# notebooks_TensorFlow_Object_Detection_API

April 7, 2023

## 1 PROJECT 1 - OBJECT DETECTION IN URBAN ENVIRONMENT

### 1.1 ROADMAP

- Install the TensorFlow Object Detection API.
- Edit the model pipeline config file and download the pre-trained model checkpoint.
- Train and evaluate the model.
- Output a video with detections

## 2 1) Import Libraries

[ ]:
```
import os
import glob
import pathlib
import pprint
import xml.etree.ElementTree as ET
import pandas as pd
import tensorflow as tf
print(tf.__version__)
```

## 3 2) Mount drive and link your folders

[ ]:
```
from google.colab import drive
drive.mount('/content/gdrive')

# this creates a symbolic link so that now the path /content/gdrive/My\ Drive/␣
  ↪is equal to /mydrive
!ln -s /content/gdrive/My\ Drive/ /mydrive
!ls /mydrive
```

## 4 3) Clone the tensorflow models git repository & Install TensorFlow Object Detection API

[ ]:
```
# clone the tensorflow models on the colab cloud vm
!git clone --q https://github.com/tensorflow/models.git

#navigate to /models/research folder to compile protos
%cd models/research

# Compile protos.
!protoc object_detection/protos/*.proto --python_out=.

# Install TensorFlow Object Detection API.
!cp object_detection/packages/tf2/setup.py .
!python -m pip install .
```

## 5 4) Test the model builder

[ ]:
```
# Testing the model builder
!python object_detection/builders/model_builder_tf2_test.py
```

## 6 5) Download pre-trained model checkpoint

Download **the model** into the **data** folder & unzip it.

A list of detection checkpoints for tensorflow 2.x can be found here.

[ ]:
```
# Working directory

%cd /mydrive/Project/customTF2/data/
```

[ ]:
```
#Download the pre-trained model into the data folder & unzip it.

# !wget -< Link of the pre-trained model >-
!wget http://download.tensorflow.org/models/object_detection/tf2/20200711/
 ↪efficientdet_d1_coco17_tpu-32.tar.gz

# !tar -xzvf -<Name_of_file.tar.gz>-
!tar -xzvf efficientdet_d1_coco17_tpu-32.tar.gz
```

## 7 6) Make changes to the model pipeline config file

Current working directory is /mydrive/Project/customTF2/data/(Model File)

**You need to make the following changes:** * change ***num_classes*** to number of your classes. * change ***test.record*** path, ***train.record*** path & ***labelmap*** path to the paths where you have

created these files (paths should be relative to your current working directory while training). * change ***fine_tune_checkpoint*** to the path of the directory where the downloaded checkpoint. * change ***fine_tune_checkpoint_type*** with value **classification** or **detection** depending on the type.. * change ***batch_size*** to any multiple of 8 depending upon the capability of your GPU. (eg:- 24,128,...,512). * change ***num_steps*** to number of steps you want the detector to train.

# 8 7) Load Tensorboard

[ ]:
```
# Load TensorBoard

%load_ext tensorboard
%tensorboard --logdir '/content/gdrive/MyDrive/Project/customTF2/logs/
 ↪resnet_logs'
```

[ ]:
```
## Incase an error occurs with TensorBoard ***UNCOMMENT*** the following lines␣
 ↪and run it to find this session PID and terminate it using !kill

# from tensorboard import notebook
# notebook.list() # View open TensorBoard instances
```

[ ]:
```
# !kill 39390
```

# 9 8) Train the model

## 9.1 Navigate to the *object_detection* folder in colab vm

[ ]:
```
%cd /content/models/research/object_detection
```

## 9.2 I - Training using model_main_tf2.py

Here **{PIPELINE_CONFIG_PATH}** points to the pipeline config and **{MODEL_DIR}** points to the directory in which training checkpoints and events will be written.

