```python
#!/usr/bin/env python
# coding: utf-8

# # 0. Setup Paths

# In[1]:


#Traning and object detectin. Found here: https://github.com/nicknochnack/TFODCourse/blob/main/2.%20Training%20and%20Detection.ipynb
import os


# In[2]:


CUSTOM_MODEL_NAME = 'efficientdet_d1_coco17_tpu-32_V12_10s'
PRETRAINED_MODEL_NAME = 'efficientdet_d1_coco17_tpu-32'
PRETRAINED_MODEL_URL = 'http://download.tensorflow.org/models/object_detection/tf2/20200711/efficientdet_d1_coco17_tpu-32.tar.gz'
TF_RECORD_SCRIPT_NAME = 'generate_tfrecord.py'
LABEL_MAP_NAME = 'label_map.pbtxt'


# In[3]:


paths = {
    'WORKSPACE_PATH': os.path.join('Tensorflow', 'workspace'),
    'SCRIPTS_PATH': os.path.join('Tensorflow','scripts'),
    'APIMODEL_PATH': os.path.join('Tensorflow','models'),
    'ANNOTATION_PATH': os.path.join('Tensorflow', 'workspace','annotations'),
    'IMAGE_PATH': os.path.join('Tensorflow', 'workspace','images'),
    'MODEL_PATH': os.path.join('Tensorflow', 'workspace','models'),
    'PRETRAINED_MODEL_PATH': os.path.join('Tensorflow', 'workspace','pre-trained-models'),
    'CHECKPOINT_PATH': os.path.join('Tensorflow', 'workspace','models',CUSTOM_MODEL_NAME),
    'OUTPUT_PATH': os.path.join('Tensorflow', 'workspace','models',CUSTOM_MODEL_NAME, 'export'),
    'TFJS_PATH':os.path.join('Tensorflow', 'workspace','models',CUSTOM_MODEL_NAME, 'tfjsexport'),
    'TFLITE_PATH':os.path.join('Tensorflow', 'workspace','models',CUSTOM_MODEL_NAME, 'tfliteexport'),
    'PROTOC_PATH':os.path.join('Tensorflow','protoc')
 }


# In[4]:


files = {
    'PIPELINE_CONFIG':os.path.join('Tensorflow', 'workspace','models', CUSTOM_MODEL_NAME, 'pipeline.config'),
    'TF_RECORD_SCRIPT': os.path.join(paths['SCRIPTS_PATH'], TF_RECORD_SCRIPT_NAME),
    'LABELMAP': os.path.join(paths['ANNOTATION_PATH'], LABEL_MAP_NAME)
}
```

```
# In[5]:


for path in paths.values():
    if not os.path.exists(path):
        if os.name == 'posix':
            get_ipython().system('mkdir -p {path}')
        if os.name == 'nt':
            get_ipython().system('mkdir {path}')


# # 1. Download TF Models Pretrained Models from Tensorflow Model Zoo and Install TFOD

# In[ ]:


# https://www.tensorflow.org/install/source_windows


# In[6]:


if os.name=='nt':
    get_ipython().system('pip install wget')
    import wget


# In[7]:


if not os.path.exists(os.path.join(paths['APIMODEL_PATH'], 'research', 'object_detection')):
    get_ipython().system("git clone https://github.com/tensorflow/models {paths['APIMODEL_PATH']}")


# In[ ]:


# Install Tensorflow Object Detection
if os.name=='posix':
    get_ipython().system('apt-get install protobuf-compiler')
    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 . ')

if os.name=='nt':
    url="https://github.com/protocolbuffers/protobuf/releases/download/v3.15.6/protoc-3.15.6-win64.zip"
    wget.download(url)
    get_ipython().system("move protoc-3.15.6-win64.zip {paths['PROTOC_PATH']}")
    get_ipython().system("cd {paths['PROTOC_PATH']} && tar -xf protoc-3.15.6-win64.zip")
    os.environ['PATH'] += os.pathsep + os.path.abspath(os.path.join(paths['PROTOC_PATH'], 'bin'))
```

```
    get_ipython().system('cd Tensorflow/models/research && protoc object_detection/protos/*.proto --python_out=. && copy object_detection\\\\packages\\\\tf2\\\\setup.py setup.py && python setup.py build && python setup.py install')
    get_ipython().system('cd Tensorflow/models/research/slim && pip install -e . ')


# In[ ]:


get_ipython().system('pip install numpy==1.18.1')


# In[24]:


VERIFICATION_SCRIPT = os.path.join(paths['APIMODEL_PATH'], 'research', 'object_detection', 'builders', 'model_builder_tf2_test.py')
# Verify Installation
get_ipython().system('python {VERIFICATION_SCRIPT}')


# In[ ]:


get_ipython().system('pip install tensorflow --upgrade')


# In[ ]:


get_ipython().system('pip uninstall protobuf matplotlib -y')
get_ipython().system('pip install protobuf matplotlib==3.2')


# In[ ]:


get_ipython().system('pip install matplotlib')


# In[ ]:


get_ipython().system('pip install -U pip wheel setuptools')


# In[ ]:


get_ipython().system('pip install pyyaml')


