|  |
| --- |
|  |
|  | 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 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 tf.\_\_version\_\_ < '1.4.0': |
|  | raise ImportError('Please upgrade your tensorflow installation to v1.4.\* or later!') |
|  |  |
|  |  |
|  | # ## Env setup |
|  |  |
|  | # In[45]: |
|  |  |
|  |  |
|  | # This is needed to display the images. |
|  |  |
|  |  |
|  | # ## Object detection imports |
|  | # Here are the imports from the object detection module. |
|  |  |
|  | # In[46]: |
|  |  |
|  |  |
|  | 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[47]: |
|  |  |
|  |  |
|  | # What model to download. |
|  | MODEL\_NAME = 'LicensePlateGraph' |
|  |  |
|  | # 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('training', 'License-Plate-Detection.pbtxt') |
|  |  |
|  | NUM\_CLASSES = 1 |
|  |  |
|  |  |
|  | # ## Download Model |
|  |  |
|  | # In[48]: |
|  |  |
|  |  |
|  |  |
|  | '''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[49]: |
|  |  |
|  |  |
|  | 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[50]: |
|  |  |
|  |  |
|  | 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[51]: |
|  |  |
|  |  |
|  | 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[52]: |
|  |  |
|  |  |
|  | # 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(3, 11) ] |
|  |  |
|  | # Size, in inches, of the output images. |
|  | IMAGE\_SIZE = (12, 8) |
|  |  |
|  |  |
|  | # In[53]: |
|  |  |
|  |  |
|  | 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[0], image.shape[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\_tensor:0') |
|  |  |
|  | # Run inference |
|  | output\_dict = sess.run(tensor\_dict, |
|  | feed\_dict={image\_tensor: np.expand\_dims(image, 0)}) |
|  |  |
|  | # 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.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[54]: |
|  |  |
|  |  |
|  | 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, detection\_graph) |
|  | # Visualization of the results of a detection. |
|  | #print(output\_dict['detection\_scores']) |
|  | 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) |
|  | print(output\_dict['detection\_scores']) |