

CANN
V100R020C10

benchmark 工具用户指南

文档版本 01
发布日期 2020-10-13



版权所有 © 华为技术有限公司 2020。保留一切权利。

非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。

商标声明



HUAWEI和其他华为商标均为华为技术有限公司的商标。

本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为公司对本文档内容不做任何明示或默示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

华为技术有限公司

地址： 深圳市龙岗区坂田华为总部办公楼 邮编： 518129

网址： <https://www.huawei.com>

客户服务邮箱： support@huawei.com

客户服务电话： 4008302118

目录

1 前言	1
2 推理 benchmark 工具	2
2.1 工具介绍	2
2.1.1 简介	2
2.1.2 工具目录说明	3
2.2 获取工具包	4
2.3 运行工具	5
2.3.1 纯推理场景	5
2.3.2 推理场景	6
2.4 使用实例	10
2.4.1 纯推理场景使用实例	10
2.4.2 推理场景使用实例	11
2.5 新模型适配 benchmark 工具方法	19
2.5.1 模型适配说明	19
2.5.2 模型适配总体流程	19
2.5.3 模型适配过程	20
3 训练 benchmark 工具	24
3.1 工具介绍	24
3.1.1 简介	24
3.1.2 工具目录说明	25
3.1.3 工具脚本说明	27
3.1.4 打点日志介绍	28
3.2 获取工具包	28
3.3 运行工具	29
3.4 使用场景	31
3.4.1 Host 单机训练场景	31
3.4.2 Docker 单机训练场景	37
3.4.3 Host 集群训练场景（TensorFlow）	40
3.4.3.1 部署前准备	40
3.4.3.2 安装 OpenMPI 开源库	40
3.4.3.3 配置服务器 SSH 免密登录	41
3.4.3.4 测试运行环境	42

3.4.3.5 运行 benchmark 工具.....	44
3.4.4 Docker 集群训练场景（ TensorFlow ）	45
3.4.4.1 部署前准备.....	45
3.4.4.2 构建容器镜像.....	45
3.4.4.3 启动训练容器.....	48
3.4.4.4 运行 benchmark 工具.....	49
3.5 新模型适配 benchmark 工具方法.....	49
3.5.1 总体流程.....	49
3.5.2 创建模型目录.....	50
3.5.3 获取训练入口脚本及启动训练方式.....	50
3.5.4 修改相关脚本.....	51
3.5.5 适配打点日志.....	51
3.5.5.1 添加打点日志步骤.....	51
3.5.5.2 修改训练过程打点日志.....	52
3.6 常见问题.....	52
3.6.1 模型训练时报错 “Open TsdClient failed, tdt error code: *****, error message: tdt client open failed”	52
3.6.2 YoloV3 模型训练时缺少预训练数据集.....	53
3.6.3 搭建训练环境时依赖 numpy 安装报错.....	53
3.6.4 ResNet50 模型训练时报错 “ValueError: Data formats other than 'NHWC' are not yet supported”	53
3.7 附录.....	54
3.7.1 适配不同硬件形态.....	54
3.7.2 安装 TensorFlow.....	55





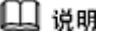
1 前言

概述

本文档详细的描述了推理benchmark工具和训练benchmark工具的使用方法及其使用实例。

符号约定

在本文中可能出现下列标志，它们所代表的含义如下。

符号	说明
 危险	表示如不避免则将会导致死亡或严重伤害的具有高等级风险的危害。
 警告	表示如不避免则可能导致死亡或严重伤害的具有中等级风险的危害。
 注意	表示如不避免则可能导致轻微或中度伤害的具有低等级风险的危害。
 须知	用于传递设备或环境安全警示信息。如不避免则可能会导致设备损坏、数据丢失、设备性能降低或其它不可预知的结果。 “须知”不涉及人身伤害。
 说明	对正文中重点信息的补充说明。 “说明”不是安全警示信息，不涉及人身、设备及环境伤害信息。

修改记录

文档版本	发布日期	修改说明
01	2020-10-10	第一次正式发布。

2 推理 benchmark 工具

- [2.1 工具介绍](#)
- [2.2 获取工具包](#)
- [2.3 运行工具](#)
- [2.4 使用实例](#)
- [2.5 新模型适配benchmark工具方法](#)

2.1 工具介绍

2.1.1 简介

功能描述

推理benchmark工具用来针对指定的推理模型运行推理程序，并能够测试推理模型的性能（包括吞吐率、时延）和精度指标。

使用场景

推理benchmark工具分为以下两种使用场景，两种场景通过不同的参数进行区分。

场景	说明
纯推理场景	该场景仅用来测试推理模型的性能指标，即模型执行的平均时间和平均吞吐率。 该场景无需准备推理数据及数据集文件，只需要准备经过ATC转换后的模型文件即可推理。
推理场景	该场景除了测试性能指标之外，还可以测试模型的精度指标。 该场景需要准备经过预处理的推理数据及数据集文件和经过ATC转换后的模型文件才能推理。

环境要求

项目	要求
产品形态	Atlas 300I 推理卡（型号 3000） Atlas 300I 推理卡（型号 3010） Atlas 800 推理服务器（型号 3000） Atlas 800 推理服务器（型号 3010） Atlas 500 智能小站
操作系统	Ubuntu 18.04 CentOS 7.6 EulerOS 2.8

2.1.2 工具目录说明

推理benchmark工具的目录结构如下：

说明

scripts目录下的脚本样例本文仅提供了一部分，实际脚本样例请用户以获取的工具为准。

```
benchmark_tools/  
├── scripts/  
│   ├── Sample/  
│   ├── bert_metric.py  
│   ├── bin_to_predict.py  
│   ├── get_info.py  
│   ├── map_calculate.py  
│   ├── nmt_metric.py  
│   ├── postprocess_tf_ssd_mobilenet_v1_fpn.py  
│   ├── preprocessing_torch_resnet50.py  
│   ├── preprocess_resnet50.py  
│   ├── preprocess_tf_ssd_mobilenet_v1_fpn.py  
│   ├── preprocess_tf_transform.py  
│   ├── preprocessing_tf_mobilenetv2.py  
│   ├── preprocessing_tf_resnet101.py  
│   ├── transform_tf_bin2de.py  
│   ├── vision_metric.py  
│   ├── yolo_caffe_postprocess.py  
│   └── yolo_caffe_preprocess.py  
├── benchmark.aarch64  
└── benchmark.x86_64
```

表 2-1 目录说明

目录名称	目录说明
Benchmark_tools	工具总目录。
scripts	工具脚本目录。
Sample	工程样例。
bert_metric.py	Bert模型精度统计脚本。

目录名称	目录说明
bin_to_predict.py	YoloV3 TensorFlow模型后处理脚本。
get_info.py	创建数据集脚本。
map_calculate.py	mAP精度统计脚本。
nmt_metric.py	NMT模型精度统计脚本。
preprocess_tf_ssd_mobilenet_v1_fpn.py	ssd_mobilenetV1_fpn模型预处理脚本。
postprocess_tf_ssd_mobilenet_v1_fpn.py	ssd_mobilenetV1_fpn模型后处理脚本。
preprocessing_torch_resnet50.py	ResNet50_pytorch模型预处理脚本。
preprocess_resnet50.py	ResNet50_caffe模型预处理脚本。
preprocess_tf_transform.py	Transform模型预处理脚本。
transform_tf_bin2de.py	Transform模型后处理脚本。
preprocessing_tf_mobilenetv2.py	MobilenetV2模型预处理脚本。
preprocessing_tf_resnet101.py	ResNet101模型预处理脚本。
vision_metric.py	Vision模型精度统计脚本。
yolo_caffe_postprocess.py	YoloV3_caffe模型后处理脚本。
yolo_caffe_preprocess.py	YoloV3_caffe模型预处理脚本。
benchmark.aarch64 benchmark.x86_64	benchmark工具执行脚本。

2.2 获取工具包

- 步骤1** 登录[软件获取](#)页面。
- 步骤2** 按需求选择对应的软件版本。
- 步骤3** 获取推理benchmark工具包infer_benchmark.zip。
- 结束

2.3 运行工具

说明

在物理机和容器内均支持运行推理benchmark工具，且运行方法相同。在容器中安装运行环境后，并将推理benchmark工具包挂载到容器内即可运行推理benchmark工具。

2.3.1 纯推理场景

前提条件

- 安装运行环境，具体安装方法请参见《CANN 软件安装指南》。
- 准备经过ATC转换后的模型OM文件，转换方法请参见《CANN 开发辅助工具指南(推理)》中的“ATC工具使用指导”章节。

运行方法

步骤1 以root用户登录服务器。

步骤2 从[2.2 获取工具包](#)章节中获取benchmark工具包，并进行解压。

步骤3 进入解压后的文件夹，获取benchmark工具**benchmark.{arch}**。

{arch}为CPU架构，取值为aarch64或x86_64。

步骤4 将benchmark工具、模型OM文件上传到服务器的任意路径下。

这些文件可以上传到同一路径，也可以上传到不同路径，以用户实际情况为准。

步骤5 设置环境变量。

环境变量设置示例如下：

```
export install_path=/usr/local/Ascend/ascend-toolkit/latest
export PATH=/usr/local/python3.7.5/bin:${install_path}/atc/cccec_compiler/bin:${install_path}/atc/bin:$PATH
export PYTHONPATH=${install_path}/atc/python/site-packages:$PYTHONPATH
export LD_LIBRARY_PATH=${install_path}/atc/lib64:${install_path}/acllib/lib64:$LD_LIBRARY_PATH
export ASCEND_OPP_PATH=${install_path}/opp
```

步骤6 进入benchmark工具所在路径，执行如下命令运行benchmark工具。

./benchmark.{arch} 运行参数

benchmark工具支持的运行参数及其说明请参见[表2-2](#)。

若显示类似如下所示信息，表示运行成功，并自动创建result文件夹（如果已存在，则不再创建），并在result文件夹下生成推理性能输出文件（文件名以**PureInfer_perf**开头），用来记录模型执行的平均时间和平均吞吐率，同时屏幕上打印的信息中也包含模型执行的平均时间和平均吞吐率。运行结果参数说明请参见[表2-3](#)。

```
[INFO][Inference] PureInfer Init SUCCESS
[INFO] Dataset number: 0 finished cost 12.273ms
...
[INFO] Dataset number: 29 finished cost 11.947ms
[INFO] PureInfer result saved in ./result/PureInfer_perf_of_yolov3_fp16_bs1_in_device_0.txt
-----PureInfer Performance Summary-----
[INFO] ave_throughputRate: 83.4736samples/s, ave_latency: 11.9798ms
```

```
-----  
[INFO][Inference] PureInfer unload model SUCCESS!
```

----结束

运行参数说明

表 2-2 运行参数说明

参数	说明	是否必填
[-batch_size, -bs]	执行一次模型推理所处理的数据量。	是
[-device_id, -di]	运行的Device编号，请根据实际使用的Device修改。缺省值为0。	否
[-om_path, -op]	模型文件所在的路径。	是
[-round, -r]	执行模型推理的次数，取值范围为1~1024。	是

运行结果参数说明

表 2-3 运行结果参数说明

参数	说明
ave_throughputRate	模型的平均吞吐率。单位为samples/s。
ave_latency	模型执行的平均时间。单位为ms。

2.3.2 推理场景

前提条件

- 安装运行环境，具体安装方法请参见《CANN 软件安装指南》。
- 准备经过ATC转换后的模型OM文件，转换方法请参见《CANN 开发辅助工具指南(推理)》中的“ATC工具使用指导”章节。
- 准备经过预处理的推理数据及数据集文件，预处理方法可以参考scripts目录下的预处理脚本样例。

