
    keypoints = result['detection_keypoints'][0]
    keypoint_scores = result['detection_keypoint_scores'][0]

  viz_utils.visualize_boxes_and_labels_on_image_array(
        image_np_with_detections[0],
        result['detection_boxes'][0],
        (result['detection_classes'][0] + label_id_offset).astype(int),
        result['detection_scores'][0],
        category_index,
        use_normalized_coordinates=True,
        max_boxes_to_draw=200,
        min_score_thresh=.30,
        agnostic_mode=False,
        keypoints=keypoints,
        keypoint_scores=keypoint_scores,
        keypoint_edges=COCO17_HUMAN_POSE_KEYPOINTS)

  #plt.figure(figsize=(24,32))
  #plt.figure(figsize=(10,10))
  axs[x, y].imshow(image_np_with_detections[0])


'Model1 ' + model_display_name1

    'Model1 Mask R-CNN Inception ResNet V2 1024x1024'


fig, axs = plt.subplots(len(IMAGES_FOR_TEST), 2, figsize=(30,40))
axs[0, 0].set_title('Model1: ' + model_display_name1, fontweight="bold", size=20)
axs[0, 1].set_title('Model2: ' + model_display_name2, fontweight="bold", size=20)

y = 0
for key in IMAGES_FOR_TEST:
  print(key)
  value = IMAGES_FOR_TEST[key]
  image_np = value['image_np']
  results1 = hub_model1(image_np)
  result1 = {key:value.numpy() for key,value in results1.items()}
  plot_results(image_np, result1, y, 0, axs)
  results2 = hub_model2(image_np)
  result2 = {key:value.numpy() for key,value in results2.items()}

