# Lint as: python2, python3

# Copyright 2017 The TensorFlow Authors. All Rights Reserved.

#

# Licensed under the Apache License, Version 2.0 (the "License");

# you may not use this file except in compliance with the License.

# You may obtain a copy of the License at

#

# http://www.apache.org/licenses/LICENSE-2.0

#

# Unless required by applicable law or agreed to in writing, software

# distributed under the License is distributed on an "AS IS" BASIS,

# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

# See the License for the specific language governing permissions and

# limitations under the License.

# ==============================================================================

r"""Tool to export an object detection model for inference.

Prepares an object detection tensorflow graph for inference using model

configuration and a trained checkpoint. Outputs inference

graph, associated checkpoint files, a frozen inference graph and a

SavedModel (https://tensorflow.github.io/serving/serving\_basic.html).

The inference graph contains one of three input nodes depending on the user

specified option.

\* `image\_tensor`: Accepts a uint8 4-D tensor of shape [None, None, None, 3]

\* `encoded\_image\_string\_tensor`: Accepts a 1-D string tensor of shape [None]

containing encoded PNG or JPEG images. Image resolutions are expected to be

the same if more than 1 image is provided.

\* `tf\_example`: Accepts a 1-D string tensor of shape [None] containing

serialized TFExample protos. Image resolutions are expected to be the same

if more than 1 image is provided.

and the following output nodes returned by the model.postprocess(..):

\* `num\_detections`: Outputs float32 tensors of the form [batch]

that specifies the number of valid boxes per image in the batch.

\* `detection\_boxes`: Outputs float32 tensors of the form

[batch, num\_boxes, 4] containing detected boxes.

\* `detection\_scores`: Outputs float32 tensors of the form

[batch, num\_boxes] containing class scores for the detections.

\* `detection\_classes`: Outputs float32 tensors of the form

[batch, num\_boxes] containing classes for the detections.

\* `raw\_detection\_boxes`: Outputs float32 tensors of the form

[batch, raw\_num\_boxes, 4] containing detection boxes without

post-processing.

\* `raw\_detection\_scores`: Outputs float32 tensors of the form

[batch, raw\_num\_boxes, num\_classes\_with\_background] containing class score

logits for raw detection boxes.

\* `detection\_masks`: (Optional) Outputs float32 tensors of the form

[batch, num\_boxes, mask\_height, mask\_width] containing predicted instance

masks for each box if its present in the dictionary of postprocessed

tensors returned by the model.

\* detection\_multiclass\_scores: (Optional) Outputs float32 tensor of shape

[batch, num\_boxes, num\_classes\_with\_background] for containing class

score distribution for detected boxes including background if any.

\* detection\_features: (Optional) float32 tensor of shape

[batch, num\_boxes, roi\_height, roi\_width, depth]

containing classifier features

Notes:

\* This tool uses `use\_moving\_averages` from eval\_config to decide which

weights to freeze.

Example Usage:

--------------

python export\_inference\_graph.py \

--input\_type image\_tensor \

--pipeline\_config\_path path/to/ssd\_inception\_v2.config \

--trained\_checkpoint\_prefix path/to/model.ckpt \

--output\_directory path/to/exported\_model\_directory

The expected output would be in the directory

path/to/exported\_model\_directory (which is created if it does not exist)

with contents:

- inference\_graph.pbtxt

- model.ckpt.data-00000-of-00001

- model.ckpt.info

- model.ckpt.meta

- frozen\_inference\_graph.pb

+ saved\_model (a directory)

Config overrides (see the `config\_override` flag) are text protobufs

(also of type pipeline\_pb2.TrainEvalPipelineConfig) which are used to override

certain fields in the provided pipeline\_config\_path. These are useful for

making small changes to the inference graph that differ from the training or

eval config.

Example Usage (in which we change the second stage post-processing score

threshold to be 0.5):

python export\_inference\_graph.py \

--input\_type image\_tensor \

--pipeline\_config\_path path/to/ssd\_inception\_v2.config \

--trained\_checkpoint\_prefix path/to/model.ckpt \

--output\_directory path/to/exported\_model\_directory \

--config\_override " \

model{ \

faster\_rcnn { \

second\_stage\_post\_processing { \

batch\_non\_max\_suppression { \

score\_threshold: 0.5 \

} \

} \

} \

}"

"""

import tensorflow.compat.v1 as tf

from google.protobuf import text\_format

from object\_detection import exporter

from object\_detection.protos import pipeline\_pb2

flags = tf.app.flags

flags.DEFINE\_string('input\_type', 'image\_tensor', 'Type of input node. Can be '