运行方法

步骤1 以root用户登录服务器。

步骤2 从[2.2 获取工具包](#)章节中获取benchmark工具包，并进行解压。

步骤3 进入解压后的文件夹，获取benchmark工具**benchmark.{arch}**。
{arch}为CPU架构，取值为aarch64或x86_64。

步骤4 将benchmark工具、模型OM文件、模型对应的数据集上传到服务器任意目录下。

这些文件可以上传到同一路径，也可以上传到不同路径，以用户实际情况为准。

步骤5 设置环境变量。

环境变量设置示例如下：

```
export install_path=/usr/local/Ascend/ascend-toolkit/latest
export PATH=/usr/local/python3.7.5/bin:${install_path}/atc/cccec_compiler/bin:${install_path}/atc/bin:$PATH
export PYTHONPATH=${install_path}/atc/python/site-packages:$PYTHONPATH
export LD_LIBRARY_PATH=${install_path}/atc/lib64:${install_path}/acllib/lib64:$LD_LIBRARY_PATH
export ASCEND_OPP_PATH=${install_path}/opp
```

步骤6 进入benchmark工具所在路径，执行如下命令运行benchmark工具。

./benchmark.{arch} 运行参数

benchmark工具支持的运行参数及其说明请参见表2-4。

若显示类似如下所示信息，表示运行成功，并自动创建result文件夹（如果已存在，则不再创建），并在result文件夹下生成记录吞吐率和时延的推理性能输出文件（文件格式为perf_<model_type>_batchsize_<batch_size>_device_<device_id>.txt，例如perf_vision_batchsize_16_device_0.txt），同时屏幕上打印的信息中也包含吞吐率和时延。运行结果参数说明请参见表2-5。

```
[INFO][Preprocess] Init SUCCESS
[INFO][DataManager] Init SUCCESS
[INFO][Inference] Init SUCCESS
[INFO][PostProcess] Init SUCCESS
[INFO][Vision] Create stream SUCCESS
[INFO][Vision] Dvpp init resource SUCCESS
[INFO][Vision] Init SUCCESS
[INFO][PostProcess] CurSampleNum is 1
-----Performance Summary-----
[e2e] throughputRate: 0.53346, latency: 1874.55
[data read] throughputRate: 2702.7, moduleLatency: 0.37
[preprocess] throughputRate: 253.485, moduleLatency: 3.945
[infer] throughputRate: 95.4381, Interface throughputRate: 168.322, moduleLatency: 6.68
[post] throughputRate: 82.6241, moduleLatency: 12.103
-----
[INFO][Vision] DeInit SUCCESS
[INFO][Preprocess] DeInit SUCCESS
[INFO][Inference] Unload model SUCCESS!
[INFO][Inference] DeInit SUCCESS
[INFO][DataManager] DeInit SUCCESS
[INFO][PostProcess] DeInit SUCCESS
```

同时会生成推理结果文件，不同的模型获得的推理结果文件格式不同，请以实际模型为准。

步骤7 在工具的scripts目录下获取后处理脚本和精度统计脚本，并将其上传到服务器任意路径下。

步骤8 进入后处理脚本和精度统计脚本所在路径，执行后处理脚本和精度统计脚本解析模型输出结果，测试精度，生成结果文件。

不同的模型执行的脚本不同，请以实际模型为准。

----结束

运行参数说明

表 2-4 运行参数说明

参数	说明	是否必填
[-model_type, -mt]	模型的类型，当前支持如下几种： <ul style="list-style-type: none">• vision：图像处理• nmt：翻译• widedeep：搜索• yolocaffe：Yolo目标检测• nlp：自然语言处理• bert：语义理解	是
[-batch_size, -bs]	执行一次模型推理所处理的数据量。	是
[-device_id, -di]	运行的Device编号，请根据实际使用的Device修改。缺省值为0。	否
[-om_path, -op]	模型OM文件所在的路径。	是
[-input_width, -iw]	输入模型的宽度。	仅vision和yolocaffe模型类型支持该参数，且必填。
[-input_height, -ih]	输入模型的高度。	仅vision和yolocaffe模型类型支持该参数，且必填。
[-input_text_path, -itp]	模型对应的数据集所在的路径。	vision和yolocaffe模型类型为可选参数，-input_text_path、-input_image_path和-input_imgFiles_path参数选择其一即可。 bert、nlp、widedeep和nmt模型类型必填。
[-input_image_path, -iip]	模型对应的单张图片所在的路径。	仅vision和yolocaffe模型类型支持该参数，且可选。-input_text_path、-input_image_path和-input_imgFiles_path参数选择其一即可。

参数	说明	是否必填
[-input_imgFiles_path, -ifp]	模型对应的图片文件夹所在的路径。	仅vision和yolocaffe模型类型支持该参数，且可选。-input_text_path、-input_image_path和-input_imgFiles_path参数选择其一即可。
[-input_vocab, -iv]	输入语言词典文件所在的路径。	仅nmt模型类型支持该参数，且必填。
[-ref_vocab, -rv]	输出语言词典文件所在的路径。	仅nmt模型类型支持该参数，且必填。
[-output_binary, -ob]	输出结果格式是否为二进制文件（即bin文件）。取值为： <ul style="list-style-type: none">• true：输出结果格式为bin文件。• false：输出结果格式为txt文件。 缺省值为false。	否
[-useDvpp, -u]	模型前处理是否适用DVPP编解码模块。取值为： <ul style="list-style-type: none">• true• false 缺省值为false。若使用DVPP编解码或图像缩放，则该参数置为true。其他情况置为false。	仅vision和yolocaffe模型类型支持该参数，且为必选。

运行结果参数说明

表 2-5 运行结果参数说明

参数		说明
[e2e]	throughputRate	端到端总吞吐率。公式为sample个数/时间。
	latency	端到端时延，即从处理第一个sample到最后一个sample的完成时间。
[data read]	throughputRate	当前模块的吞吐率。
[preprocess] [post]	moduleLatency	执行一次当前模块的时延。

参数		说明
[infer]	throughputRate	推理模块的吞吐率。公式为sample个数/执行一次推理的时间。
	moduleLatency	推理模块的平均时延。公式为执行一次推理的时间/batch size。
	Interface throughputRate	aclmdlExecute接口的吞吐率。公式为sample个数/aclmdlExecute接口的平均执行时间。

须知

根据目前的统计规则，一般情况下sample个数=batch size。但是对于NMT模型，sample个数=整个batch中所有句子的总单词个数。

2.4 使用实例

2.4.1 纯推理场景使用实例

以YoloV3 caffe模型为例进行说明：

步骤1 将工具运行脚本benchmark.x86_64、模型OM文件上传到服务器的同一路径下（如“/home/work”）。

步骤2 进入“/home/work”，执行如下命令运行benchmark工具。

```
./benchmark.x86_64 -device_id=0 -om_path=./yolov3_bs1.om -round=30 -batch_size=1
```

若出现如下显示信息，表示运行成功。从显示信息可以看到模型执行的平均时间和平均吞吐率，同时在result文件夹下也会生成记录该信息的文件

PureInfer_perf_of_yolov3_bs1_in_device_0.txt。

```
[INFO][Inference] PureInfer Init SUCCESS
[INFO] Dataset number: 0 finished cost 12.273ms
[INFO] Dataset number: 1 finished cost 11.951ms
[INFO] Dataset number: 2 finished cost 11.964ms
[INFO] Dataset number: 3 finished cost 12.015ms
[INFO] Dataset number: 4 finished cost 12.002ms
[INFO] Dataset number: 5 finished cost 12.056ms
[INFO] Dataset number: 6 finished cost 11.98ms
[INFO] Dataset number: 7 finished cost 11.916ms
[INFO] Dataset number: 8 finished cost 11.906ms
[INFO] Dataset number: 9 finished cost 12.021ms
[INFO] Dataset number: 10 finished cost 11.956ms
[INFO] Dataset number: 11 finished cost 11.974ms
[INFO] Dataset number: 12 finished cost 11.982ms
[INFO] Dataset number: 13 finished cost 11.993ms
[INFO] Dataset number: 14 finished cost 11.904ms
[INFO] Dataset number: 15 finished cost 11.91ms
[INFO] Dataset number: 16 finished cost 11.867ms
[INFO] Dataset number: 17 finished cost 11.887ms
[INFO] Dataset number: 18 finished cost 11.958ms
```

```
[INFO] Dataset number: 19 finished cost 11.989ms
[INFO] Dataset number: 20 finished cost 12.035ms
[INFO] Dataset number: 21 finished cost 11.963ms
[INFO] Dataset number: 22 finished cost 11.989ms
[INFO] Dataset number: 23 finished cost 11.997ms
[INFO] Dataset number: 24 finished cost 12.028ms
[INFO] Dataset number: 25 finished cost 11.918ms
[INFO] Dataset number: 26 finished cost 11.962ms
[INFO] Dataset number: 27 finished cost 11.975ms
[INFO] Dataset number: 28 finished cost 12.077ms
[INFO] Dataset number: 29 finished cost 11.947ms
[INFO] PureInfer result saved in ./result/PureInfer_perf_of_yolov3_bs1_in_device_0.txt
-----PureInfer Performance Summary-----
[INFO] ave_throughputRate: 83.4736samples/s, ave_latency: 11.9798ms
-----
[INFO][Inference] PureInfer unload model SUCCESS!
```

----结束

2.4.2 推理场景使用实例

下面对推理benchmark工具支持的几种模型类型分别进行举例说明。

Vision 类型

以AlexNet模型和YoloV3 TensorFlow模型为例进行说明：

- AlexNet模型
 - a. 将工具运行脚本benchmark.x86_64、模型OM文件、数据集、精度统计脚本vision_metric.py等相关文件上传到服务器的同一路径下（如“/home/work”）。
 - b. 进入“/home/work”，执行如下命令运行benchmark工具。

```
./benchmark.x86_64 -model_type=vision -batch_size=1 -device_id=1 -  
om_path=./alexnet.om -input_width=224 -input_height=224 -  
input_text_path=./ImageNet128.info -useDvpp=true
```

若显示如下信息，说明运行成功。从显示信息可以看到吞吐率和时延信息。
同时在result文件夹也会生成记录这些信息的文件
perf_vision_batchsize_1_device_1.txt。

```
[INFO][Preprocess] Init SUCCESS
[INFO][DataManager] Init SUCCESS
[INFO][Inference] Init SUCCESS
[INFO][PostProcess] Init SUCCESS
[INFO][Vision] Create stream SUCCESS
[INFO][Vision] Dvpp init resource SUCCESS
[INFO][Vision] Init SUCCESS
[INFO][PostProcess] CurSampleNum is 1
[INFO][PostProcess] CurSampleNum is 2
...
[INFO][PostProcess] CurSampleNum is 127
-----Performance Summary-----
[e2e] throughputRate: 66.3689, latency: 1913.55
[data read] throughputRate: 7504.14, moduleLatency: 0.13326
[preprocess] throughputRate: 327.251, moduleLatency: 3.05576
[infer] throughputRate: 156.237, Interface throughputRate: 178.117, moduleLatency: 6.40864
[post] throughputRate: 156.084, moduleLatency: 6.4068
-----
[INFO][Vision] DeInit SUCCESS
[INFO][Preprocess] DeInit SUCCESS
[INFO][Inference] DeInit SUCCESS
[INFO][DataManager] DeInit SUCCESS
[INFO][PostProcess] DeInit SUCCESS
```

同时在“result/dumpOutput”下生成各数据的推理结果文件（与各数据同名的txt文件）。

- c. 执行如下命令计算并生成精度结果文件，该文件记录模型的TOP-1至TOP-5精度。

```
python3 vision_metric.py ./result/dumpOutput ./HiAIAnnotations ./result.json
```

参数	含义
./result/dumpOutput	b 生成的推理结果文件所在路径。
./HiAIAnnotations	真实结果文件所在路径。用户根据实际路径替换。
./ result.json	生成的精度结果文件存放路径和精度结果文件的名称（如 result.json ），用户可以自定义。

