ДОДАТОК А

Програмна реалізація

```
from __future__ import division
import io
import os
import unittest
from absl.testing import parameterized
import numpy as np
from PIL import Image
import six
import tensorflow.compat.v2 as tf
from object_detection import exporter_lib_v2
from object_detection.builders import model_builder
from object_detection.core import model
from object_detection.core import standard_fields as
fields
from object_detection.protos import pipeline_pb2
from object_detection.utils import dataset_util
from object_detection.utils import tf_version
if six.PY2:
  import mock  # pylint: disable=g-importing-
member,g-import-not-at-top
else:
  from unittest import mock  # pylint: disable=g-
importing-member,g-import-not-at-top

class FakeModel(model.DetectionModel):
  def __init__(self, conv_weight_scalar=1.0):
    super(FakeModel, self).__init__(num_classes=2)
    self._conv = tf.keras.layers.Conv2D(
        filters=1, kernel_size=1, strides=(1, 1),
padding='valid',

kernel_initializer=tf.keras.initializers.Constant(
            value=conv_weight_scalar))
  def preprocess(self, inputs):
    return tf.identity(inputs),
exporter_lib_v2.get_true_shapes(inputs)
  def predict(self, preprocessed_inputs,
true_image_shapes, **side_inputs):
    return_dict = {'image':
self._conv(preprocessed_inputs)}
```

```
    if 'side_inp_1' in side_inputs:
      return_dict['image'] +=
side_inputs['side_inp_1']
    return return_dict
  def postprocess(self, prediction_dict,
true_image_shapes):
    predict_tensor_sum =
tf.reduce_sum(prediction_dict['image'])
    with
tf.control_dependencies(list(prediction_dict.values()
)):
      postprocessed_tensors = {
          'detection_boxes': tf.constant([[[0.0, 0.0,
0.5, 0.5],
                                           [0.5, 0.5,
0.8, 0.8]],
                                          [[0.5, 0.5,
1.0, 1.0],
                                           [0.0, 0.0,
0.0, 0.0]]], tf.float32),
          'detection_scores': predict_tensor_sum +
tf.constant(
              [[0.7, 0.6], [0.9, 0.0]], tf.float32),
          'detection_classes': tf.constant([[0, 1],
                                            [1, 0]],
tf.float32),
          'num_detections': tf.constant([2, 1],
tf.float32),
      }
    return postprocessed_tensors
  def predict_masks_from_boxes(self, prediction_dict,
true_image_shapes, boxes):
    output_dict = self.postprocess(prediction_dict,
true_image_shapes)
    output_dict.update({
        'detection_masks': tf.ones(shape=(1, 2, 16),
dtype=tf.float32),
    })
    return output_dict
  def restore_map(self, checkpoint_path,
fine_tune_checkpoint_type):
    pass
  def restore_from_objects(self,
fine_tune_checkpoint_type):
```

```
    pass
  def loss(self, prediction_dict, true_image_shapes):
    pass
  def regularization_losses(self):
    pass
  def updates(self):
    pass

@unittest.skipIf(tf_version.is_tf1(), 'Skipping TF2.X
only test.')
