Faculty of Engineering

Credit Hours Engineering Programs

Mechatronics Engineering and Automation

Academic Year 2019/2020 – Spring 2019

CSE 489

Machine Vision

Project No. (4)

Deep learning and object detection

Submitted by:

----------	---
----------	---
----------	---
----------	---
----------	---

Contents

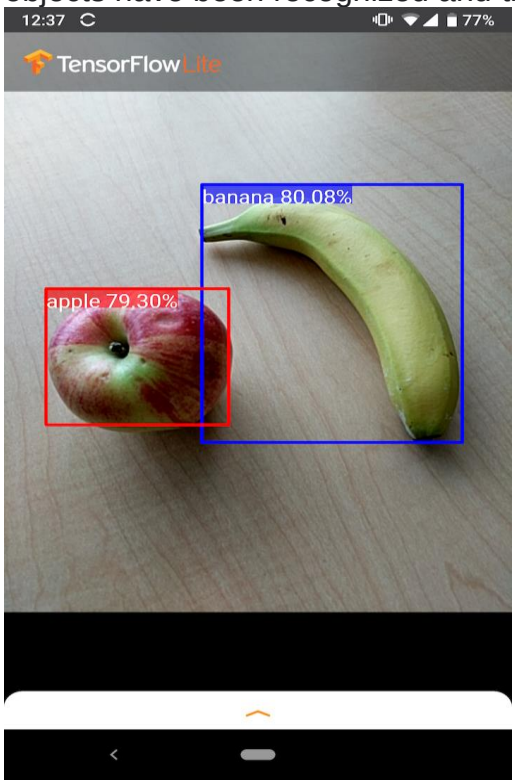
1 PROBLEM DEFINITON AND IMPORTANCE

Creating accurate Machine Learning Models which are capable of identifying and localizing multiple objects in a single image remained a core challenge in computer vision. But, with recent advancements in **Deep Learning, Object Detection** applications are easier to develop than ever before. TensorFlow's Object Detection API is an open source framework built on top of TensorFlow that makes it easy to construct, train and deploy object detection models.

1.1 WHAT IS OBJECT DETECTION?

Given an image or a video stream, an object detection model can identify which of a known set of objects might be present and provide information about their positions within the image.

For example, this screenshot of our example application shows how two objects have been recognized and their positions annotated:



An object detection model is trained to detect the presence and location of multiple classes of objects. For example, a model might be trained with images that contain various pieces of fruit, along with a *label* that specifies

the class of fruit they represent (e.g. an apple, a banana, or a strawberry), and data specifying where each object appears in the image.

When we subsequently provide an image to the model, it will output a list of the objects it detects, the location of a bounding box that contains each object, and a score that indicates the confidence that detection was correct.

1.1.1 Model output

Imagine a model has been trained to detect apples, bananas, and strawberries. When we pass it an image, it will output a set number of detection results - in this example.

Class	Score	Location
Apple	0.92	[18, 21, 57, 63]
Banana	0.88	[100, 30, 180, 150]
Strawberry	0.87	[7, 82, 89, 163]
Banana	0.23	[42, 66, 57, 83]
Apple	0.11	[6, 42, 31, 58]

1.1.2 Confidence score

To interpret these results, we can look at the score and the location for each detected object. The score is a number between 0 and 1 that indicates confidence that the object was genuinely detected. The closer the number is to 1, the more confident the model is.

Depending on your application, you can decide a cut-off threshold below which you will discard detection results. For our example, we might decide a sensible cut-off is a score of 0.5 (meaning a 50% probability that the detection is valid). In that case, we would ignore the last two objects in the array, because those confidence scores are below 0.5:

Class	Score	Location
Apple	0.92	[18, 21, 57, 63]
Banana	0.88	[100, 30, 180, 150]
Strawberry	0.87	[7, 82, 89, 163]
~~Banana~~	~~0.23~~	~~[42, 66, 57, 83]~~
~~Apple~~	~~0.11~~	~~[6, 42, 31, 58]~~

1.1.3 Uses and limitations

The object detection model we provide can identify and locate up to 10 objects in an image. It is trained to recognize 80 classes of object. For a full list of classes, see the labels file in the model zip.

If you want to train a model to recognize new classes, see Customize model.

For the following use cases, you should use a different type of model:

- Predicting which single label the image most likely represents (see image classification)
- Predicting the composition of an image, for example subject versus background (see segmentation)

2 METHODS AND ALGORITHM

2.1 TRAINING RCNN

What is the input to an RCNN?