精度结果文件信息显示示例如下：

```
{"title": "Overall statistical evaluation", "value":  
[{"key": "Number of images", "value": "127"},  
{"key": "Number of classes", "value": "1000"},  
{"key": "Top1 accuracy", "value": "27.56%"},  
{"key": "Top2 accuracy", "value": "39.37%"},  
{"key": "Top3 accuracy", "value": "45.67%"},  
{"key": "Top4 accuracy", "value": "47.24%"},  
{"key": "Top5 accuracy", "value": "50.39%"}]}
```

- YoloV3 TensorFlow模型

- a. 将工具运行脚本benchmark.x86_64、模型OM文件、数据集、后处理脚本bin_to_predict.py和精度统计脚本map_calculate.py等相关文件上传到服务器的同一路径下（如“/home/work”）。
- b. 进入“/home/work”，执行如下命令运行benchmark工具。

```
./benchmark.x86_64 -model_type=vision -batch_size=1 -device_id=0 -  
om_path=./yolov3_bs1.om -input_text_path=./yolo_tf.info -  
input_width=416 -input_height=416 -useDvpp=false -  
output_binary=true
```

若显示如下信息，说明运行成功。从显示信息可以看到吞吐率和时延信息。同时在result文件夹也会生成记录这些信息的文件

perf_vision_batchsize_1_device_0.txt。

```
[INFO][Preprocess] Init SUCCESS  
[INFO][DataManager] Init SUCCESS  
[INFO][Inference] Init SUCCESS  
[INFO][PostProcess] Init SUCCESS  
[INFO][Vision] Create stream SUCCESS  
[INFO][Vision] Dvpp init resource SUCCESS  
[INFO][Vision] Init SUCCESS  
[INFO][PostProcess] CurSampleNum is 1  
[INFO][PostProcess] CurSampleNum is 2  
...  
[INFO][PostProcess] CurSampleNum is 35504  
-----Performance Summary-----  
[e2e] throughputRate: 38.251, lantency: 928186  
[data read] throughputRate: 61.9421, moduleLatency: 16.1441  
[preprocess] throughputRate: 39.3065, moduleLatency: 25.4411  
[infer] throughputRate: 38.3595, Interface throughputRate: 72.4677, moduleLatency: 11.0464  
[post] throughputRate: 38.3588, moduleLatency: 26.0696  
-----
```



```
[INFO][Vision] Delnit SUCCESS  
[INFO][Preprocess] Delnit SUCCESS  
[INFO][Inference] Delnit SUCCESS  
[INFO][DataManager] Delnit SUCCESS  
[INFO][PostProcess] Delnit SUCCESS
```

同时在“result/dumpOutput”下生成各数据的推理结果文件（与各数据同名的bin文件）。

- c. 执行如下命令将**b**的推理结果bin文件转换为txt文件。

```
python3 bin_to_predict.py --bin_data_path ./result/dumpOutput --  
test_annotation ./yolo_tf.info --det_results_path ./detection-results --  
coco_class_names ./coco.names --voc_class_names ./voc.names --  
net_input_width 416 --net_input_height 416
```

参数	说明
--bin_data_path	b 生成的推理结果文件所在的路径。
--test_annotation	模型对应的数据集。
--det_results_path	bin文件转换为txt文件后结果保存的路径。
--coco_class_names	coco数据集所在的路径。
--voc_class_names	VOC数据集所在的路径。
--net_input_width	模型的输入宽度。
--net_input_height	模型的输入高度。

- d. 执行如下命令根据**c**推理结果和真实结果计算出精度mAP值，并保存在result_yolo.txt文件中（用户可以自定义文件名）。

```
python3 map_calculate.py --label_path ./ground-truth/ --  
npu_txt_path ./detection-results -na -np > result_yolo.txt
```

其中--label_path为真实结果文件所在的路径；--npu_txt_path为**c**生成的txt文件所在的路径。

查看结果，显示示例如下：

```
45.82% = aeroplane AP  
52.68% = bicycle AP  
34.51% = bird AP  
16.42% = boat AP  
22.11% = bottle AP  
61.43% = bus AP  
49.49% = car AP  
24.56% = cat AP  
18.98% = chair AP  
23.94% = cow AP  
42.61% = diningtable AP  
35.86% = dog AP  
45.38% = horse AP  
48.54% = motorbike AP  
34.51% = person AP  
14.64% = pottedplant AP  
19.08% = sheep AP  
33.47% = sofa AP  
63.26% = train AP  
34.28% = tvmonitor AP  
mAP = 36.08%
```

NMT 类型

步骤1 将工具运行脚本benchmark.x86_64、模型OM文件、数据集、精度统计脚本nmt_metric.py等相关文件上传到服务器的同一路径下（如“/home/work”）。

步骤2 进入“/home/work”，执行如下命令运行benchmark工具。

```
./benchmark.x86_64 -model_type=nmt -batch_size=1 -device_id=0 -om_path=./nmt_tf_bs1.om -input_text_path=./tst2013.en -input_vocab=./vocab.en -ref_vocab=./vocab.vi
```

若显示如下信息，说明运行成功。从显示信息可以看到吞吐率和时延信息，同时在result文件夹也会生成记录这些信息的文件perf_nmt_batchsize_1_device_0.txt。

```
[INFO][Preprocess] Init SUCCESS
[INFO][DataManager] Init SUCCESS
[INFO][Inference] Init SUCCESS
[INFO][PostProcess] Init SUCCESS
[INFO][NmtPreprocess] Init SUCCESS
[INFO][PostProcess] CurSampleNum is 1
[INFO][PostProcess] CurSampleNum is 2
...
[INFO][PostProcess] CurSampleNum is 1268
-----Performance Summary-----
[e2e] throughputRate: 9.86887, latency: 128485
[data read] throughputRate: 93946.8, moduleLatency: 0.0106443
[preprocess] throughputRate: 3310.05, moduleLatency: 0.30211
[infer] throughputRate: 639.599, Interface throughputRate: 644.313, moduleLatency: 100.057
[post] throughputRate: 9.99372, moduleLatency: 100.063
-----
[INFO][NmtPreprocess] Delnit SUCCESS
[INFO][Preprocess] Delnit SUCCESS
[INFO][Inference] Delnit SUCCESS
[INFO][DataManager] Delnit SUCCESS
[INFO][PostProcess] Deinit SUCCESS
```

同时在result文件夹中会生成推理结果文件nmt_output_file.txt。

步骤3 执行如下命令根据**步骤2**中生成的翻译结果和人工翻译结果计算出精度。

```
python3 nmt_metric.py ./tst2013.vi ./result/nmt_output_file.txt > nmt.txt
```

其中./tst2013.vi为人工翻译结果文件所在路径，./result/nmt_output_file.txt为**步骤2**中生成的翻译结果文件所在路径，nmt.txt为精度结果保存的文件名（用户可以自定义文件名）。

查看结果，显示示例如下：

```
bleu: 0.56
```

----结束

Widedeep 类型

步骤1 将工具运行脚本benchmark.x86_64、模型OM文件、数据集等相关文件上传到服务器的同一路径下（如“/home/work”）。

说明

在benchmark工具所在目录下需要有权重文件occupation_embedding_weights.bin。

步骤2 进入“/home/work”，执行如下命令运行benchmark工具。

```
./benchmark.x86_64 -model_type=widedeep -batch_size=1 -device_id=1 -  
om_path=./widedeep_tf_bs1.om -input_text_path=./adult.test
```

若显示如下信息，说明运行成功。从显示信息可以看到吞吐率和时延信息，同时在result文件夹也会生成记录这些信息的文件

perf_widedeep_batchsize_1_device_1.txt。

```
[INFO][Preprocess] Init SUCCESS  
[INFO][DataManager] Init SUCCESS  
[INFO][Inference] Init SUCCESS  
[INFO][PostProcess] Init SUCCESS  
[INFO][WidePreProcess] Init SUCCESS  
[INFO][DeepPreprocess] Init SUCCESS  
[INFO][PostProcess] CurSampleNum is 1  
[INFO][PostProcess] CurSampleNum is 2  
...  
[INFO][PostProcess] CurSampleNum is 16281  
-----Performance Summary-----  
[e2e] throughputRate: 1107.93, latency: 14695  
[data read] throughputRate: 152354, moduleLatency: 0.00656366  
[wide preprocess] throughputRate: 151969, moduleLatency: 0.00658028  
[deep preprocess] throughputRate: 6317.22, moduleLatency: 0.158297  
[infer] throughputRate: 1214.36, Interface throughputRate: 2857.56, moduleLatency: 0.813056  
[post] throughputRate: 1214.34, moduleLatency: 0.823495  
-----  
[INFO][WidePreProcess] Delnit SUCCESS  
[INFO][DeepPreprocess] Delnit SUCCESS  
[INFO][Preprocess] Delnit SUCCESS  
[INFO][Inference] Delnit SUCCESS  
[INFO][DataManager] Delnit SUCCESS  
[INFO][PostProcess] Deinit SUCCESS
```

此外，还会在result文件夹下生成精度结果文件**widedeep_outputfile.txt**。精度结果文件显示信息示例如下：

```
rightCount: 7241  
totalCount: 16281  
accuracy: 0.444752
```

参数	含义
rightCount	预测正确的数据个数。
totalCount	数据集集中的数据总数。
accuracy	精度，公式为rightCount/totalCount。

----结束

YOLOCaffe 类型

以YoloV3 caffe模型为例进行说明：

步骤1 将工具运行脚本benchmark.x86_64、模型OM文件、数据集、后处理脚本yolo_caffe_postprocess.py和精度统计脚本map_calculate.py等相关文件上传到服务器的同一路径下（如“/home/work”）。

步骤2 进入“/home/work”，执行如下命令运行benchmark工具。

```
./benchmark.x86_64 -model_type=yolocaffe -batch_size=1 -device_id=0 -  
om_path=./yolov3_bs1.om -input_width=416 -input_height=416 -  
input_text_path=./test_VOC2007.info -useDvpp=false -output_binary=true
```

若显示如下信息，说明运行成功。从显示信息可以看到吞吐率和时延信息，同时在result文件夹也会生成记录这些信息的文件
perf_yolocaffe_batchsize_1_device_0.txt。

```
[INFO][Preprocess] Init SUCCESS
[INFO][DataManager] Init SUCCESS
[INFO][Inference] Init SUCCESS
[INFO][PostProcess] Init SUCCESS
[INFO][Vision] Create stream SUCCESS
[INFO][Vision] Dvpp init resource SUCCESS
[INFO][Vision] Init SUCCESS
[INFO][PostProcess] CurSampleNum is 1
[INFO][PostProcess] CurSampleNum is 2
...
[INFO][PostProcess] CurSampleNum is 4952
-----Performance Summary-----
[e2e] throughputRate: 63.5358, latency: 77940.4
[data read] throughputRate: 3452.21, moduleLatency: 0.28967
[preprocess] throughputRate: 2257.39, moduleLatency: 0.442989
[infer] throughputRate: 71.5616, Interface throughputRate: 91.2882, moduleLatency: 6.87263
[post] throughputRate: 64.5317, moduleLatency: 15.4963
-----
[INFO][Vision] Delnit SUCCESS
[INFO][Preprocess] Delnit SUCCESS
[INFO][Inference] Delnit SUCCESS
[INFO][DataManager] Delnit SUCCESS
[INFO][PostProcess] Deinit SUCCESS
```

同时在“result/dumpOutput”下生成各数据的推理结果文件（与各数据同名的两份bin文件）。

步骤3 执行如下命令将**步骤2**的推理结果bin文件转换为txt文件。

```
python3 yolo_caffe_postprocess.py --bin_data_path ./result/dumpOutput --
test_annotation ./test_VOC2007.info --det_results_path ./detection-results --
coco_class_names ./coco.names --voc_class_names ./voc.names --
net_input_width 416 --net_input_height 416
```

步骤4 执行如下命令根据**步骤3**的推理结果和真实结果计算出精度mAP值。

```
python3 map_calculate.py --label_path ./ground-truth/ --npu_txt_path ./
detection-results -na -np
```