class ExportInferenceGraphTest(tf.test.TestCase,
parameterized.TestCase):
  def _save_checkpoint_from_mock_model(
      self, checkpoint_dir, conv_weight_scalar=6.0):
    mock_model = FakeModel(conv_weight_scalar)
    fake_image = tf.zeros(shape=[1, 10, 10, 3],
dtype=tf.float32)
    preprocessed_inputs, true_image_shapes =
mock_model.preprocess(fake_image)
    predictions =
mock_model.predict(preprocessed_inputs,
true_image_shapes)
    mock_model.postprocess(predictions,
true_image_shapes)
    ckpt = tf.train.Checkpoint(model=mock_model)
    exported_checkpoint_manager =
tf.train.CheckpointManager(
        ckpt, checkpoint_dir, max_to_keep=1)

exported_checkpoint_manager.save(checkpoint_number=0)
  @parameterized.parameters(
      {'input_type': 'image_tensor'},
      {'input_type': 'encoded_image_string_tensor'},
      {'input_type': 'tf_example'},
  )
  def test_export_yields_correct_directory_structure(
      self, input_type='image_tensor'):
    tmp_dir = self.get_temp_dir()
    self._save_checkpoint_from_mock_model(tmp_dir)
    with mock.patch.object(
        model_builder, 'build', autospec=True) as
mock_builder:
      mock_builder.return_value = FakeModel()
```

```
exporter_lib_v2.INPUT_BUILDER_UTIL_MAP['model_build']
= mock_builder
      output_directory = os.path.join(tmp_dir,
'output')
      pipeline_config =
pipeline_pb2.TrainEvalPipelineConfig()
      exporter_lib_v2.export_inference_graph(
          input_type=input_type,
          pipeline_config=pipeline_config,
          trained_checkpoint_dir=tmp_dir,
          output_directory=output_directory)
      self.assertTrue(os.path.exists(os.path.join(
          output_directory, 'saved_model',
'saved_model.pb')))
      self.assertTrue(os.path.exists(os.path.join(
          output_directory, 'saved_model',
'variables', 'variables.index')))
      self.assertTrue(os.path.exists(os.path.join(
          output_directory, 'saved_model',
'variables',
          'variables.data-00000-of-00001')))
      self.assertTrue(os.path.exists(os.path.join(
          output_directory, 'checkpoint', 'ckpt-
0.index')))
      self.assertTrue(os.path.exists(os.path.join(
          output_directory, 'checkpoint', 'ckpt-
0.data-00000-of-00001')))
      self.assertTrue(os.path.exists(os.path.join(
          output_directory, 'pipeline.config')))
  def get_dummy_input(self, input_type):
    """Get dummy input for the given input type."""
    if input_type == 'image_tensor':
      return np.zeros((1, 20, 20, 3), dtype=np.uint8)
    if input_type == 'float_image_tensor':
      return np.zeros((1, 20, 20, 3),
dtype=np.float32)
    elif input_type == 'encoded_image_string_tensor':
      image = Image.new('RGB', (20, 20))
      byte_io = io.BytesIO()
      image.save(byte_io, 'PNG')
      return [byte_io.getvalue()]
    elif input_type == 'tf_example':
      image_tensor = tf.zeros((20, 20, 3),
dtype=tf.uint8)
```

```
      encoded_jpeg =
tf.image.encode_jpeg(tf.constant(image_tensor)).numpy
()
      example = tf.train.Example(
          features=tf.train.Features(
              feature={
                  'image/encoded':

dataset_util.bytes_feature(encoded_jpeg),
                  'image/format':

dataset_util.bytes_feature(six.b('jpeg')),
                  'image/source_id':

dataset_util.bytes_feature(six.b('image_id')),
              })).SerializeToString()
      return [example]
  @parameterized.parameters(
      {'input_type': 'image_tensor'},
      {'input_type': 'encoded_image_string_tensor'},
      {'input_type': 'tf_example'},
      {'input_type': 'float_image_tensor'},
  )
  def test_export_saved_model_and_run_inference(
      self, input_type='image_tensor'):
    tmp_dir = self.get_temp_dir()
    self._save_checkpoint_from_mock_model(tmp_dir)
    with mock.patch.object(
        model_builder, 'build', autospec=True) as
mock_builder:
      mock_builder.return_value = FakeModel()

exporter_lib_v2.INPUT_BUILDER_UTIL_MAP['model_build']
= mock_builder
      output_directory = os.path.join(tmp_dir,
'output')
      pipeline_config =
pipeline_pb2.TrainEvalPipelineConfig()
      exporter_lib_v2.export_inference_graph(
          input_type=input_type,
          pipeline_config=pipeline_config,
          trained_checkpoint_dir=tmp_dir,
          output_directory=output_directory)
      saved_model_path =
```

```
os.path.join(output_directory, 'saved_model')
      detect_fn =
tf.saved_model.load(saved_model_path)
      image = self.get_dummy_input(input_type)
