

```

1  #!/usr/bin/env python
2  # coding: utf-8
3
4  # # 0. Setup Paths
5
6  # In[1]:
7
8
9  #Traning and object detectin. Found here:
10 https://github.com/nicknochnack/TFODCourse/blob/main/2.%20Training%20and%20Detection.ipynb
11 import os
12
13 # In[2]:
14
15
16 CUSTOM_MODEL_NAME = 'efficientdet_d1_coco17_tpu-32_v12_10s'
17 PRETRAINED_MODEL_NAME = 'efficientdet_d1_coco17_tpu-32'
18 PRETRAINED_MODEL_URL =
19 'http://download.tensorflow.org/models/object_detection/tf2/20200711/efficientdet_d1_coco17_tpu-32.tar.gz'
20 TF_RECORD_SCRIPT_NAME = 'generate_tfrecord.py'
21 LABEL_MAP_NAME = 'label_map.pbtxt'
22
23 # In[3]:
24
25
26 paths = {
27     'WORKSPACE_PATH': os.path.join('Tensorflow', 'workspace'),
28     'SCRIPTS_PATH': os.path.join('Tensorflow', 'scripts'),
29     'APIMODEL_PATH': os.path.join('Tensorflow', 'models'),
30     'ANNOTATION_PATH': os.path.join('Tensorflow', 'workspace', 'annotations'),
31     'IMAGE_PATH': os.path.join('Tensorflow', 'workspace', 'images'),
32     'MODEL_PATH': os.path.join('Tensorflow', 'workspace', 'models'),
33     'PRETRAINED_MODEL_PATH': os.path.join('Tensorflow', 'workspace', 'pre-trained-models'),
34     'CHECKPOINT_PATH': os.path.join('Tensorflow', 'workspace', 'models', CUSTOM_MODEL_NAME),
35     'OUTPUT_PATH': os.path.join('Tensorflow', 'workspace', 'models', CUSTOM_MODEL_NAME, 'export'),
36     'TFJS_PATH': os.path.join('Tensorflow', 'workspace', 'models', CUSTOM_MODEL_NAME, 'tfjsexport'),
37     'TFLITE_PATH': os.path.join('Tensorflow', 'workspace', 'models', CUSTOM_MODEL_NAME, 'tfliteexport'),
38     'PROTOC_PATH': os.path.join('Tensorflow', 'protoc')
39 }
40
41
42 # In[4]:
43
44
45 files = {
46     'PIPELINE_CONFIG': os.path.join('Tensorflow', 'workspace', 'models', CUSTOM_MODEL_NAME, 'pipeline.config'),
47     'TF_RECORD_SCRIPT': os.path.join(paths['SCRIPTS_PATH'], TF_RECORD_SCRIPT_NAME),
48     'LABELMAP': os.path.join(paths['ANNOTATION_PATH'], LABEL_MAP_NAME)
49 }

```

```

50
51
52 # In[5]:
53
54
55 for path in paths.values():
56     if not os.path.exists(path):
57         if os.name == 'posix':
58             get_ipython().system('mkdir -p {path}')
59         if os.name == 'nt':
60             get_ipython().system('mkdir {path}')
61
62
63 # # 1. Download TF Models Pretrained Models from Tensorflow Model Zoo and Install TFOD
64
65 # In[ ]:
66
67
68 # https://www.tensorflow.org/install/source\_windows
69
70
71 # In[6]:
72
73
74 if os.name=='nt':
75     get_ipython().system('pip install wget')
76     import wget
77
78
79 # In[7]:
80
81
82 if not os.path.exists(os.path.join(paths['APIMODEL_PATH'], 'research', 'object_detection')):
83     get_ipython().system("git clone https://github.com/tensorflow/models {paths['APIMODEL_PATH']}")
84
85
86 # In[ ]:
87
88
89 # Install Tensorflow Object Detection
90 if os.name=='posix':
91     get_ipython().system('apt-get install protobuf-compiler')
92     get_ipython().system('cd Tensorflow/models/research && protoc object_detection/protos/*.proto --python_out=. && cp object_detection/packages/tf2/setup.py . && python -m pip install .')
93
94 if os.name=='nt':
95     url="https://github.com/protocolbuffers/protobuf/releases/download/v3.15.6/protoc-3.15.6-win64.zip"
96     wget.download(url)
97     get_ipython().system("move protoc-3.15.6-win64.zip {paths['PROTOC_PATH']}")
98     get_ipython().system("cd {paths['PROTOC_PATH']} && tar -xf protoc-3.15.6-win64.zip")
99     os.environ['PATH'] += os.pathsep + os.path.abspath(os.path.join(paths['PROTOC_PATH'], 'bin'))

