Guide to Tensorflow Object Detection (Tensorflow 2)

Rohit Prakash
Oct 5, 2020 · 8 min read ★

There are many guides out there that are very good to help you get started with setting up the TF Object Detection API, but unfortunately, most of them are written for the TF v1 API.

We will take a look at how to use the TF v2 Object Detection API to build a model for a custom dataset on a Google Colab Notebook.

Before we begin the setup, make sure to change the runtime-type in Colab to GPU so that we can make use of the free GPU provided.

1. Installing Dependencies and setting up the workspace.

Create a folder for your workspace

```
%mkdir workspace
%cd /content/workspace
```

We will be cloning the TF repository from GitHub

```
!git clone --q https://github.com/tensorflow/models.git
```

And before we install TF Object Detection we must install Protobuf.

“The Tensorflow Object Detection API uses Protobufs to configure model and training parameters. Before the framework can be used, the Protobuf libraries must be downloaded and compiled”

```
!apt-get install -qq protobuf-compiler python-pil python-lxml
python-tk
!pip install -qq Cython contextlib2 pillow lxml matplotlib

!pip install -qq pycocotools
%cd models/research/
!protoc object_detection/protos/*.proto --python_out=.
```

Now we install the TF Object Detection API

```
%cp object_detection/packages/tf2/setup.py .
!python -m pip install .
!python object_detection/builders/model_builder_tf2_test.py
```

```
[ RUN      ] ModelBuilderTF2Test.test_invalid_faster_rcnn_batchnorm_update
INFO:tensorflow:time(__main__.ModelBuilderTF2Test.test_invalid_faster_rcnn_batchnorm_update): 0.0s
I0929 14:36:41.920821 139723948119936 test_util.py:1973] time(__main__.ModelBuilderTF2Test.test_invalid_faster_rcnn_batchnorm_update): 0.0s
[       OK ] ModelBuilderTF2Test.test_invalid_faster_rcnn_batchnorm_update
[ RUN      ] ModelBuilderTF2Test.test_invalid_first_stage_nms_iou_threshold
INFO:tensorflow:time(__main__.ModelBuilderTF2Test.test_invalid_first_stage_nms_iou_threshold): 0.0s
I0929 14:36:41.922938 139723948119936 test_util.py:1973] time(__main__.ModelBuilderTF2Test.test_invalid_first_stage_nms_iou_threshold): 0.0s
[       OK ] ModelBuilderTF2Test.test_invalid_first_stage_nms_iou_threshold
[ RUN      ] ModelBuilderTF2Test.test_invalid_model_config_proto
INFO:tensorflow:time(__main__.ModelBuilderTF2Test.test_invalid_model_config_proto): 0.0s
I0929 14:36:41.923602 139723948119936 test_util.py:1973] time(__main__.ModelBuilderTF2Test.test_invalid_model_config_proto): 0.0s
[       OK ] ModelBuilderTF2Test.test_invalid_model_config_proto
[ RUN      ] ModelBuilderTF2Test.test_invalid_second_stage_batch_size
INFO:tensorflow:time(__main__.ModelBuilderTF2Test.test_invalid_second_stage_batch_size): 0.0s
I0929 14:36:41.925252 139723948119936 test_util.py:1973] time(__main__.ModelBuilderTF2Test.test_invalid_second_stage_batch_size): 0.0s
[       OK ] ModelBuilderTF2Test.test_invalid_second_stage_batch_size
[ RUN      ] ModelBuilderTF2Test.test_session
[  SKIPPED ] ModelBuilderTF2Test.test_session
[ RUN      ] ModelBuilderTF2Test.test_unknown_faster_rcnn_feature_extractor
INFO:tensorflow:time(__main__.ModelBuilderTF2Test.test_unknown_faster_rcnn_feature_extractor): 0.0s
I0929 14:36:41.926978 139723948119936 test_util.py:1973] time(__main__.ModelBuilderTF2Test.test_unknown_faster_rcnn_feature_extractor): 0.0s
[       OK ] ModelBuilderTF2Test.test_unknown_faster_rcnn_feature_extractor
[ RUN      ] ModelBuilderTF2Test.test_unknown_meta_architecture
INFO:tensorflow:time(__main__.ModelBuilderTF2Test.test_unknown_meta_architecture): 0.0s
I0929 14:36:41.927641 139723948119936 test_util.py:1973] time(__main__.ModelBuilderTF2Test.test_unknown_meta_architecture): 0.0s
[       OK ] ModelBuilderTF2Test.test_unknown_meta_architecture
[ RUN      ] ModelBuilderTF2Test.test_unknown_ssd_feature_extractor
INFO:tensorflow:time(__main__.ModelBuilderTF2Test.test_unknown_ssd_feature_extractor): 0.0s
I0929 14:36:41.929182 139723948119936 test_util.py:1973] time(__main__.ModelBuilderTF2Test.test_unknown_ssd_feature_extractor): 0.0s
[       OK ] ModelBuilderTF2Test.test_unknown_ssd_feature_extractor
----------------------------------------------------------------------
Ran 20 tests in 41.672s

OK (skipped=1)
```

