This folder contains a custom training loop (CTL) implementation for ResNet50.

Before you begin

Please refer to the README in the parent directory for information on setup and preparing the data.

ResNet (custom training loop)

Similar to the estimator implementation, the Keras implementation has code for the ImageNet dataset. The ImageNet version uses a ResNet50 model implemented in `resnet_model.py`.

Pretrained Models

- ResNet50 Checkpoints
- ResNet50 TFHub: feature vector and classification

Again, if you did not download the data to the default directory, specify the location with the `--data_dir` flag:

```
python3 resnet_ctl_imagenet_main.py --data_dir=/path/to/imagenet
```

There are more flag options you can specify. Here are some examples:

- `--use_synthetic_data`: when set to true, synthetic data, rather than real data, are used;
- `--batch_size`: the batch size used for the model;
- `--model_dir`: the directory to save the model checkpoint;
- `--train_epochs`: number of epoches to run for training the model;
- `--train_steps`: number of steps to run for training the model. We now only support a number that is smaller than the number of batches in an epoch.
- `--skip_eval`: when set to true, evaluation as well as validation during training is skipped

For example, this is a typical command line to run with ImageNet data with batch size 128 per GPU:

```
python3 -m resnet_ctl_imagenet_main.py \
    --model_dir=/tmp/model_dir/something \
    --num_gpus=2 \
    --batch_size=128 \
    --train_epochs=90 \
    --train_steps=10 \
    --use_synthetic_data=false
```

See `common.py` for full list of options.

Using multiple GPUs

You can train these models on multiple GPUs using `tf.distribute.Strategy` API. You can read more about them in this guide.

In this example, we have made it easier to use is with just a command line flag `--num_gpus`. By default this flag is 1 if TensorFlow is compiled with CUDA, and 0 otherwise.

- --num_gpus=0: Uses tf.distribute.OneDeviceStrategy with CPU as the device.

- --num_gpus=1: Uses tf.distribute.OneDeviceStrategy with GPU as the device.
- --num_gpus=2+: Uses tf.distribute.MirroredStrategy to run synchronous distributed training across the GPUs.

If you wish to run without `tf.distribute.Strategy`, you can do so by setting `--distribution_strategy=off`.

Running on multiple GPU hosts

You can also train these models on multiple hosts, each with GPUs, using `tf.distribute.Strategy`.

The easiest way to run multi-host benchmarks is to set the `TF_CONFIG` appropriately at each host. e.g., to run using `MultiWorkerMirroredStrategy` on 2 hosts, the `cluster` in `TF_CONFIG` should have 2 `host:port` entries, and host `i` should have the `task` in `TF_CONFIG` set to `{"type": "worker", "index": i}`. `MultiWorkerMirroredStrategy` will automatically use all the available GPUs at each host.

Running on Cloud TPUs

Note: This model will **not** work with TPUs on Colab.

You can train the ResNet CTL model on Cloud TPUs using `tf.distribute.TPUStrategy`. If you are not familiar with Cloud TPUs, it is strongly recommended that you go through the quickstart to learn how to create a TPU and GCE VM.

To run ResNet model on a TPU, you must set `--distribution_strategy=tpu` and `--tpu=$TPU_NAME`, where `$TPU_NAME` the name of your TPU in the Cloud Console. From a GCE VM, you can run the following command to train ResNet for one epoch on a v2-8 or v3-8 TPU by setting `TRAIN_EPOCHS` to 1:

```
python3 resnet_ctl_imagenet_main.py \
  --tpu=$TPU_NAME \
  --model_dir=$MODEL_DIR \
  --data_dir=$DATA_DIR \
  --batch_size=1024 \
  --steps_per_loop=500 \
  --train_epochs=$TRAIN_EPOCHS \
  --use_synthetic_data=false \
  --dtype=fp32 \
  --enable_eager=true \
  --enable_tensorboard=true \
  --distribution_strategy=tpu \
  --log_steps=50 \
  --single_l2_loss_op=true \
  --use_tf_function=true
```

To train the ResNet to convergence, run it for 90 epochs by setting `TRAIN_EPOCHS` to 90.

Note: `$MODEL_DIR` and `$DATA_DIR` must be GCS paths.