  plot_results(image_np, result2, y, 1, axs)
  y += 1
plt.show()
```

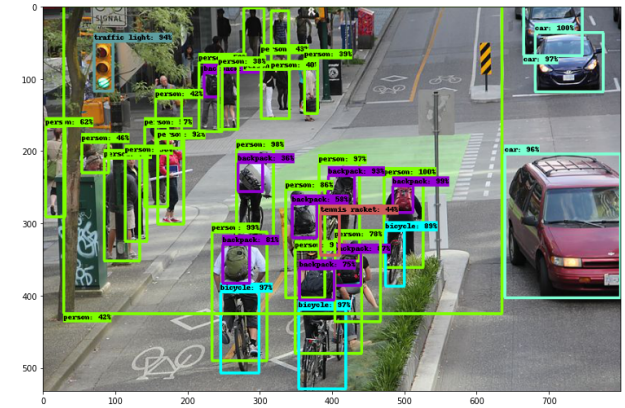
Bike-Race
Street1
Naxos_Taverna
Kalua-Bay

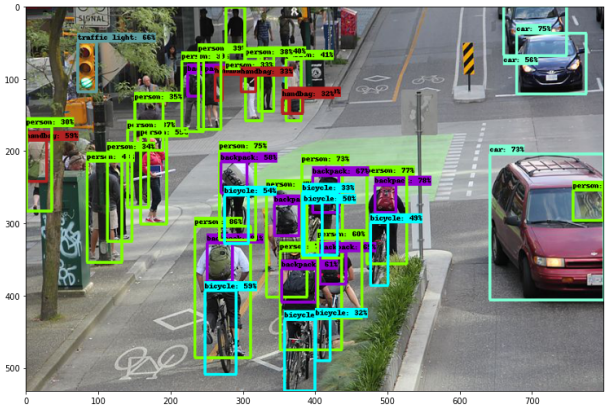
Model1: Mask R-CNN Inception ResNet V2 1024x1024

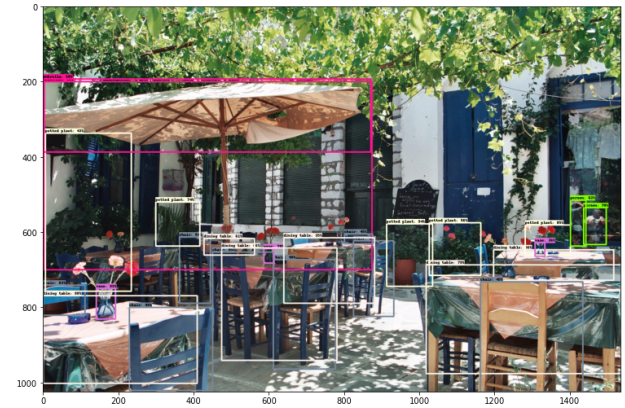


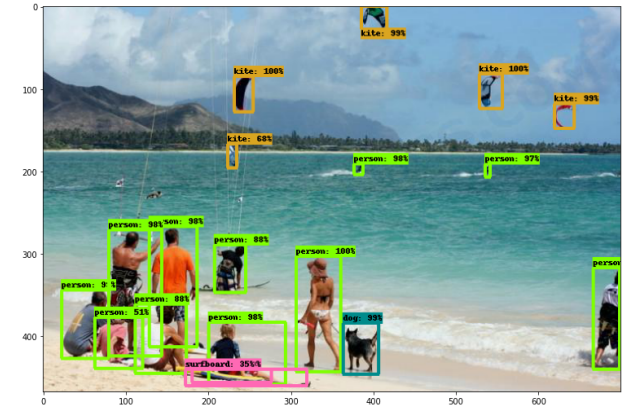
Model2: EfficientDet D7 1536x1536

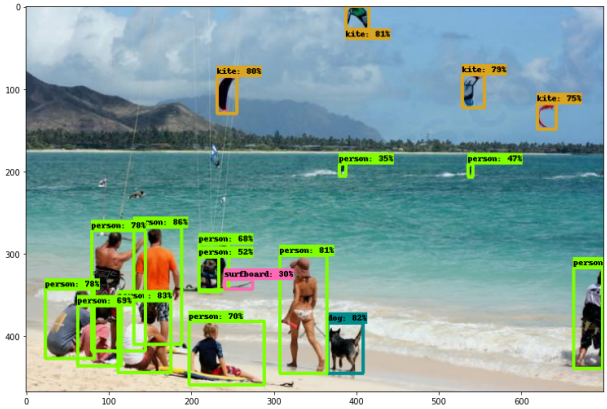












[Optional]

Among the available object detection models there's Mask R-CNN and the output of this model allows instance segmentation.

To visualize it we will use the same method we did before but adding an aditional parameter:

```
 instance_masks=output_dict.get('detection_masks_reframed', None)
```

```
# Handle models with masks:
image_np_with_mask = image_np.copy()

if 'detection_masks' in result:
  # we need to convert np.arrays to tensors
  detection_masks = tf.convert_to_tensor(result['detection_masks'][0])
  detection_boxes = tf.convert_to_tensor(result['detection_boxes'][0])

  # Reframe the the bbox mask to the image size.
  detection_masks_reframed = utils_ops.reframe_box_masks_to_image_masks(
            detection_masks, detection_boxes,
              image_np.shape[1], image_np.shape[2])
  detection_masks_reframed = tf.cast(detection_masks_reframed > 0.5,
                                     tf.uint8)
  result['detection_masks_reframed'] = detection_masks_reframed.numpy()

viz_utils.visualize_boxes_and_labels_on_image_array(
      image_np_with_mask[0],
      result['detection_boxes'][0],
      (result['detection_classes'][0] + label_id_offset).astype(int),
      result['detection_scores'][0],
      category_index,
      use_normalized_coordinates=True,
      max_boxes_to_draw=200,
      min_score_thresh=.30,
      agnostic_mode=False,
      instance_masks=result.get('detection_masks_reframed', None),
      line_thickness=8)

#plt.figure(figsize=(24,32))
plt.figure(figsize=(15,15))

plt.imshow(image_np_with_mask[0])
plt.show()
```