The output should be similar to this.

2. Preparing the Dataset

There are two ways to go about this:

- Use a Public Labelled Dataset
- Create a Custom Labelled Dataset

You can find Public Labelled Datasets online, which are already labeled and saved in the right format, ready to be used to train.

For this tutorial, we will be creating our own dataset from scratch.

First things first, gather the images for the dataset. I will assume this step has already been done.

Now we need to label the images. There are many popular labeling tools, we will be using LabelIMG.

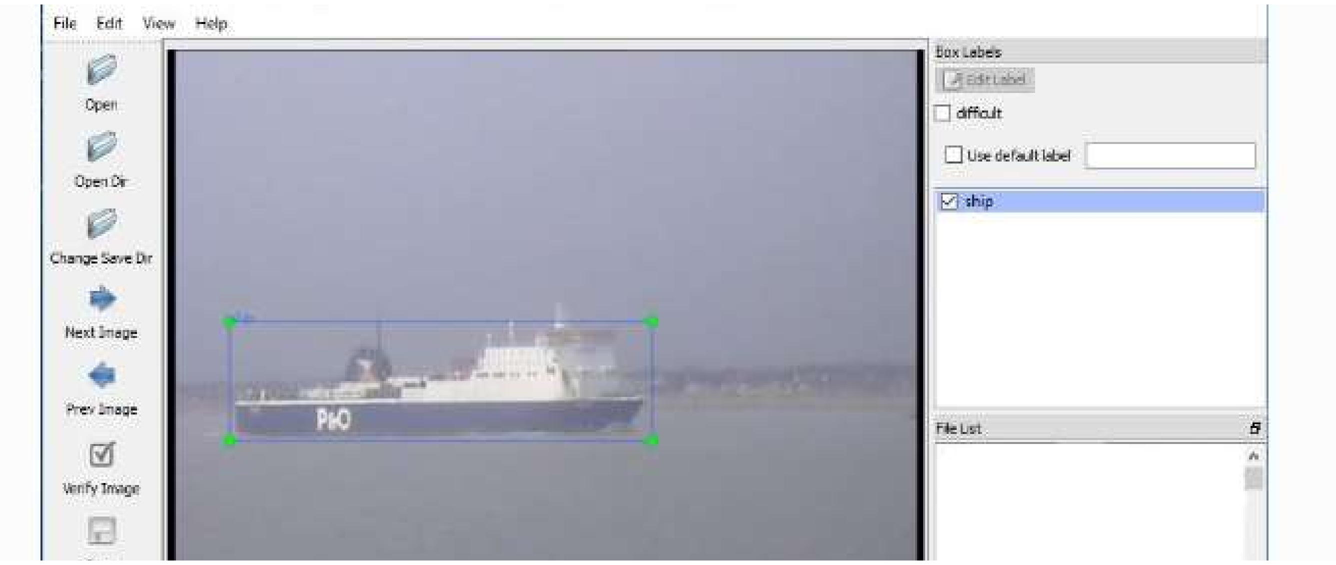
To install LabelIMG, execute the following code (Do it on your local Terminal since Colab does not support GUI applications):

```
pip install labelImg
```

Launch LabelImg in the folder where your images are stored.

```
labelImg imagesdir
```

Now you can start labeling your images, for more info on how to label the images follow this link (LabelImg Repository).



