**TFOD2 Custom Training Running on Google Colab**

**Steps:**

1. Install latest version of tensorflow: !pip install tensorflow==”2.13.0”
2. Clone the tensorflow models repository

|  |
| --- |
| import os  import pathlib  # Clone the tensorflow models repository if it doesn't already exist  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> |

1. Install the object detection API

|  |
| --- |
| # Install the Object Detection API  %%bash  cd models/research/  protoc object\_detection/protos/\*.proto --python\_out=.  cp object\_detection/packages/tf2/setup.py .  python -m pip install . |

1. Run model builder test

|  |
| --- |
| ## run model builder test  !python /content/models/research/object\_detection/builders/model\_builder\_tf2\_test.py |

1. Prepare data

|  |
| --- |
| %cd /content/models/research/object\_detection/  from google.colab import drive  drive.mount('/content/drive')  #path that contains folder you want to copy  %cp -av /content/drive/MyDrive/TFOD2/pet\_dataset pet\_dataset  # run xml-to-csv.py to convert annotation to csv file  import os  import glob  import pandas as pd  import xml.etree.ElementTree as ET  def xml\_to\_csv(xml\_files):      xml\_list = []      for xml\_file in xml\_files:          tree = ET.parse(xml\_file)          root = tree.getroot()          for member in root.findall('object'):              value = (root.find('filename').text,                       int(root.find('size')[0].text),                       int(root.find('size')[1].text),                       member[0].text,                       int(member[4][0].text),                       int(member[4][1].text),                       int(member[4][2].text),                       int(member[4][3].text)                       )              xml\_list.append(value)      column\_name = ['filename', 'width', 'height',                     'class', 'xmin', 'ymin', 'xmax', 'ymax']      xml\_df = pd.DataFrame(xml\_list, columns=column\_name)      return xml\_df  def main():      for folder in ['train', 'valid', 'test']:          image\_path = os.path.join('artifacts/data\_split', folder)          image\_files = glob.glob(image\_path + '/\*.jpg')          xml\_files = [os.path.join('artifacts/data/annotations/xmls', f.split(              os.sep)[-1].replace('.jpg', '.xml')) for f in image\_files]          xml\_df = xml\_to\_csv(xml\_files)          print(xml\_df)          outpath = os.path.join('artifacts/data\_split',                                 folder, folder+'\_labels.csv')          print(outpath)          xml\_df.to\_csv(outpath, index=None)          print('Successfully converted xml to csv.')  # main() |

