TensorFlow 2.1 3 . 0 test

1. **Automatic testing:**

Copy 2.script, 3.yaml, and 4.data to the current path.

Execute the following command to give all script files execution permissions

|  |
| --- |
| chmod +x 2.script/\* |

Execute the following command to build the test file.

|  |
| --- |
| ./2.script/build.sh​​ |

Start the test and execute the commands one by one.

## examples

|  |
| --- |
| ./examples.sh |

## models

|  |
| --- |
| ./recommendation.sh  ./vision.sh  ./projects.sh |

## Benchmark

|  |
| --- |
| ./benchmark.sh |

## others

|  |
| --- |
| ./others.sh |

1. **Manual testing:**

Note: The path should be switched to your own path, and cannot be directly copied and pasted.

## examples

This repository has four models in the tensorflow\_examples/models/ path: dcgan, densenet, nmt\_with\_attention, and pix2pix. They are tested separately below.

First clone the repository and set the environment variable `PYTHONPATH` as follows:

Clone the repository

|  |
| --- |
| git clone https://github.com/tensorflow/examples |

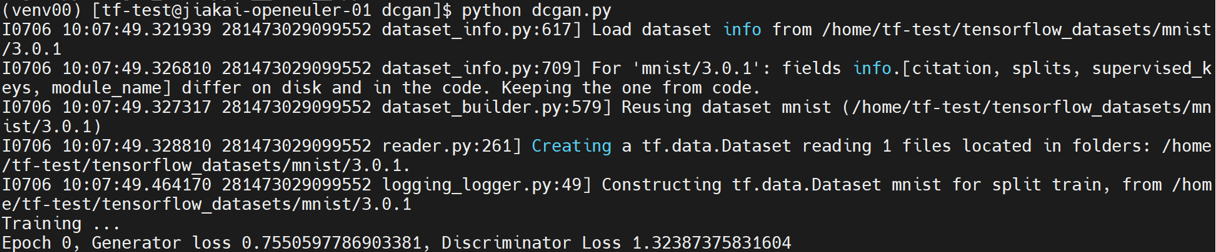
Add the examples repository to the PYTHONPATH environment variable

|  |
| --- |
| export PYTHONPATH="$PYTHONPATH:/home/tf-test/file/examples" |

### dcgan

Here is the training.

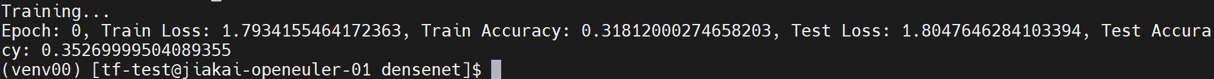
|  |
| --- |
| python dcgan.py |



### densenet

Here is the training.

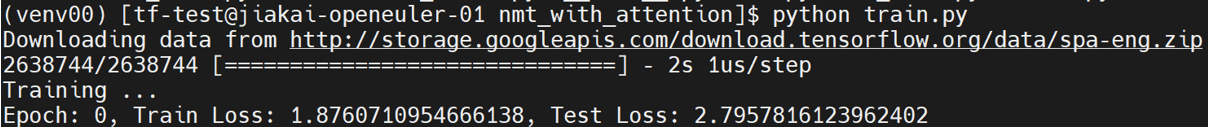
|  |
| --- |
| python train.py |



### nmt\_with\_attention

Here is the training.

|  |
| --- |
| python train.py |

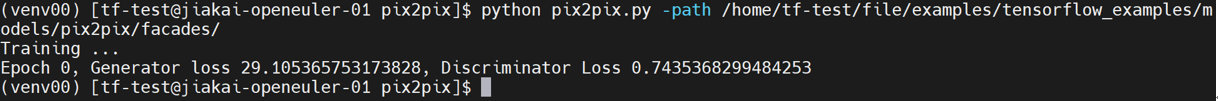


### pix2pix

Here is the training.

Download the dataset [facades] to the path tensorflow\_examples/models/pix2pix/ and unzip facades.tar.

|  |
| --- |
| python pix2pix.py -path /home/tf-test/file/examples/tensorflow\_examples/models/pix2pix/facades/ |



## models

This time, we mainly tested the models in recommendation, nlp, vision, and projects under this repository. First, clone the repository and set the environment variables:

Clone the repository

|  |
| --- |
| git clone https://github.com/tensorflow/models |

Add the models repository to the PYTHONPATH environment variable

|  |
| --- |
| export PYTHONPATH="$PYTHONPATH:/home/tf-test/file/models" |

### recommendation

The models in recommendation are under the path models/official/recommendation.

#### NCF

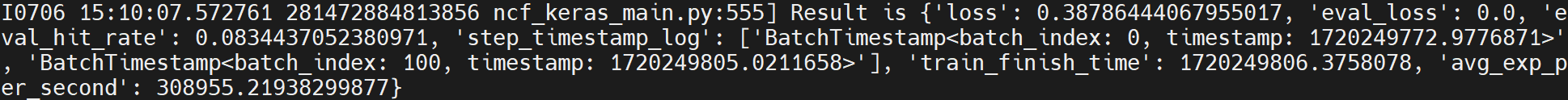
NCF downloads and preprocesses the dataset and trains and evaluates the model as follows. The working path is models/official/recommendation/:

Download and preprocess the dataset, here we choose ml-1m dataset

|  |
| --- |
| python movielens.py --data\_dir /home/tf-test/models/dataset/ncf --dataset ml-1m |

Training and evaluating the model

|  |
| --- |
| python ncf\_keras\_main.py --model\_dir /home/tf-test/models/model\_dir/ncf --data\_dir /home/tf-test/models/dataset/ncf/ --dataset ml-1m --num\_gpus 0 |



#### DCN v2

DCN v2 has the following steps for training with synthetic data:

|  |
| --- |
| python3 models/official/recommendation/ranking/train.py --mode=eval \  --model\_dir=/home/tf-test/models/model\_dir/dcn\_v2 --params\_override="  runtime:  distribution\_strategy: 'one\_device'  task:  use\_synthetic\_data: true  model:  num\_dense\_features: 13  bottom\_mlp: [512,256,2]  embedding\_dim: 2  top\_mlp: [1024,1024,512,256,1]  interaction: 'cross'  vocab\_sizes: [39884406, 39043, 17289, 7420, 20263, 3, 7120, 1543, 63,  38532951, 2953546, 403346, 10, 2208, 11938, 155, 4, 976, 14,  39979771, 25641295, 39664984, 585935, 12972, 108, 36]  trainer:  validation\_steps: 2  " |

1720405253457

#### DLRM

DLRM has the following steps for training with synthetic data:

|  |
| --- |
| python3 models/official/recommendation/ranking/train.py --mode=eval \  --model\_dir=/home/tf-test/models/model\_dir/dlrm --params\_override="  runtime:  distribution\_strategy: 'one\_device'  task:  use\_synthetic\_data: true  model:  num\_dense\_features: 13  bottom\_mlp: [512,256,2]  embedding\_dim: 2  top\_mlp: [1024,1024,512,256,1]  interaction: 'dot'  vocab\_sizes: [39884406, 39043, 17289, 7420, 20263, 3, 7120, 1543, 63,  38532951, 2953546, 403346, 10, 2208, 11938, 155, 4, 976, 14,  39979771, 25641295, 39664984, 585935, 12972, 108, 36]  trainer:  validation\_steps: 2  " |

1720405316275

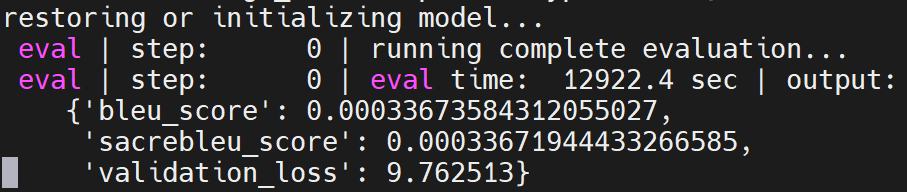
### nlp

Models in nlp are run under the path models/official/nlp/.

#### Transformer

Transformer is as follows:

|  |
| --- |
| python train.py --experiment wmt\_transformer/large --mode eval --model\_dir /home/tf-test/models/model\_dir/transformer --params\_override task.sentencepiece\_model\_path='gs://tf\_model\_garden/nlp/transformer\_wmt/ende\_bpe\_32k.model' |



### vision

Models in vision are run under the path models/official/vision/.

