SESSION 11 - Project on Object Detection - Part I

TOC

PART I

Real-time object detection and tracking is a vast, vibrant yet inconclusive and complex area of computer vision. Due to its increased utilization in surveillance, tracking system used in security and many others applications have propelled researchers to continuously devise more efficient and competitive algorithms. The latest research on this area has been making great progress in many directions. In the current manuscript, we give an overview of the necessity of object detection in today's computing systems, outline the current main research directions, discuss on how our API algorithm works, and discuss open problems and possible future directions. Key words: - Object Detection, Computer Vision, Tracking Systems, API algorithm.

1. Setup

Packages to be installed:

```
1. pip install protobuf
2. pip install pillow
3. pip install lxml
4. pip install Cython
5. pip install jupyter
6. pip install matplotlib
7. pip install pandas
8. pip install opencv-python
9. pip install tensorflow
```

2. Install

```
pip install -U --pre tensorflow=="2.*"
```

Make sure you have `pycocotools` installed

```
pip install pycocotools
```

Python Implementation

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

In [ ]:

```
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
```

```
Cloning into 'models'...
Updating files:   2% (100/3487)
Updating files:   3% (105/3487)
Updating files:   4% (140/3487)
Updating files:   4% (171/3487)
Updating files:   5% (175/3487)
Updating files:   6% (210/3487)
Updating files:   6% (236/3487)
Updating files:   7% (245/3487)
Updating files:   8% (279/3487)
Updating files:   9% (314/3487)
Updating files:  10% (349/3487)
Updating files:  11% (384/3487)
Updating files:  11% (393/3487)
Updating files:  12% (419/3487)
Updating files:  13% (454/3487)
Updating files:  14% (489/3487)
Updating files:  15% (524/3487)
Updating files:  15% (556/3487)
Updating files:  16% (558/3487)
Updating files:  17% (593/3487)
Updating files:  18% (628/3487)
Updating files:  19% (663/3487)
Updating files:  19% (677/3487)
```

```
Updating files:  20% (698/3487)
Updating files:  21% (733/3487)
Updating files:  22% (768/3487)
Updating files:  23% (803/3487)
Updating files:  24% (837/3487)
Updating files:  25% (872/3487)
Updating files:  25% (877/3487)
Updating files:  26% (907/3487)
Updating files:  27% (942/3487)
Updating files:  28% (977/3487)
Updating files:  29% (1012/3487)
Updating files:  29% (1037/3487)
Updating files:  30% (1047/3487)
Updating files:  31% (1081/3487)
Updating files:  32% (1116/3487)
Updating files:  33% (1151/3487)
Updating files:  33% (1177/3487)
Updating files:  34% (1186/3487)
Updating files:  35% (1221/3487)
Updating files:  36% (1256/3487)
Updating files:  37% (1291/3487)
Updating files:  38% (1326/3487)
Updating files:  38% (1345/3487)
Updating files:  39% (1360/3487)
Updating files:  39% (1394/3487)
Updating files:  40% (1395/3487)
Updating files:  41% (1430/3487)
Updating files:  42% (1465/3487)
Updating files:  43% (1500/3487)
Updating files:  44% (1535/3487)
Updating files:  44% (1546/3487)
Updating files:  45% (1570/3487)
Updating files:  46% (1605/3487)
Updating files:  47% (1639/3487)
Updating files:  48% (1674/3487)
Updating files:  49% (1709/3487)
Updating files:  50% (1744/3487)
Updating files:  51% (1779/3487)
Updating files:  51% (1788/3487)
Updating files:  52% (1814/3487)
Updating files:  53% (1849/3487)
Updating files:  54% (1883/3487)
Updating files:  55% (1918/3487)
Updating files:  56% (1953/3487)
Updating files:  56% (1985/3487)
Updating files:  57% (1988/3487)
Updating files:  58% (2023/3487)
```

```
Updating files:  59% (2058/3487)
Updating files:  60% (2093/3487)
Updating files:  61% (2128/3487)
Updating files:  62% (2162/3487)
Updating files:  62% (2191/3487)
Updating files:  63% (2197/3487)
Updating files:  64% (2232/3487)
Updating files:  65% (2267/3487)
Updating files:  66% (2302/3487)
Updating files:  67% (2337/3487)
Updating files:  67% (2342/3487)
Updating files:  68% (2372/3487)
Updating files:  69% (2407/3487)
Updating files:  70% (2441/3487)
Updating files:  71% (2476/3487)
Updating files:  72% (2511/3487)
Updating files:  72% (2542/3487)
Updating files:  73% (2546/3487)
Updating files:  74% (2581/3487)
Updating files:  75% (2616/3487)
Updating files:  76% (2651/3487)
Updating files:  77% (2685/3487)
Updating files:  77% (2715/3487)
Updating files:  78% (2720/3487)
Updating files:  79% (2755/3487)
Updating files:  80% (2790/3487)
Updating files:  81% (2825/3487)
Updating files:  81% (2854/3487)
Updating files:  82% (2860/3487)
Updating files:  83% (2895/3487)
Updating files:  84% (2930/3487)
Updating files:  85% (2964/3487)
Updating files:  86% (2999/3487)
Updating files:  87% (3034/3487)
Updating files:  88% (3069/3487)
Updating files:  88% (3086/3487)
Updating files:  89% (3104/3487)
Updating files:  90% (3139/3487)
Updating files:  91% (3174/3487)
Updating files:  92% (3209/3487)
Updating files:  92% (3219/3487)
Updating files:  93% (3243/3487)
Updating files:  94% (3278/3487)
Updating files:  95% (3313/3487)
Updating files:  96% (3348/3487)
Updating files:  97% (3383/3487)
Updating files:  97% (3401/3487)
```

```
Updating files:  98% (3418/3487)
Updating files:  99% (3453/3487)
Updating files: 100% (3487/3487)
Updating files: 100% (3487/3487), done.
```