LabelImg

Create a label map in notepad as follows (label_map.pbtxt) with two classes for example cars and bikes:

```
item {
    id: 1
    name: 'car'
}

item {
    id: 2
    name: 'bike'
}
```

Now for creating the TFRecord files.

We can do the following:

- Create TFRecord ourselves
- Upload the annotations to Roboflow and get the dataset in TFRecord Format.

Creating the TFRecords ourselves is a bit tedious as the XML created after annotating may sometimes vary, so for the sake of ease, I suggest using Roboflow to perform the above task. They also provide an option to perform additional Data Augmentation which will increase the size of the dataset.

For your reference, here is a sample .py script to create the TFRecords manually.

```
import pandas as pd
import numpy as np
import csv
import re
import cv2
import os
import glob
import xml.etree.ElementTree as ET

import io
```

```python
import tensorflow as tf
from collections import namedtuple, OrderedDict

import shutil
import urllib.request
import tarfile
import argparse

# os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'    # Suppress TensorFlow logging (1)
import tensorflow.compat.v1 as tf
from PIL import Image
from object_detection.utils import dataset_util, label_map_util
from collections import namedtuple
xml_dir = 'images/test'
image_dir = 'images/test'

label_map = label_map_util.load_labelmap('annotations/label_map.pbtxt')
label_map_dict = label_map_util.get_label_map_dict(label_map)

output_path = 'annotations/test.record'

def xml_to_csv(path):
    """Iterates through all .xml files (generated by labelImg) in a given directory and combir
    them in a single Pandas dataframe.

    Parameters:
    ----------
    path : str
        The path containing the .xml files
    Returns
    -------
    Pandas DataFrame
        The produced dataframe
    """

    xml_list = []
    for xml_file in glob.glob(path + '/*.xml'):
        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 class_text_to_int(row_label):
    return label_map_dict[row_label]


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

csv_path = None
def main(_):

    writer = tf.python_io.TFRecordWriter(output_path)
    path = os.path.join(image_dir)
    examples = xml_to_csv(xml_dir)
    grouped = split(examples, 'filename')
    for group in grouped:
        tf_example = create_tf_example(group, path)
        writer.write(tf_example.SerializeToString())
    writer.close()
    print('Successfully created the TFRecord file: {}'.format(output_path))
    if csv_path is not None:
        examples.to_csv(csv_path, index=None)
```

```
xml_dir = 'images/test'
image_dir = 'images/test'
output_path = 'annotations/test.record'
```

By using Roboflow you will be provided the TFRecord files automatically.

Setting up on Colab

Create folders to store all the necessary files we have just created.

```
%mkdir annotations exported-models pre-trained-models
models/my_mobilenet # my_mobilenet folder is where our training
results will be stored
```

Now upload the newly created TFRecord files along with the images and annotations to Google Colab by clicking upload files.

You could use Google Drive to store your necessary files and importing those to Google Colab should be as simple as doing a !cp command.

Download Pre-Trained Model

There are many models ready to download from the Tensorflow Model Zoo.

Be careful in choosing which model to use as some are not made for Object Detection. For this tutorial we will be using the following model:

SSD MobileNet V2 FPNLite 320x320.