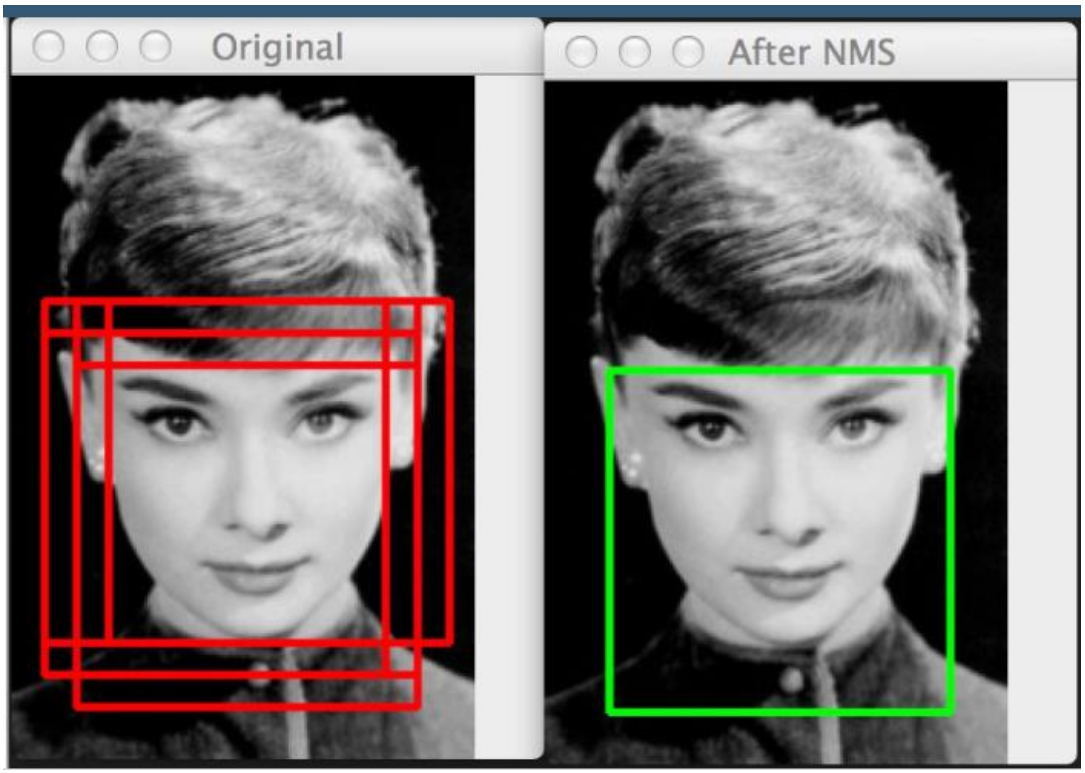
So we have got an image, Region Proposals from the RPN strategy and the ground truths of the labels (labels, ground truth boxes)

Next we treat all region proposals with ≥ 0.5 IoU(Intersection over union) overlap with a ground-truth box as positive training example for that box's class and the rest as negative. We train class specific SVM's

So every region proposal becomes a training example. and the convnet gives a feature vector for that region proposal. We can then train our n-SVMs using the class specific data.

2.2 TEST TIME RCNN

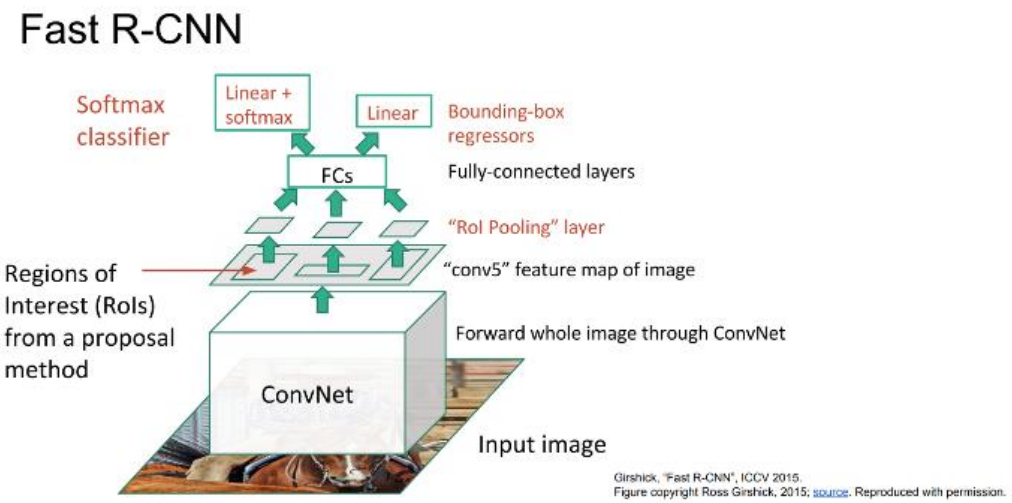
At test time we predict detection boxes using class specific SVMs. We will be getting a lot of overlapping detection boxes at the time of testing. Non-maximum suppression is an integral part of the object detection pipeline. First, it sorts all detection boxes on the basis of their scores. The detection box M with the maximum score is selected and all other detection boxes with a significant overlap (using a pre-defined threshold) with M are suppressed. This process is recursively applied on the remaining boxes