```

```
# In[ ]:


get_ipython().system('pip install kaggle')


# In[8]:


import object_detection


# In[ ]:


get_ipython().system('pip list')


# In[9]:


if os.name =='posix':
    get_ipython().system('wget {PRETRAINED_MODEL_URL}')
    get_ipython().system("mv {PRETRAINED_MODEL_NAME+'.tar.gz'} {paths['PRETRAINED_MODEL_PATH']}")
    get_ipython().system("cd {paths['PRETRAINED_MODEL_PATH']} && tar -zxvf {PRETRAINED_MODEL_NAME+'.tar.gz'}")
if os.name == 'nt':
    wget.download(PRETRAINED_MODEL_URL)
    get_ipython().system("move {PRETRAINED_MODEL_NAME+'.tar.gz'} {paths['PRETRAINED_MODEL_PATH']}")
    get_ipython().system("cd {paths['PRETRAINED_MODEL_PATH']} && tar -zxvf {PRETRAINED_MODEL_NAME+'.tar.gz'}")


# # 2. Create Label Map

# In[10]:


labels = [{'name':'Sveis', 'id':1},{'name':'SveisSlipt', 'id':2},{'name':'SveisSliptLitt', 'id':3}]

with open(files['LABELMAP'], 'w') as f:
    for label in labels:
        f.write('item { \n')
        f.write('\tname:\'{}\'\n'.format(label['name']))
        f.write('\tid:{}\n'.format(label['id']))
        f.write('}\n')


# # 3. Create TF records

# In[ ]:


```

```
# OPTIONAL IF RUNNING ON COLAB
ARCHIVE_FILES = os.path.join(paths['IMAGE_PATH'], 'archive.tar.gz')
if os.path.exists(ARCHIVE_FILES):
  get_ipython().system('tar -zxvf {ARCHIVE_FILES}')


# In[11]:


if not os.path.exists(files['TF_RECORD_SCRIPT']):
    get_ipython().system("git clone https://github.com/nicknochnack/GenerateTFRecord {paths['SCRIPTS_PATH']}")


# In[12]:


get_ipython().system("python {files['TF_RECORD_SCRIPT']} -x {os.path.join(paths['IMAGE_PATH'], 'train')} -l {files['LABELMAP']} -o {os.path.join(paths['ANNOTATION_PATH'], 'train.record')} ")
get_ipython().system("python {files['TF_RECORD_SCRIPT']} -x {os.path.join(paths['IMAGE_PATH'], 'test')} -l {files['LABELMAP']} -o {os.path.join(paths['ANNOTATION_PATH'], 'test.record')} ")


# In[ ]:


get_ipython().system('pip install pytz')


# # 4. Copy Model Config to Training Folder

# In[13]:


if os.name =='posix':
    get_ipython().system("cp {os.path.join(paths['PRETRAINED_MODEL_PATH'], PRETRAINED_MODEL_NAME, 'pipeline.config')} {os.path.join(paths['CHECKPOINT_PATH'])}")
if os.name == 'nt':
    get_ipython().system("copy {os.path.join(paths['PRETRAINED_MODEL_PATH'], PRETRAINED_MODEL_NAME, 'pipeline.config')} {os.path.join(paths['CHECKPOINT_PATH'])}")


# # 5. Update Config For Transfer Learning

# In[14]:


import tensorflow as tf
from object_detection.utils import config_util
from object_detection.protos import pipeline_pb2
from google.protobuf import text_format
```

```
# In[15]:


config = config_util.get_configs_from_pipeline_file(files['PIPELINE_CONFIG'])


# In[ ]:


config


# In[16]:


pipeline_config = pipeline_pb2.TrainEvalPipelineConfig()
with tf.io.gfile.GFile(files['PIPELINE_CONFIG'], "r") as f:
    proto_str = f.read()

    text_format.Merge(proto_str, pipeline_config)


# In[ ]:


pipeline_config.model.ssd.num_classes = len(labels)
pipeline_config.train_config.batch_size = 4
pipeline_config.train_config.fine_tune_checkpoint = os.path.join(paths['PRETRAINED_MODEL_PATH'], PRETRAINED_MODEL_NAME, 'checkpoint', 'ckpt-0')
pipeline_config.train_config.fine_tune_checkpoint_type = "detection"
pipeline_config.train_input_reader.label_map_path= files['LABELMAP']
pipeline_config.train_input_reader.tf_record_input_reader.input_path[:] = [os.path.join(paths['ANNOTATION_PATH'], 'train.record')]
pipeline_config.eval_input_reader[0].label_map_path = files['LABELMAP']
pipeline_config.eval_input_reader[0].tf_record_input_reader.input_path[:] = [os.path.join(paths['ANNOTATION_PATH'], 'test.record')]


# In[ ]:


config


# 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[
    'CHECKPOINT_PATH'],files['PIPELINE_CONFIG'])
309
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
```

```
# In[ ]:


command = "python {} --model_dir={} --pipeline_config_path={} --checkpoint_dir={}".format(TRAINING_SCRIPT, paths['CHECKPOINT_PATH'],files['PIPELINE_CONFIG'], paths['CHECKPOINT_PATH'])


# In[ ]:


print(command)


# In[ ]:


get_ipython().system('{command}')


# In[ ]:


get_ipython().system('pip conda install cudnn')


# # 8. Load Train Model From Checkpoint

# In[ ]:


import os
import tensorflow as tf
from object_detection.utils import label_map_util
from object_detection.utils import visualization_utils as viz_utils
from object_detection.builders import model_builder
from object_detection.utils import config_util


# In[ ]:


# Load pipeline config and build a detection model
configs = config_util.get_configs_from_pipeline_file(files['PIPELINE_CONFIG'])
detection_model = model_builder.build(model_config=configs['model'], is_training=False)

# Restore checkpoint
ckpt = tf.compat.v2.train.Checkpoint(model=detection_model)
ckpt.restore(os.path.join(paths['CHECKPOINT_PATH'], 'ckpt-41')).expect_partial() ##Viktig å endre ckpt-(nummer) til den site treningsmodellen