Download it into your Colab Notebook and extract it by executing:

```
%cd pre-trained-models
!curl
"http://download.tensorflow.org/models/object_detection/tf2/20200711
/ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8.tar.gz" --output
"ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8.tar.gz"

model_name = 'ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8'

model_file = model_name + '.tar.gz'

tar = tarfile.open(model_file)

tar.extractall()

tar.close()

os.remove(model_file)
```

Your directory structure should now look like this:

```
workspace/
├─ models/
│   ├─ community/
│   ├─ official/
│   ├─ orbit/
│   ├─ research/
│   ├─ my_mobilenet/
│   └─ ...
├─ annotations/
```

```
│   ├─ train/
│   └─ test/
├─ pre-trained-model/
├─ exported-models/
```

Editing the Configuration file

In TF Object Detection API, all the settings and required information for training the model and evaluating is situated in the pipeline.config file.

Let us take a look at it:

The most important ones we will need to change are

```
batch_size: 128

fine_tune_checkpoint: "PATH_TO_BE_CONFIGURED"

num_steps: 50000
num_classes: 2

fine_tune_checkpoint_type: "classification"

train_input_reader {
  label_map_path: "PATH_TO_BE_CONFIGURED"
  tf_record_input_reader {
    input_path: "PATH_TO_BE_CONFIGURED"
  }
}
eval_input_reader {
  label_map_path: "PATH_TO_BE_CONFIGURED"
  shuffle: false
  num_epochs: 1
  tf_record_input_reader {
    input_path: "PATH_TO_BE_CONFIGURED"
  }
}
```

batch_size is the number of batches the model will train in parallel. A suitable number to use is 8. It could be more/less depending on the computing power available.

A good suggestion given on StackOverflow is:

> Max batch size= available GPU memory bytes / 4 / (size of tensors + trainable parameters)

fine_tune_checkpoint is the last trained checkpoint (a checkpoint is how the model is stored by Tensorflow).

If you are starting the training for the first time, set this to the pre-trained-model.

If you want to continue training on a previously trained checkpoint, set it to the respective checkpoint path. (This will continue training, building upon the features and loss instead of starting from scratch).

```
# For Fresh Training
fine_tune_checkpoint: "pre-trained-
model/ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8/checkpoint/ckpt-
0"

# For Contuining the Training
fine_tune_checkpoint:
"exported_models/your_latest_batch/checkpoint/ckpt-0"

batch_size = 8 # Increase/Decrease this value depending on how fast
your train job runs and the availability of the Compute Resources.

num_steps: 25000 # 25000 is a good number of steps to get a good
loss.

fine_tune_checkpoint_type: "detection" # Set this to detection

train_input_reader {
  label_map_path: "annotations/label_map.pbtxt"   # Set to location
of label map
  tf_record_input_reader {
    input_path: "annotations/train.tfrecord"   # Set to location of
train TFRecord file
  }
}
# Similarly do the same for the eval input reader
eval_input_reader {
  label_map_path: "annotations/label_map.pbtxt"
  shuffle: false
  num_epochs: 1
  tf_record_input_reader {
    input_path: "annotations/test.tfrecord"
  }
}
```