2.3 PROBLEMS WITH RCNN:

- Training is slow.
- Inference (detection) is slow. 47s / image with VGG16—Since the Convnet needs to be run many times.

2.4 FAST RCNN



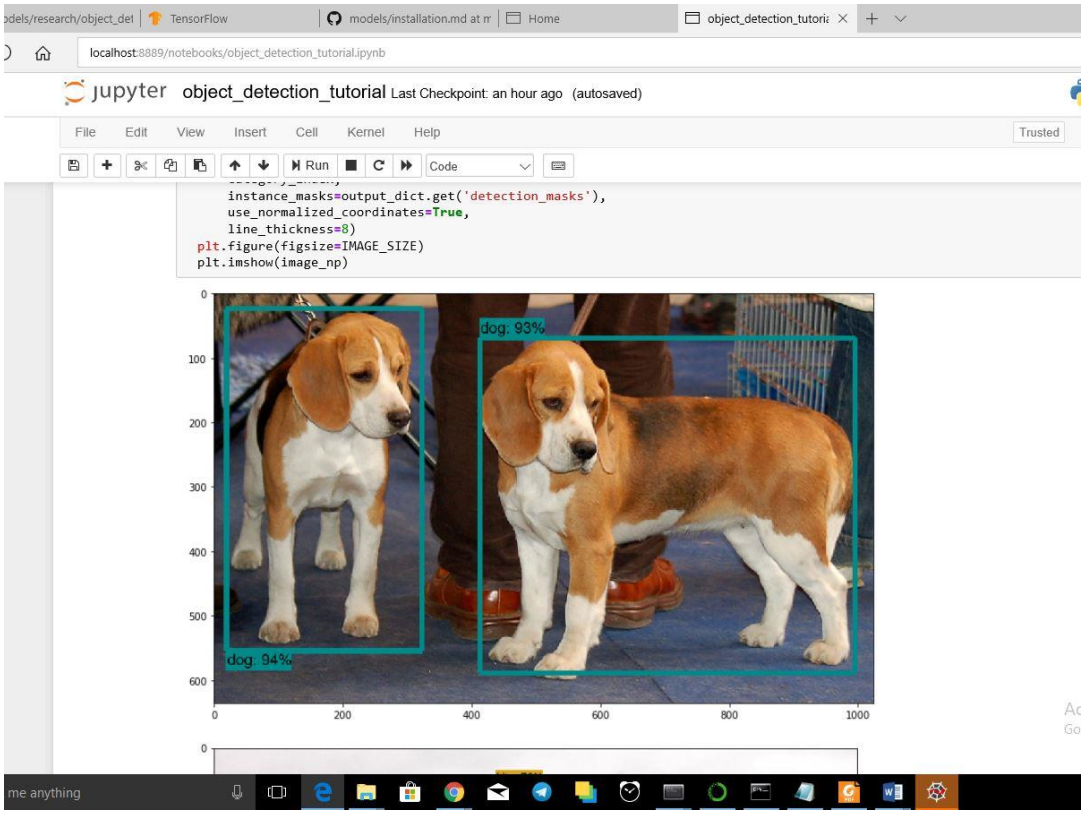
illustrates the Fast R-CNN architecture. A Fast R-CNN network takes as input an entire image and a set of object proposals. The network first processes the whole image with several convolutional (conv) and max pooling layers to produce a conv feature map. Then, for each object proposal a region of interest (RoI) pooling layer extracts a fixed-length feature vector from the feature map. Each feature vector is fed into a sequence of fully connected (fc) layers that finally branch into two sibling output layers: one that produces softmax probability estimates over K object classes plus a catch-all "background" class and another layer that outputs four real-valued numbers for each of the K object classes. Each set of 4 values encodes refined bounding-box positions for one of the K classes.

The last pooling layer is 7x7x512. This is the layer the network authors intend to replace by the ROI pooling layers. This pooling layer has got as input the location of the region proposal (xmin_roi,ymin_roi,h_roi,w_roi) and the previous feature map(14x14x512).