      detections = detect_fn(tf.constant(image))
      detection_fields = fields.DetectionResultFields

self.assertAllClose(detections[detection_fields.detec
tion_boxes],
                          [[[0.0, 0.0, 0.5, 0.5],
                            [0.5, 0.5, 0.8, 0.8]],
                           [[0.5, 0.5, 1.0, 1.0],
                            [0.0, 0.0, 0.0, 0.0]]])

self.assertAllClose(detections[detection_fields.detec
tion_scores],
                          [[0.7, 0.6], [0.9, 0.0]])

self.assertAllClose(detections[detection_fields.detec
tion_classes],
                          [[1, 2], [2, 1]])

self.assertAllClose(detections[detection_fields.num_d
etections], [2, 1])
  @parameterized.parameters(
      {'use_default_serving': True},
      {'use_default_serving': False}
  )
  def
test_export_saved_model_and_run_inference_with_side_i
nputs(
      self, input_type='image_tensor',
use_default_serving=True):
    tmp_dir = self.get_temp_dir()
    self._save_checkpoint_from_mock_model(tmp_dir)
    with mock.patch.object(
        model_builder, 'build', autospec=True) as
mock_builder:
      mock_builder.return_value = FakeModel()

exporter_lib_v2.INPUT_BUILDER_UTIL_MAP['model_build']
= mock_builder
      output_directory = os.path.join(tmp_dir,
'output')
```

```
      pipeline_config =
pipeline_pb2.TrainEvalPipelineConfig()
      exporter_lib_v2.export_inference_graph(
          input_type=input_type,
          pipeline_config=pipeline_config,
          trained_checkpoint_dir=tmp_dir,
          output_directory=output_directory,
          use_side_inputs=True,
          side_input_shapes='1/2,2',
          side_input_names='side_inp_1,side_inp_2',
          side_input_types='tf.float32,tf.uint8')
      saved_model_path =
os.path.join(output_directory, 'saved_model')
      detect_fn =
tf.saved_model.load(saved_model_path)
      detect_fn_sig =
detect_fn.signatures['serving_default']
      image =
tf.constant(self.get_dummy_input(input_type))
      side_input_1 = np.ones((1,), dtype=np.float32)
      side_input_2 = np.ones((2, 2), dtype=np.uint8)
      if use_default_serving:
        detections =
detect_fn_sig(input_tensor=image,

side_inp_1=tf.constant(side_input_1),

side_inp_2=tf.constant(side_input_2))
      else:
        detections = detect_fn(image,

tf.constant(side_input_1),

tf.constant(side_input_2))
      detection_fields = fields.DetectionResultFields

self.assertAllClose(detections[detection_fields.detec
tion_boxes],
                          [[[0.0, 0.0, 0.5, 0.5],
                            [0.5, 0.5, 0.8, 0.8]],
                           [[0.5, 0.5, 1.0, 1.0],
                            [0.0, 0.0, 0.0, 0.0]]])

self.assertAllClose(detections[detection_fields.detec
```

```
tion_scores],
                            [[400.7, 400.6], [400.9,
400.0]])

self.assertAllClose(detections[detection_fields.detec
tion_classes],
                            [[1, 2], [2, 1]])

self.assertAllClose(detections[detection_fields.num_d
etections], [2, 1])
  def
test_export_checkpoint_and_run_inference_with_image(s
elf):
    tmp_dir = self.get_temp_dir()
    self._save_checkpoint_from_mock_model(tmp_dir,
conv_weight_scalar=2.0)
    with mock.patch.object(
        model_builder, 'build', autospec=True) as
mock_builder:
      mock_builder.return_value = FakeModel()

exporter_lib_v2.INPUT_BUILDER_UTIL_MAP['model_build']
= mock_builder
      output_directory = os.path.join(tmp_dir,
'output')
      pipeline_config =
pipeline_pb2.TrainEvalPipelineConfig()
      exporter_lib_v2.export_inference_graph(
          input_type='image_tensor',
          pipeline_config=pipeline_config,
          trained_checkpoint_dir=tmp_dir,
          output_directory=output_directory)
      mock_model = FakeModel()
      ckpt = tf.compat.v2.train.Checkpoint(
          model=mock_model)
      checkpoint_dir = os.path.join(tmp_dir,
'output', 'checkpoint')
      manager = tf.compat.v2.train.CheckpointManager(
          ckpt, checkpoint_dir, max_to_keep=7)

ckpt.restore(manager.latest_checkpoint).expect_partia
l()
      fake_image = tf.ones(shape=[1, 5, 5, 3],
dtype=tf.float32)
```

```
      preprocessed_inputs, true_image_shapes =
mock_model.preprocess(fake_image)
      predictions =
mock_model.predict(preprocessed_inputs,
true_image_shapes)
      detections =
mock_model.postprocess(predictions,
true_image_shapes)
      # 150 = conv_weight_scalar * height * width *
channels = 2 * 5 * 5 * 3.