For best results, you should stop the training when the loss is less than 0.1 if possible, else train the model until the loss does not show any significant change for a while. The ideal loss should be below 0.05 (Try to get the loss as low as possible without overfitting the model. Don't go too high on training steps to try and lower the loss if the model has already converged viz. if it does not reduce loss significantly any further and takes a while to go down. )

[ ]:
```
# Run the command below from the content/models/research/object_detection␣
 ↪directory
"""
PIPELINE_CONFIG_PATH=path/to/pipeline.config
MODEL_DIR=path to training checkpoints directory
NUM_TRAIN_STEPS=2000
SAMPLE_1_OF_N_EVAL_EXAMPLES=1
```

```
python model_main_tf2.py -- \
  --model_dir=$MODEL_DIR --num_train_steps=$NUM_TRAIN_STEPS \
  --sample_1_of_n_eval_examples=$SAMPLE_1_OF_N_EVAL_EXAMPLES \
  --pipeline_config_path=$PIPELINE_CONFIG_PATH \
  --alsologtostderr
"""

!python model_main_tf2.py --pipeline_config_path=/content/gdrive/MyDrive/
 ↪Project/customTF2/data/ssd_resnet50_v1_fpn_640x640_coco17_tpu-8/pipeline.
 ↪config --model_dir=/content/gdrive/MyDrive/Project/customTF2/logs/
 ↪resnet_logs --alsologtostderr
```

#### 9.2.1 TROUBLESHOOTING:

If you get an error for _registerMatType cv2 above, this might be because of OpenCV version mismatches in Colab. Run `!pip list|grep opencv` to see the versions of OpenCV packages installed i.e. `opencv-python`, `opencv-contrib-python` & `opencv-python-headless`. The versions will be different which is causing this error. This error will go away when colab updates it supported versions. For now, you can fix this by simply uninstalling and installing OpenCV packages.

**Check versions:**

!pip list|grep opencv

**Use the following 2 commands if only the opencv-python-headless is of different version**:

!pip uninstall opencv-python-headless –y

!pip install opencv-python-headless==4.1.2.30

**Or use the following commands if other opencv packages are of different versions. Uninstall and install all with the same version**:

!pip uninstall opencv-python –y

!pip uninstall opencv-contrib-python –y

!pip uninstall opencv-python-headless –y

!pip install opencv-python==4.5.4.60

!pip install opencv-contrib-python==4.5.4.60

!pip install opencv-python-headless==4.5.4.60

### 9.3 II - Evaluation using model_main_tf2.py

You can run this in parallel by opening another colab notebook and running this command simultaneously along with the training command above (don't forget to mount drive, clone the TF git repo and install the TF2 object detection API there as well). This will give you validation loss, mAP, etc so you have a better idea of how your model is performing.

Here **{CHECKPOINT_DIR}** points to the directory with checkpoints produced by the training job. Evaluation events are written to **{MODEL_DIR/eval}**.

```
[ ]: # Run the command below from the content/models/research/object_detection␣
      ↪directory
     """
     PIPELINE_CONFIG_PATH=path/to/pipeline.config
     MODEL_DIR=path to training checkpoints directory
     CHECKPOINT_DIR=${MODEL_DIR}
     NUM_TRAIN_STEPS=2000
     SAMPLE_1_OF_N_EVAL_EXAMPLES=1

     python model_main_tf2.py -- \
       --model_dir=$MODEL_DIR --num_train_steps=$NUM_TRAIN_STEPS \
       --checkpoint_dir=${CHECKPOINT_DIR} \
       --sample_1_of_n_eval_examples=$SAMPLE_1_OF_N_EVAL_EXAMPLES \
       --pipeline_config_path=$PIPELINE_CONFIG_PATH \
       --alsologtostderr
     """

     !python model_main_tf2.py --pipeline_config_path=/mydrive/Project/customTF2/
      ↪data/ssd_resnet50_v1_fpn_640x640_coco17_tpu-8/pipeline.config --model_dir=/
      ↪mydrive/Project/customTF2/logs/resnet_logs --alsologtostderr
```

### 9.4 RETRAINING THE MODEL ( in case you get disconnected )

If you get disconnected or lose your session on colab vm, you can start your training where you left off as the checkpoint is saved on your drive inside the ***training*** folder. To restart the training simply run **steps 1 till 7.**

Note that since we have all the files required for training like the record files,our edited pipeline config file,the label_map file and the model checkpoint folder, therefore we do not need to create these again.