```

```
100     get_ipython().system('cd Tensorflow/models/research && protoc object_detection/protos/*.proto --python_out=. &&
101     copy object_detection\\\\\\\\packages\\\\\\\\tf2\\\\\\\\setup.py setup.py && python setup.py build && python setup.py install')
102     get_ipython().system('cd Tensorflow/models/research/slim && pip install -e . ')
103
104     # In[ ]:
105
106
107     get_ipython().system('pip install numpy==1.18.1')
108
109
110     # In[24]:
111
112
113     VERIFICATION_SCRIPT = os.path.join(paths['APIMODEL_PATH'], 'research', 'object_detection', 'builders',
114     'model_builder_tf2_test.py')
115     # Verify Installation
116     get_ipython().system('python {VERIFICATION_SCRIPT}')
117
118     # In[ ]:
119
120
121     get_ipython().system('pip install tensorflow --upgrade')
122
123
124     # In[ ]:
125
126
127     get_ipython().system('pip uninstall protobuf matplotlib -y')
128     get_ipython().system('pip install protobuf matplotlib==3.2')
129
130
131     # In[ ]:
132
133
134     get_ipython().system('pip install matplotlib')
135
136
137     # In[ ]:
138
139
140     get_ipython().system('pip install -U pip wheel setuptools')
141
142
143     # In[ ]:
144
145
146     get_ipython().system('pip install pyyaml')
```

```

149 # In[ ]:
150
151
152 get_ipython().system('pip install kaggle')
153
154
155 # In[8]:
156
157
158 import object_detection
159
160
161 # In[ ]:
162
163
164 get_ipython().system('pip list')
165
166
167 # In[9]:
168
169
170 if os.name == 'posix':
171     get_ipython().system('wget {PRETRAINED_MODEL_URL}')
172     get_ipython().system("mv {PRETRAINED_MODEL_NAME+'.tar.gz'} {paths['PRETRAINED_MODEL_PATH']}")
173     get_ipython().system("cd {paths['PRETRAINED_MODEL_PATH']} && tar -zxvf {PRETRAINED_MODEL_NAME+'.tar.gz'}")
174 if os.name == 'nt':
175     wget.download(PRETRAINED_MODEL_URL)
176     get_ipython().system("move {PRETRAINED_MODEL_NAME+'.tar.gz'} {paths['PRETRAINED_MODEL_PATH']}")
177     get_ipython().system("cd {paths['PRETRAINED_MODEL_PATH']} && tar -zxvf {PRETRAINED_MODEL_NAME+'.tar.gz'}")
178
179
180 # # 2. Create Label Map
181
182 # In[10]:
183
184
185 labels = [{'name':'Sveis', 'id':1},{'name':'SveisSlipT', 'id':2},{'name':'SveisSlipTLitt', 'id':3}]
186
187 with open(files['LABELMAP'], 'w') as f:
188     for label in labels:
189         f.write('item { \n')
190         f.write('\tname:\'{}\'\n'.format(label['name']))
191         f.write('\tid:{ }\n'.format(label['id']))
192         f.write('{}\n')
193
194
195 # # 3. Create TF records
196
197 # In[ ]:
198
199