1. Create “labelmap.pbtxt” according to annotation classes

|  |
| --- |
| item {  id: 1  name: 'cat'  }  item {  id: 2  name: 'dog'  } |

1. Edit “class\_text\_to\_int” function in generate\_tfrecord.py as per annotation classes

|  |
| --- |
| """  Usage:    # From tensorflow/models/    # Create train data:    python generate\_tfrecord.py --csv\_input=images/train\_labels.csv --image\_dir=images/train --output\_path=train.record    # Create test data:    python generate\_tfrecord.py --csv\_input=images/test\_labels.csv  --image\_dir=images/test --output\_path=test.record  """  from \_\_future\_\_ import division  from \_\_future\_\_ import print\_function  from \_\_future\_\_ import absolute\_import  import os  import io  import pandas as pd  import tensorflow as tf  import tensorflow.compat.v1 as tf  from PIL import Image  from object\_detection.utils import dataset\_util  from collections import namedtuple, OrderedDict  flags = tf.app.flags  flags.DEFINE\_string('csv\_input', '', 'Path to the CSV input')  flags.DEFINE\_string('image\_dir', '', 'Path to the image directory')  flags.DEFINE\_string('output\_path', '', 'Path to output TFRecord')  FLAGS = flags.FLAGS  # TO-DO replace this with label map  def class\_text\_to\_int(row\_label):      if row\_label == 'cat':          return 1      elif row\_label == 'dog':          return 2      else:          return 0  def split(df, group):      data = namedtuple('data', ['filename', 'object'])      gb = df.groupby(group)      return [data(filename, gb.get\_group(x)) for filename, x in zip(gb.groups.keys(), gb.groups)]  def create\_tf\_example(group, path):      with tf.gfile.GFile(os.path.join(path, '{}'.format(group.filename)), 'rb') as fid:          encoded\_jpg = fid.read()      encoded\_jpg\_io = io.BytesIO(encoded\_jpg)      image = Image.open(encoded\_jpg\_io)      width, height = image.size      filename = group.filename.encode('utf8')      image\_format = b'jpg'      xmins = []      xmaxs = []      ymins = []      ymaxs = []      classes\_text = []      classes = []      for index, row in group.object.iterrows():          xmins.append(row['xmin'] / width)          xmaxs.append(row['xmax'] / width)          ymins.append(row['ymin'] / height)          ymaxs.append(row['ymax'] / height)          classes\_text.append(row['class'].encode('utf8'))          classes.append(class\_text\_to\_int(row['class']))      tf\_example = tf.train.Example(features=tf.train.Features(feature={          'image/height': dataset\_util.int64\_feature(height),          'image/width': dataset\_util.int64\_feature(width),          'image/filename': dataset\_util.bytes\_feature(filename),          'image/source\_id': dataset\_util.bytes\_feature(filename),          'image/encoded': dataset\_util.bytes\_feature(encoded\_jpg),          'image/format': dataset\_util.bytes\_feature(image\_format),          'image/object/bbox/xmin': dataset\_util.float\_list\_feature(xmins),          'image/object/bbox/xmax': dataset\_util.float\_list\_feature(xmaxs),          'image/object/bbox/ymin': dataset\_util.float\_list\_feature(ymins),          'image/object/bbox/ymax': dataset\_util.float\_list\_feature(ymaxs),          'image/object/class/text': dataset\_util.bytes\_list\_feature(classes\_text),          'image/object/class/label': dataset\_util.int64\_list\_feature(classes),      }))      return tf\_example  def main(\_):      writer = tf.python\_io.TFRecordWriter(FLAGS.output\_path)      path = os.path.join(os.getcwd(), FLAGS.image\_dir)      examples = pd.read\_csv(FLAGS.csv\_input)      grouped = split(examples, 'filename')      for group in grouped:          tf\_example = create\_tf\_example(group, path)          writer.write(tf\_example.SerializeToString())      writer.close()      output\_path = os.path.join(os.getcwd(), FLAGS.output\_path)      print('Successfully created the TFRecords: {}'.format(output\_path))  if \_\_name\_\_ == '\_\_main\_\_':      tf.app.run() |

1. Retrieve generate\_tfrecord.py and labelmap

|  |
| --- |
| %cp -av /content/drive/MyDrive/TFOD2/generate\_tfrecord.py generate\_tfrecord.py  %cp -av /content/drive/MyDrive/TFOD2/labelmap.pbtxt labelmap.pbtxt  !python generate\_tfrecord.py --csv\_input=pet\_dataset/train/train\_labels.csv --image\_dir=pet\_dataset/train --output\_path=train.record  !python generate\_tfrecord.py --csv\_input=pet\_dataset/valid/valid\_labels.csv --image\_dir=pet\_dataset/valid --output\_path=test.record |

1. Configuring training pipeline

|  |
| --- |
| train\_record\_path = 'train.record'  test\_record\_path = 'test.record'  labelmap\_path = 'labelmap.pbtxt'  batch\_size = 8  num\_steps = 1000  num\_eval\_steps = 500  # find pre-trained object detection model from here: <https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf2_detection_zoo.md>  # Select model as per requirement -> then right-click on that -> copy link address -> paste below after “wget” to download  !wget <http://download.tensorflow.org/models/object_detection/tf2/20200711/efficientdet_d0_coco17_tpu-32.tar.gz>  !tar -xf efficientdet\_d0\_coco17\_tpu-32.tar.gz  fine\_tune\_checkpoint = 'efficientdet\_d0\_coco17\_tpu-32/checkpoint/ckpt-0'  !wget <https://raw.githubusercontent.com/tensorflow/models/master/research/object_detection/configs/tf2/ssd_efficientdet_d0_512x512_coco17_tpu-8.config>  base\_config\_path = 'ssd\_efficientdet\_d0\_512x512\_coco17\_tpu-8.config'  # edit base\_config\_path configuration file  import re  with open(base\_config\_path) as f:      config = f.read()  with open('model\_config.config', 'w') as f:      # Set labelmap path    config = re.sub('label\_map\_path: ".\*?"', 'label\_map\_path: "{}"'.format(labelmap\_path), config)      # Set fine\_tune\_checkpoint path    config = re.sub('fine\_tune\_checkpoint: ".\*?"', 'fine\_tune\_checkpoint: "{}"'.format(fine\_tune\_checkpoint), config)      # Set train tf-record file path    config = re.sub('(input\_path: ".\*?)(PATH\_TO\_BE\_CONFIGURED/train)(.\*?")', 'input\_path: "{}"'.format(train\_record\_path), config)      # Set test tf-record file path    config = re.sub('(input\_path: ".\*?)(PATH\_TO\_BE\_CONFIGURED/val)(.\*?")', 'input\_path: "{}"'.format(test\_record\_path), config)      # Set number of classes.    config = re.sub('num\_classes: [0-9]+', 'num\_classes: {}'.format(2), config)      # Set batch size    config = re.sub('batch\_size: [0-9]+', 'batch\_size: {}'.format(batch\_size), config)      # Set training steps    config = re.sub('num\_steps: [0-9]+', 'num\_steps: {}'.format(num\_steps), config)      # Set fine-tune checkpoint type to detection    config = re.sub('fine\_tune\_checkpoint\_type: "classification"', 'fine\_tune\_checkpoint\_type: "{}"'.format('detection'), config)      f.write(config)  %cat model\_config.config  model\_dir = 'training/'  pipeline\_config\_path = 'model\_config.config' |