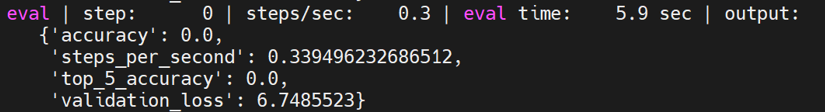
Before running the models, you need to download and preprocess the ILSVRC2012 and coco2017 datasets, and finally rename the ILSVRC2012 dataset folder containing `train\*` and `val\*` to `imagenet-2012-tfrecord` and put it in the vision directory; rename the coco2017 dataset folder containing `val\*` and `instances\_val2017.json` to coco and put it in the vision directory.

Then run the following model.

#### vit\_imagenet\_finetune

The specific test method is as follows:

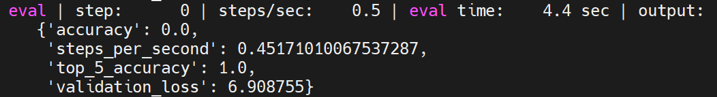
|  |
| --- |
| python train.py -experiment vit\_imagenet\_finetune -mode eval -model\_dir /home/tf-test/models/model\_dir/vit\_imagenet\_finetune --params\_override="  runtime:  distribution\_strategy: 'one\_device'  task:  validation\_data:  global\_batch\_size: 2  trainer:  validation\_steps: 2  validation\_interval: 2  " |



#### vit\_imagenet\_pretrain

The specific test method is as follows:

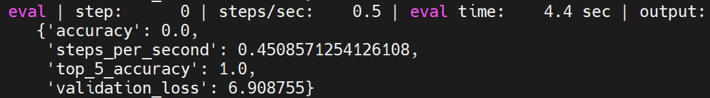
|  |
| --- |
| python train.py -experiment vit\_imagenet\_pretrain -mode eval -model\_dir /home/tf-test/models/model\_dir/vit\_imagenet\_pretrain --params\_override="  runtime:  distribution\_strategy: 'one\_device'  task:  validation\_data:  global\_batch\_size: 2  trainer:  validation\_steps: 2  validation\_interval: 2  " |



#### deit\_imagenet\_pretrain

The specific test method is as follows:

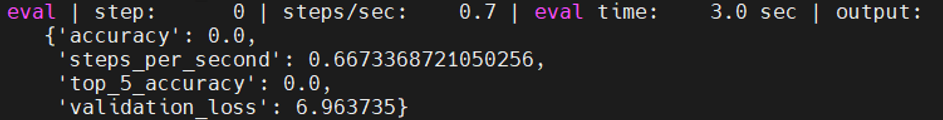
|  |
| --- |
| python train.py -experiment deit\_imagenet\_pretrain -mode eval -model\_dir /home/tf-test/models/model\_dir/deit\_imagenet\_pretrain --params\_override="  runtime:  distribution\_strategy: 'one\_device'  task:  validation\_data:  global\_batch\_size: 2  trainer:  validation\_steps: 2  validation\_interval: 2  " |



#### mobilenet\_imagenet

The specific test method is as follows:

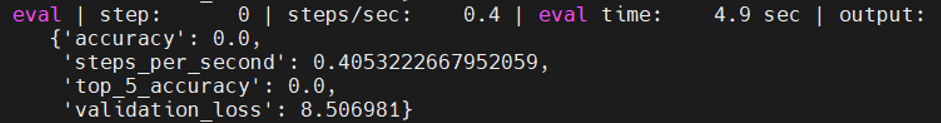
|  |
| --- |
| python train.py -experiment mobilenet\_imagenet -mode eval -model\_dir /home/tf-test/models/model\_dir/mobilenet\_imagenet --params\_override="  runtime:  distribution\_strategy: 'one\_device'  task:  validation\_data:  global\_batch\_size: 2  trainer:  validation\_steps: 2  validation\_interval: 2  " |



#### revnet\_imagenet

The specific test method is as follows:

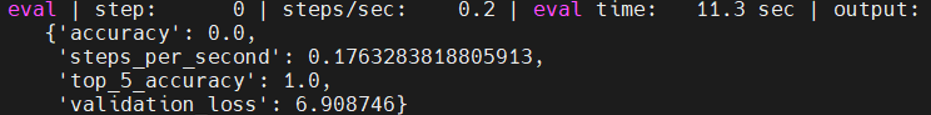
|  |
| --- |
| python train.py -experiment revnet\_imagenet -mode eval -model\_dir /home/tf-test/models/model\_dir/revnet\_imagenet --params\_override="  runtime:  distribution\_strategy: 'one\_device'  task:  validation\_data:  global\_batch\_size: 2  trainer:  validation\_steps: 2  validation\_interval: 2  " |



#### resnet\_rs\_imagenet

The specific test method is as follows:

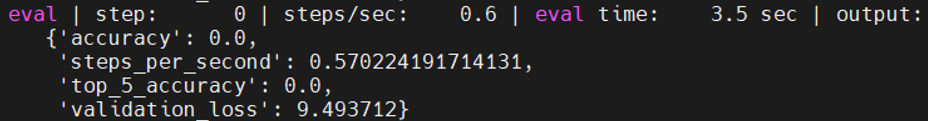
|  |
| --- |
| python train.py -experiment resnet\_rs\_imagenet -mode eval -model\_dir /home/tf-test/models/model\_dir/resnet\_rs\_imagenet --params\_override="  runtime:  distribution\_strategy: 'one\_device'  task:  validation\_data:  global\_batch\_size: 2  trainer:  validation\_steps: 2  validation\_interval: 2  " |



#### resnet\_imagenet

The specific test method is as follows:

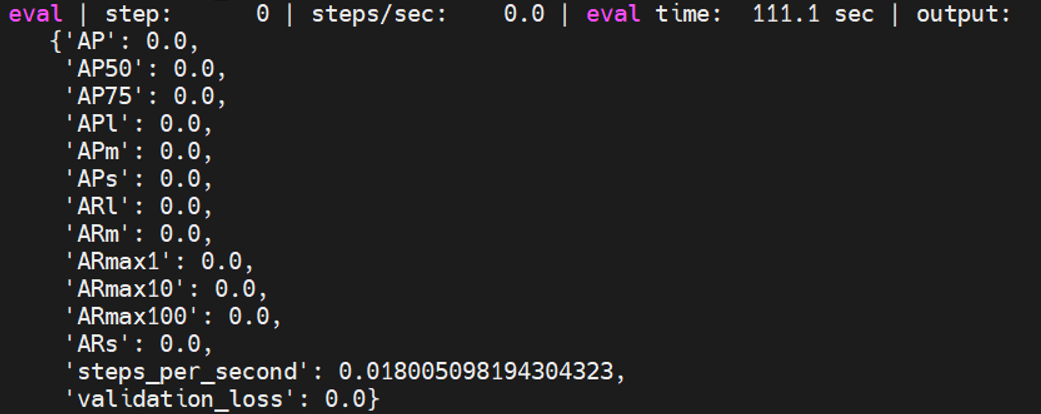
|  |
| --- |
| python train.py -experiment resnet\_imagenet -mode eval -model\_dir /home/tf-test/models/model\_dir/resnet\_imagenet --params\_override="  runtime:  distribution\_strategy: 'one\_device'  task:  validation\_data:  global\_batch\_size: 2  trainer:  validation\_steps: 2  validation\_interval: 2  " |



#### fasterrcnn\_resnetfpn\_coco

The specific test method is as follows:

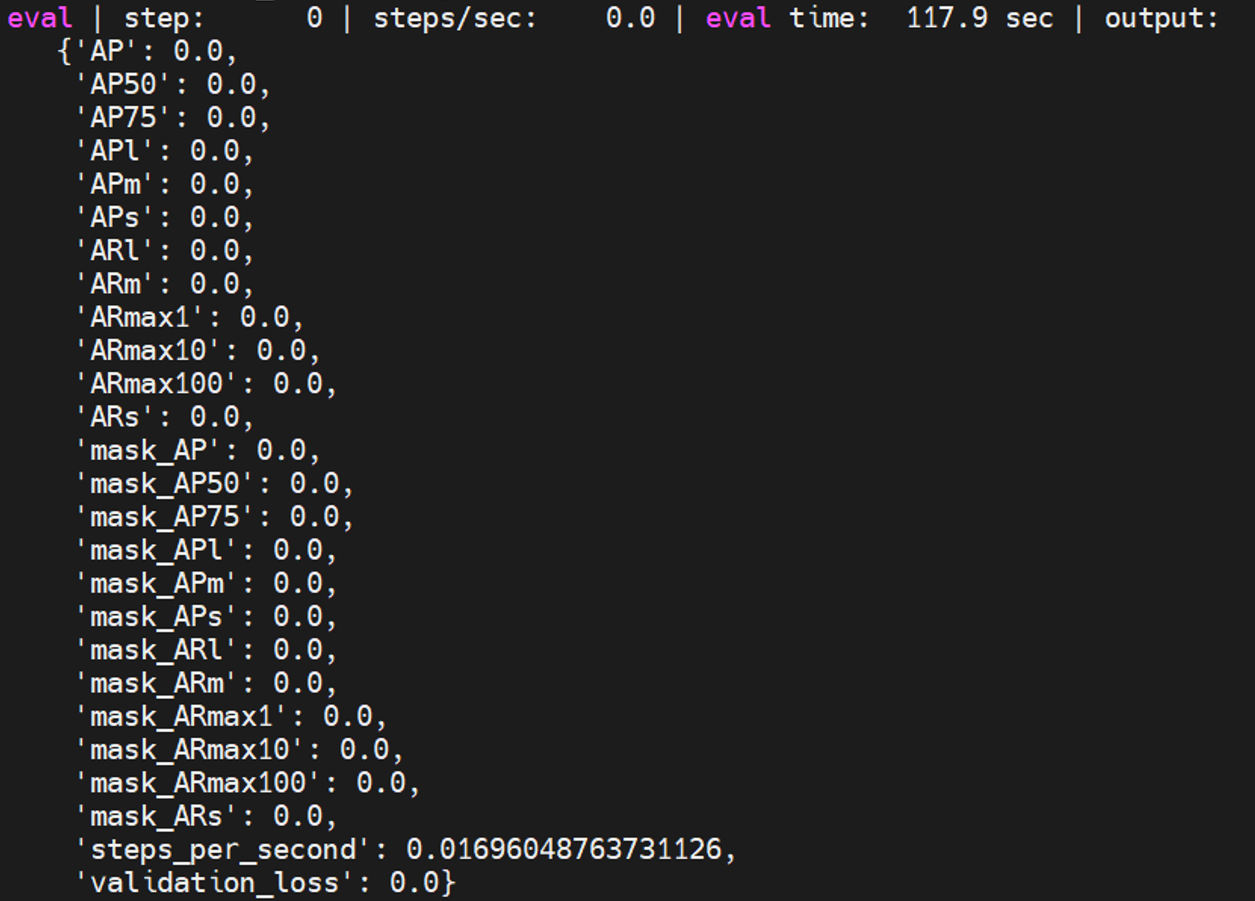
|  |
| --- |
| python train.py -experiment fasterrcnn\_resnetfpn\_coco -mode eval -model\_dir /home/tf-test/models/model\_dir/fasterrcnn\_resnetfpn\_coco --params\_override="  runtime:  distribution\_strategy: 'one\_device'  task:  validation\_data:  global\_batch\_size: 2  trainer:  validation\_steps: 2  validation\_interval: 2  " |



#### maskrcnn\_resnetfpn\_coco

The specific test method is as follows:

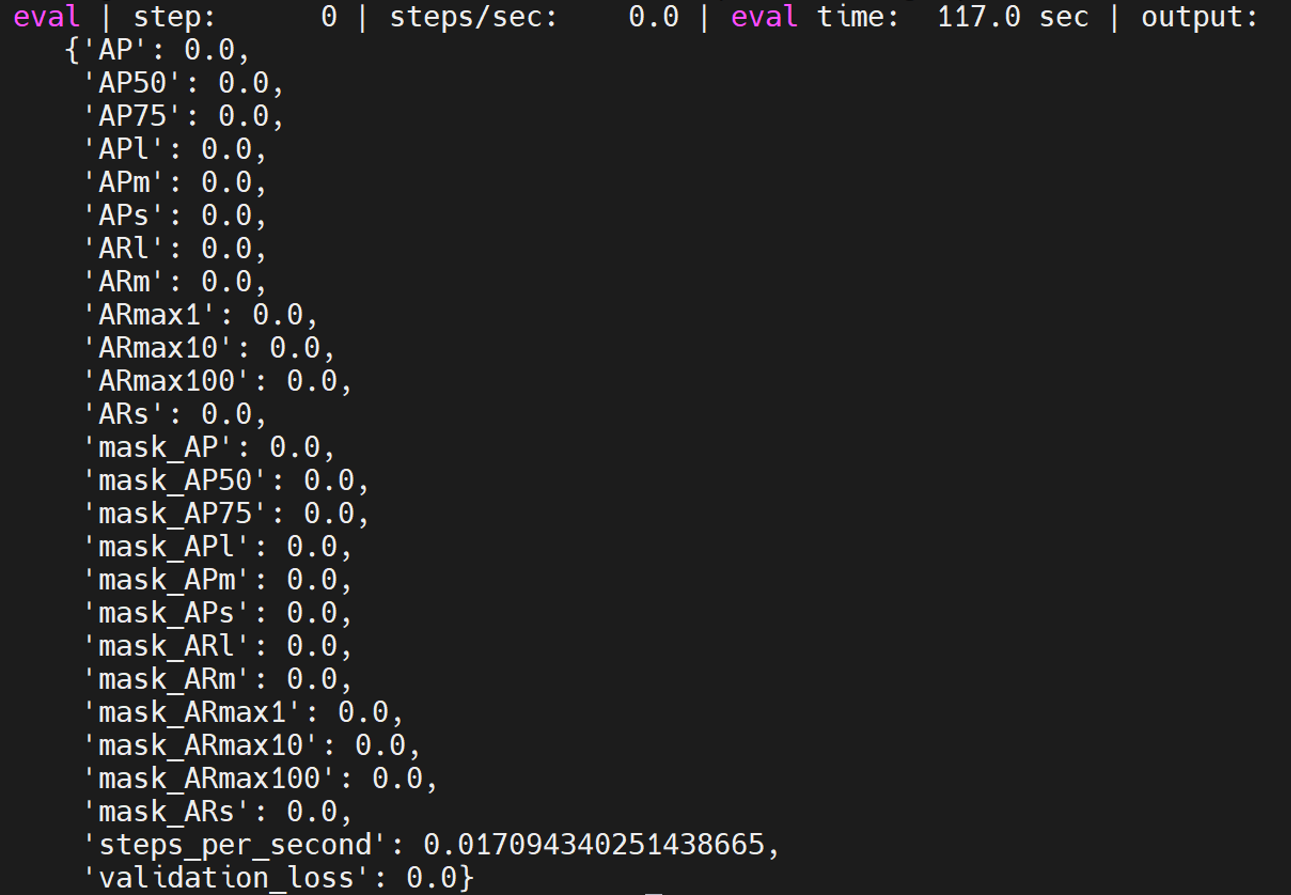
|  |
| --- |
| python train.py -experiment maskrcnn\_resnetfpn\_coco -mode eval -model\_dir /home/tf-test/models/model\_dir/maskrcnn\_resnetfpn\_coco --params\_override="  runtime:  distribution\_strategy: 'one\_device'  task:  validation\_data:  global\_batch\_size: 2  trainer:  validation\_steps: 2  validation\_interval: 2  " |



#### maskrcnn\_spinenet\_coco

The specific test method is as follows:

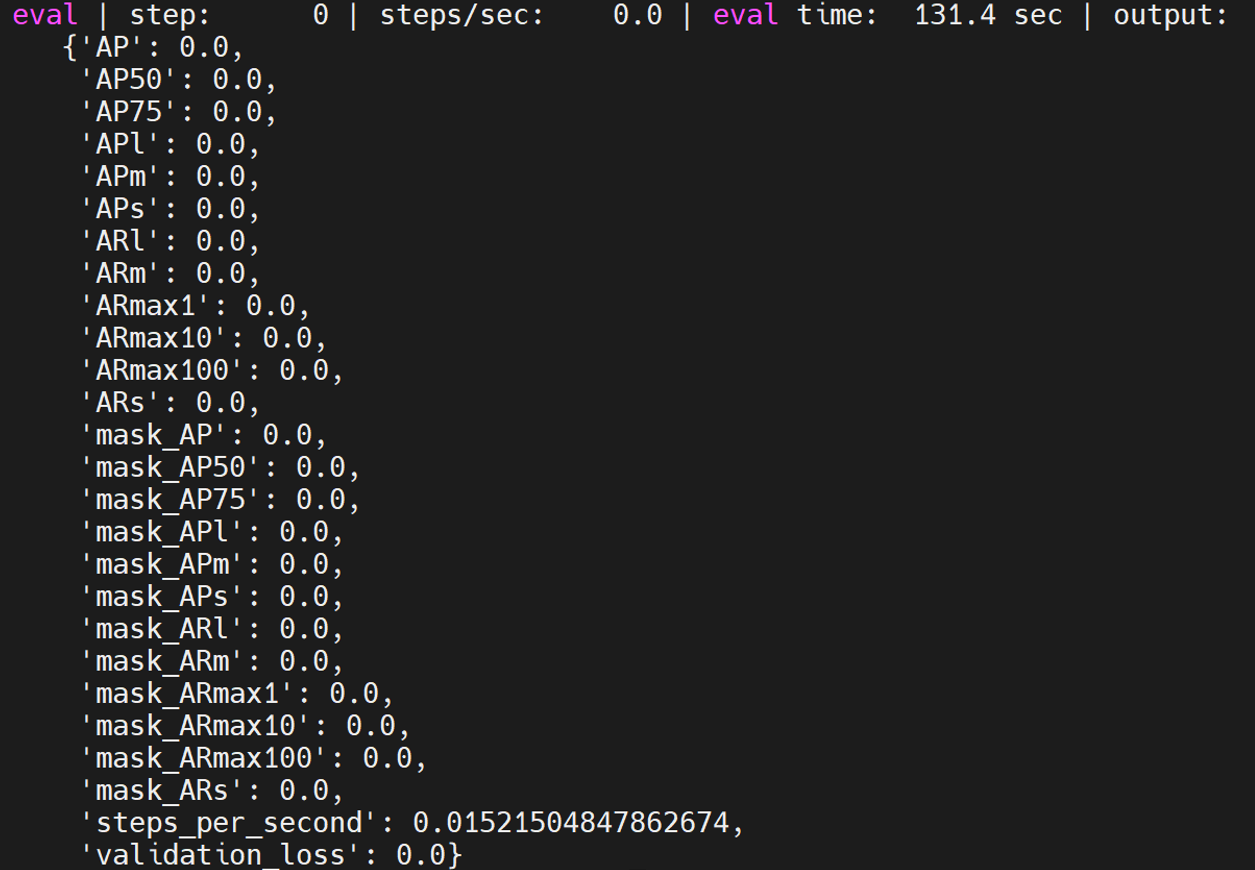
|  |
| --- |
| python train.py -experiment maskrcnn\_spinenet\_coco -mode eval -model\_dir /home/tf-test/models/model\_dir/maskrcnn\_spinenet\_coco --params\_override="  runtime:  distribution\_strategy: 'one\_device'  task:  validation\_data:  global\_batch\_size: 2  trainer:  validation\_steps: 2  validation\_interval: 2  " |



#### cascadercnn\_spinenet\_coco

The specific test method is as follows:

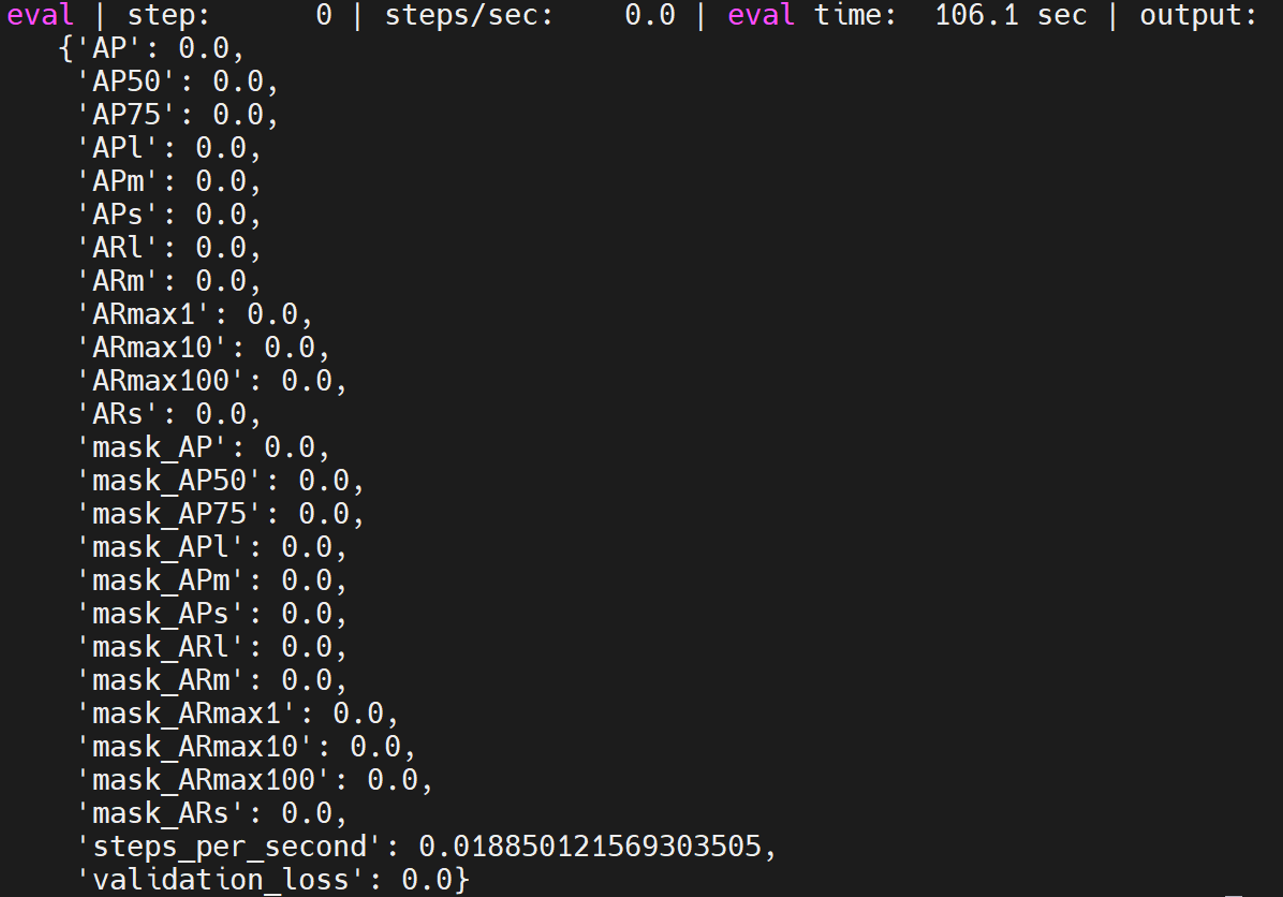
|  |
| --- |
| python train.py -experiment cascadercnn\_spinenet\_coco -mode eval -model\_dir /home/tf-test/models/model\_dir/cascadercnn\_spinenet\_coco --params\_override="  runtime:  distribution\_strategy: 'one\_device'  task:  validation\_data:  global\_batch\_size: 2  trainer:  validation\_steps: 2  validation\_interval: 2  " |



#### maskrcnn\_mobilenet\_coco

The specific test method is as follows:

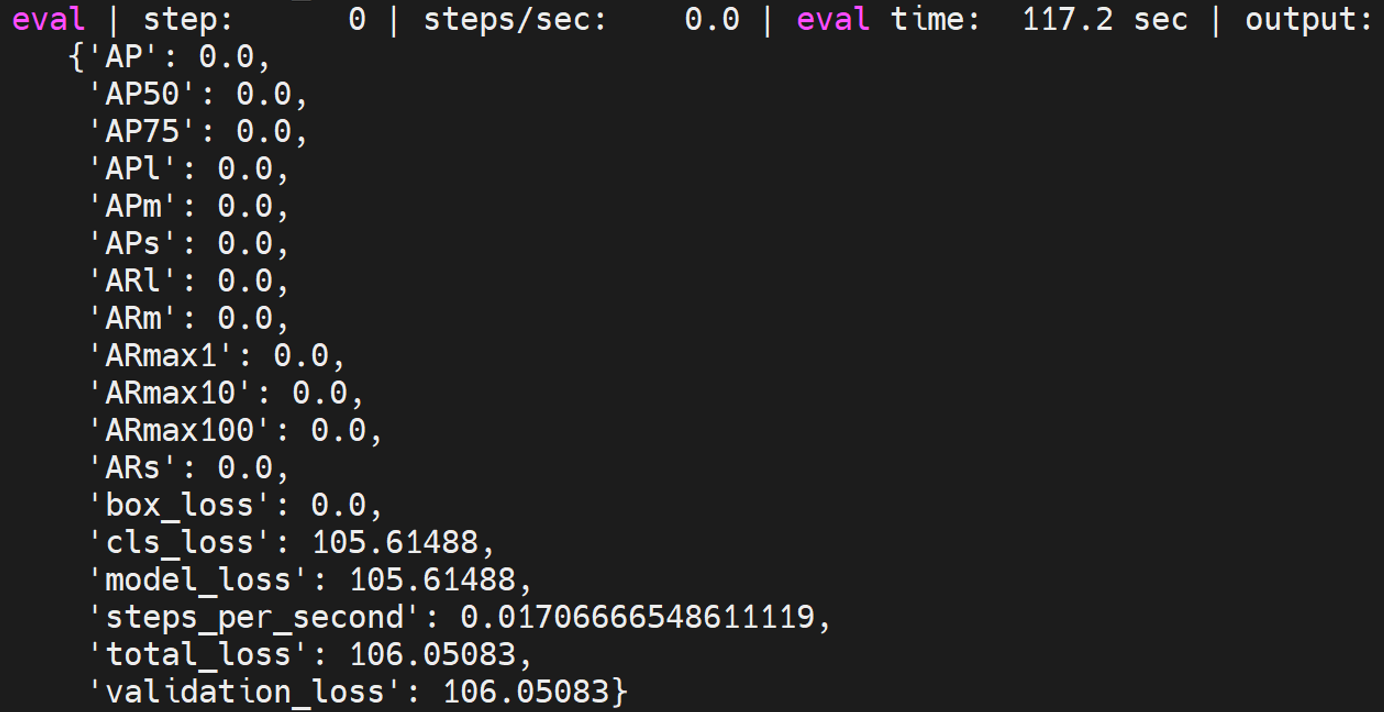
|  |
| --- |
| python train.py -experiment maskrcnn\_mobilenet\_coco -mode eval -model\_dir /home/tf-test/models/model\_dir/maskrcnn\_mobilenet\_coco --params\_override="  runtime:  distribution\_strategy: 'one\_device'  task:  validation\_data:  global\_batch\_size: 2  trainer:  validation\_steps: 2  validation\_interval: 2  " |



