==

r"""Training executable for detection models.

This executable is used to train DetectionModels. There are two ways of

configuring the training job:

1) A single pipeline_pb2.TrainEvalPipelineConfig configuration file

can be specified by --pipeline_config_path.

Example usage:

```
./train \
    --logtostderr \
    --train_dir=path/to/train_dir \
    --pipeline_config_path=pipeline_config.pbtxt
```

2) Three configuration files can be provided: a model_pb2.DetectionModel configuration file to define what type of DetectionModel is being trained, an input_reader_pb2.InputReader file to specify what training data will be used and a train_pb2.TrainConfig file to configure training parameters.

Example usage:

```
    ./train \
        --logtostderr \
        --train_dir=path/to/train_dir \
        --model_config_path=model_config.pbtxt \
        --train_config_path=train_config.pbtxt \
        --input_config_path=train_input_config.pbtxt
"""

import functools
import json
import os
import tensorflow as tf

from object_detection.builders import dataset_builder
from object_detection.builders import graph_rewriter_builder
from object_detection.builders import model_builder
from object_detection.legacy import trainer
from object_detection.utils import config_util
```

```
tf.logging.set_verbosity(tf.logging.INFO)


flags = tf.app.flags
flags.DEFINE_string('master', '', 'Name of the TensorFlow master to use.')
flags.DEFINE_integer('task', 0, 'task id')
flags.DEFINE_integer('num_clones', 1, 'Number of clones to deploy per worker.')
flags.DEFINE_boolean('clone_on_cpu', False,
                     'Force clones to be deployed on CPU.  Note that even if '
                     'set to False (allowing ops to run on gpu), some ops may '
                     'still be run on the CPU if they have no GPU kernel.')
flags.DEFINE_integer('worker_replicas', 1, 'Number of worker+trainer '
                     'replicas.')
flags.DEFINE_integer('ps_tasks', 0,
                     'Number of parameter server tasks. If None, does not use '
                     'a parameter server.')
flags.DEFINE_string('train_dir', '',
                    'Directory to save the checkpoints and training summaries.')


flags.DEFINE_string('pipeline_config_path', '',
                    'Path to a pipeline_pb2.TrainEvalPipelineConfig config '
                    'file. If provided, other configs are ignored')


flags.DEFINE_string('train_config_path', '',
                    'Path to a train_pb2.TrainConfig config file.')
```

```
flags.DEFINE_string('input_config_path', '',
                    'Path to an input_reader_pb2.InputReader config file.')
flags.DEFINE_string('model_config_path', '',
                    'Path to a model_pb2.DetectionModel config file.')


FLAGS = flags.FLAGS



@tf.contrib.framework.deprecated(None, 'Use object_detection/model_main.py.')
def main(_):
  assert FLAGS.train_dir, '`train_dir` is missing.'
  if FLAGS.task == 0: tf.gfile.MakeDirs(FLAGS.train_dir)
  if FLAGS.pipeline_config_path:
    configs = config_util.get_configs_from_pipeline_file(
        FLAGS.pipeline_config_path)
    if FLAGS.task == 0:
      tf.gfile.Copy(FLAGS.pipeline_config_path,
                    os.path.join(FLAGS.train_dir, 'pipeline.config'),
                    overwrite=True)
  else:
    configs = config_util.get_configs_from_multiple_files(
        model_config_path=FLAGS.model_config_path,
        train_config_path=FLAGS.train_config_path,
        train_input_config_path=FLAGS.input_config_path)
    if FLAGS.task == 0:
```

```
  for name, config in [('model.config', FLAGS.model_config_path),
                       ('train.config', FLAGS.train_config_path),
                       ('input.config', FLAGS.input_config_path)]:
    tf.gfile.Copy(config, os.path.join(FLAGS.train_dir, name),
                  overwrite=True)

model_config = configs['model']
train_config = configs['train_config']
input_config = configs['train_input_config']

model_fn = functools.partial(
  model_builder.build,
  model_config=model_config,
  is_training=True)

def get_next(config):
 return dataset_builder.make_initializable_iterator(
    dataset_builder.build(config)).get_next()

create_input_dict_fn = functools.partial(get_next, input_config)

env = json.loads(os.environ.get('TF_CONFIG', '{}'))
cluster_data = env.get('cluster', None)
cluster = tf.train.ClusterSpec(cluster_data) if cluster_data else None
task_data = env.get('task', None) or {'type': 'master', 'index': 0}
```

```
task_info = type('TaskSpec', (object,), task_data)

# Parameters for a single worker.
ps_tasks = 0
worker_replicas = 1
worker_job_name = 'lonely_worker'
task = 0
is_chief = True
master = ''

if cluster_data and 'worker' in cluster_data:
  # Number of total worker replicas include "worker"s and the "master".
  worker_replicas = len(cluster_data['worker']) + 1
if cluster_data and 'ps' in cluster_data:
  ps_tasks = len(cluster_data['ps'])

if worker_replicas > 1 and ps_tasks < 1:
  raise ValueError('At least 1 ps task is needed for distributed training.')

if worker_replicas >= 1 and ps_tasks > 0:
  # Set up distributed training.
  server = tf.train.Server(tf.train.ClusterSpec(cluster), protocol='grpc',
                           job_name=task_info.type,
                           task_index=task_info.index)
  if task_info.type == 'ps':
```

```
    server.join()
    return


  worker_job_name = '%s/task:%d' % (task_info.type, task_info.index)
  task = task_info.index
  is_chief = (task_info.type == 'master')
  master = server.target

graph_rewriter_fn = None
if 'graph_rewriter_config' in configs:
  graph_rewriter_fn = graph_rewriter_builder.build(
      configs['graph_rewriter_config'], is_training=True)

trainer.train(
    create_input_dict_fn,
    model_fn,
    train_config,
    master,
    task,
    FLAGS.num_clones,
    worker_replicas,
    FLAGS.clone_on_cpu,
    ps_tasks,
    worker_job_name,
    is_chief,
```

```
      FLAGS.train_dir,

      graph_hook_fn=graph_rewriter_fn)



if __name__ == '__main__':

  tf.app.run()
```