其中--label_path为真实结果文件所在的路径；--npu_txt_path为**步骤3**生成的txt文件所在的路径。

显示示例如下：

```
86.82% = aeroplane AP
81.21% = bicycle AP
72.86% = bird AP
51.86% = boat AP
70.65% = bottle AP
92.49% = bus AP
80.39% = car AP
86.77% = cat AP
53.64% = chair AP
55.40% = cow AP
68.55% = diningtable AP
81.80% = dog AP
88.01% = horse AP
84.77% = motorbike AP
85.07% = person AP
43.31% = pottedplant AP
67.76% = sheep AP
73.35% = sofa AP
```

```
81.66% = train AP
73.00% = tvmonitor AP
mAP = 73.97%
```

----结束

Bert 类型

以Bert_base模型为例进行说明：

- 步骤1** 将工具运行脚本benchmark.x86_64、模型OM文件、数据集、精度统计脚本bert_metric.py等相关文件上传到服务器的同一路径下（如“/home/work”）。
- 步骤2** 进入“/home/work”，执行如下命令运行benchmark工具。

```
./benchmark.x86_64 -model_type=bert -batch_size=1 -device_id=0 -om_path=./bert_base_bs1.om -input_text_path=./MrpcData.info
```

若显示如下信息，说明运行成功。从显示信息可以看到吞吐率和时延信息，同时在result文件夹也会生成记录这些信息的文件perf_bert_batchsize_1_device_0.txt。

```
[INFO][Preprocess] Init SUCCESS
[INFO][DataManager] Init SUCCESS
[INFO][Inference] Init SUCCESS
[INFO][PostProcess] Init SUCCESS
[INFO][BertPreprocess] Init SUCCESS
[INFO][PostProcess] CurSampleNum is 1
[INFO][PostProcess] CurSampleNum is 2
...
[INFO][PostProcess] CurSampleNum is 3668
-----Performance Summary-----
[e2e] throughputRate: 238.919, latency: 4.18551
[data read] throughputRate: 15699.5, moduleLatency: 0.0636963
[preprocess] throughputRate: 796.15, moduleLatency: 1.25604
[infer] throughputRate: 82.1084, Interface throughputRate: 87.6803, moduleLatency: 12.1488
[post] throughputRate: 82.1079, moduleLatency: 12.1791
-----
[INFO][BertPreprocess] DelInit SUCCESS
[INFO][Preprocess] DelInit SUCCESS
[INFO][Inference] DelInit SUCCESS
[INFO][DataManager] DelInit SUCCESS
[INFO][PostProcess] Deinit SUCCESS
```

同时在“result/dumpOutput”下生成各数据的推理结果文件（与各数据同名的txt文件）。

- 步骤3** 执行如下命令计算精度。

```
python3 bert_metric.py ./data.txt ./result/dumpOutput ./ result_BERT_Base.txt
```

参数	含义
./data.txt	真实结果文件所在的路径。
./result/dumpOutput	步骤2 中生成的推理结果文件。
./ result_BERT_Base.txt	生成的精度结果文件存放路径和精度结果文件的名称（如result_BERT_Base.txt），用户可以自定义。

查看结果，显示示例如下：

```
rightNum: 3041  
toatlNum: 3668  
rightRate: 0.829062159214831
```

参数	含义
rightNum	预测正确的数据个数。
toatlNum	数据集集中的数据总数。
rightRate	精度，公式为rightNum/toatlNum。

----结束

NLP 类型

以Transform模型为例进行说明：

步骤1 将工具运行脚本benchmark.x86_64、模型OM文件、数据集、后处理脚本transform_tf_bin2de.py等相关文件上传到服务器的同一路径下（如“/home/work”）。

步骤2 进入“/home/work”，执行如下命令运行benchmark工具。

```
./benchmark.x86_64 -model_type=nlp -batch_size=1 -device_id=3 -om_path=./  
transformer_bs1.om -input_text_path=./transform_3003.info -  
output_binary=true
```

若显示如下信息，说明运行成功。从显示信息可以看到吞吐率和时延信息，同时在result文件夹也会生成记录这些信息的文件perf_nlp_batchsize_1_device_3.txt。

```
[INFO][Preprocess] Init SUCCESS  
[INFO][DataManager] Init SUCCESS  
[INFO][Inference] Init SUCCESS  
[INFO][PostProcess] Init SUCCESS  
[INFO][NLPPreprocess] Init SUCCESS  
[INFO][PostProcess] CurSampleNum is 1  
[INFO][PostProcess] CurSampleNum is 2  
...  
[INFO][PostProcess] CurSampleNum is 3003  
-----Performance Summary-----  
[e2e] throughputRate: 0.173933, latency: 5749.33  
[data read] throughputRate: 3248.83, moduleLatency: 0.307803  
[preprocess] throughputRate: 3246.9, moduleLatency: 0.307986  
[infer] throughputRate: 0.0579881, Interface throughputRate: 0.0583952, moduleLatency: 5748.3  
[post] throughputRate: 0.173964, moduleLatency: 5748.31  
-----  
[INFO][Preprocess] Delnit SUCCESS  
[INFO][Inference] Delnit SUCCESS  
[INFO][DataManager] Delnit SUCCESS  
[INFO][PostProcess] Deinit SUCCESS
```

同时在“result/dumpOutput”下生成各数据的推理结果文件（与各数据同名的bin文件）。

步骤3 执行如下命令将生成的bin文件转换为翻译后的文本文件。

```
python3 transform_tf_bin2de.py ./result/dumpOutput ./  
vocab.translate_ende_wmt32k.32768.subwords ./benchmark_test
```

其中./result/dumpOutput为步骤2生成的推理结果文件所在的路径； ./vocab.translate_ende_wmt32k.32768.subwords为翻译词典所在的路径； ./benchmark_test为生成的翻译后文件所在的路径。

步骤4 执行如下命令根据生成的翻译后文本文件和真实的翻译结果计算出精度。

```
t2t-bleu --translation=./benchmark_test --reference=./newstest2014.de
```

其中./benchmark_test为步骤3生成的翻译后文件所在的路径； ./newstest2014.de为真实的翻译结果所在路径。

精度结果显示示例如下：

```
BLEU_uncased = 24.26
```

----结束

2.5 新模型适配 benchmark 工具方法

2.5.1 模型适配说明

推理benchmark工具框架设计包含数据输入、前处理、模型推理、后处理和精度统计等模块。不同模型任务的前处理和后处理存在较大差异，因此在设计上支持添加不同的模型类别来做相应的适配开发。

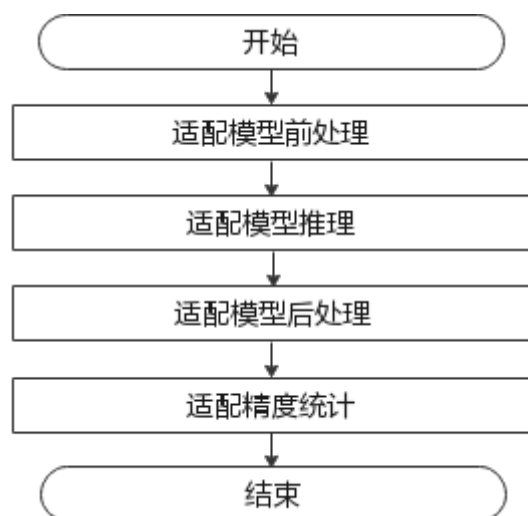
新模型适配开发工作主要涉及模型前处理、模型推理、模型后处理等三个方面。

- 模型前处理适配：模型推理前需要对数据进行前处理，前处理的方法与原始工程中的方法一致，处理好的数据传入模型推理模块。
- 模型推理适配：根据不同的模型结构，构造对应的输入数据结构，并传入模型推理接口执行推理。
- 模型后处理适配：保存模型输出层信息，提取有效输出特征，并根据输出特征进行精度统计。

2.5.2 模型适配总体流程

新模型适配benchmark工具的总体流程如图2-1所示：

图 2-1 总体流程



2.5.3 模型适配过程

下面以ResNet50为例说明模型适配操作步骤。ResNet50为Vision类型模型，只需要开发用于适配benchmark工具的前处理脚本和后处理脚本即可，无需适配模型推理。此外，ResNet50模型适配模型后处理和精度统计，采用一个脚本实现。

前提条件

- 获取ResNet50模型
获取地址：<https://github.com/KaimingHe/deep-residual-networks>
说明：从该地址下载模型网络文件prototxt和权重文件caffemodel。
- 获取测试集
获取地址：<http://www.image-net.org/challenges/LSVRC/2012/>
说明：从该地址下载2012年的图像测试集，并利用scripts目录下get_info.py脚本将图像测试集生成模型推理所需的输入数据集ImageNet2012_bin.list。输入数据集的格式为“图像编号 图像路径 宽 高”，示例如下：

```
1 dataset/000001.bin 224 224
2 dataset/000002.bin 224 224
...
```

操作步骤

步骤1 适配模型前处理。

模型前处理支持以下两种方法：

- 方法1：支持采用脚本将JPEG图像预先做图像处理并保存成bin格式文件，然后拷贝到NPU的Device端，构造成模型输入数据结构。
- 方法2：支持读取JPEG图像，通过DVPP做图像预处理，并拷贝到NPU的Device端，构造成模型输入数据结构。

本例采用方法1进行模型前处理，即对图像依次进行固定宽高比缩放、居中裁剪和减均值，并保存成bin格式文件。

1. 固定宽高比缩放。

在模型输入大小基础上，对图像先放大后缩小，做固定宽高比例缩放。如下所示：

```
def resize_with_aspectratio(img, out_height, out_width, scale=87.5,
inter_pol=cv2.INTER_LINEAR):
    height, width, _ = img.shape
    new_height = int(100. * out_height / scale)
    new_width = int(100. * out_width / scale)
    if height > width:
        w = new_width
        h = int(new_height * height / width)
    else:
        h = new_height
        w = int(new_width * width / height)
    img = cv2.resize(img, (w, h), interpolation=inter_pol)
    return im
```

2. 居中裁剪。

```
def center_crop(img, out_height, out_width):
    height, width, _ = img.shape
    left = int((width - out_width) / 2)
    right = int((width + out_width) / 2)
```



```
top = int((height - out_height) / 2)
bottom = int((height + out_height) / 2)
img = img[top:bottom, left:right]
return img
```

3. 减均值。

对图像三个颜色通道减去对应的平均值。

```
means = np.array([103.94, 116.78, 126.68], dtype=np.float16)
```

4. 模型前处理完整步骤如下：

```
import numpy as np
import os
import cv2
import sys

def preprocess_resnet50(img, need_transpose=True, precision="fp16"):
    output_height = 224
    output_width = 224
    cv2_interpol = cv2.INTER_AREA
    img = resize_with_aspectratio(img, output_height, output_width, inter_pol=cv2_interpol)
    img = center_crop(img, output_height, output_width)

    if precision == "fp32":
        img = np.asarray(img, dtype='float32')
    if precision == "fp16":
        img = np.asarray(img, dtype='float16')

    means = np.array([103.94, 116.78, 126.68], dtype=np.float16)
    img -= means
    if need_transpose:
        img = img.transpose([2, 0, 1])
    return img

def preprocess_resnet50(input_path, output_path):
    img = cv2.imread(input_path)
    h, w, _ = img.shape
    img_name = input_path.split('/')[-1]
    bin_name = img_name.split('.')[0] + ".bin"
    output_bin = os.path.join(output_path, bin_name)
    imgdst = preprocess_resnet50 (output_bin, img)
    imgdst.tofile(output_bin)

if __name__ == "__main__":
    pathSrcImgFD = sys.argv[1]
    pathDstBinFD = sys.argv[2]
    images = os.listdir(pathSrcImgFD)
    for image_name in images:
        if not image_name.endswith(".jpeg"):
            continue
        print("start to process image {}".format(image_name))
        path_image = os.path.join(pathSrcImgFD, image_name)
        resnet50(path_image, pathDstBinFD)
```