```

```
@tf.function
def detect_fn(image):
    image, shapes = detection_model.preprocess(image)
    prediction_dict = detection_model.predict(image, shapes)
    detections = detection_model.postprocess(prediction_dict, shapes)
    return detections


# # 9. Detect from an Image

# In[ ]:


import cv2
import numpy as np
from matplotlib import pyplot as plt
get_ipython().run_line_magic('matplotlib', 'inline')


# In[ ]:


category_index = label_map_util.create_category_index_from_labelmap(files['LABELMAP'])


# In[ ]:


IMAGE_PATH = os.path.join(paths['IMAGE_PATH'], 'test', 'IMG_20220307_130610.jpg')


# In[ ]:


IMAGE_PATH


# In[ ]:




img = cv2.imread(IMAGE_PATH)
image_np = np.array(img)

input_tensor = tf.convert_to_tensor(np.expand_dims(image_np, 0), dtype=tf.float32)
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()}
```

```
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
```

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

    label_id_offset = 1
    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']+label_id_offset,
                detections['detection_scores'],
                category_index,
                use_normalized_coordinates=True,
                max_boxes_to_draw=5,
                min_score_thresh=.8,
                agnostic_mode=False)

    cv2.imshow('object detection',  cv2.resize(image_np_with_detections, (800, 600)))

    if cv2.waitKey(10) & 0xFF == ord('q'):
        cap.release()
        cv2.destroyAllWindows()
        break


# # 10. Freezing the Graph

# In[ ]:


FREEZE_SCRIPT = os.path.join(paths['APIMODEL_PATH'], 'research', 'object_detection', 'exporter_main_v2.py ')


# In[ ]:


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'])


# In[ ]:


print(command)


# In[ ]:


get_ipython().system('{command}')
```

```
# # 11. Conversion to TFJS

# In[ ]:


get_ipython().system('pip install tensorflowjs')


# In[ ]:


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'])


# In[ ]:


print(command)


# In[ ]:


get_ipython().system('{command}')


# In[ ]:


# Test Code: https://github.com/nicknochnack/RealTimeSignLanguageDetectionwithTFJS

# # 12. Conversion to TFLite

# In[ ]:


TFLITE_SCRIPT = os.path.join(paths['APIMODEL_PATH'], 'research', 'object_detection', 'export_tflite_graph_tf2.py ')


# In[ ]:


command = "python {} --pipeline_config_path={} --trained_checkpoint_dir={} --output_directory={}".format(TFLITE_SCRIPT,files['PIPELINE_CONFIG'], paths['CHECKPOINT_PATH'], paths['TFLITE_PATH'])


```

```
# In[ ]:


print(command)


# In[ ]:


get_ipython().system('{command}')


# In[ ]:


FROZEN_TFLITE_PATH = os.path.join(paths['TFLITE_PATH'], 'saved_model')
TFLITE_MODEL = os.path.join(paths['TFLITE_PATH'], 'saved_model', 'detect.tflite')


# In[ ]:


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','TFLit
e_Detection_PostProcess:3' --inference_type=FLOAT --allow_custom_ops".format(FROZEN_TFLITE_PATH, TFLITE_MODEL, )


# In[ ]:


print(command)


# In[ ]:


get_ipython().system('{command}')


# # 13. Zip and Export Models

# In[ ]:


get_ipython().system("tar -czf models.tar.gz {paths['CHECKPOINT_PATH']}")


# In[ ]:
```

```
from google.colab import drive
drive.mount('/content/drive')
```