**The model_main_tf2.py script saves the checkpoint every 1000 steps.** The training automatically restarts from the last saved checkpoint itself.

However, if you see that it doesn't restart training from the last checkpoint you can make 1 change in the pipeline config file. Change **fine_tune_checkpoint** to where your latest trained checkpoints have been written and have it point to the latest checkpoint as shown below:

`fine_tune_checkpoint: "/mydrive/Project/customTF2/logs/ckpt-X" (where ckpt-X is the latest che`

## 10 9) Test your trained model

### 10.1 Export inference graph

Current working directory is /content/models/research/object_detection

***CHANGE*** < Model_File > in < –pipeline_config_path > to the name of you model folder

[ ]:
```
# Export inference graph

!python exporter_main_v2.py --trained_checkpoint_dir=/content/gdrive/MyDrive/
 ↪Project/customTF2/logs --pipeline_config_path=/content/gdrive/MyDrive/
 ↪Project/customTF2/data/< Model_File >/pipeline.config --output_directory /
 ↪mydrive/Project/customTF2/data/inference_graph_resnet
```

### 10.2 Test your trained Object Detection model on a video

[ ]:
```
# Importing libraries

import tensorflow as tf
import time
import numpy as np
import warnings
warnings.filterwarnings('ignore')
from PIL import Image
from google.colab.patches import cv2_imshow
from object_detection.utils import label_map_util
from object_detection.utils import visualization_utils as viz_utils
import cv2
```

[ ]:
```
# Output display size as you want

IMAGE_SIZE = (12, 8)
```

[ ]:
```
# Load the model

PATH_TO_SAVED_MODEL="/content/gdrive/MyDrive/Project/customTF2/data/
 ↪inference_graph/saved_model"
detect_fn=tf.saved_model.load(PATH_TO_SAVED_MODEL)
print('Done!')
```

[ ]:
```
category_index = {
                    1:{'id': 1, 'name': 'vehicle'},
                    2: {'id': 2, 'name': 'pedestrian'},
                    4: {'id': 4, 'name': 'cyclist'}
                }
```

[ ]:
```
# Convert each frame into a np array and save it in list 'x'

frames_path = sorted(glob.glob('/mydrive/Project/test_video/*.png'),
                     key = lambda k: int(os.path.basename(k).split('.')[0].
 ↪split('_')[1]))

x = np.array([np.array(Image.open(fname)) for fname in frames_path])
```

[ ]:
```
# Objects detected on each frame

images = []

for i in x:

  input_tensor = tf.convert_to_tensor(i)
  input_tensor = input_tensor[tf.newaxis, ...]

  detections = detect_fn(input_tensor)

  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

  detections['detection_classes'] = detections['detection_classes'].astype(np.
 ↪int64)

  image_np_with_detections = i.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 classified as True
      agnostic_mode=False)

  images.append(image_np_with_detections)

## The following lines visualize the detections on the image

  # %matplotlib inline
  # plt.figure(figsize=IMAGE_SIZE, dpi=200)
  # plt.axis("off")
  # plt.imshow(image_np_with_detections)
  # plt.show()
```

[ ]:
```
# Each frame is exported to 'images_new' directory

from PIL import Image
```

```
import numpy as np

w = 0

for i in images:
  img = Image.fromarray(i, "RGB")
  w = w + 1
  img.save('/content/gdrive/MyDrive/Project/images_new/' + str(w) + '.png')
```

[ ]:
```
#  Frames from 'images_new' directory are converted into a video

import numpy as np
import glob

img_array = []
for filename in glob.glob('/content/gdrive/MyDrive/Project/images_new/*.png'):
    img = cv2.imread(filename)
    height, width, layers = img.shape
    size = (width,height)
    img_array.append(img)


out = cv2.VideoWriter('/content/gdrive/MyDrive/Project/images_new/project_1.
 ↪avi',cv2.VideoWriter_fourcc(*'DIVX'), 15, size)

for i in range(len(img_array)):
    out.write(img_array[i])
out.release()
```