其中：

- `resize_with_aspectratio`的实现方法参考[步骤1.1](#)
- `center_crop`的实现方法参考[步骤1.2](#)。

步骤2 适配模型推理。

ResNet50模型推理，benchmark工具已经适配支持，无需再适配。推理方法如下：

1. 将开源caffe模型转换为ATC模型。[步骤1](#)中已将图像处理为float16类型，模型输入类型同样指定为float16类型。模型转换方法详细请参见《CANN 开发辅助工具指南(推理)》中的“ATC工具使用指导”章节。模型转换命令如下：

```
atc --model=./ResNet50_bs8.prototxt --weight=./ResNet-50-  
model.caffemodel --framework=0 --input_shape=data:8,3,224,224 --  
input_fp16_nodes=data --soc_version=Ascend310 --input_format=NCHW --  
output_type=FP32 --output=./ResNet50_bs8_ip16op32
```

2. 使用benchmark工具进行模型推理。推理过程详细请参见[Vision类型](#)。

模型推理命令如下：

```
./benchmark -model_type=vision -batch_size=8 -device_id=0 -om_path=./  
ResNet50_bs8_ip16op32.om -input_width=224 -input_height=224 -  
input_text_path=./dataset/ImageNet2012_bin.list -useDvpp=false
```

步骤3 适配模型后处理与精度统计。

ResNet50模型用于图像分类，该模型最后一层为softmax层，softmax层输出所有类别的预测概率值。模型后处理与精度统计的主要步骤如下，具体代码可以从benchmark工具包scripts目录中的vision_metric.py脚本获取。执行benchmark工具保存模型输出层信息，再执行vision_metric.py脚本解析模型输出层信息，并根据解析出来的信息统计精度。

1. 读取数据集真实标签。

```
def cre_groundtruth_dict_fromtxt(gtfile_path):  
    """  
    :param filename: file contains the imagename and label number  
    :return: dictionary key imagename, value is label number  
    """  
    img_gt_dict = {}  
    with open(gtfile_path, 'r') as f:  
        for line in f.readlines():  
            temp = line.strip().split(" ")  
            imgName = temp[0].split(".")[0]  
            imgLab = temp[1]  
            img_gt_dict[imgName] = imgLab  
    return img_gt_dict
```

2. 加载数据预测结果，并按照预测概率从高到低排序。

```
for tfile_name in os.listdir(prediction_file_path):  
    count += 1  
    temp = tfile_name.split('.')[0]  
    index = temp.rfind('_')  
    img_name = temp[:index]  
    filepath = os.path.join(prediction_file_path, tfile_name)  
    ret = load_statistical_predict_result(filepath)  
    prediction = ret[0]  
    n_labels = ret[1]  
    sort_index = np.argsort(-prediction)
```

3. 逐个比较预测结果与真实标签是否一致，统计模型分类精度，输出准确率。

```
sort_index = np.argsort(-prediction)  
gt = img_gt_dict[img_name]  
if (n_labels == 1000):  
    realLabel = int(gt)  
elif (n_labels == 1001):  
    realLabel = int(gt) + 1  
else:  
    realLabel = int(gt)  
resCnt = min(len(sort_index), topn)  
# print(sort_index[:5])
```

```
for i in range(resCnt):
    if (str(realLabel) == str(sort_index[i])):
        count_hit[i] += 1
        break
accuracy = np.cumsum(count_hit) / count
```

4. 精度统计完整步骤如下:

```
if __name__ == '__main__':
    start = time.time()
    folder_davinci_target = sys.argv[1] # txt file path
    annotation_file_path = sys.argv[2] # annotation files path, "val_label.txt"
    result_json_path = sys.argv[3] # the path to store the results json path
    json_file_name = sys.argv[4] # result json file name
    if not (os.path.exists(folder_davinci_target)):
        print("target file folder does not exist.")
    if not (os.path.exists(annotation_file_path)):
        print("Ground truth file does not exist.")
    if not (os.path.exists(result_json_path)):
        print("Result folder doesn't exist.")
    img_label_dict = cre_groundtruth_dict_fromtxt(annotation_file_path)
    create_visualization_statistical_result(folder_davinci_target,
        result_json_path, json_file_name, img_label_dict, topn=5)
    elapsed = (time.time() - start)
```

----结束

3 训练 benchmark 工具

- [3.1 工具介绍](#)
- [3.2 获取工具包](#)
- [3.3 运行工具](#)
- [3.4 使用场景](#)
- [3.5 新模型适配benchmark工具方法](#)
- [3.6 常见问题](#)
- [3.7 附录](#)

3.1 工具介绍

3.1.1 简介

功能描述

训练benchmark工具用来对模型进行训练，并对模型的训练过程进行统计和分析，记录训练结果、运行时间等信息，同时能够测试训练模型的性能和精度指标。

训练benchmark工具自带不同框架模型，支持在不同的操作系统和硬件形态下进行训练，同时也支持用户适配新的模型。

使用场景

使用benchmark工具进行模型训练可以分为以下几种场景：

场景	说明
Host单机训练	由1台服务器完成训练。 以Atlas 800 训练服务器（型号 9000）为例，该服务器包含8块芯片（即昇腾AI处理器），其中0-3卡和4-7卡各为一个组网。根据训练的芯片数量不同，又可以分为： <ul style="list-style-type: none">• 单机单卡，即1p• 单机多卡，即2p、4p和8p
Docker单机训练	由1台容器完成训练。根据训练的芯片数量不同，又可以分为： <ul style="list-style-type: none">• 单机单卡，即1p• 单机多卡，即2p、4p和8p
Host集群训练	即Host多机多卡场景，由多台服务器使用同一数据集进行训练。 集群训练目前使用OpenMPI开源库的mpirun命令调用同一网段内的多台服务器中train.sh脚本来实现。在一台服务器上执行OpenMPI的mpirun命令负责启动train.sh脚本，其他服务器上则会自动被mpirun命令启动train.sh脚本。
Docker集群训练	即Docker多机多卡场景，由多台容器使用同一数据集完成训练。

环境要求

项目	要求
产品形态	Atlas 800 训练服务器（型号 9000） Atlas 800 训练服务器（型号 9010） Atlas 300T 训练卡（型号 9000）
操作系统	Ubuntu 18.04 CentOS 7.6 EulerOS 2.8

3.1.2 工具目录说明

训练benchmark工具的目录结构如下：

```
train/
├── atlas_benchmark-master/
│   ├── image_classification/
│   │   └── ResNet50/
│   │       ├── tensorflow/
│   │       │   ├── code/
│   │       │   ├── config/
│   │       │   └── npu_set_env.sh
│   │       └── scripts/
│   │           └── run.sh
```

```
├── train.sh
├── utils/
│   ├── shell/
│   │   ├── start.sh
│   │   ├── set_json.sh
│   │   ├── hccl_sample.json
│   │   └── get_params_for_yaml.sh
│   ├── atlasboost/
│   ├── benchmark_log/
│   └── Dockerfile/
├── result/
├── yaml/
├── benchmark.sh
└── docker_ct_build.sh
```

表 3-1 目录说明

目录名称	目录说明
train	模型训练的总目录。
atlas_benchmark-master	模型训练的执行代码存放目录。
image_classification	模型类型存放目录，以模型类型区分目录。目前有如下三个目录： <ul style="list-style-type: none">• image_classification：图片分类• nlp：自然语言处理• object_detection：目标检测
ResNet50	模型代码存放目录，以模型名称区分目录。
tensorflow	训练框架存放目录，以框架区分目录。目前有如下三个目录： <ul style="list-style-type: none">• tensorflow• pytorch• mindspore
code	各模型训练工具代码存放目录。
config	各模型环境变量配置文件存放目录。该目录中的 npu_set_env.sh 为系统环境变量配置脚本。
scripts	各模型训练调度相关脚本存放目录。该目录中包含如下脚本： <ul style="list-style-type: none">• run.sh：模型训练启动脚本• train.sh：模型训练调度脚本
utils	公共模块存放目录。

目录名称	目录说明
shell	公共流程控制脚本存放目录。该目录中包含如下脚本： <ul style="list-style-type: none">• start.sh：调度控制脚本• set_json.sh：RANK_TABLE_FILE配置脚本• hccl_sample.json：RANK_TABLE_FILE模板文件• get_params_for_yaml.sh：读取yaml配置文件的脚本
atlasboost	集群训练所需要的库存放目录。
benchmark_log	打点日志公共脚本存放目录。
Dockerfile	Docker集群训练场景下构建容器镜像所需的Dockerfile文件。
result	各模型日志与训练结果存放目录。
yaml	yaml文件存放目录。
benchmark.sh	训练benchmark工具的执行脚本。
docker_ct_build.sh	Docker集群训练场景下启动训练容器的执行脚本。

3.1.3 工具脚本说明

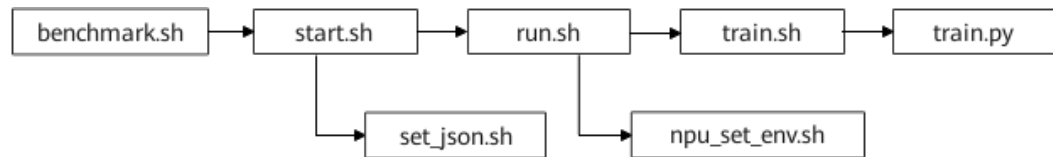
脚本功能

脚本	说明
benchmark.sh	工具执行脚本。负责解析用户参数，并将执行参数传递至模型接口脚本。
start.sh	各模型对外统一接口脚本。负责生成训练脚本使用的RANK_TABLE_FILE文件以及区分容器docker和物理机host。根据训练框架和模型，以不同方式调用run.sh脚本。
set_json.sh	自动根据当前硬件环境生成训练脚本使用的RANK_TABLE_FILE文件。RANK_TABLE_FILE文件包含训练的芯片信息。
run.sh	负责创建日志目录、配置参数以及区分单机和集群。根据单卡、多卡、单机和多机场景，以不同方式调用train.sh脚本，在集群场景下负责拷贝配置文件。
npu_set_env.sh	负责配置系统环境变量。
train.sh	负责配置各模型相关的环境变量、拉起训练任务。根据单卡、多卡、单机和多机场景，以不同方式调用train.py训练脚本。

脚本调用顺序

各脚本调用顺序如图3-1所示：

图 3-1 脚本调用顺序



3.1.4 打点日志介绍

打点日志模块位置为“train/atlas_benchmark-master/utils”目录下的 **benchmark_log**，具体包括：

- hwlog.py：包含日志打点中预设置的键，即要打印的内容名称；生成logger对象，构造日志打印函数。
- basic_utils.py：日志打点中获取一些键的值函数。

打点日志主要打印以下几点信息：

- 硬件与环境参数
- 训练初始化参数
- 模型配置参数
- 训练性能指标
- 训练精度指标和训练总时长

当要打印的信息不在hwlog.py的预设置键中时，例如要打印的信息为GLOBAL_BATCH_SIZE，可以执行如下操作将其增加到hwlog.py中：

步骤1 将要打印的信息按照如下格式增加到hwlog.py中。

```
GLOBAL_BATCH_SIZE = "global_batch_size"
```

其中GLOBAL_BATCH_SIZE是日志打印函数调用时传入的键，"global_batch_size"是在日志中显示的键。

步骤2 将新增的键加入到STDOUT_TAG_SET和REMARK_TAGS元组中。

----结束

3.2 获取工具包

步骤1 登录[软件获取](#)页面。

步骤2 按需求选择对应的软件版本。

步骤3 获取训练benchmark工具包train_benchmark.tar.gz。

----结束

3.3 运行工具

前提条件

- 安装运行环境，具体安装方法请参见《CANN 软件安装指南》。
- 运行benchmark工具启动模型训练之前，请先参照benchmark工具中各模型的README文件（例如YoLoV3模型路径为“/train/atlas_benchmark-master/object_detection/yolov3/tensorflow/README.md”）做好训练前准备工作。每个模型需要做的准备工作不同，请以实际README文件内容为准。例如YoLoV3模型，需要做的准备工作包括安装依赖、生成标注文件和转化ckpt文件。
- 获取测试模型对应的数据集。
- 确保模型对应环境变量路径下的环境变量与环境中安装的CANN软件包路径一致（如ResNet50模型的环境变量路径为“/home/train/atlas_benchmark-master/image_classification/ResNet50/tensorflow/config/npu_set_env.sh”）。

