FIN42110: HomeWork 5

Ross Kearney

5th April 2022

1 Image Classification

4 street level images of people outside shops have been used to run the classification tools on, as required.

Figure 1: Image 1 of people outside shop (left) object classification complete (right).

A highly confident classification for each person is observed (92%+)

Figure 2: Image 2 of worker inside shop (left) object classification complete (right).

A less confident classification (69%) for the worker, perhaps due to his stance as well as the lighting/colours in the image. Also a notable classification of the overall bookshelf as a singular book, however, with a lower confidence. (67%)

Figure 3: Image 3 of customers queuing outside a shop (left) object classification complete (right).

A varied classification of people ranging from 69% to 92%. A notable classification of a poster of a person with a reasonable confidence (67%)

Figure 4: Image 4 of a large crowd outside a shop (left) object classification complete (right).

A large range of confidence in the classifications. Presumably due to the crowd with some people only partially in view. 56% to 99% confidence.

2 Python Code

```
# -*- coding: utf-8 -*-
"""
Created on  Apr  1 11:45:34 2022

@author: rosskearney
"""
import numpy as np
import os
import six.moves.urllib as urllib
import sys
import tarfile
import tensorflow as tf
import zipfile
import pathlib


from collections import defaultdict
from io import StringIO
from matplotlib import pyplot as plt
from PIL import Image
from IPython.display import display
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

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))

  return model

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
  model_fn = model.signatures['serving_default']
  output_dict = model_fn(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


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))

if __name__ == '__main__':

    # List of the strings that is used to add correct label for each box.
    PATH_TO_LABELS = 'models-master/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)

    # 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('TESTIMAGES')
    PATH_TO_TEST_IMAGES_DIR = pathlib.Path('models-master/research/object_detection/test_images')
    TEST_IMAGE_PATHS = sorted(list(PATH_TO_TEST_IMAGES_DIR.glob("*.jpeg")))

    model_name = 'faster_rcnn_inception_v2_coco_2018_01_28' #'ssd_mobilenet_v1_coco_2017_11_17'
    detection_model = load_model(model_name)


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