#### retinanet\_resnetfpn\_coco

The specific test method is as follows:

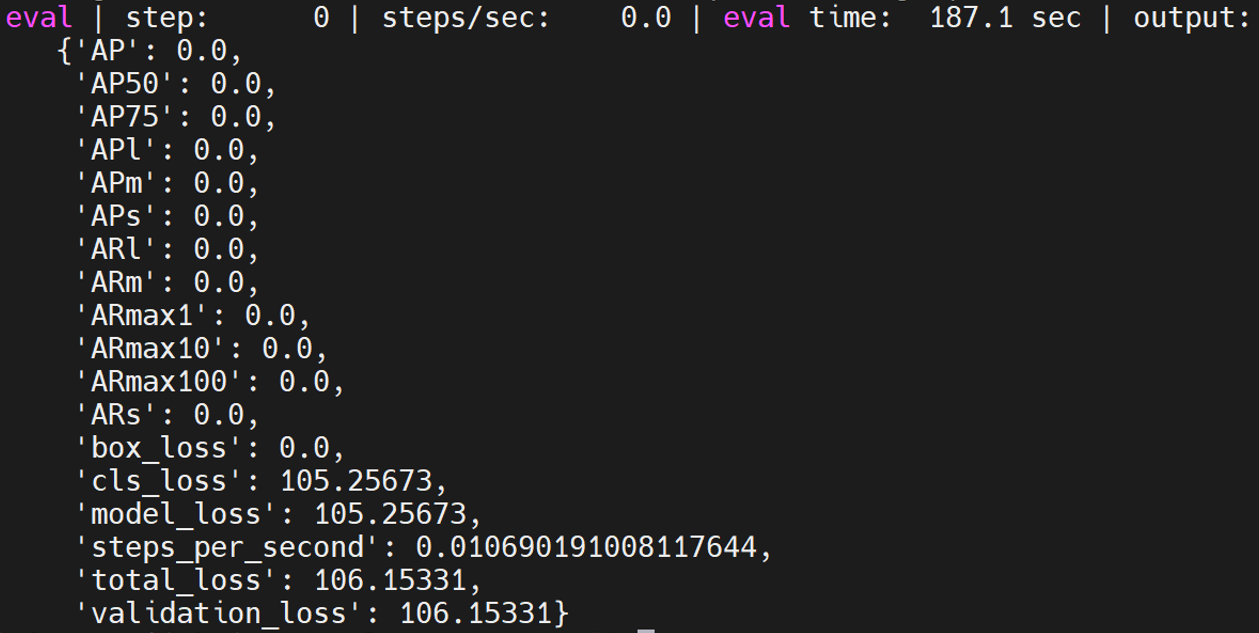
|  |
| --- |
| python train.py -experiment retinanet\_resnetfpn\_coco -mode eval -model\_dir /home/tf-test/models/model\_dir/retinanet\_resnetfpn\_coco --params\_override="  runtime:  distribution\_strategy: 'one\_device'  task:  validation\_data:  global\_batch\_size: 2  trainer:  validation\_steps: 2  validation\_interval: 2  " |



#### retinanet\_spinenet\_coco

The specific test method is as follows:

|  |
| --- |
| python train.py -experiment retinanet\_spinenet\_coco -mode eval -model\_dir /home/tf-test/models/model\_dir/retinanet\_spinenet\_coco --params\_override="  runtime:  distribution\_strategy: 'one\_device'  task:  validation\_data:  global\_batch\_size: 2  trainer:  validation\_steps: 2  validation\_interval: 2  " |



#### retinanet\_mobile\_coco

The specific test method is as follows:

|  |
| --- |
| python train.py -experiment retinanet\_mobile\_coco -mode eval -model\_dir /home/tf-test/models/model\_dir/retinanet\_mobile\_coco --params\_override="  runtime:  distribution\_strategy: 'one\_device'  task:  validation\_data:  global\_batch\_size: 2  trainer:  validation\_steps: 2  validation\_interval: 2  " |



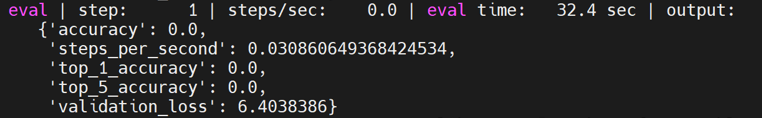
### Projects

The models in projects are in the path models/official/projects/.

#### \*\*assemblenet\*\*

The specific eval is as follows:

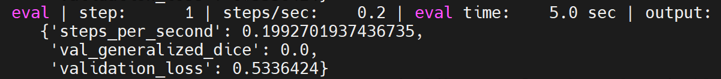
|  |
| --- |
| python train\_test.py |



#### \*\*volumetric\_models\*\*

The specific eval is as follows:

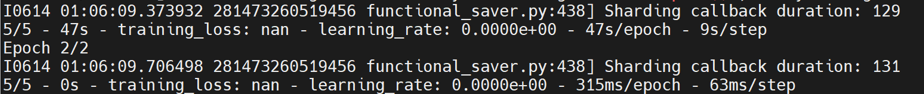
|  |
| --- |
| python train\_test.py |



#### \*\*nhnet\*\*

The specific eval is as follows:

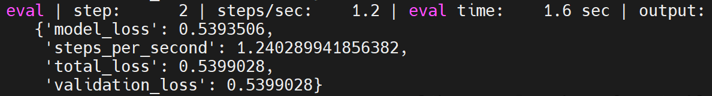
|  |
| --- |
| python trainer\_test.py |



#### \*\*yt8m\*\*

The specific eval is as follows:

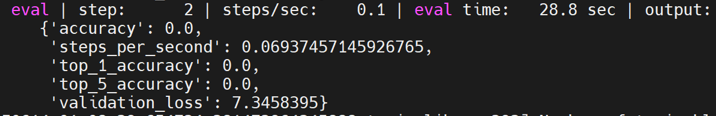
|  |
| --- |
| python train\_test.py |



#### \*\*Movient\*\*

The specific eval is as follows:

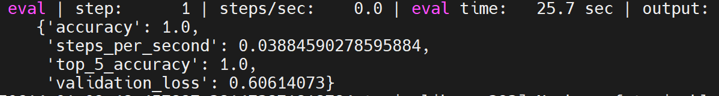
|  |
| --- |
| python train\_test.py |



#### \*\*maxvit\*\*

The specific eval is as follows:

|  |
| --- |
| python train\_test.py |



## benchmark

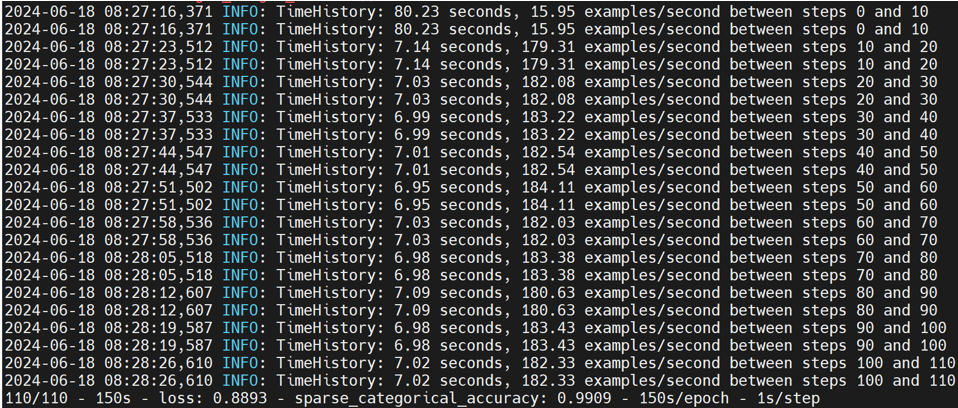
The benchmark model needs to clone the benchmark repository and switch the cloned models repository to the v2.15.0 tags with the benchmark module, as follows:

|  |
| --- |
| git clone https://github.com/tensorflow/benchmarks  cd models  git checkout v2.15.0 |