运行方法

步骤1 登录安装了昇腾AI设备的服务器。

步骤2 将[3.2 获取工具包](#)章节中获取的benchmark工具包上传到服务器某一目录下（如“/home”）。

步骤3 在工具包所在目录下执行如下命令解压工具包。

```
tar -zxvf train_benchmark.tar.gz
```

步骤4 执行如下命令进入train目录。

```
cd train
```

步骤5 执行如下命令增加train目录脚本权限。

```
chmod -R u+x *
```

步骤6 将模型对应的数据集上传到服务器某一目录下（如“/home”）。

步骤7 修改“/home/train/yaml/”目录下模型对应的yaml文件中的参数。

yaml文件以模型名称命名。各模型yaml文件中的参数以框架区分，请根据不同框架的模型训练，修改相应框架下的参数。不同模型和框架包含的参数不同。

须知

- 本文以ResNet50模型、TensorFlow框架为例说明各个参数，如[表3-2](#)所示。其他框架和模型的参数解释及其特殊说明请用户参考yaml文件中的注释，其中必须要修改的参数可参考下表。
- 请根据yaml文件中的注释说明和实际情况修改各个参数，切不可随意修改，否则可能导致训练失败或训练结果偏离实际。

表 3-2 参数说明

参数	说明	是否必填
data_url	指定ResNet50模型的数据集所在路径。例如“/home/imagenet_TF/”，用户需根据实际路径进行替换。	是
batch_size	指定模型的批处理数。	否
epoches	指定训练周期。	如果要测试精度，该参数设置为90。
max_train_steps	指定一个epoch跑多少步，默认1000。	如果要测试精度，该参数设置为None。
epochs_between_evals	指定多少epoch做一次评测精度。	否
iterations_per_loop	指定在每个estimator调用中执行多少步。	否
save_checkpoints_steps	指定保存checkpoint的步数。	否
mpirun_ip	指定设备Host IP和Device的数量，例如10.90.176.152:8。	集群场景必填
docker_image	指定镜像名称和标签，例如mpirun:latest。	Docker场景必填
device_group_1p device_group_2p device_group_4p	指定Device ID，用户需要根据实际设备的Device ID修改。多个ID使用空格分隔。执行 cat /etc/hccn.conf 命令可以查看设备的Device ID。 <ul style="list-style-type: none">● 如果为1p，需要修改device_group_1p。● 如果为2p，需要修改device_group_2p。● 如果为4p，需要修改device_group_4p。● 如果为8p，则无需修改该参数。	是

步骤8 执行如下任一命令运行benchmark工具，启动模型训练。

bash benchmark.sh 运行参数或者**./benchmark.sh** 运行参数

其中benchmark工具的运行参数可以通过执行**./benchmark.sh -h**或者**./benchmark.sh --help**命令查看支持的参数，各参数解释请参见表3-3。

步骤9 查看模型训练结果。

模型训练结果信息可以通过查看`home/train/result/tf_resnet50/training_job_*`目录下的日志文件获取。其中`tf_resnet50`文件夹仅为示例，该文件夹以模型和框架来命名。日志文件包括：

- 打点日志：记录每个模型训练过程打点。日志名以`hw`开头，例如`hw_resnet50.log`。打点日志主要记录如下信息：
 - 硬件芯片、系统软件和训练框架信息
 - 模型的学习率、数据集、优化器、loss scale和batch size等信息
 - 总训练步数、总训练轮数、性能和精度指标
- 训练日志：记录每次训练的训练过程信息，记录内容较为详细。日志名为`train_${deviceid}.log`。

----结束

运行参数说明

表 3-3 运行参数说明

参数	含义	是否必填
<code>--execmodel, -e</code>	指定模型名称，默认为ResNet50。	否
<code>--hardware, -hw</code>	指定芯片资源，取值为1p、2p、4p、8p、cluster ct（集群），默认为1p。	否
<code>--yamlpath, -y</code>	指定yaml文件所在的路径。默认路径为 <code>train/yaml/{execmodel}.yaml</code> 。其中 <code>{execmodel}</code> 为模型名称。	否
<code>--framework, -f</code>	指定模型训练的框架，取值为tensorflow、mindspore和pytorch。默认为tensorflow。	否
<code>-docker, -host</code>	选择训练是在容器docker内执行还是在物理机host执行，默认为host。	否
<code>--help, -h</code>	显示帮助信息。	-
<code>--list, -l</code>	显示当前支持的训练模型与训练框架。	-

3.4 使用场景

3.4.1 Host 单机训练场景

前提条件

请参照[3.3 运行工具](#)章节完成运行工具前的准备工作。包括：

- 安装运行环境，具体安装方法请参见《CANN 软件安装指南》中的“安装运行环境（训练）> 物理机安装”章节。

- 参照benchmark工具中各模型的README文件做好训练前准备工作。
- 获取测试模型对应的数据集并上传到服务器。
- 获取benchmark工具包并上传到服务器，解压benchmark工具包，并增加权限。
- 确保环境变量与环境中安装的CANN软件包路径一致。

使用实例

以Host单机单卡场景下启动框架为TensorFlow、模型为ResNet50训练为例：

步骤1 修改“/home/train/yaml/”目录下ResNet50.yaml文件中的**tensorflow_config**配置参数。

tensorflow_config:

- **data_url:** /home/imagenet_TF/
- **device_group_1p:** 0

步骤2 在train目录下执行如下命令运行benchmark工具，启动训练。

bash benchmark.sh -e ResNet50 -hw 1p

若显示如下所示的信息，表示ResNet50模型训练成功。

```
find script path success
run train script file path is /home/train/atlas_benchmark-master/image_classification/ResNet50/tensorflow/
scripts/run.sh
[20200918-22:25:16] [INFO] /home/train/result/tf_resnet50/training_job_20200918222516/ &
start to modify inner config file
[20200918-22:25:16] [INFO] train job is working, wait more 5s
modify inner config file success
[20200918-22:25:21] [INFO] train job is working, wait more 5s
[20200918-22:25:26] [INFO] train job is working, wait more 5s
...
[20200907-07:08:43] [INFO] train job is working, wait more 5s
:::ABK 1.0.0 resnet50 train success
:::ABK 1.0.0 resnet50 train total time: 0:46:40
[20200907-07:08:48] [INFO] process end
```

通过查看“/home/train/result/tf_resnet50/training_job_20200918222516”下的日志文件hw_resnet50.log，可以看到训练结果、训练时长、吞吐率（fps）和精度（eval_accuracy_top1和eval_accuracy_top5）等信息。显示示例如下：

```
:::ABK 1.0.0 resnet50 2020-09-18 22:25:22.518150 (resnet_tf_r1_benchmark.py:235) cpu_info:
{"Architecture": "aarch64Byte", "Order": "LittleEndian", "CPU(s)": "144-167NUMAnode7", "list":
"0-191Thread(s)per", "core": "1Core(s)per", "socket": "48", "Socket(s)": "4NUMA", "node(s)": "8Vendor",
"ID": "0x48", "Model": "0", "Stepping": "0x1CPUMax", "MHz": "200.0000", "BogoMIPS": "200.00L1d",
"cache": "49152KNUMAnode0"}
:::ABK 1.0.0 resnet50 2020-09-18 22:25:22.518922 (resnet_tf_r1_benchmark.py:236) npu_info: "Ascend910"
:::ABK 1.0.0 resnet50 2020-09-18 22:25:22.519340 (resnet_tf_r1_benchmark.py:237) os_info: "Linux version
4.15.0-55-generic (buildd@bos02-arm64-065) (gcc version 7.4.0 (Ubuntu/Linaro 7.4.0-1ubuntu1~18.04.1))
#60-Ubuntu SMP Tue Jul 2 18:23:38 UTC 2019"
:::ABK 1.0.0 resnet50 2020-09-18 22:25:22.519763 (resnet_tf_r1_benchmark.py:238) framework_info:
"tensorflow 1.15.0"
:::ABK 1.0.0 resnet50 2020-09-18 22:25:22.520164 (resnet_tf_r1_benchmark.py:239) benchmark_version:
"v1.0.0"
:::ABK 1.0.0 resnet50 2020-09-18 22:25:22.520638 (resnet_tf_r1_benchmark.py:240) config_info: {"data_url":
"/home/imagenet_TF/", "batch_size": 32, "epoches": 90, "max_train_steps": 1000, "epochs_between_evals":
5, "iterations_per_loop": 100, "save_checkpoints_steps": 115200, "device_group_1p": 0, "device_group_2p":
"0 1", "device_group_4p": "0 1 2 3"}
:::ABK 1.0.0 resnet50 2020-09-18 22:25:22.521071 (resnet_tf_r1_benchmark.py:241) base_lr: 0.128
:::ABK 1.0.0 resnet50 2020-09-18 22:25:22.521464 (resnet_tf_r1_benchmark.py:242) dataset: "imagenet1024"
:::ABK 1.0.0 resnet50 2020-09-18 22:25:22.521853 (resnet_tf_r1_benchmark.py:243) opt_name: "SGD"
:::ABK 1.0.0 resnet50 2020-09-18 22:25:22.522254 (resnet_tf_r1_benchmark.py:244) loss_scale: 512
:::ABK 1.0.0 resnet50 2020-09-18 22:25:22.522652 (resnet_tf_r1_benchmark.py:245) input_batch_size: 32
```

```
...ABK 1.0.0 resnet50 2020-09-18 22:25:24.766399 (resnet_run_loop.py:817) current_epoch: 5
...ABK 1.0.0 resnet50 2020-09-18 22:27:14.317073 (hooks.py:146) fps: 1534.8606114388047
...ABK 1.0.0 resnet50 2020-09-18 22:27:16.418724 (hooks.py:146) fps: 1519.122719508856
...ABK 1.0.0 resnet50 2020-09-18 22:27:18.520968 (hooks.py:146) fps: 1522.7351427284875
...ABK 1.0.0 resnet50 2020-09-18 22:27:20.618978 (hooks.py:146) fps: 1525.3831138071898
...ABK 1.0.0 resnet50 2020-09-18 22:27:22.716328 (hooks.py:146) fps: 1525.777782425998
...ABK 1.0.0 resnet50 2020-09-18 22:27:24.813244 (hooks.py:146) fps: 1525.4765603751496
...ABK 1.0.0 resnet50 2020-09-18 22:27:26.910933 (hooks.py:146) fps: 1526.0425120663092
...ABK 1.0.0 resnet50 2020-09-18 22:27:29.010216 (hooks.py:146) fps: 1524.3298150421258
...ABK 1.0.0 resnet50 2020-09-18 22:27:31.108209 (hooks.py:146) fps: 1525.4744798074655
...ABK 1.0.0 resnet50 2020-09-18 22:28:41.467371 (resnet_run_loop.py:858) eval_accuracy_top1:
0.0010796545539051294
...ABK 1.0.0 resnet50 2020-09-18 22:28:41.468240 (resnet_run_loop.py:859) eval_accuracy_top5:
0.006297984626144171
...ABK 1.0.0 resnet50 2020-09-18 22:28:42.098713 (resnet_run_loop.py:817) current_epoch: 5
...
...ABK 1.0.0 resnet50 train success
...ABK 1.0.0 resnet50 train total time: 0:46:40
```

----结束

以Host单机多卡场景下启动框架为TensorFlow、模型为Bert-Base训练为例：

步骤1 修改“/home/train/yaml/”目录下Bert-Base.yaml文件中的**tensorflow_config**参数。

tensorflow_config:

- **input_files_dir:** /home/BertData/cn-wiki-128/
- **eval_files_dir:** /home/BertData/cn-wiki-128/
- **device_group_2p:** 0 1

步骤2 在train目录下执行如下命令运行benchmark工具，启动训练。

bash benchmark.sh -e Bert-Base -hw 2p

若显示如下所示的信息，表示Bert-Base模型训练成功。

```
find script path success
run train script file path is /home/train/atlas_benchmark-master/nlp/Bert-Base/tensorflow/scripts/run.sh
[20200922-16:24:32] [INFO] see more config info in /home/train/atlas_benchmark-master/nlp/Bert-Base/
tensorflow/config
[20200922-16:24:32] [INFO] train result in /home/train/result/tf_bert_base/training_job_20200922162432/
[20200922-16:24:32] [INFO] train job is working, wait more 5s
[20200922-16:24:37] [INFO] train job is working, wait more 5s
[20200922-16:24:42] [INFO] train job is working, wait more 5s
...
[20200907-08:18:13] [INFO] train job is working, wait more 5s
...ABK 1.0.0 bert train success
...ABK 1.0.0 bert train total time 0:10:54
...ABK 1.0.0 bert train success
...ABK 1.0.0 bert train total time 0:10:54
[20200907-08:18:18] [INFO] process end
```

通过查看“/home/train/result/tf_bert_base/training_job_20200922162432”下的日志文件hw_bert.log，可以看到训练结果、训练时长、吞吐率（throwout）和精度（masked_lm_accuracy）等信息。显示示例如下：

```
...ABK 1.0.0 bert 2020-09-22 16:24:37.084792 (run_pretraining.py:769) cpu_info: {"Architecture":
"x86_64CPU", "op-mode(s)": "32-bit,64-bitByte", "Order": "LittleEndian", "CPU(s)":
"0-19,40-59NUMANode1", "list": "0-79Thread(s)per", "core": "2Core(s)per", "socket": "20", "Socket(s)":
"2NUMA", "node(s)": "2Vendor", "ID": "GenuineIntelCPU", "family": "6", "Model": "85Model", "name":
"Intel(R)Xeon(R)Gold6148CPU@2.40GHz", "Stepping": "4CPU", "MHz": "1000.0000", "BogoMIPS":
"4800.00", "Virtualization": "VT-xL1d", "cache": "28160KNUMANode0"}
...ABK 1.0.0 bert 2020-09-22 16:24:37.085350 (run_pretraining.py:770) npu_info: "Ascend910"
...ABK 1.0.0 bert 2020-09-22 16:24:37.085667 (run_pretraining.py:771) os_info: "Linux version
3.10.0-1127.13.1.el7.x86_64 (mockbuild@kbuilder.bsys.centos.org) (gcc version 4.8.5 20150623 (Red Hat
```

```
4.8.5-39) (GCC) #1 SMP Tue Jun 23 15:46:38 UTC 2020"
:::ABK 1.0.0 bert 2020-09-22 16:24:37.085966 (run_pretraining.py:772) framework_info: "tensorflow 1.15.0"
:::ABK 1.0.0 bert 2020-09-22 16:24:37.086266 (run_pretraining.py:773) benchmark_version: "v1.0.0"
:::ABK 1.0.0 bert 2020-09-22 16:24:37.086601 (run_pretraining.py:774) config_info: {"bert_config_file":
"bert_base_layer6_cn.json", "max_seq_length": 128, "max_predictions_per_seq": 20, "train_batch_size": 160,
"learning_rate": 3.125e-05, "num_warmup_steps": 100, "num_train_steps": 1000, "optimizer_type": "adam",
"manual_fp16": true, "use_fp16_cls": true, "input_files_dir": "/home/BertData/cn-wiki-128/", "eval_files_dir":
"/home/BertData/cn-wiki-128/", "npu_bert_debug": false, "npu_bert_use_tdt": true, "distributed": true,
"do_train": true, "do_eval": true, "num_accumulation_steps": 1, "iterations_per_loop": 100,
"npu_bert_loss_scale": 0, "save_checkpoints_steps": 1000, "npu_bert_clip_by_global_norm": false,
"device_group_1p": 0, "device_group_2p": "0 1", "device_group_4p": "0 1 2 3", "PROFILING_MODE": false,
"AICPU_PROFILING_MODE": false, "PROFILING_OPTIONS": "training_trace", "FP_POINT": "bert/embeddings/
GatherV2", "BP_POINT": "gradients/bert/embeddings/IdentityN_1_grad/UnsortedSegmentSum"}
:::ABK 1.0.0 bert 2020-09-18 16:43:26.839171 (run_pretraining.py:775) base_lr: 0.01
:::ABK 1.0.0 bert 2020-09-18 16:43:26.839409 (run_pretraining.py:776) dataset: "cn-clue/en-wiki"
:::ABK 1.0.0 bert 2020-09-18 16:43:26.839633 (run_pretraining.py:777) opt_name: "Adam"
:::ABK 1.0.0 bert 2020-09-18 16:43:26.839862 (run_pretraining.py:778) loss_scale: 512
:::ABK 1.0.0 bert 2020-09-18 16:45:09.928856 (run_pretraining.py:237) current_step: " 79"
:::ABK 1.0.0 bert 2020-09-18 16:45:09.930988 (run_pretraining.py:238) throwout: " 2589.4"
:::ABK 1.0.0 bert 2020-09-18 16:45:20.475623 (run_pretraining.py:237) current_step: " 179"
:::ABK 1.0.0 bert 2020-09-18 16:45:20.477533 (run_pretraining.py:238) throwout: " 12146.9"
:::ABK 1.0.0 bert 2020-09-18 16:45:31.015402 (run_pretraining.py:237) current_step: " 279"
:::ABK 1.0.0 bert 2020-09-18 16:45:31.016302 (run_pretraining.py:238) throwout: " 12151.5"
:::ABK 1.0.0 bert 2020-09-18 16:45:41.583725 (run_pretraining.py:237) current_step: " 379"
:::ABK 1.0.0 bert 2020-09-18 16:45:41.585642 (run_pretraining.py:238) throwout: " 12118.8"
:::ABK 1.0.0 bert 2020-09-18 16:45:52.133378 (run_pretraining.py:237) current_step: " 479"
:::ABK 1.0.0 bert 2020-09-18 16:45:52.134726 (run_pretraining.py:238) throwout: " 12140.9"
:::ABK 1.0.0 bert 2020-09-18 16:46:02.689664 (run_pretraining.py:237) current_step: " 579"
:::ABK 1.0.0 bert 2020-09-18 16:46:02.690570 (run_pretraining.py:238) throwout: " 12131.7"
:::ABK 1.0.0 bert 2020-09-18 16:46:13.252272 (run_pretraining.py:237) current_step: " 679"
:::ABK 1.0.0 bert 2020-09-18 16:46:13.254195 (run_pretraining.py:238) throwout: " 12125.2"
:::ABK 1.0.0 bert 2020-09-18 16:46:23.798265 (run_pretraining.py:237) current_step: " 779"
:::ABK 1.0.0 bert 2020-09-18 16:46:23.800281 (run_pretraining.py:238) throwout: " 12146.1"
:::ABK 1.0.0 bert 2020-09-18 16:46:34.356962 (run_pretraining.py:237) current_step: " 879"
:::ABK 1.0.0 bert 2020-09-18 16:46:34.358889 (run_pretraining.py:238) throwout: " 12132.1"
:::ABK 1.0.0 bert 2020-09-18 16:46:44.910931 (run_pretraining.py:237) current_step: " 979"
:::ABK 1.0.0 bert 2020-09-18 16:46:44.912744 (run_pretraining.py:238) throwout: " 12136.0"
:::ABK 1.0.0 bert 2020-09-18 16:46:55.463024 (run_pretraining.py:237) current_step: " 1077"
:::ABK 1.0.0 bert 2020-09-18 16:46:55.464920 (run_pretraining.py:238) throwout: " 12138.5"
:::ABK 1.0.0 bert 2020-09-18 16:47:33.190724 (run_pretraining.py:748) global_step: "1076"
:::ABK 1.0.0 bert 2020-09-18 16:47:33.191398 (run_pretraining.py:750) loss: "7.067054"
:::ABK 1.0.0 bert 2020-09-18 16:47:33.191961 (run_pretraining.py:744) masked_lm_accuracy: "0.11443632"
:::ABK 1.0.0 bert 2020-09-18 16:47:33.192522 (run_pretraining.py:752) masked_lm_loss: "6.4371023"
:::ABK 1.0.0 bert train success
:::ABK 1.0.0 bert train total time 0:4:12
```

----结束

以Host单机多卡场景下启动框架为TensorFlow、模型为YOLOv3训练为例：

步骤1 修改“/home/train/yaml/”目录下YOLOv3.yaml文件中的**tensorflow_config**参数。

tensorflow_config:

- **data_url:** /opt/npu/dataset
- **runmode:** train

步骤2 在train目录下执行如下命令运行benchmark工具，启动训练。

```
bash benchmark.sh -e YOLOv3 -hw 8p
```

若显示如下所示的信息，表示YOLOv3模型训练成功。

```
find script path success
run train script file path is /home/train/atlas_benchmark-master/object_detection/yolov3/tensorflow/scripts/
run.sh
[20200922-14:29:18] [INFO] train job is working, wait more 5s
[20200922-14:29:18] [INFO] /home/train/result/tf_yolov3/training_job_20200922142918/ &
```

```
[20200922-14:29:23] [INFO] train job is working, wait more 5s
[20200922-14:29:29] [INFO] train job is working, wait more 5s
[20200922-14:29:34] [INFO] train job is working, wait more 5s
...
[20200922-14:37:42] [INFO] train job is working, wait more 5s
:::ABK 1.0.0 yolov3 train success
0:8:28
[20200922-14:37:48] [INFO] train job is working, wait more 5s
:::ABK 1.0.0 yolov3 train success
0:8:29
:::ABK 1.0.0 yolov3 train success
0:8:31
:::ABK 1.0.0 yolov3 train success
0:8:31
:::ABK 1.0.0 yolov3 train success
0:8:33
:::ABK 1.0.0 yolov3 train success
0:8:33
[20200922-14:37:53] [INFO] train job is working, wait more 5s
:::ABK 1.0.0 yolov3 train success
0:8:34
:::ABK 1.0.0 yolov3 train success
0:8:35
[20200922-14:37:58] [INFO] process end
```