Compile protobufs and install the object_detection package

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

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

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

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

3. Imports

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

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

4. 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.

4.1 Loader

In [ ]:
```
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
```

4.2 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 [ ]:
```
# 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_display_name=True)
```

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

In [ ]:
```
# 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 = pathlib.Path('models/research/object_detection/test_images')
TEST_IMAGE_PATHS = sorted(list(PATH_TO_TEST_IMAGES_DIR.glob("*.jpg")))
TEST_IMAGE_PATHS
```

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

5. Detection

5.1 Load an object detection model:

In [ ]:
```
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
```

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

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

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

And retuns several outputs:

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

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

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

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

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

In [ ]:
```
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
```

5.4 Run it on each test image and show the results:

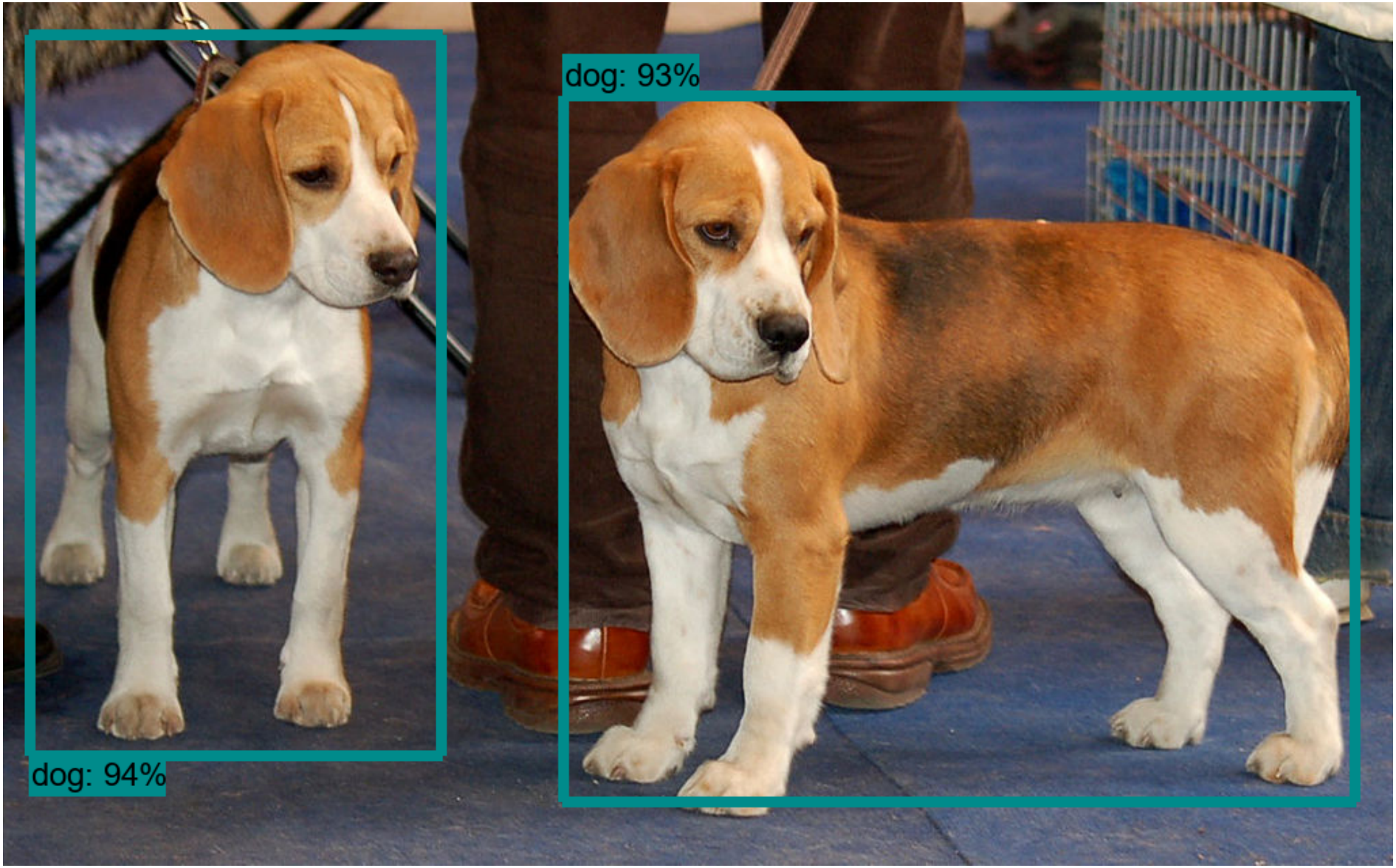
In [ ]:

```
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 [ ]:

```
for image_path in TEST_IMAGE_PATHS:
  show_inference(detection_model, image_path)
```


dog: 93%
dog: 94%

6. Instance Segmentation

In [ ]:
```
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")
```

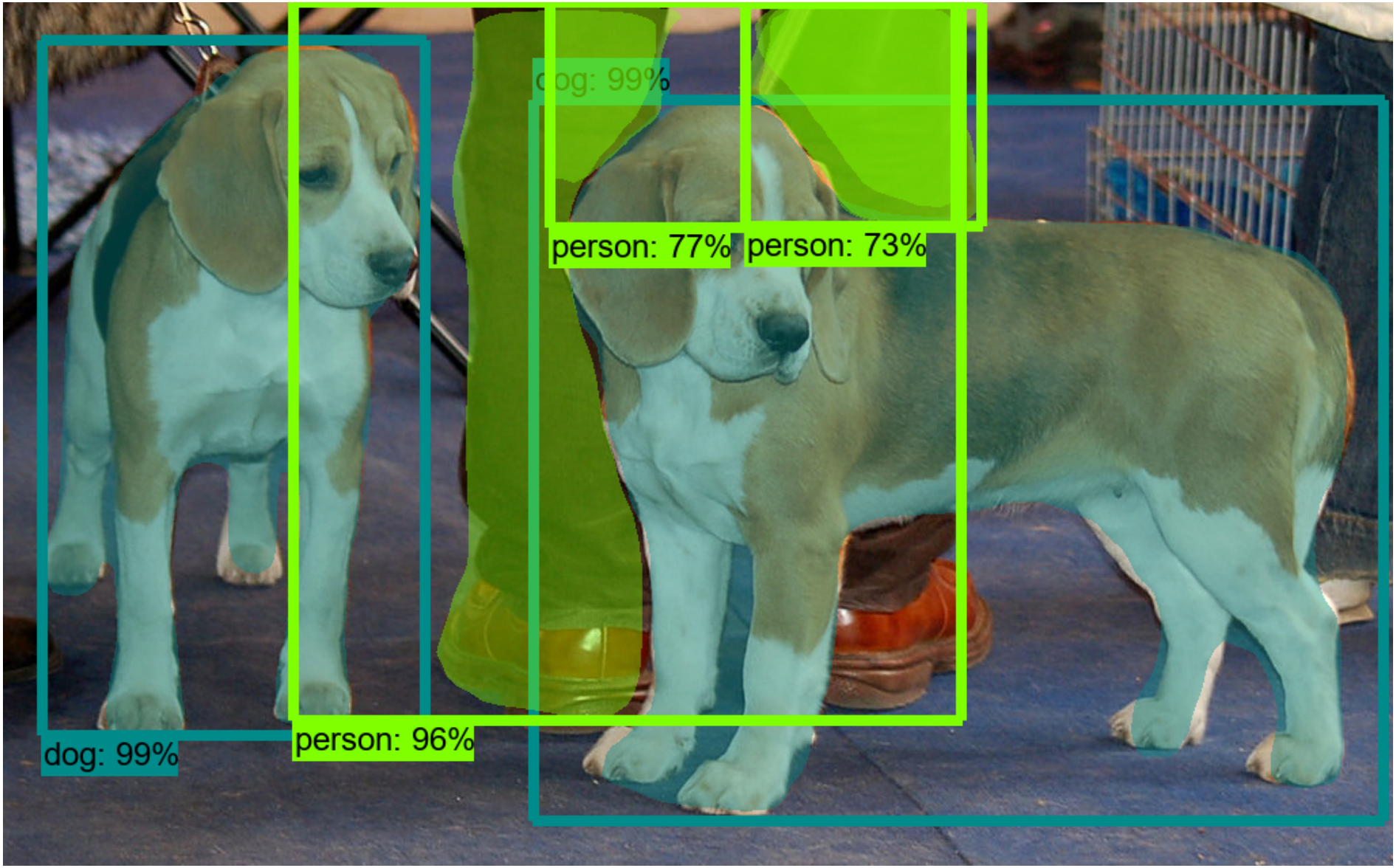
```
INFO:tensorflow:Saver not created because there are no variables in the graph to restore
```

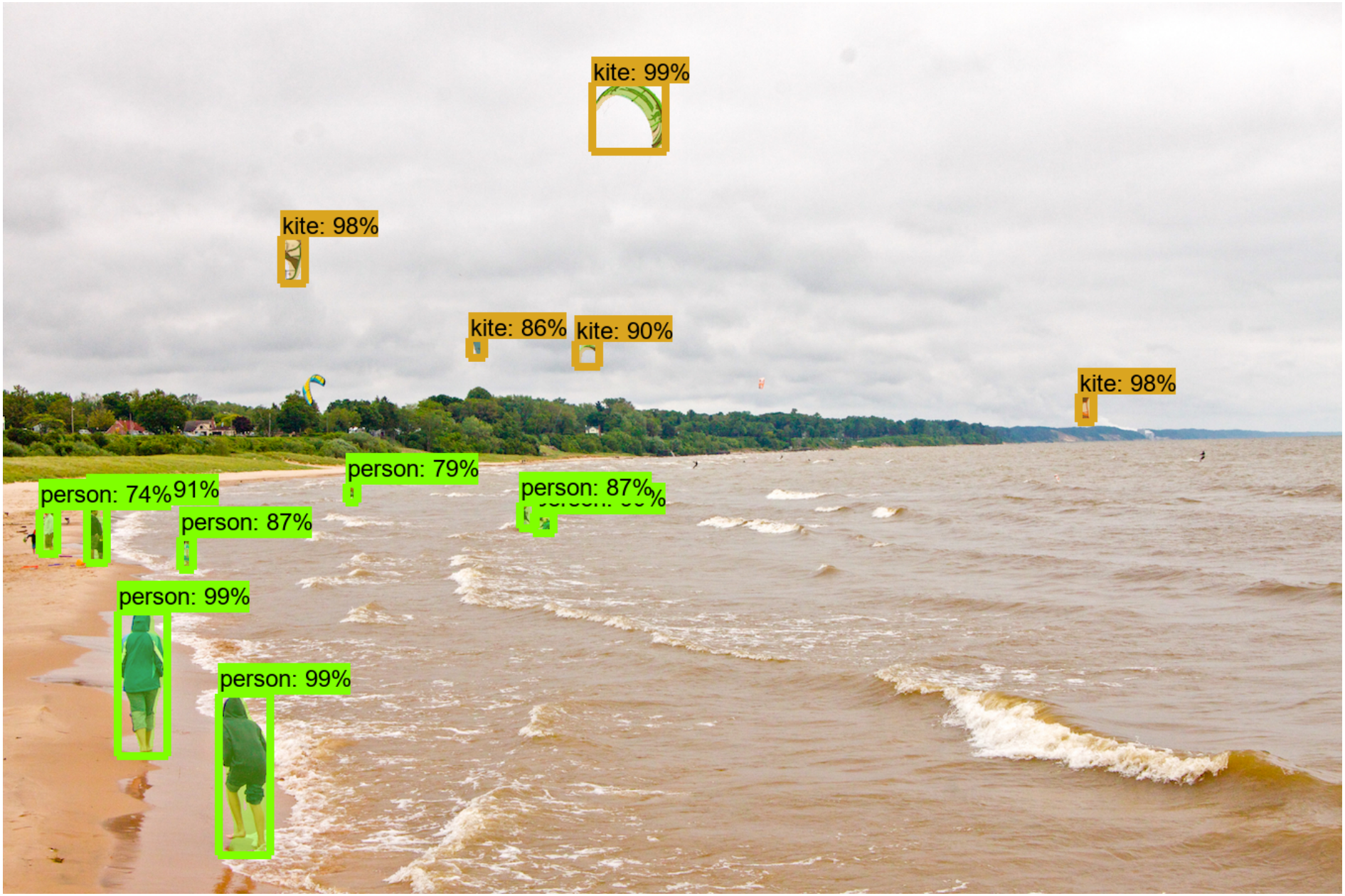
The instance segmentation model includes a `detection_masks` output:

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

Out[ ]:
```
{'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 [ ]:
```
for image_path in TEST_IMAGE_PATHS:
  show_inference(masking_model, image_path)
```


dog: 99%
person: 96%
dog: 99%
person: 77%
person: 73%


kite: 99%
kite: 98%
kite: 86%
kite: 90%
kite: 98%
person: 79%
person: 74%
91%
person: 87%
person: 87%
person: 99%
person: 99%