```

```

200 # OPTIONAL IF RUNNING ON COLAB
201 ARCHIVE_FILES = os.path.join(paths['IMAGE_PATH'], 'archive.tar.gz')
202 if os.path.exists(ARCHIVE_FILES):
203     get_ipython().system('tar -zxvf {ARCHIVE_FILES}')
204
205
206 # In[11]:
207
208
209 if not os.path.exists(files['TF_RECORD_SCRIPT']):
210     get_ipython().system("git clone https://github.com/nicknochnack/GenerateTFRecord {paths['SCRIPTS_PATH']}")
211
212
213 # In[12]:
214
215
216 get_ipython().system("python {files['TF_RECORD_SCRIPT']} -x {os.path.join(paths['IMAGE_PATH'], 'train')} -l
217 {files['LABELMAP']} -o {os.path.join(paths['ANNOTATION_PATH'], 'train.record')} ")
218 get_ipython().system("python {files['TF_RECORD_SCRIPT']} -x {os.path.join(paths['IMAGE_PATH'], 'test')} -l
219 {files['LABELMAP']} -o {os.path.join(paths['ANNOTATION_PATH'], 'test.record')} ")
220
221 # In[ ]:
222
223 get_ipython().system('pip install pytz')
224
225
226 # # 4. Copy Model Config to Training Folder
227
228 # In[13]:
229
230
231 if os.name == 'posix':
232     get_ipython().system("cp {os.path.join(paths['PRETRAINED_MODEL_PATH'], PRETRAINED_MODEL_NAME, 'pipeline.config')}
233 {os.path.join(paths['CHECKPOINT_PATH'])}")
234 if os.name == 'nt':
235     get_ipython().system("copy {os.path.join(paths['PRETRAINED_MODEL_PATH'], PRETRAINED_MODEL_NAME,
236 'pipeline.config')} {os.path.join(paths['CHECKPOINT_PATH'])}")
237
238
239 # # 5. Update Config For Transfer Learning
240
241 # In[14]:
242
243 import tensorflow as tf
244 from object_detection.utils import config_util
245 from object_detection.protos import pipeline_pb2
246 from google.protobuf import text_format

```

```
247
248 # In[15]:
249
250
251 config = config_util.get_configs_from_pipeline_file(files['PIPELINE_CONFIG'])
252
253
254 # In[ ]:
255
256
257 config
258
259
260 # In[16]:
261
262
263 pipeline_config = pipeline_pb2.TrainEvalPipelineConfig()
264 with tf.io.gfile.GFile(files['PIPELINE_CONFIG'], "r") as f:
265
266     proto_str = f.read()
267
268
269     text_format.Merge(proto_str, pipeline_config)
270
271
272 # In[ ]:
273
274
275 pipeline_config.model.ssd.num_classes = len(labels)
276 pipeline_config.train_config.batch_size = 4
277 pipeline_config.train_config.fine_tune_checkpoint = os.path.join(paths['PRETRAINED_MODEL_PATH'], PRETRAINED_MODEL_NAME,
278     'checkpoint', 'ckpt-0')
279 pipeline_config.train_config.fine_tune_checkpoint_type = "detection"
280 pipeline_config.train_input_reader.label_map_path= files['LABELMAP']
281 pipeline_config.train_input_reader.tf_record_input_reader.input_path[:] = [os.path.join(paths['ANNOTATION_PATH'],
282     'train.record')]
283 pipeline_config.eval_input_reader[0].label_map_path = files['LABELMAP']
284 pipeline_config.eval_input_reader[0].tf_record_input_reader.input_path[:] = [os.path.join(paths['ANNOTATION_PATH'],
285     'test.record')]
286
287
288 # In[ ]:
289
290
291 config
292
293
294 # In[17]:
```

```
290
291 config_text = text_format.MessageToString(pipeline_config)

292 with tf.io.gfile.GFile(files['PIPELINE_CONFIG'], "wb") as f:

293     f.write(config_text)
294
295
296 # # 6. Train the model
297
298 # In[18]:
299
300
301 TRAINING_SCRIPT = os.path.join(paths['APIMODEL_PATH'], 'research', 'object_detection', 'model_main_tf2.py')
302 os.environ['CUDA_VISIBLE_DEVICES'] = '-1'
303
304
305 # In[22]:
306
307
308 command = "python {} --model_dir={} --pipeline_config_path={} --num_train_steps=100".format(TRAINING_SCRIPT, paths[
309 'CHECKPOINT_PATH'],files['PIPELINE_CONFIG'])
310
311 # In[23]:
312
313
314 print(command)
315
316
317 # In[ ]:
318
319
320 get_ipython().system('{command}')
321
322
323 # In[ ]:
324
325
326 get_ipython().system('pip install tensorflow-addons')
327
328
329 # In[ ]:
330
331
332
333
334
335 # # 7. Evaluate the Model
```

```

336
337 # In[ ]:
338
339
340 command = "python {} --model_dir={} --pipeline_config_path={} --checkpoint_dir={} ".format(TRAINING_SCRIPT, paths[
'CHECKPOINT_PATH'], files['PIPELINE_CONFIG'], paths['CHECKPOINT_PATH'])
341
342
343 # In[ ]:
344
345
346 print(command)
347
348
349 # In[ ]:
350
351
352 get_ipython().system('{command}')
353
354
355 # In[ ]:
356
357
358 get_ipython().system('pip conda install cudnn')
359
360
361 # # 8. Load Train Model From Checkpoint
362
363 # In[ ]:
364
365
366 import os
367 import tensorflow as tf
368 from object_detection.utils import label_map_util
369 from object_detection.utils import visualization_utils as viz_utils
370 from object_detection.builders import model_builder
371 from object_detection.utils import config_util
372
373
374 # In[ ]:
375
376
377 # Load pipeline config and build a detection model
378 configs = config_util.get_configs_from_pipeline_file(files['PIPELINE_CONFIG'])
379 detection_model = model_builder.build(model_config=configs['model'], is_training=False)
380
381 # Restore checkpoint
382 ckpt = tf.compat.v2.train.Checkpoint(model=detection_model)
383 ckpt.restore(os.path.join(paths['CHECKPOINT_PATH'], 'ckpt-41')).expect_partial() ##Viktig å endre ckpt-(nummer) til
den site treningsmodellen
384