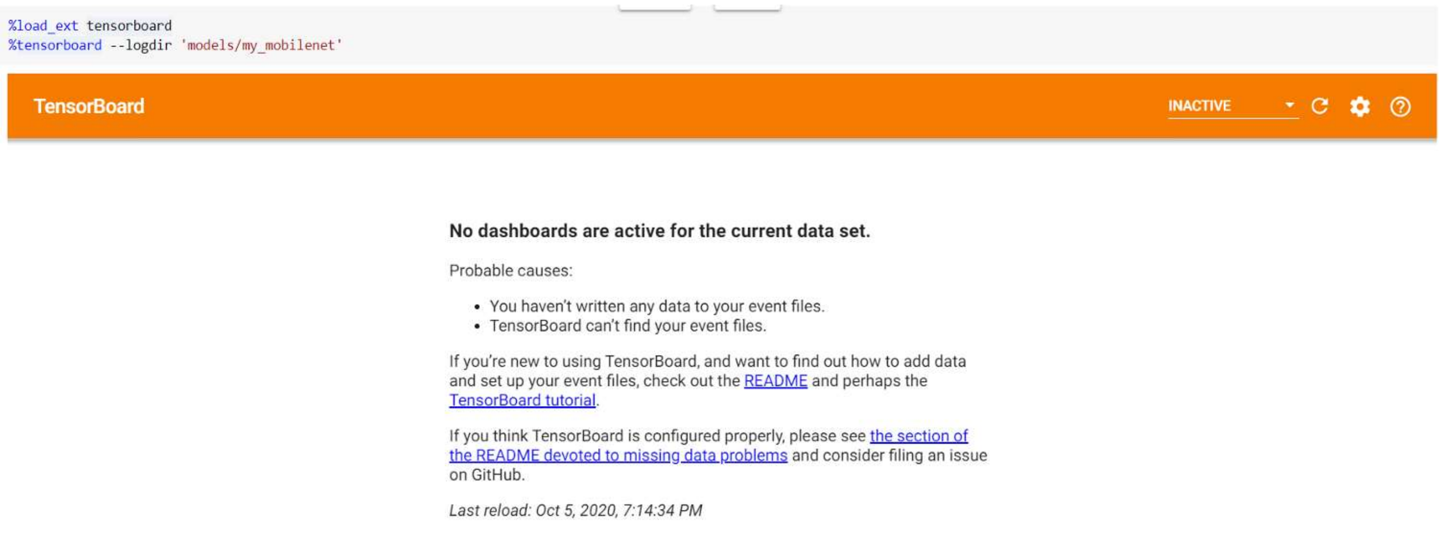
After editing the config file, we need to add the TensorFlow object detection folders to the python path.

```
import os

os.environ['PYTHONPATH'] +=
':/content/window_detection/models/:/content/window_detection/models
/research/:/content/window_detection/models/research/slim/'
```

Setting up TensorBoard on Colab to monitor the training process

Colab has introduced inbuilt support for TensorBoard and can now be called with a simple magic command as follows

```
%load_ext tensorboard
%tensorboard --logdir 'models/my_mobilenet'
```



Cell running Tensorboard

This is how the cell will look once you execute the above command, but nothing to worry, once we start the training job, click refresh on the Tensorboard cell(Top Right) after a few minutes(The .tfevent files need to be created for us to monitor the TensorFlow logs) and you will see the output on the TensorBoard magic cell

Running the Training Job

We will copy the TensorFlow training python script to the workspace directory for ease of access.

```
!cp
'/content/window_detection/models/research/object_detection/model_ma
in_tf2.py' .
```

The training job requires command-line arguments, namely:

- *model_dir* : This refers to the path where the training process will store the checkpoint files.
- *pipeline_config_path* : This refers to the path where the pipeline.config file is stored

Execute the following command to start the training job

```
# If you are training from scratch
!python model_main_tf2.py --model_dir=models/my_mobilenet --
pipeline_config_path=pre-trained-
model/ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8/pipeline.config

# Or if you are continuing from a previous training
!python model_main_tf2.py --model_dir=models/my_mobilenet --
pipeline_config_path=exported_models/pipeline.config
```

