In [ ]:
```
Welcome to the object Detection API!
```

Setup

In [ ]:
```
The setup has been mentioned in the readme file
```

Install

In [ ]:
```
!pip install -U --pre tensorflow=="2.*"
```

Make sure you have `pycocotools` installed

In [ ]:
```
!pip install pycocotools
```

Get `tensorflow/models` or `cd` to parent directory of the repository.

In [6]:
```
import os
import pathlib


if "models" in pathlib.Path.cwd().parts:
  while "models" in pathlib.Path.cwd().parts:
    os.chdir('..')
elif not pathlib.Path('models').exists():
  !git clone --depth 1 https://github.com/tensorflow/models
```

Compile protobufs and install the object_detection package

In [7]:
```
%%bash
cd models/research/
object_detection/protos/*.proto --python_out=.
```

```
Couldn't find program: 'bash'
```

In [9]:
```
%%bash
cd models/research
pip install .
```

```
Couldn't find program: 'bash'
```

Imports

In [15]:
```
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
from IPython.display import display
```

Import the object detection module.

In [16]:
```
from object_detection.utils import ops as utils_ops
from object_detection.utils import label_map_util
from object_detection.utils import visualization_utils as vis_util
```

Patches:

In [17]:
```
# patch tf1 into `utils.ops`
utils_ops.tf = tf.compat.v1

# Patch the location of gfile
tf.gfile = tf.io.gfile
```

Model preparation

Variables

Any model exported using the `export_inference_graph.py` tool can be loaded here simply by changing the path.

By default we use an "SSD with Mobilenet" model here. See the detection model zoo for a list of other models that can be run out-of-the-box with varying speeds and accuracies.

Loader

In [18]:
```
def load_model(model_name):
  base_url = 'http://download.tensorflow.org/models/object_detection/'
  model_file = model_name + '.tar.gz'
  model_dir = tf.keras.utils.get_file(
    fname=model_name,
    origin=base_url + model_file,
    untar=True)

  model_dir = pathlib.Path(model_dir)/"saved_model"

  model = tf.saved_model.load(str(model_dir))
  model = model.signatures['serving_default']

  return model
```

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 [19]:
```
# List of the strings that is used to add correct label for each box.
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_disp
lay_name=True)
```

For the sake of simplicity we will test on 2 images:

In [20]:
```
# If you want to test the code with your images, just add path to the images to the TEST_IMA
GE_PATHS.
PATH_TO_TEST_IMAGES_DIR = pathlib.Path('models/research/object_detection/test_images')
TEST_IMAGE_PATHS = sorted(list(PATH_TO_TEST_IMAGES_DIR.glob("*.jpg")))
TEST_IMAGE_PATHS
```

Out[20]:
```
[WindowsPath('models/research/object_detection/test_images/image1.jpg'),
 WindowsPath('models/research/object_detection/test_images/image2.jpg')]
```

Detection

Load an object detection model:

In [21]:
```
model_name = 'ssd_mobilenet_v1_coco_2017_11_17'
detection_model = load_model(model_name)
```

```
INFO:tensorflow:Saver not created because there are no variables in the graph to restore
```

Check the model's input signature, it expects a batch of 3-color images of type uint8:

In [22]:
```
print(detection_model.inputs)
```

```
[<tf.Tensor 'image_tensor:0' shape=(None, None, None, 3) dtype=uint8>]
```

And returns several outputs:

In [23]:
```
detection_model.output_dtypes
```

Out[23]:
```
{'detection_scores': tf.float32,
 'detection_classes': tf.float32,
 'num_detections': tf.float32,
 'detection_boxes': tf.float32}
```

In [24]:
```
detection_model.output_shapes
```

Out[24]:
```
{'detection_scores': TensorShape([None, 100]),
 'detection_classes': TensorShape([None, 100]),
 'num_detections': TensorShape([None]),
 'detection_boxes': TensorShape([None, 100, 4])}
```

Add a wrapper function to call the model, and cleanup the outputs:

In [25]:
```
def run_inference_for_single_image(model, image):
  image = np.asarray(image)
  # The input needs to be a tensor, convert it using `tf.convert_to_tensor`.
  input_tensor = tf.convert_to_tensor(image)
  # The model expects a batch of images, so add an axis with `tf.newaxis`.
  input_tensor = input_tensor[tf.newaxis,...]

  # Run inference
  output_dict = model(input_tensor)

  # All outputs are batches tensors.
  # Convert to numpy arrays, and take index [0] to remove the batch dimension.
  # We're only interested in the first num_detections.
  num_detections = int(output_dict.pop('num_detections'))
  output_dict = {key:value[0, :num_detections].numpy()
                 for key,value in output_dict.items()}
  output_dict['num_detections'] = num_detections

  # detection_classes should be ints.
  output_dict['detection_classes'] = output_dict['detection_classes'].astype(np.int64)

  # Handle models with masks:
  if 'detection_masks' in output_dict:
    # Reframe the the bbox mask to the image size.
    detection_masks_reframed = utils_ops.reframe_box_masks_to_image_masks(
              output_dict['detection_masks'], output_dict['detection_boxes'],
               image.shape[0], image.shape[1])
    detection_masks_reframed = tf.cast(detection_masks_reframed > 0.5,
                                       tf.uint8)
    output_dict['detection_masks_reframed'] = detection_masks_reframed.numpy()

  return output_dict
```

Run it on each test image and show the results:

In [26]:
```
def show_inference(model, 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 = np.array(Image.open(image_path))
  # Actual detection.
  output_dict = run_inference_for_single_image(model, image_np)
  # Visualization of the results of a detection.
  vis_util.visualize_boxes_and_labels_on_image_array(
      image_np,
      output_dict['detection_boxes'],
      output_dict['detection_classes'],
      output_dict['detection_scores'],
      category_index,
      instance_masks=output_dict.get('detection_masks_reframed', None),
      use_normalized_coordinates=True,
      line_thickness=8)

  display(Image.fromarray(image_np))
```

In [27]:
```
for image_path in TEST_IMAGE_PATHS:
  show_inference(detection_model, image_path)
```

Instance Segmentation

In [28]:
```
model_name = "mask_rcnn_inception_resnet_v2_atrous_coco_2018_01_28"
masking_model = load_model("mask_rcnn_inception_resnet_v2_atrous_coco_2018_01_28")
```

```
INFO:tensorflow:Saver not created because there are no variables in the graph to restore
```

The instance segmentation model includes a `detection_masks` output:

In [29]:
```
masking_model.output_shapes
```

Out[29]:
```
{'detection_scores': TensorShape([None, 100]),
 'detection_classes': TensorShape([None, 100]),
 'detection_masks': TensorShape([None, None, None, None]),
 'num_detections': TensorShape([None]),
 'detection_boxes': TensorShape([None, 100, 4])}
```

In [30]:
```
for image_path in TEST_IMAGE_PATHS:
  show_inference(masking_model, image_path)
```


In [ ]:
```

```