A Fast R-CNN network has two sibling output layers. The first outputs a discrete probability distribution (per RoI), p = (p0, . . . , pK), over K + 1 categories. As usual, p is computed by a softmax over the K+1 outputs of a fully connected layer. The second sibling layer outputs bounding-box regression offsets, t= (tx , ty , tw, th), for each of the K object classes. Each training RoI is labeled with a ground-truth class u and a ground-truth bounding-box regression target v. We use a multi-task loss L on each labeled RoI to jointly train for classification and bounding-box regression

3 EXPERIMENTAL RESULTS AND DISCUSSION

3.1 DEMO FOR TENSORFLOW OBJECT DETECTION API



3.2 REAL TIME VIDEO DETECTION

There were multiple errors that prevented us from running the code. The errors were mainly problems in installations that we could not solve in the given time.

4 APPENDIX

4.1 OBJECT DETECTION SAMPLE CODE

```
#!/usr/bin/env python

# coding: utf-8



# # 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[16]:




import numpy as np

import os

import six.moves.urllib as urllib

import sys
```

```
import tarfile
import tensorflow as tf
import zipfile

from distutils.version import StrictVersion
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 folder.
sys.path.append("..")
from object_detection.utils import ops as utils_ops

if StrictVersion(tf.__version__) < StrictVersion('1.12.0'):
  raise ImportError('Please upgrade your TensorFlow installation to v1.12.*.')



# ## Env setup

# In[17]:
```

```
# This is needed to display the images.
get_ipython().run_line_magic('matplotlib', 'inline')



# ## Object detection imports
# Here are the imports from the object detection module.

# In[18]:


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_FROZEN_GRAPH` 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[19]:

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_detection/'

Path to frozen detection graph. This is the actual model that is used for the object detection.

PATH_TO_FROZEN_GRAPH = 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')

Download Model

In[20]:

```
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[21]:

```
detection_graph = tf.Graph()
```

```
with detection_graph.as_default():
  od_graph_def = tf.GraphDef()
  with tf.gfile.GFile(PATH_TO_FROZEN_GRAPH, '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[22]:



category_index =
label_map_util.create_category_index_from_labelmap(PATH_TO_LABELS,
use_display_name=True)



# ## Helper code
```

In[23]:

```
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[24]:

```
# 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 images to the TEST_IMAGE_PATHS.
PATH_TO_TEST_IMAGES_DIR = 'test_images'
```

```
TEST_IMAGE_PATHS = [ os.path.join(PATH_TO_TEST_IMAGES_DIR, 'image{}.jpg'.format(i)) for i in range(1, 3) ]

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

# In[25]:

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.outputs}
      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], tf.int32)
        detection_boxes = tf.slice(detection_boxes, [0, 0], [real_num_detection, -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[1], image.shape[2])
        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_tensor:0')
```

```
      # Run inference
      output_dict = sess.run(tensor_dict,
                             feed_dict={image_tensor: image})

      # all outputs are float32 numpy arrays, so convert types as appropriate
      output_dict['num_detections'] = int(output_dict['num_detections'][0])
      output_dict['detection_classes'] = output_dict[
          'detection_classes'][0].astype(np.int64)
      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[26]:


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, None, None, 3]
  image_np_expanded = np.expand_dims(image_np, axis=0)
  # Actual detection.
  output_dict = run_inference_for_single_image(image_np_expanded, 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)
```

4.2 OBJECT DETECTION FROM A VIDEO

```
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


import cv2

cap = cv2.VideoCapture(“testvideo.mp4”)


# 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) 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_11_06_2017'

MODEL_FILE = MODEL_NAME + '.tar.gz'

DOWNLOAD_BASE = 'http://download.tensorflow.org/models/object_detection/'

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())



# ## 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_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)
```

Detection

In[9]:

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 images to the TEST_IMAGE_PATHS.

PATH_TO_TEST_IMAGES_DIR = 'test_images'

TEST_IMAGE_PATHS = [os.path.join(PATH_TO_TEST_IMAGES_DIR, 'image{}.jpg'.format(i)) for i in range(1, 3)]

Size, in inches, of the output images.

IMAGE_SIZE = (12, 8)

In[10]:

with detection_graph.as_default():

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

```
while True:
    ret, image_np = cap.read()

    # Expand dimensions since the model expects images to have shape: [1, None, 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 detected.
    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=8)

cv2.imshow('object detection', cv2.resize(image_np, (800,400)))
if cv2.waitKey(25) & 0xFF == ord('q'):
  cv2.destroyAllWindows()
  cap.release()
  break
```