3 Object Detection

Object Detection Demo

Welcome to the object detection inference walkthrough! This notebook will walk you step by step through the process of using a pre-trained model to detect objects in an image. Make sure to follow the installation instructions (https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/installation.md) before you start.

Imports

In [229]:
```
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

# This is needed since the notebook is stored in the object_detection fo
lder.
sys.path.append("..")
from object_detection.utils import ops as utils_ops

if tf.__version__ < '1.4.0':
  raise ImportError('Please upgrade your tensorflow installation to v1.
4.* or later!')
```

Env setup

In [2]:
```
# This is needed to display the images.
%matplotlib inline
```

Object detection imports

Here are the imports from the object detection module.

In [230]:
```
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/research/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]:
```
# What model to download.
MODEL_NAME = 'ssd_mobilenet_v1_coco_2017_11_17'
MODEL_FILE = MODEL_NAME + '.tar.gz'
DOWNLOAD_BASE = 'http://download.tensorflow.org/models/object_detectio
n/'

# Path to frozen detection graph. This is the actual model that is used
 for the object 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]:
```
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())
```

Change to other Models

In [225]:
```
#MODEL_NAME = 'ssd_mobilenet_v1_coco_2017_11_17'
MODEL_NAME = 'mask_rcnn_inception_v2_coco_2018_01_28'
#MODEL_NAME = 'faster_rcnn_inception_v2_coco_2018_01_28'
#MODEL_NAME = 'rfcn_resnet101_coco_2018_01_28'
PATH_TO_CKPT = MODEL_NAME + '/frozen_inference_graph.pb'
MODEL_NAME_ABB = MODEL_NAME.split('_')[0] + '_' + MODEL_NAME.split('_')[
1] if MODEL_NAME.split('_')[1] == 'rcnn' else  MODEL_NAME.split('_')[0]
```

Load a (frozen) Tensorflow model into memory.

In [226]:
```
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 [20]:
```
label_map = label_map_util.load_labelmap(PATH_TO_LABELS)
categories = label_map_util.convert_label_map_to_categories(label_map, m
ax_num_classes=NUM_CLASSES, use_display_name=True)
category_index = label_map_util.create_category_index(categories)
```

Helper code

In [8]:
```
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)
```

Image collages

In [160]:
```
def img_collage(input_dir,output_dir,n):
    image = Image.open(input_dir)
    image_np = load_image_into_numpy_array(image)
    image_n = np.tile(image_np,(n,n,1))
    image_n = Image.fromarray(image_n)
    image_n = image_n.resize((image_np.shape[1]*4,image_np.shape[0]*4))
    image_n.save(output_dir)
```

Image rotation

In [68]:
```
def img_rotation(input_dir,output_dir,degree):
    image = Image.open(input_dir)
    image_rotate = image.rotate(degree)
    image_rotate.save(output_dir)
```

Detection

In [227]:
```
# For the sake of simplicity we will use only 2 images:
# image1.jpg
# image2.jpg
# If you want to test the code with your images, just add path to the im
ages to the TEST_IMAGE_PATHS.
PATH_TO_TEST_IMAGES_DIR = 'test_images'
#TEST_IMAGE_PATHS = [ os.path.join(PATH_TO_TEST_IMAGES_DIR, 'image{}.jp
g'.format(i)) for i in range(1, 3) ]
img_names = [img_name for img_name in os.listdir(PATH_TO_TEST_IMAGES_DIR
) if img_name.endswith('.jpg') or img_name.endswith('.png')]
TEST_IMAGE_PATHS = [ os.path.join(PATH_TO_TEST_IMAGES_DIR, img_name) for
 img_name in img_names ]