```

```

385 @tf.function
386 def detect_fn(image):
387     image, shapes = detection_model.preprocess(image)
388     prediction_dict = detection_model.predict(image, shapes)
389     detections = detection_model.postprocess(prediction_dict, shapes)
390     return detections
391
392
393 # # 9. Detect from an Image
394
395 # In[ ]:
396
397
398 import cv2
399 import numpy as np
400 from matplotlib import pyplot as plt
401 get_ipython().run_line_magic('matplotlib', 'inline')
402
403
404 # In[ ]:
405
406
407 category_index = label_map_util.create_category_index_from_labelmap(files['LABELMAP'])
408
409
410 # In[ ]:
411
412
413 IMAGE_PATH = os.path.join(paths['IMAGE_PATH'], 'test', 'IMG_20220307_130610.jpg')
414
415
416 # In[ ]:
417
418
419 IMAGE_PATH
420
421
422 # In[ ]:
423
424
425
426
427 img = cv2.imread(IMAGE_PATH)
428 image_np = np.array(img)
429
430 input_tensor = tf.convert_to_tensor(np.expand_dims(image_np, 0), dtype=tf.float32)
431 detections = detect_fn(input_tensor)
432
433 num_detections = int(detections.pop('num_detections'))
434 detections = {key: value[0, :num_detections].numpy()
435               for key, value in detections.items()}

```

```

436 detections['num_detections'] = num_detections
437
438 # detection_classes should be ints.
439 detections['detection_classes'] = detections['detection_classes'].astype(np.int64)
440
441 label_id_offset = 1
442 image_np_with_detections = image_np.copy()
443
444 viz_utils.visualize_boxes_and_labels_on_image_array(
445     image_np_with_detections,
446     detections['detection_boxes'],
447     detections['detection_classes']+label_id_offset,
448     detections['detection_scores'],
449     category_index,
450     use_normalized_coordinates=True,
451     max_boxes_to_draw=5,
452     min_score_thresh=.8,
453     agnostic_mode=False)
454
455 plt.imshow(cv2.cvtColor(image_np_with_detections, cv2.COLOR_BGR2RGB))
456 plt.show()
457
458
459 # # 10. Real Time Detections from your Webcam
460
461 # In[ ]:
462
463
464 get_ipython().system('pip install opencv-python-headless -y')
465
466
467 # In[ ]:
468
469
470 import numpy as np
471 cap = cv2.VideoCapture(1)
472 width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
473 height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
474
475 while cap.isOpened():
476     ret, frame = cap.read()
477     image_np = np.array(frame)
478
479     input_tensor = tf.convert_to_tensor(np.expand_dims(image_np, 0), dtype=tf.float32)
480     detections = detect_fn(input_tensor)
481
482     num_detections = int(detections.pop('num_detections'))
483     detections = {key: value[0, :num_detections].numpy()
484                   for key, value in detections.items()}
485     detections['num_detections'] = num_detections
486