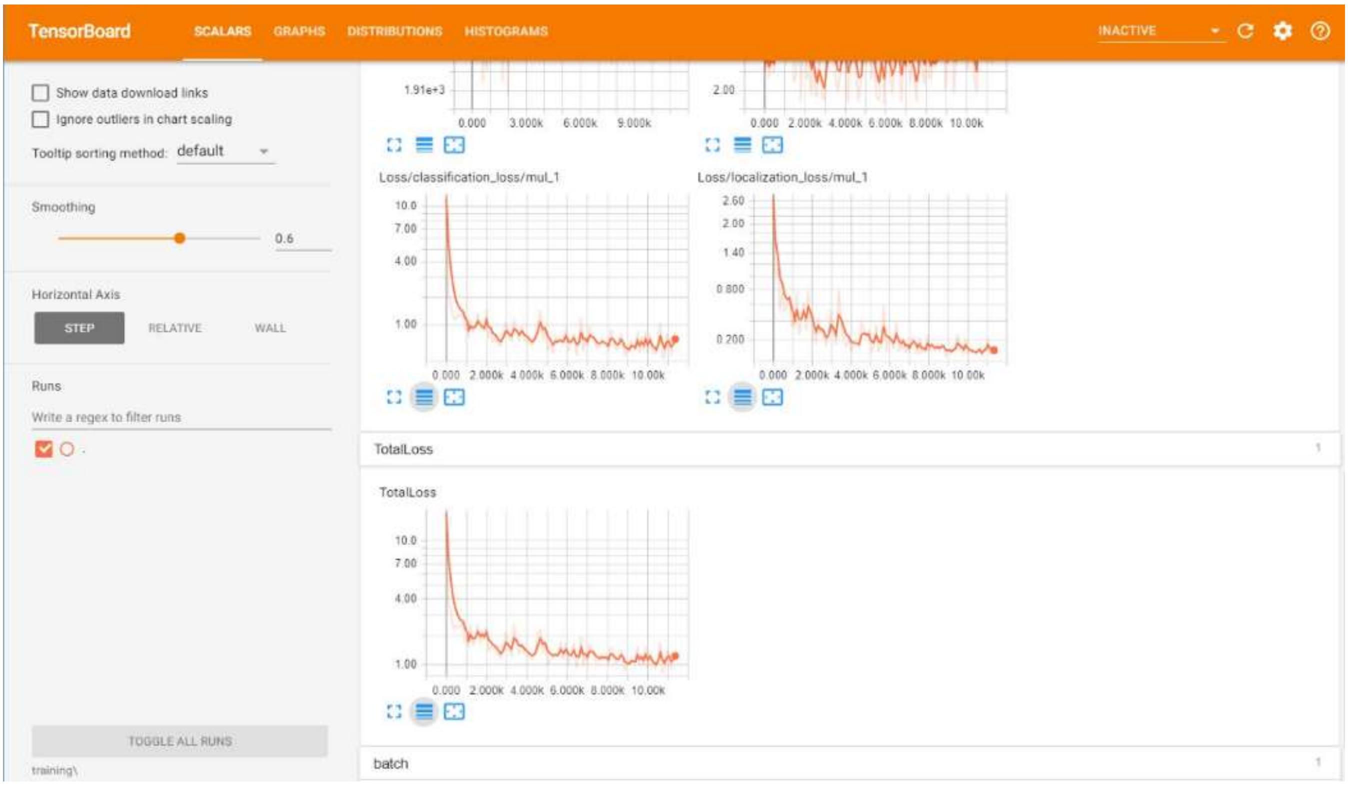
If everything goes well, the training output cell should look like this

```
I0929 19:14:03.429471 140008397502336 model_lib_v2.py:652] Step 24300 per-step time 0.355s loss=0.056
INFO:tensorflow:Step 24400 per-step time 0.335s loss=0.047
I0929 19:14:37.807487 140008397502336 model_lib_v2.py:652] Step 24400 per-step time 0.335s loss=0.047
INFO:tensorflow:Step 24500 per-step time 0.337s loss=0.056
I0929 19:15:12.283904 140008397502336 model_lib_v2.py:652] Step 24500 per-step time 0.337s loss=0.056
INFO:tensorflow:Step 24600 per-step time 0.308s loss=0.052
I0929 19:15:46.930425 140008397502336 model_lib_v2.py:652] Step 24600 per-step time 0.308s loss=0.052
INFO:tensorflow:Step 24700 per-step time 0.361s loss=0.046
I0929 19:16:21.292615 140008397502336 model_lib_v2.py:652] Step 24700 per-step time 0.361s loss=0.046
INFO:tensorflow:Step 24800 per-step time 0.338s loss=0.048
I0929 19:16:55.302360 140008397502336 model_lib_v2.py:652] Step 24800 per-step time 0.338s loss=0.048
INFO:tensorflow:Step 24900 per-step time 0.373s loss=0.062
I0929 19:17:29.647791 140008397502336 model_lib_v2.py:652] Step 24900 per-step time 0.373s loss=0.062
INFO:tensorflow:Step 25000 per-step time 0.311s loss=0.054
I0929 19:18:04.469359 140008397502336 model_lib_v2.py:652] Step 25000 per-step time 0.311s loss=0.054
```

Training Output

The output will normally update slowly. The training outputs logs only every 100 steps by default, therefore if you wait for a while, you should see a log for the loss at step 100. The speed depends on whether a GPU is being used to train and the available VRAM and many other factors, so be patient.