self.assertAllClose(detections['detection_scores'],
                          [[150 + 0.7, 150 + 0.6],
[150 + 0.9, 150 + 0.0]])

class
DetectionFromImageAndBoxModuleTest(tf.test.TestCase):
  def get_dummy_input(self, input_type):
    """Get dummy input for the given input type."""
    if input_type == 'image_tensor' or input_type ==
'image_and_boxes_tensor':
      return np.zeros((1, 20, 20, 3), dtype=np.uint8)
    if input_type == 'float_image_tensor':
      return np.zeros((1, 20, 20, 3),
dtype=np.float32)
    elif input_type == 'encoded_image_string_tensor':
      image = Image.new('RGB', (20, 20))
      byte_io = io.BytesIO()
      image.save(byte_io, 'PNG')
      return [byte_io.getvalue()]
    elif input_type == 'tf_example':
      image_tensor = tf.zeros((20, 20, 3),
dtype=tf.uint8)
      encoded_jpeg =
tf.image.encode_jpeg(tf.constant(image_tensor)).numpy
()
      example = tf.train.Example(
          features=tf.train.Features(
              feature={
                  'image/encoded':

dataset_util.bytes_feature(encoded_jpeg),
                  'image/format':
```

```
dataset_util.bytes_feature(six.b('jpeg')),
                  'image/source_id':

dataset_util.bytes_feature(six.b('image_id')),
              })).SerializeToString()
      return [example]
  def _save_checkpoint_from_mock_model(self,

checkpoint_dir,

conv_weight_scalar=6.0):
    mock_model = FakeModel(conv_weight_scalar)
    fake_image = tf.zeros(shape=[1, 10, 10, 3],
dtype=tf.float32)
    preprocessed_inputs, true_image_shapes =
mock_model.preprocess(fake_image)
    predictions =
mock_model.predict(preprocessed_inputs,
true_image_shapes)
    mock_model.postprocess(predictions,
true_image_shapes)
    ckpt = tf.train.Checkpoint(model=mock_model)
    exported_checkpoint_manager =
tf.train.CheckpointManager(
        ckpt, checkpoint_dir, max_to_keep=1)

exported_checkpoint_manager.save(checkpoint_number=0)
  def
test_export_saved_model_and_run_inference_for_segment
ation(
      self, input_type='image_and_boxes_tensor'):
    tmp_dir = self.get_temp_dir()
    self._save_checkpoint_from_mock_model(tmp_dir)
    with mock.patch.object(
        model_builder, 'build', autospec=True) as
mock_builder:
      mock_builder.return_value = FakeModel()

exporter_lib_v2.INPUT_BUILDER_UTIL_MAP['model_build']
= mock_builder
      output_directory = os.path.join(tmp_dir,
'output')
      pipeline_config =
pipeline_pb2.TrainEvalPipelineConfig()
```

```
      exporter_lib_v2.export_inference_graph(
          input_type=input_type,
          pipeline_config=pipeline_config,
          trained_checkpoint_dir=tmp_dir,
          output_directory=output_directory)
      saved_model_path =
os.path.join(output_directory, 'saved_model')
      detect_fn =
tf.saved_model.load(saved_model_path)
      image = self.get_dummy_input(input_type)
      boxes = tf.constant([
          [
              [0.0, 0.0, 0.5, 0.5],
              [0.5, 0.5, 0.8, 0.8],
          ],
      ])
      detections = detect_fn(tf.constant(image),
boxes)
      detection_fields = fields.DetectionResultFields
      self.assertIn(detection_fields.detection_masks,
detections)
      self.assertListEqual(

list(detections[detection_fields.detection_masks].sha
pe), [1, 2, 16])

if __name__ == '__main__':
  tf.enable_v2_behavior()
  tf.test.main()

from __future__ import absolute_import
from __future__ import division
from __future__ import print_function
from absl import flags
import tensorflow.compat.v1 as tf
from object_detection import model_lib
flags.DEFINE_string(
    'model_dir', None, 'Path to output model
directory '
    'where event and checkpoint files will be
written.')
flags.DEFINE_string('pipeline_config_path', None,
'Path to pipeline config '
                    'file.')