'one of [`image\_tensor`, `encoded\_image\_string\_tensor`, '

'`tf\_example`]')

flags.DEFINE\_string('input\_shape', None,

'If input\_type is `image\_tensor`, this can explicitly set '

'the shape of this input tensor to a fixed size. The '

'dimensions are to be provided as a comma-separated list '

'of integers. A value of -1 can be used for unknown '

'dimensions. If not specified, for an `image\_tensor, the '

'default shape will be partially specified as '

'`[None, None, None, 3]`.')

flags.DEFINE\_string('pipeline\_config\_path', None,

'Path to a pipeline\_pb2.TrainEvalPipelineConfig config '

'file.')

flags.DEFINE\_string('trained\_checkpoint\_prefix', None,

'Path to trained checkpoint, typically of the form '

'path/to/model.ckpt')

flags.DEFINE\_string('output\_directory', None, 'Path to write outputs.')

flags.DEFINE\_string('config\_override', '',

'pipeline\_pb2.TrainEvalPipelineConfig '

'text proto to override pipeline\_config\_path.')

flags.DEFINE\_boolean('write\_inference\_graph', False,

'If true, writes inference graph to disk.')

flags.DEFINE\_string('additional\_output\_tensor\_names', None,

'Additional Tensors to output, to be specified as a comma '

'separated list of tensor names.')

flags.DEFINE\_boolean('use\_side\_inputs', False,

'If True, uses side inputs as well as image inputs.')

flags.DEFINE\_string('side\_input\_shapes', None,

'If use\_side\_inputs is True, this explicitly sets '

'the shape of the side input tensors to a fixed size. The '

'dimensions are to be provided as a comma-separated list '

'of integers. A value of -1 can be used for unknown '

'dimensions. A `/` denotes a break, starting the shape of '

'the next side input tensor. This flag is required if '

'using side inputs.')

flags.DEFINE\_string('side\_input\_types', None,

'If use\_side\_inputs is True, this explicitly sets '

'the type of the side input tensors. The '

'dimensions are to be provided as a comma-separated list '

'of types, each of `string`, `integer`, or `float`. '

'This flag is required if using side inputs.')

flags.DEFINE\_string('side\_input\_names', None,

'If use\_side\_inputs is True, this explicitly sets '

'the names of the side input tensors required by the model '

'assuming the names will be a comma-separated list of '

'strings. This flag is required if using side inputs.')

tf.app.flags.mark\_flag\_as\_required('pipeline\_config\_path')

tf.app.flags.mark\_flag\_as\_required('trained\_checkpoint\_prefix')

tf.app.flags.mark\_flag\_as\_required('output\_directory')

FLAGS = flags.FLAGS

def main(\_):

pipeline\_config = pipeline\_pb2.TrainEvalPipelineConfig()

with tf.gfile.GFile(FLAGS.pipeline\_config\_path, 'r') as f:

text\_format.Merge(f.read(), pipeline\_config)

text\_format.Merge(FLAGS.config\_override, pipeline\_config)

if FLAGS.input\_shape:

input\_shape = [

int(dim) if dim != '-1' else None

for dim in FLAGS.input\_shape.split(',')

]

else:

input\_shape = None

if FLAGS.use\_side\_inputs:

side\_input\_shapes, side\_input\_names, side\_input\_types = (

exporter.parse\_side\_inputs(

FLAGS.side\_input\_shapes,

FLAGS.side\_input\_names,

FLAGS.side\_input\_types))

else:

side\_input\_shapes = None

side\_input\_names = None

side\_input\_types = None

if FLAGS.additional\_output\_tensor\_names:

additional\_output\_tensor\_names = list(

FLAGS.additional\_output\_tensor\_names.split(','))

else:

additional\_output\_tensor\_names = None

exporter.export\_inference\_graph(

FLAGS.input\_type, pipeline\_config, FLAGS.trained\_checkpoint\_prefix,

FLAGS.output\_directory, input\_shape=input\_shape,

write\_inference\_graph=FLAGS.write\_inference\_graph,

additional\_output\_tensor\_names=additional\_output\_tensor\_names,

use\_side\_inputs=FLAGS.use\_side\_inputs,

side\_input\_shapes=side\_input\_shapes,

side\_input\_names=side\_input\_names,

side\_input\_types=side\_input\_types)

if \_\_name\_\_ == '\_\_main\_\_':

tf.app.run()