Refresh the TensorBoard while the training is running and you will be able to monitor the progress



Tensorboard Logs

Once the loss reaches a fairly constant value or becomes lower than 0.05(in my case), then you can stop the training cell.

Evaluating the model

Now you can run the evaluation script to find out the mAP (Mean Average Precision) and the Loss.

Run the following in a cell:

```
!python model_main_tf2.py --model_dir=exported-models/checkpoint --
pipeline_config_path=exported-models/pipeline.config --
checkpoint_dir=models/my_mobilenet/checkpoint # The folder where the
model has saved the checkpoints during training
```

You should get an output that looks like this

```
Accumulating evaluation results...
DONE (t=0.02s).
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.833
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.911
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.859
```

```
Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000
Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.800
Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.838
Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.567
Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.873
Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.894
Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000
Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.800
Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.896
```

Evaluation results

Now the evaluation script has a default timeout of 3600 seconds to wait for a new checkpoint to be generated as the script was initially intended to be running in parallel to the training job, but we are running it after the training process on Colab

You may go ahead and stop the evaluation cell from running.

Exporting the model

Now that we have our model ready, we need to save it in a format we can use it later.

We now have a bunch of checkpoints in the *models/my_mobilenet* folder. To remove all the older checkpoints and keep the latest checkpoint, I have attached a neat little python script that will do the task automatically.

```python
output_directory = 'exported-models/'

# goes through the model is the training/ dir and gets the last one.
# you could choose a specfic one instead of the last
lst = os.listdir("models/my_mobilenet/")
# print(lst)
lst = [l for l in lst if 'ckpt-' in l and '.index' not in l]
steps=np.array([int(re.findall('\d+', l)[0]) for l in lst])
last_model = lst[steps.argmax()]
last_model_path = os.path.join('models/my_mobilenet', last_model)
# print(last_model_path)
```

GetLatestCheckpoint.py hosted with ♡ by GitHub view raw

Now to export the model, we run the export script provided by TF2, as follows:

```
!python
/content/workspace/models/research/object_detection/exporter_main_v2
.py --input_type=image_tensor \
--pipeline_config_path=pre-trained-
model/ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8/pipeline.config
--output_directory=exported_models \
--trained_checkpoint_dir=models/my_mobilenet
```

The export script will save the model in the *exported_models* folder with the following structure

```
workspace/
├─ exported_models/
   ├─ checkpoint/
   ├─ saved_model/
   ├─ pipeline.config
```

You can now upload this folder to Google Drive or download it to save it for future use.