```

```
flags.DEFINE_integer('num_train_steps', None, 'Number
of train steps.')
flags.DEFINE_boolean('eval_training_data', False,
                     'If training data should be
evaluated for this job. Note '
                     'that one call only use this in
eval-only mode, and '
                     '`checkpoint_dir` must be
supplied.')
flags.DEFINE_integer('sample_1_of_n_eval_examples',
1, 'Will sample one of '
                     'every n eval input examples,
where n is provided.')
flags.DEFINE_integer('sample_1_of_n_eval_on_train_exa
mples', 5, 'Will sample '
                     'one of every n train input
examples for evaluation, '
                     'where n is provided. This is
only used if '
                     '`eval_training_data` is True.')
flags.DEFINE_string(
    'checkpoint_dir', None, 'Path to directory
holding a checkpoint.  If '
    '`checkpoint_dir` is provided, this binary
operates in eval-only mode, '
    'writing resulting metrics to `model_dir`.')
flags.DEFINE_boolean(
    'run_once', False, 'If running in eval-only mode,
whether to run just '
    'one round of eval vs running continuously
(default).'
)
flags.DEFINE_integer(
    'max_eval_retries', 0, 'If running continuous
eval, the maximum number of '
    'retries upon encountering
tf.errors.InvalidArgumentError. If negative, '
    'will always retry the evaluation.'
)
FLAGS = flags.FLAGS

def main(unused_argv):
  flags.mark_flag_as_required('model_dir')
  flags.mark_flag_as_required('pipeline_config_path')
```

```
  config =
tf.estimator.RunConfig(model_dir=FLAGS.model_dir)
  train_and_eval_dict =
model_lib.create_estimator_and_inputs(
      run_config=config,

pipeline_config_path=FLAGS.pipeline_config_path,
      train_steps=FLAGS.num_train_steps,

sample_1_of_n_eval_examples=FLAGS.sample_1_of_n_eval_
examples,
      sample_1_of_n_eval_on_train_examples=(

FLAGS.sample_1_of_n_eval_on_train_examples))
  estimator = train_and_eval_dict['estimator']
  train_input_fn =
train_and_eval_dict['train_input_fn']
  eval_input_fns =
train_and_eval_dict['eval_input_fns']
  eval_on_train_input_fn =
train_and_eval_dict['eval_on_train_input_fn']
  predict_input_fn =
train_and_eval_dict['predict_input_fn']
  train_steps = train_and_eval_dict['train_steps']
  if FLAGS.checkpoint_dir:
    if FLAGS.eval_training_data:
      name = 'training_data'
      input_fn = eval_on_train_input_fn
    else:
      name = 'validation_data'
      # The first eval input will be evaluated.
      input_fn = eval_input_fns[0]
    if FLAGS.run_once:
      estimator.evaluate(input_fn,
                         steps=None,

checkpoint_path=tf.train.latest_checkpoint(
                              FLAGS.checkpoint_dir))
    else:
      model_lib.continuous_eval(estimator,
FLAGS.checkpoint_dir, input_fn,
                                train_steps, name,
FLAGS.max_eval_retries)
  else:
```

```
    train_spec, eval_specs =
model_lib.create_train_and_eval_specs(
        train_input_fn,
        eval_input_fns,
        eval_on_train_input_fn,
        predict_input_fn,
        train_steps,
        eval_on_train_data=False)
    # Currently only a single Eval Spec is allowed.
    tf.estimator.train_and_evaluate(estimator,
train_spec, eval_specs[0])

if __name__ == '__main__':
  tf.app.run()

import unittest
import tensorflow.compat.v1 as tf
from google.protobuf import text_format
from object_detection.builders import
hyperparams_builder
from object_detection.models import
faster_rcnn_resnet_v1_fpn_keras_feature_extractor as
frcnn_res_fpn
from object_detection.protos import hyperparams_pb2
from object_detection.utils import tf_version

@unittest.skipIf(tf_version.is_tf1(), 'Skipping TF2.X
only test.')