### resnet56

resnet56 is in the models/official/benchmark/keras\_cifar\_benchmark.py file. Here, synthetic data is used for testing. It comes with benchmark\_cpu, so it can be tested directly, as follows:

|  |
| --- |
| python3 benchmarks/perfzero/lib/benchmark.py --git\_repos="https://github.com/tensorflow/models.git;benchmark" --gcloud\_key\_file\_url="" --benchmark\_methods=official.benchmark.keras\_cifar\_benchmark.Resnet56KerasBenchmarkSynth.benchmark\_cpu |

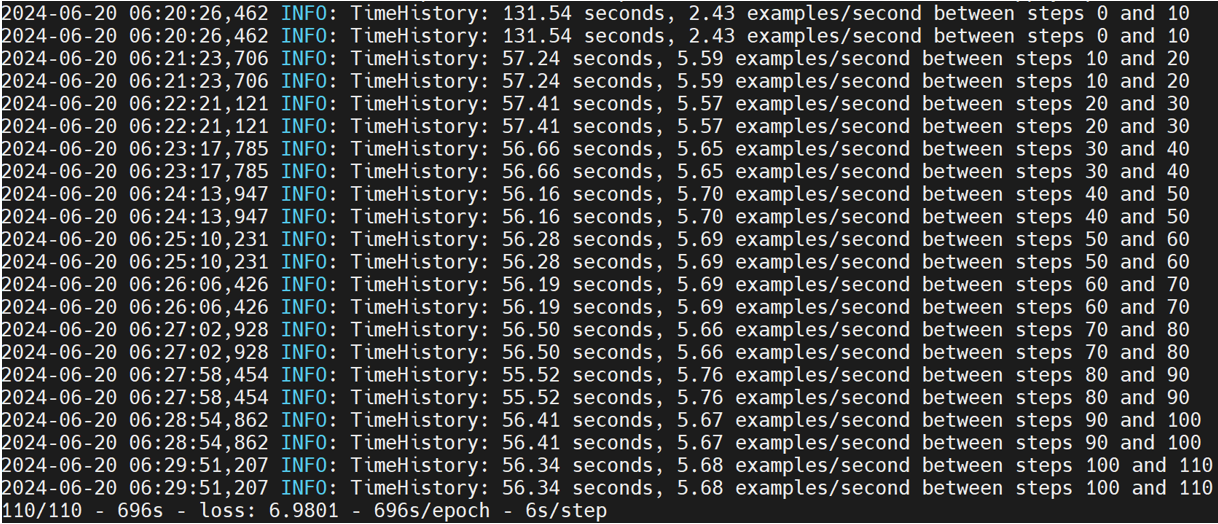


### resnet50

resnet50 is in the models/official/benchmark/keras\_imagenet\_benchmark.py file. Here, synthetic data is used for testing. It comes with benchmark\_1\_gpu. When the program is running, it will detect whether there is an available gpu. If not, it will use the cpu for testing, so you can test it directly, as follows:

Note: You can reduce the batch\_size to shorten the test time by modifying the per\_replica\_batch\_size value of the benchmark\_1\_gpu method in the KerasClassifierBenchmarkBase class in the keras\_imagenet\_benchmark.py file.

|  |
| --- |
| python3 benchmarks/perfzero/lib/benchmark.py --git\_repos="https://github.com/tensorflow/models.git;benchmark" --gcloud\_key\_file\_url="" --benchmark\_methods=official.benchmark.keras\_imagenet\_benchmark.Resnet50KerasBenchmarkSynth.benchmark\_1\_gpu |



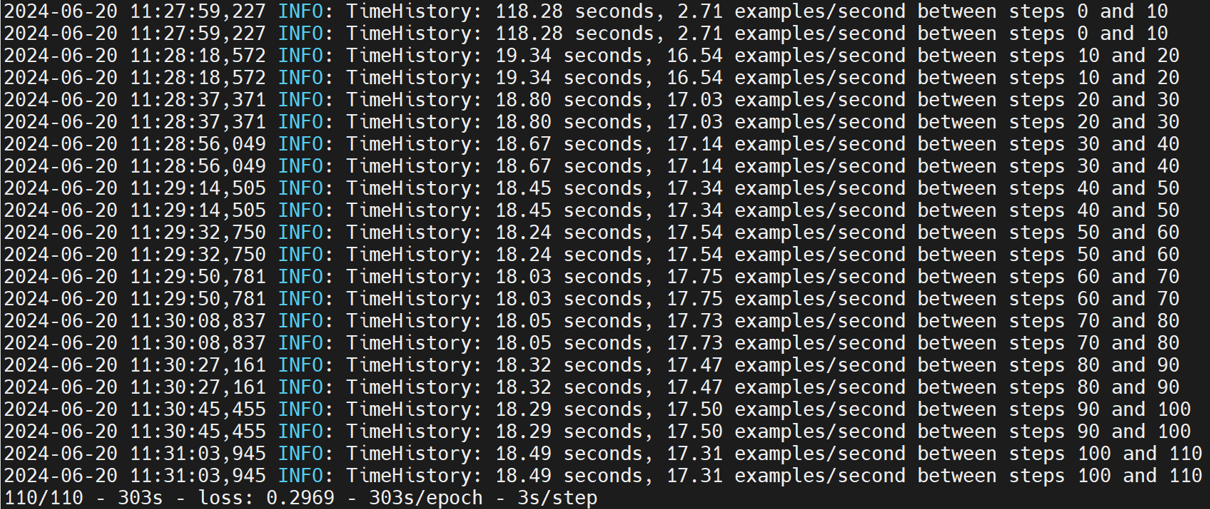
### MobileNetV1

Mobilenetv1 is in the models/official/benchmark/keras\_imagenet\_benchmark.py file. For the convenience of testing, synthetic data is used here, so you need to write the corresponding KerasPruningBenchmarkSynthBase and MobilenetV1KerasPruningBenchmarkSynth classes yourself, as follows:

|  |
| --- |
| class KerasPruningBenchmarkSynthBase(Resnet50KerasBenchmarkBase):  """Pruning method benchmarks."""  def \_\_init\_\_(self, root\_data\_dir=None, default\_flags=None, \*\*kwargs):  if default\_flags is None:  default\_flags = {}  default\_flags.update({  'skip\_eval': True,  'report\_accuracy\_metrics': False,  'use\_synthetic\_data': True,  'train\_steps': 110,  'log\_steps': 10,  'pruning\_method': 'polynomial\_decay',  'pruning\_begin\_step': 0,  'pruning\_end\_step': 50000,  'pruning\_initial\_sparsity': 0,  'pruning\_final\_sparsity': 0.5,  'pruning\_frequency': 100,  })  super(KerasPruningBenchmarkSynthBase, self).\_\_init\_\_(  default\_flags=default\_flags, \*\*kwargs)  class MobilenetV1KerasPruningBenchmarkSynth(KerasPruningBenchmarkSynthBase):  """Pruning method benchmarks for MobilenetV1."""  def \_\_init\_\_(self, \*\*kwargs):  default\_flags = {  'model': 'mobilenet',  'optimizer': 'mobilenet\_default',  }  super(MobilenetV1KerasPruningBenchmarkSynth, self).\_\_init\_\_(  default\_flags=default\_flags, \*\*kwargs) |

You can also adjust batch\_size to shorten the test time.

|  |
| --- |
| python3 benchmarks/perfzero/lib/benchmark.py --git\_repos="https://github.com/tensorflow/models.git;benchmark" --gcloud\_key\_file\_url="" --benchmark\_methods=official.benchmark.keras\_imagenet\_benchmark.MobilenetV1KerasPruningBenchmarkSynth.benchmark\_1\_gpu |