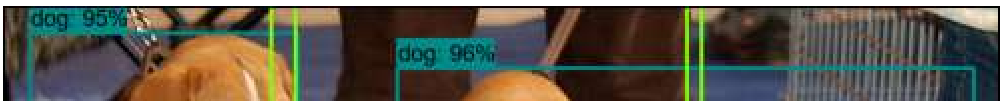
Inference on the model

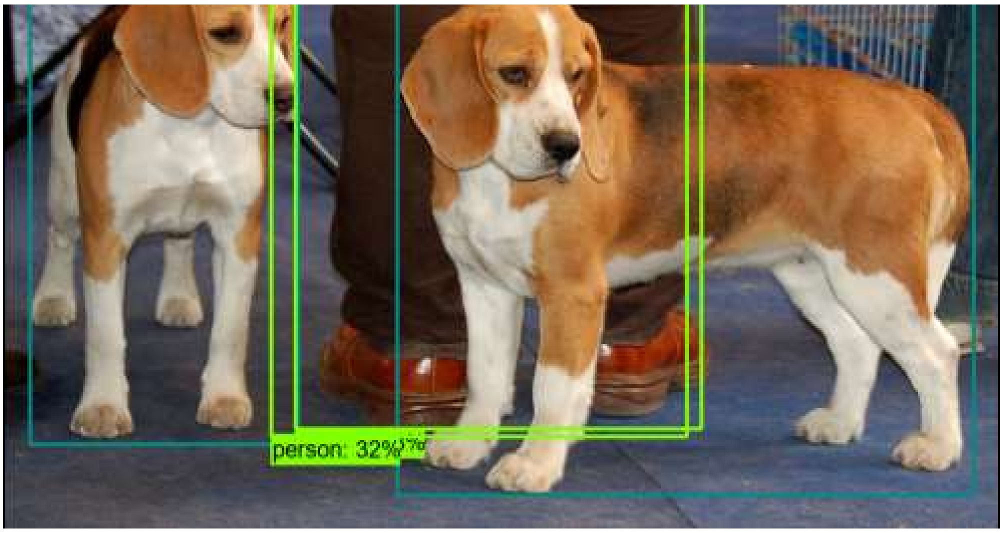
The final step, the step that fills you with a sense of accomplishment, in this step we will test our model on a random input image and see the model predict the type of object and give its bounding box.

The entire process is a little tedious but I will attach a script that will let you perform inference directly on Google Colab

```
import numpy as np
from PIL import Image
from google.colab.patches import cv2_imshow

def load_image_into_numpy_array(path):
    """Load an image from file into a numpy array.

    Puts image into numpy array to feed into tensorflow graph.
    Note that by convention we put it into a numpy array with shape
    (height, width, channels), where channels=3 for RGB.

    Args:
      path: the file path to the image

    Returns:
      uint8 numpy array with shape (img_height, img_width, 3)
    """
    return np.array(Image.open(path))

image_path = "PATH TO YOUR INFERENCE IMAGE"
print('Running inference for {}... '.format(image_path), end='')

```

```
image_np = load_image_into_numpy_array(image_path)

# Things to try:
# Flip horizontally
# image_np = np.fliplr(image_np).copy()

# Convert image to grayscale, (You could uncomment this to try and see how the model reacts to
# image_np = np.tile(
#     np.mean(image_np, 2, keepdims=True), (1, 1, 3)).astype(np.uint8)

# The input needs to be a tensor, convert it using `tf.convert_to_tensor`.
input_tensor = tf.convert_to_tensor(image_np)
# The model expects a batch of images, so add an axis with `tf.newaxis`.
input_tensor = input_tensor[tf.newaxis, ...]

detections = detect_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(detections.pop('num_detections'))
detections = {key: value[0, :num_detections].numpy()
              for key, value in detections.items()}
detections['num_detections'] = num_detections

# detection_classes should be ints.
detections['detection_classes'] = detections['detection_classes'].astype(np.int64)

image_np_with_detections = image_np.copy()

viz_utils.visualize_boxes_and_labels_on_image_array(
      image_np_with_detections,
      detections['detection_boxes'],
      detections['detection_classes'],
      detections['detection_scores'],
      category_index,
      use_normalized_coordinates=True,
      max_boxes_to_draw=200,
      min_score_thresh=.4, # Adjust this value to set the minimum probability boxes to be class
      agnostic_mode=False)

cv2.imshow(image_np_with_detections)
```





Inference Result

You can use the above script to fashion it into using a video as an input and perform inference on that.

Conclusion

Congratulations! You have built an object detection model with TensorFlow 2.

That's it for the tutorial! Hope you face no issues while following along, if there are any questions please comment and I will respond to your queries.

Refer to the Tensorflow Github page.

My socials -> LinkedIn, Medium

If you like this article or any of my other articles, please consider supporting me on my Patreon!

Thank you for reading!

TensorFlow Object Detection Python Deep Learning Data Science