# Size, in inches, of the output images.
IMAGE_SIZE = (15, 10)
```

In [162]:
```
for img_name in TEST_IMAGE_PATHS:
    dir_parts = img_name.split('/')
    for n in [4,8,16]:
        img_parts = dir_parts[1].split('.')
        output_name = dir_parts[0] + '/collages/' + img_parts[0] + '_' +
 str(n) + '.' + img_parts[1]
        img_collage(img_name,output_name,n)
    for degree in range(30,360,30):
        img_parts = dir_parts[1].split('.')
        output_name = dir_parts[0] + '/rotation/' + img_parts[0] + '_' +
 str(degree) + '.' + img_parts[1]
        img_rotation(img_name,output_name,degree)
```

In [22]:
```
def run_inference_for_single_image(image, graph):
  with graph.as_default():
    with tf.Session() as sess:
      # Get handles to input and output tensors
      ops = tf.get_default_graph().get_operations()
      all_tensor_names = {output.name for op in ops for output in op.out
puts}
      tensor_dict = {}
      for key in [
          'num_detections', 'detection_boxes', 'detection_scores',
          'detection_classes', 'detection_masks'
      ]:
        tensor_name = key + ':0'
        if tensor_name in all_tensor_names:
          tensor_dict[key] = tf.get_default_graph().get_tensor_by_name(
              tensor_name)
      if 'detection_masks' in tensor_dict:
        # The following processing is only for single image
        detection_boxes = tf.squeeze(tensor_dict['detection_boxes'], [0
])
        detection_masks = tf.squeeze(tensor_dict['detection_masks'], [0
])
        # Reframe is required to translate mask from box coordinates to
 image coordinates and fit the image size.
        real_num_detection = tf.cast(tensor_dict['num_detections'][0], t
f.int32)
        detection_boxes = tf.slice(detection_boxes, [0, 0], [real_num_de
tection, -1])
        detection_masks = tf.slice(detection_masks, [0, 0, 0], [real_num
_detection, -1, -1])
        detection_masks_reframed = utils_ops.reframe_box_masks_to_image_
masks(
            detection_masks, detection_boxes, image.shape[0], image.shap
e[1])
        detection_masks_reframed = tf.cast(
            tf.greater(detection_masks_reframed, 0.5), tf.uint8)
        # Follow the convention by adding back the batch dimension
        tensor_dict['detection_masks'] = tf.expand_dims(
            detection_masks_reframed, 0)
      image_tensor = tf.get_default_graph().get_tensor_by_name('image_te
nsor:0')

      # Run inference
      output_dict = sess.run(tensor_dict,
                             feed_dict={image_tensor: np.expand_dims(ima
ge, 0)})

      # all outputs are float32 numpy arrays, so convert types as approp
riate
      output_dict['num_detections'] = int(output_dict['num_detections'][
0])
      output_dict['detection_classes'] = output_dict[
          'detection_classes'][0].astype(np.uint8)
      output_dict['detection_boxes'] = output_dict['detection_boxes'][0]
      output_dict['detection_scores'] = output_dict['detection_scores'][
0]
```

```
      if 'detection_masks' in output_dict:
        output_dict['detection_masks'] = output_dict['detection_masks'][
0]
  return output_dict
```

In [228]:
```
for image_path in TEST_IMAGE_PATHS:
  image = Image.open(image_path)
  # the array based representation of the image will be used later in or
der 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,
 None, None, 3]
  image_np_expanded = np.expand_dims(image_np, axis=0)
  # Actual detection.
  output_dict = run_inference_for_single_image(image_np, detection_graph
)
  # 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'),
      use_normalized_coordinates=True,
      line_thickness=8)
  plt.figure(figsize=IMAGE_SIZE)
  plt.imshow(image_np)
  plt.axis('off')
  plt.gca().xaxis.set_major_locator(plt.NullLocator())
  plt.gca().yaxis.set_major_locator(plt.NullLocator())
  #plt.subplots_adjust(top=1,bottom=0,right=1,left=0,hspace=0,wspace=0)
  #plt.margins(0,0)
  dir_parts = image_path.split('/')
  new_dir = dir_parts[0]+'/'+MODEL_NAME_ABB+'_detection_'+dir_parts[1]
  #new_dir = dir_parts[0]+'/'+dir_parts[1]+'/'+MODEL_NAME_ABB+'_detectio
n_'+dir_parts[2]
  plt.savefig(new_dir,dpi=100,bbox_inches='tight',pad_inches=0)
  image.close()
```

cat: 99%

dog: 99%
dog: 98%
cow: 57%
person: 32%

cell phone: 73%
book: 73%
scissors: 52%
cup: 95%

kite: 99%
kite: 94%
kite: 56%
kite: 84%
kite: 73%
kite: 65%
person: 54%
person: 87%
person: 57%
person: 50%
person: 99%
person: 99%

airplane: 93%
airplane: 69%
airplane: 62%
airplane: 61%
truck: 94%
truck
airplane: 55%

dog: 99%

person: 99%
person: 96%
car: 99%
perso

bottle: 94%
book: 61%
bottle: 94%
cup: 62%
dining tab
book: 75%
chair: 74%
chair: 88%

Results for scale invariance and rotation invariance are included in the report.