```

```

487     # detection_classes should be ints.
488     detections['detection_classes'] = detections['detection_classes'].astype(np.int64)
489
490     label_id_offset = 1
491     image_np_with_detections = image_np.copy()
492
493     viz_utils.visualize_boxes_and_labels_on_image_array(
494         image_np_with_detections,
495         detections['detection_boxes'],
496         detections['detection_classes']+label_id_offset,
497         detections['detection_scores'],
498         category_index,
499         use_normalized_coordinates=True,
500         max_boxes_to_draw=5,
501         min_score_thresh=.8,
502         agnostic_mode=False)
503
504     cv2.imshow('object detection', cv2.resize(image_np_with_detections, (800, 600)))
505
506     if cv2.waitKey(10) & 0xFF == ord('q'):
507         cap.release()
508         cv2.destroyAllWindows()
509         break
510
511
512     # # 10. Freezing the Graph
513
514     # In[ ]:
515
516
517     FREEZE_SCRIPT = os.path.join(paths['APIMODEL_PATH'], 'research', 'object_detection', 'exporter_main_v2.py ')
518
519
520     # In[ ]:
521
522
523     command = "python {} --input_type=image_tensor --pipeline_config_path={} --trained_checkpoint_dir={}
--output_directory={}".format(FREEZE_SCRIPT ,files['PIPELINE_CONFIG'], paths['CHECKPOINT_PATH'], paths['OUTPUT_PATH'])
524
525
526     # In[ ]:
527
528
529     print(command)
530
531
532     # In[ ]:
533
534
535     get_ipython().system('{command}')
536

```

```

537
538 # # 11. Conversion to TFJS
539
540 # In[ ]:
541
542
543 get_ipython().system('pip install tensorflowjs')
544
545
546 # In[ ]:
547
548
549 command = "tensorflowjs_converter --input_format=tf_saved_model
--output_node_names='detection_boxes,detection_classes,detection_features,detection_multiclass_scores,detection_scores,
num_detections,raw_detection_boxes,raw_detection_scores' --output_format=tfjs_graph_model
--signature_name=serving_default {} {}".format(os.path.join(paths['OUTPUT_PATH'], 'saved_model'), paths['TFJS_PATH'])
550
551
552 # In[ ]:
553
554
555 print(command)
556
557
558 # In[ ]:
559
560
561 get_ipython().system('{command}')
562
563
564 # In[ ]:
565
566
567 # Test Code: https://github.com/nicknochnack/RealTimeSignLanguageDetectionwithTFJS
568
569
570 # # 12. Conversion to TFLite
571
572 # In[ ]:
573
574
575 TFLITE_SCRIPT = os.path.join(paths['APIMODEL_PATH'], 'research', 'object_detection', 'export_tflite_graph_tf2.py ')
576
577
578 # In[ ]:
579
580
581 command = "python {} --pipeline_config_path={} --trained_checkpoint_dir={} --output_directory={} {}".format(TFLITE_SCRIPT
,files['PIPELINE_CONFIG'], paths['CHECKPOINT_PATH'], paths['TFLITE_PATH'])
582
583

```

```

584 # In[ ]:
585
586
587 print(command)
588
589
590 # In[ ]:
591
592
593 get_ipython().system('{command}')
594
595
596 # In[ ]:
597
598
599 FROZEN_TFLITE_PATH = os.path.join(paths['TFLITE_PATH'], 'saved_model')
600 TFLITE_MODEL = os.path.join(paths['TFLITE_PATH'], 'saved_model', 'detect.tflite')
601
602
603 # In[ ]:
604
605
606 command = "tflite_convert --saved_model_dir={} --output_file={} --input_shapes=1,300,300,3
--input_arrays=normalized_input_image_tensor
--output_arrays='TFLite_Detection_PostProcess','TFLite_Detection_PostProcess:1','TFLite_Detection_PostProcess:2','TFLite_Detection_PostProcess:3' --inference_type=FLOAT --allow_custom_ops".format(FROZEN_TFLITE_PATH, TFLITE_MODEL, )
607
608
609 # In[ ]:
610
611
612 print(command)
613
614
615 # In[ ]:
616
617
618 get_ipython().system('{command}')
619
620
621 # # 13. Zip and Export Models
622
623 # In[ ]:
624
625
626 get_ipython().system("tar -czf models.tar.gz {paths['CHECKPOINT_PATH']}")
627
628
629 # In[ ]:
630
631

```

```
632 from google.colab import drive
633 drive.mount('/content/drive')
634
635
```