class
FasterRCNNResnetV1FpnKerasFeatureExtractorTest(tf.tes
t.TestCase):
  def _build_conv_hyperparams(self):
    conv_hyperparams = hyperparams_pb2.Hyperparams()
    conv_hyperparams_text_proto = """
      regularizer {
        l2_regularizer {
        }
      }
      initializer {
        truncated_normal_initializer {
        }
      }
    """
    text_format.Parse(conv_hyperparams_text_proto,
```

```
conv_hyperparams)
    return
hyperparams_builder.KerasLayerHyperparams(conv_hyperp
arams)
  def _build_feature_extractor(self):
    return
frcnn_res_fpn.FasterRCNNResnet50FpnKerasFeatureExtrac
tor(
        is_training=False,

conv_hyperparams=self._build_conv_hyperparams(),
        first_stage_features_stride=16,
        batch_norm_trainable=False,
        weight_decay=0.0)
  def
test_extract_proposal_features_returns_expected_size(
self):
    feature_extractor =
self._build_feature_extractor()
    preprocessed_inputs = tf.random_uniform(
        [2, 448, 448, 3], maxval=255,
dtype=tf.float32)
    rpn_feature_maps =
feature_extractor.get_proposal_feature_extractor_mode
l(
        name='TestScope')(preprocessed_inputs)
    features_shapes = [tf.shape(rpn_feature_map)
                       for rpn_feature_map in
rpn_feature_maps]
    self.assertAllEqual(features_shapes[0].numpy(),
[2, 112, 112, 256])
    self.assertAllEqual(features_shapes[1].numpy(),
[2, 56, 56, 256])
    self.assertAllEqual(features_shapes[2].numpy(),
[2, 28, 28, 256])
    self.assertAllEqual(features_shapes[3].numpy(),
[2, 14, 14, 256])
    self.assertAllEqual(features_shapes[4].numpy(),
[2, 7, 7, 256])
  def
test_extract_proposal_features_half_size_input(self):
    feature_extractor =
self._build_feature_extractor()
    preprocessed_inputs = tf.random_uniform(
```

```
        [2, 224, 224, 3], maxval=255,
dtype=tf.float32)
    rpn_feature_maps =
feature_extractor.get_proposal_feature_extractor_mode
l(
        name='TestScope')(preprocessed_inputs)
    features_shapes = [tf.shape(rpn_feature_map)
                       for rpn_feature_map in
rpn_feature_maps]
    self.assertAllEqual(features_shapes[0].numpy(),
[2, 56, 56, 256])
    self.assertAllEqual(features_shapes[1].numpy(),
[2, 28, 28, 256])
    self.assertAllEqual(features_shapes[2].numpy(),
[2, 14, 14, 256])
    self.assertAllEqual(features_shapes[3].numpy(),
[2, 7, 7, 256])
    self.assertAllEqual(features_shapes[4].numpy(),
[2, 4, 4, 256])
  def
test_extract_box_classifier_features_returns_expected
_size(self):
    feature_extractor =
self._build_feature_extractor()
    proposal_feature_maps = tf.random_uniform(
        [3, 7, 7, 1024], maxval=255,
dtype=tf.float32)
    model =
feature_extractor.get_box_classifier_feature_extracto
r_model(
        name='TestScope')
    proposal_classifier_features = (
        model(proposal_feature_maps))
    features_shape =
tf.shape(proposal_classifier_features)
    self.assertAllEqual(features_shape.numpy(), [3,
1, 1, 1024])
```

ДОДАТОК Б
Відомість кваліфікаційної роботи

Позначення	Найменування	Дод. відомості
	Текстові документи	
1.	Пояснювальна записка	93 с.
	Інші документи	
2.	Презентаційні матеріали	19 плакатів

.	.	Прізвище та ініціали.	Підп.	Дата	
Розробив		Байкалов М.О.			Дослідження методів маркування візуальних об'єктів текстом в задачах розпізнавання зображень
Перевірив		Шевченко О. Ю.			
Н.контр.		Малєєва І.А.			
Затв.		Філатов В.О.			

Шифр групи	Код напр./спец.
СШІм-19-2	122
ХНУРЕ кафедра ШІ	