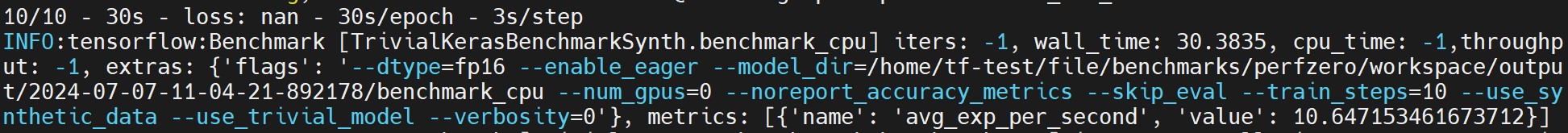


### trivial

trivial is in the models/official/benchmark/keras\_imagenet\_benchmark.py file. In order to facilitate testing, synthetic data is used here, so you need to write the corresponding TrivialKerasBenchmarkSynth class yourself, as follows:

|  |
| --- |
| class TrivialKerasBenchmarkSynth(keras\_benchmark.KerasBenchmark):  """Trivial model with synth data benchmark tests."""  def \_\_init\_\_(self, output\_dir=None, root\_data\_dir=None, \*\*kwargs):  flag\_methods = [resnet\_imagenet\_main.define\_imagenet\_keras\_flags]  def\_flags = {}  def\_flags['use\_trivial\_model'] = True  def\_flags['skip\_eval'] = True  def\_flags['report\_accuracy\_metrics'] = False  def\_flags['dtype'] = 'fp16'  def\_flags['use\_synthetic\_data'] = True  def\_flags['train\_steps'] = 600  def\_flags['log\_steps'] = 100  def\_flags['distribution\_strategy'] = 'mirrored'  super(TrivialKerasBenchmarkSynth, self).\_\_init\_\_(  output\_dir=output\_dir,  flag\_methods=flag\_methods,  default\_flags=def\_flags)  @benchmark\_wrappers.enable\_runtime\_flags  def \_run\_and\_report\_benchmark(self):  start\_time\_sec = time.time()  stats = resnet\_imagenet\_main.run(FLAGS)  wall\_time\_sec = time.time() - start\_time\_sec  super(TrivialKerasBenchmarkSynth, self).\_report\_benchmark(  stats,  wall\_time\_sec,  total\_batch\_size=FLAGS.batch\_size,  log\_steps=FLAGS.log\_steps)  def benchmark\_cpu(self):  self.\_setup()  FLAGS.num\_gpus = 0  FLAGS.enable\_eager = True  FLAGS.model\_dir = self.\_get\_model\_dir('benchmark\_cpu')  FLAGS.batch\_size = 32  FLAGS.train\_steps = 10  self.\_run\_and\_report\_benchmark()  def fill\_report\_object(self, stats):  super(TrivialKerasBenchmarkSynth, self).fill\_report\_object(  stats,  total\_batch\_size=FLAGS.batch\_size,  log\_steps=FLAGS.log\_steps) |

|  |
| --- |
| python3 benchmarks/perfzero/lib/benchmark.py --git\_repos="https://github.com/tensorflow/models.git;benchmark" --gcloud\_key\_file\_url="" --benchmark\_methods=official.benchmark.keras\_imagenet\_benchmark.TrivialKerasBenchmarkSynth.benchmark\_cpu |

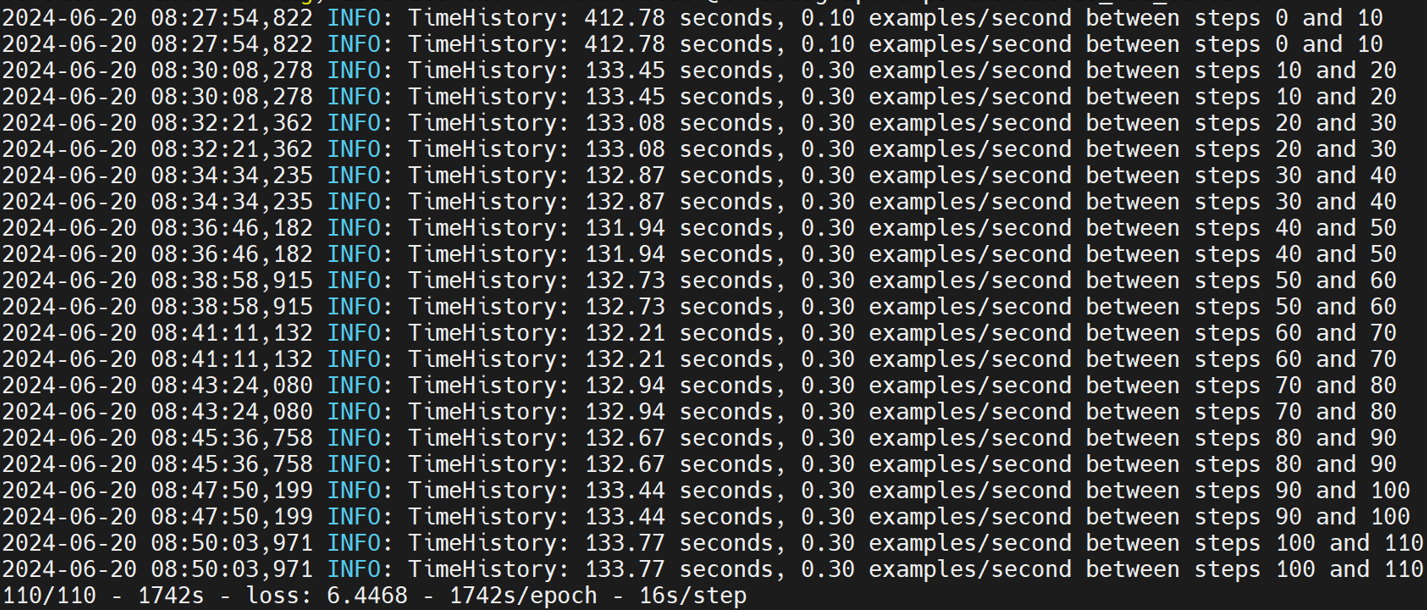


### efficientnet

efficientnet is in the models/official/benchmark/keras\_imagenet\_benchmark.py file. In order to facilitate testing, synthetic data is used here, so you need to write the corresponding EfficientNetKerasBenchmarkSynth class yourself, as follows:

|  |
| --- |
| class EfficientNetKerasBenchmarkSynth(KerasClassifierBenchmarkBase):  """EfficientNet synth data benchmark tests."""  def \_\_init\_\_(self, output\_dir=None, root\_data\_dir=None, tpu=None, \*\*kwargs):  def\_flags = {}  def\_flags['log\_steps'] = 10  super(EfficientNetKerasBenchmarkSynth, self).\_\_init\_\_(  model='efficientnet', output\_dir=output\_dir, default\_flags=def\_flags,  tpu=tpu, dataset\_builder='synthetic', train\_epochs=1, train\_steps=110)  def benchmark\_cpu(self):  self.\_setup()  self.\_run\_and\_report\_benchmark(  experiment\_name='benchmark\_cpu',  model\_variant='efficientnet-b7',  dtype = 'bfloat16',  num\_tpus=0,  distribution\_strategy='one\_device',  per\_replica\_batch\_size=4) |

|  |
| --- |
| python3 benchmarks/perfzero/lib/benchmark.py --git\_repos="https://github.com/tensorflow/models.git;benchmark" --gcloud\_key\_file\_url="" --benchmark\_methods=official.benchmark.keras\_imagenet\_benchmark.EfficientNetKerasBenchmarkSynth.benchmark\_cpu |

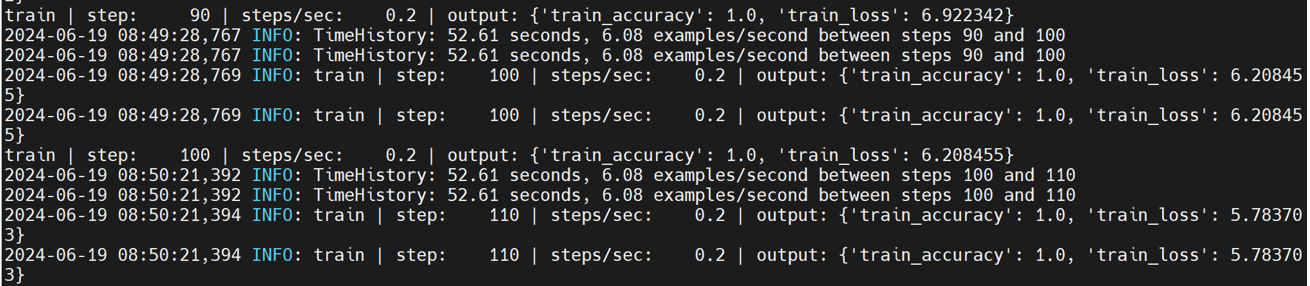


### resnet50ctl

resnet50ctl is in the models/official/benchmark/resnet\_ctl\_imagenet\_benchmark.py file. Here, synthetic data is used for testing. It comes with benchmark\_1\_gpu. When the program is running, it will detect whether there is an available gpu. If not, it will use the cpu for testing, so you can test it directly, as follows:

In addition, you need to change the batch\_size of the benchmark\_1\_gpu method in the Resnet50CtlBenchmarkBase class to 32, otherwise there will be a problem of insufficient RAM.

|  |
| --- |
| python3 benchmarks/perfzero/lib/benchmark.py --git\_repos="https://github.com/tensorflow/models.git;benchmark" --gcloud\_key\_file\_url="" --benchmark\_methods=official.benchmark.resnet\_ctl\_imagenet\_benchmark.Resnet50CtlBenchmarkSynth.benchmark\_1\_gpu |

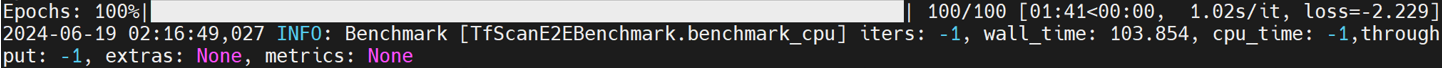


### TfScanE2E

A benchmark to test and evaluate the overall end-to-end performance of TensorFlow models.

TFScanE2E has its own benchmark\_cpu in the models/official/benchamrk/tf\_scan\_benchmark.py file, as follows:

|  |
| --- |
| python3 benchmarks/perfzero/lib/benchmark.py --git\_repos="https://github.com/tensorflow/models.git;benchmark" --gcloud\_key\_file\_url="" --benchmark\_methods=official.benchmark.tf\_scan\_benchmark.TfScanE2EBenchmark.benchmark\_cpu |



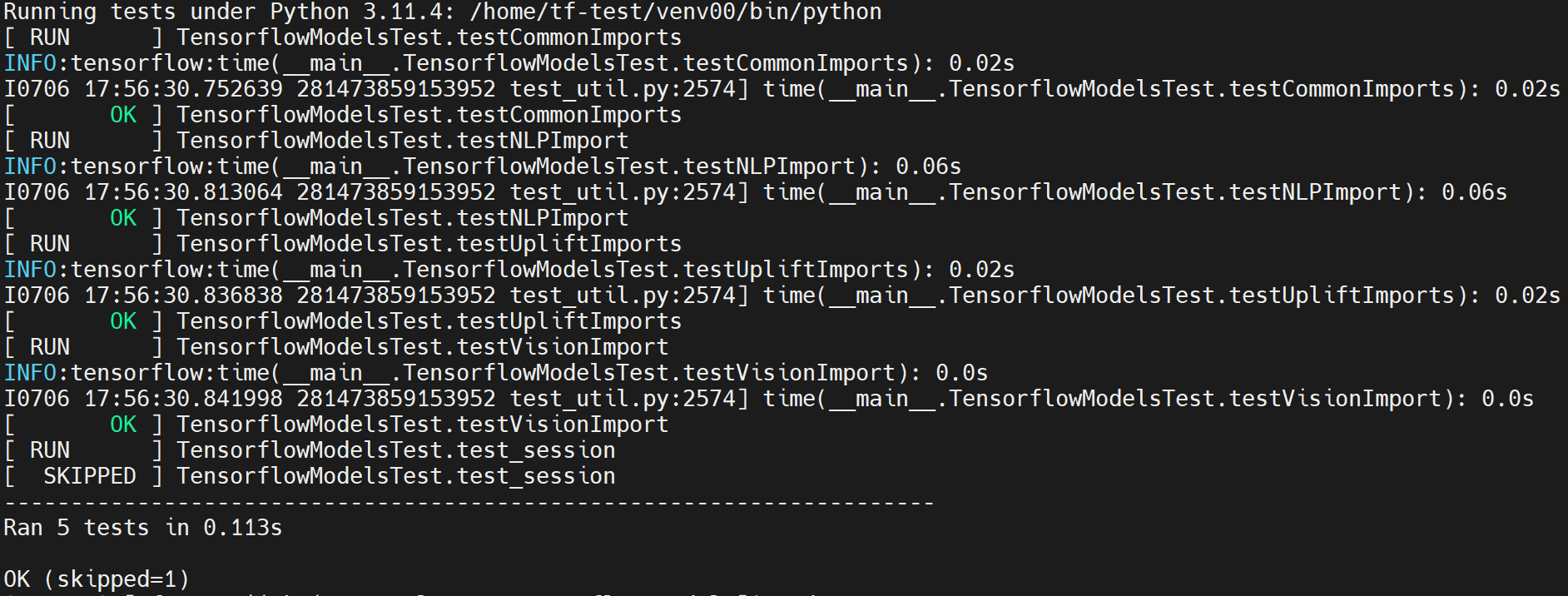
## others

Here are some small tests.

### \*\*tensorflow\_models\*\*

Working path: /home/tf-test/file/models/tensorflow\_models

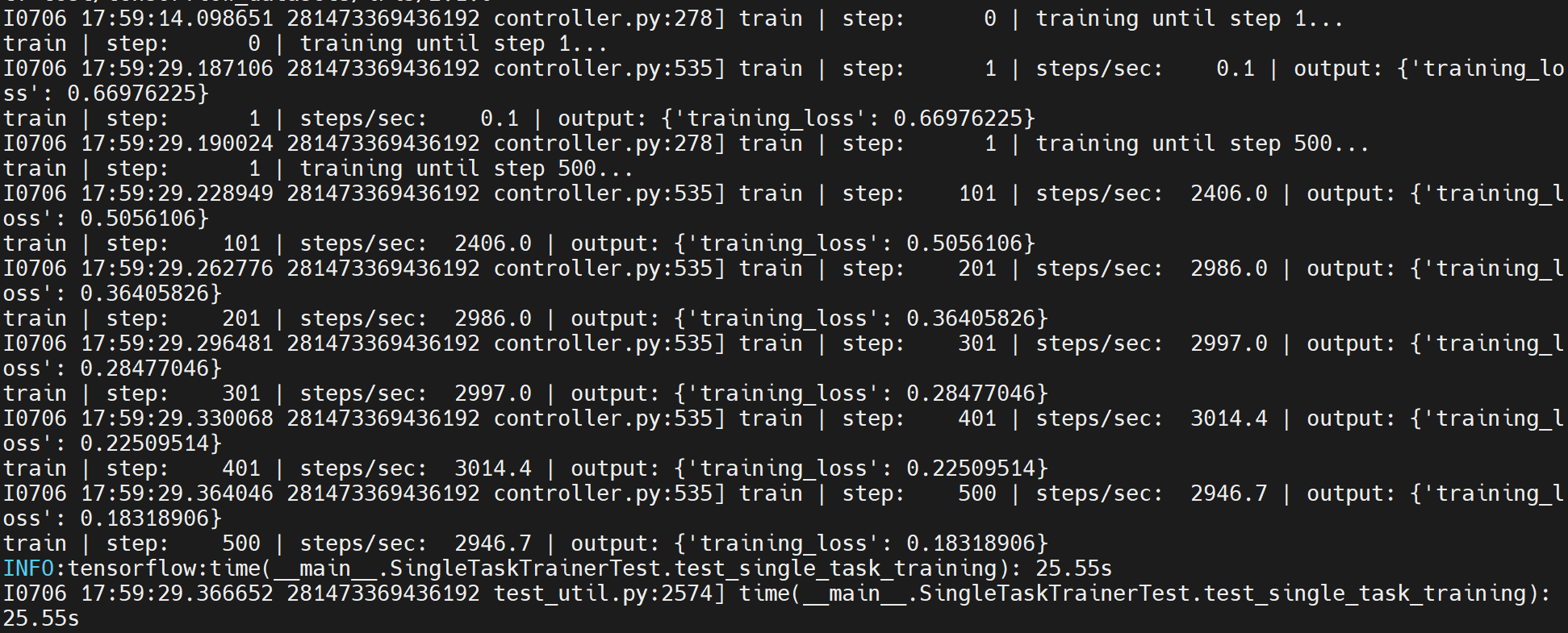
|  |
| --- |
| python tensorflow\_models\_test.py |



### ORBIT

Working path: /home/tf-test/file/models/orbit/examples/single\_task

|  |
| --- |
| python single\_task\_trainer\_test.py |



|  |
| --- |
| python single\_task\_evaluator\_test.py |