1. Train object detector

|  |
| --- |
| import os  os.environ['TF\_CPP\_MIN\_LOG\_LEVEL'] = '2'  # Run this block first and use the refresh arrow that will appear in the header once the training below starts  %load\_ext tensorboard  %tensorboard --logdir 'training/train'  !python /content/models/research/object\_detection/model\_main\_tf2.py \  --pipeline\_config\_path={pipeline\_config\_path} \  --model\_dir={model\_dir} \  --alsologtostderr \  --num\_train\_steps={num\_steps} \  --sample\_1\_of\_n\_eval\_examples=1 \  --num\_eval\_steps={num\_eval\_steps} |

1. Export model inference graph

|  |
| --- |
| output\_directory = 'inference\_graph'  !python /content/models/research/object\_detection/exporter\_main\_v2.py \      --trained\_checkpoint\_dir {model\_dir} \      --output\_directory {output\_directory} \      --pipeline\_config\_path {pipeline\_config\_path} |

1. Download model

|  |
| --- |
| # Zip and download your new model to your system  from google.colab import files  !zip -r new\_model.zip /content/models/research/object\_detection/{output\_directory}/saved\_model  files.download(f'new\_model.zip')  # Optional: save a copy of the training data to your drive in case you want to re-train later  from google.colab import drive  drive.mount('/content/drive')  !cp -r training/ /content/drive/MyDrive/TFOD2/ |

1. Inferencing

|  |
| --- |
| # Custom Model Loading  import tensorflow as tf  def load\_custom\_model(model\_name):    model\_file = model\_name    model\_dir = pathlib.Path(model\_file)/"saved\_model"    model = tf.saved\_model.load(str(model\_dir))    return model  from object\_detection.utils import label\_map\_util  # List of the strings that is used to add correct label for each box.  PATH\_TO\_LABELS = 'labelmap.pbtxt'  category\_index = label\_map\_util.create\_category\_index\_from\_labelmap(PATH\_TO\_LABELS, use\_display\_name=True)  import pathlib  # 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('pet\_dataset/test')  TEST\_IMAGE\_PATHS = sorted(list(PATH\_TO\_TEST\_IMAGES\_DIR.glob("\*.jpg")))  model\_dir = 'inference\_graph'  detection\_model = load\_custom\_model(model\_dir)  import numpy as np  from object\_detection.utils import ops as utils\_ops  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  from PIL import Image  from object\_detection.utils import visualization\_utils as vis\_util  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))    return  image\_path = TEST\_IMAGE\_PATHS[100]  show\_inference(detection\_model, image\_path) |

1. Download complete project from google colab

|  |
| --- |
| # Zip and download the complete models folder to local system  from google.colab import files  !zip -r Custom\_TFOD2\_Model.zip /content/models  files.download(f'Custom\_TFOD2\_Model.zip') |

Error handling:

1. In latest tensorflow library, edit “tfexample\_decoder.py” file
   1. This file can be found in “usr/local/lib/python3.10/dist-packages/tf\_slim/data/ tfexample\_decoder.py” under “Files” section in Google Colab.
   2. Add “import tensorflow as tf” line in the script
   3. Replace “control\_flow\_ops” by “tf” everywhere in the script
   4. Finally, comment out “from tensorflow.python.ops import control\_flow\_ops” line from the script