通过查看 “/home/train/result/tf_yolov3/training_job_20200922142918/” 下的日志文件hw_yolov3.log，可以看到训练结果、训练时长、吞吐率（fps）等信息。显示示例如下：

```
:::ABK 1.0.0 yolov3 2020-09-22 14:29:21.367563 (train.py:40) cpu_info: {"Architecture": "x86_64CPU", "op-
mode(s)": "32-bit,64-bitByte", "Order": "LittleEndian", "CPU(s)": "0-19,40-59NUMAnode1", "list":
"0-79Thread(s)per", "core": "2Core(s)per", "socket": "20", "Socket(s)": "2NUMA", "node(s)": "2Vendor",
"ID": "GenuineIntelCPU", "family": "6", "Model": "85Model", "name":
"Intel(R)Xeon(R)Gold6148CPU@2.40GHz", "Stepping": "4CPU", "MHz": "1000.0000", "BogoMIPS":
"4800.00", "Virtualization": "VT-xL1d", "cache": "28160KNUMAnode0"}
:::ABK 1.0.0 yolov3 2020-09-22 14:29:21.368202 (train.py:41) npu_info: "Ascend910"
:::ABK 1.0.0 yolov3 2020-09-22 14:29:21.368398 (train.py:42) os_info: "Linux version
3.10.0-1127.13.1.el7.x86_64 (mockbuild@kbuilder.bsys.centos.org) (gcc version 4.8.5 20150623 (Red Hat
4.8.5-39) (GCC) ) #1 SMP Tue Jun 23 15:46:38 UTC 2020"
:::ABK 1.0.0 yolov3 2020-09-22 14:29:21.368576 (train.py:43) framework_info: "tensorflow 1.15.0"
:::ABK 1.0.0 yolov3 2020-09-22 14:29:21.368750 (train.py:44) benchmark_version: "v1.0.0"
:::ABK 1.0.0 yolov3 2020-09-22 14:29:21.368962 (train.py:45) config_info: {"mode": "single", "data_url":
"/opt/npu/dataset", "runmode": "train", "ckpt_path": "/train/benchmark-master720/train/atlas_benchmark-
master/object_detection/yolov3/tensorflow/result/TrainingJob-20200724115042", "total_epoches": 1,
"save_epoch": 3, "batch_size": 32, "device_group_1p": 0, "device_group_2p": "0 1", "device_group_4p": "0 1 2
3", "profiling_mode": false, "profiling_options": "training_trace:task_trace", "fp_point": "yolov3/
darknet53_body/Conv/Conv2D", "bp_point": "cond_1/Momentum/update_yolov3/yolov3_head/Conv_9/
weights/ApplyMomentum", "aicpu_profiling_mode": false}
:::ABK 1.0.0 yolov3 2020-09-22 14:29:21.369162 (train.py:46) base_lr: 0.128
:::ABK 1.0.0 yolov3 2020-09-22 14:29:21.369339 (train.py:47) dataset: "coco1024"
:::ABK 1.0.0 yolov3 2020-09-22 14:29:21.369514 (train.py:48) opt_name: "Adam"
:::ABK 1.0.0 yolov3 2020-09-22 14:29:21.369688 (train.py:49) loss_scale: 512
:::ABK 1.0.0 yolov3 2020-09-22 14:29:21.369864 (train.py:50) input_batch_size: 32
:::ABK 1.0.0 yolov3 2020-09-22 14:29:58.904349 (train.py:239) total_train_epoch: "1"
:::ABK 1.0.0 yolov3 2020-09-22 14:33:35.713389 (train.py:269) fps: "12.408741465186743"
:::ABK 1.0.0 yolov3 2020-09-22 14:33:35.713955 (train.py:270) global_step: "10"
:::ABK 1.0.0 yolov3 2020-09-22 14:33:40.910602 (train.py:269) fps: "492.84705325937443"
:::ABK 1.0.0 yolov3 2020-09-22 14:33:40.923378 (train.py:270) global_step: "20"
:::ABK 1.0.0 yolov3 2020-09-22 14:33:48.302835 (train.py:269) fps: "347.98206910498493"
:::ABK 1.0.0 yolov3 2020-09-22 14:33:48.307875 (train.py:270) global_step: "30"
...
:::ABK 1.0.0 yolov3 2020-09-22 14:37:30.356924 (train.py:269) fps: "556.3515907683327"
:::ABK 1.0.0 yolov3 2020-09-22 14:37:30.369520 (train.py:270) global_step: "450"
:::ABK 1.0.0 yolov3 train success
:::ABK 1.0.0 yolov3 train total time 0:8:33
```

如需对YoLoV3模型进行精度评测，还要修改YoLoV3.yaml文件中的如下参数，并再次执行**bash benchmark.sh -e YoLoV3 -hw 8p**命令运行benchmark工具。

- **runmode**: 修改为evaluate
- **ckpt_path**: 修改为训练生成的日志路径 “/home/train/result/tf_yolov3/training_job_20200922142918/”

运行成功后，通过查看 “/home/train/result/tf_yolov3/training_job_20200922142918/” 路径下的eval_`\${deviceid}`.out文件，可以看到精度信息。例如：

```
Average Precision (AP) @[ IoU=0.75 | area= all | maxDets=100 ] = 0.315
Average Precision (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.142
Average Precision (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.344
Average Precision (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.465
Average Recall (AR) @[ IoU=0.50:0.95 | area= all | maxDets= 1 ] = 0.277
Average Recall (AR) @[ IoU=0.50:0.95 | area= all | maxDets= 10 ] = 0.447
Average Recall (AR) @[ IoU=0.50:0.95 | area= all | maxDets=100 ] = 0.491
Average Recall (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.289
Average Recall (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.546
Average Recall (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.663
```

----结束

以Host单机单卡场景下启动框架为Pytorch、模型为ResNet50训练为例：

步骤1 修改 “/home/train/yaml/” 目录下ResNet50.yaml文件中的**pytorch_config**配置参数。

pytorch_config:

- **data_url**: /home/imagenet/
- **mode**: train_and_evaluate
- **device_group_1p**: 0

步骤2 在train目录下执行如下命令运行benchmark工具，启动训练。

./benchmark.sh -e ResNet50 -hw 1p -f pytorch

若显示如下所示的信息，表示ResNet50模型训练成功。

```
find script path success
run train script file path is /home/train/atlas_benchmark-master/image_classification/ResNet50/pytorch/
scripts/run.sh
[20200929-11:59:09] [INFO] /home/train/result/pt_resnet50/training_job_20200929115909/ &
[20200929-11:59:09] [INFO] train job is working, wait more 5s
device group is 0
rank size is 1
1p
[20200929-11:59:14] [INFO] train job is working, wait more 5s
[20200929-11:59:19] [INFO] train job is working, wait more 5s
...
[20200907-07:08:43] [INFO] train job is working, wait more 5s
:::ABK 1.0.0 resnet50 train success
:::ABK 1.0.0 resnet50 train total time: 0:46:40
[20200907-07:08:48] [INFO] process end
```

通过查看 “/home/train/result/pt_resnet50/training_job_20200929115909” 下的日志文件hw_resnet50.log，可以看到训练结果、训练时长、吞吐率（fps）等信息。显示示例如下：

```
:::ABK 1.0.0 resnet50 2020-09-29 11:59:10.750179 (pytorch-resnet50-apex.py:600) cpu_info: {"Architecture":
"aarch64Byte", "Order": "LittleEndian", "CPU(s)": "144-167NUMAnode7", "list": "0-191Thread(s)per", "core":
"1Core(s)per", "socket": "48", "Socket(s)": "4NUMA", "node(s)": "8Vendor", "ID": "0x48", "Model": "0",
"Stepping": "0x1CPUmax", "MHz": "200.0000", "BogoMIPS": "200.00L1d", "cache": "49152KNUMAnode0"}
:::ABK 1.0.0 resnet50 2020-09-29 11:59:10.750917 (pytorch-resnet50-apex.py:601) npu_info: "Ascend910"
:::ABK 1.0.0 resnet50 2020-09-29 11:59:10.751246 (pytorch-resnet50-apex.py:602) os_info: "Linux version
```



```
4.15.0-55-generic (buildd@bos02-arm64-065) (gcc version 7.4.0 (Ubuntu/Linaro 7.4.0-1ubuntu1~18.04.1))
#60-Ubuntu SMP Tue Jul 2 18:23:38 UTC 2019"
:::ABK 1.0.0 resnet50 2020-09-29 11:59:10.751568 (pytorch-resnet50-apex.py:603) framework_info: "pytorch
1.5.0+ascend"
:::ABK 1.0.0 resnet50 2020-09-29 11:59:10.751878 (pytorch-resnet50-apex.py:604) benchmark_version:
"v1.0.0"
:::ABK 1.0.0 resnet50 2020-09-29 11:59:10.752214 (pytorch-resnet50-apex.py:605) config_info: {"data_url": "/
home/imagenet/", "batch_size": 512, "epoche": 90, "mode": "train_and_evaluate", "ckpt_path": "/home/
train/result/pt_resnet50/training_job_20200916042624/7/checkpoint_npu7model_best.pth.tar", "lr": 0.2}
:::ABK 1.0.0 resnet50 2020-09-29 11:59:10.752548 (pytorch-resnet50-apex.py:606) base_lr: 0.1
:::ABK 1.0.0 resnet50 2020-09-29 11:59:10.752858 (pytorch-resnet50-apex.py:607) dataset: "imagenet"
:::ABK 1.0.0 resnet50 2020-09-29 11:59:10.753166 (pytorch-resnet50-apex.py:608) opt_name: "SGD"
:::ABK 1.0.0 resnet50 2020-09-29 11:59:10.753480 (pytorch-resnet50-apex.py:609) loss_scale: 1024
:::ABK 1.0.0 resnet50 2020-09-29 12:04:43.613476 (pytorch-resnet50-apex.py:514) fps: 2.837303896305409
:::ABK 1.0.0 resnet50 2020-09-29 12:04:43.980156 (pytorch-resnet50-apex.py:514) fps: 7.040413624300427
:::ABK 1.0.0 resnet50 2020-09-29 12:04:55.278455 (pytorch-resnet50-apex.py:514) fps: 1398.9071038251366
:::ABK 1.0.0 resnet50 2020-09-29 12:04:55.644022 (pytorch-resnet50-apex.py:514) fps: 45.31775535493008
:::ABK 1.0.0 resnet50 2020-09-29 12:04:56.068004 (pytorch-resnet50-apex.py:514) fps: 1398.9071038251366
...
:::ABK 1.0.0 resnet50 train success
:::ABK 1.0.0 resnet50 train total time: 1:46:40
```

----结束

3.4.2 Docker 单机训练场景

前提条件

请参照[3.3 运行工具](#)章节完成运行工具前的准备工作。包括：

- 安装运行环境，具体安装方法请参见《CANN 软件安装指南》中的“安装运行环境（训练）> 容器安装”章节。
- 参照benchmark工具中各模型的README文件做好训练前准备工作。
- 获取测试模型对应的数据集并上传到服务器。
- 获取benchmark工具包并上传到服务器，解压benchmark工具包，并增加权限。
- 确保环境变量与环境中安装的CANN软件包路径一致。

使用实例

以Docker单机单卡场景下启动框架为TensorFlow、模型为ResNet50训练为例：

步骤1 修改“/home/train/yaml/”目录下ResNet50.yaml文件中的**tensorflow_config**参数。

tensorflow_config:

- **data_url:** “/home/dataset/imagenet_TF/”
- **docker_image:** ubuntu:b030
- **device_group_1p:** 0

步骤2 在train目录下执行如下命令运行benchmark工具，启动训练。例如：

bash benchmark.sh -e ResNet50 -hw 1p -docker

若显示如下所示的信息，表示ResNet50模型训练成功。

```
find script path success
run train script file path is /home/train/atlas_benchmark-master/image_classification/ResNet50/tensorflow/
scripts/run.sh
[20200916-03:22:06] [INFO] train job is working, wait more 5s
[20200916-07:22:08] [INFO] /home/train/result/tf_resnet50/training_job_20200916072208/ &
```

```
start to modify inner config file
modify inner config file success
[20200907-09:04:42] [INFO] train job is working, wait more 5s
[20200907-09:04:47] [INFO] train job is working, wait more 5s
...
[20200907-09:54:23] [INFO] train job is working, wait more 5s
:::ABK 1.0.0 resnet50 train success
:::ABK 1.0.0 resnet50 train total time: 0:49:43
[20200907-09:54:28] [INFO] process end
```

通过查看 “/home/train/result/tf_resnet50/training_job_20200916072208” 下的日志文件hw_resnet50.log，可以看到训练结果、训练时长、吞吐率（fps）和精度（eval_accuracy_top1和eval_accuracy_top5）等信息。显示示例如下：

```
:::ABK 1.0.0 resnet50 2020-09-07 13:04:59.893007 (resnet_tf_r1_benchmark.py:235) cpu_info:
{"Architecture": "x86_64CPU", "op-mode(s)": "32-bit,64-bitByte", "Order": "LittleEndi an", "CPU(s)":
"0-23,48-71NUMAnode1", "list": "0-95Thread(s)per", "core": "2Core(s)per", "socket": "24", "Socket(s)":
"2NUMA", "node(s)": "2Vendor", "ID": "GenuineIntelCPU", "family": "6", "Model": "85Model", "name":
"Intel(R)Xeon(R)Platinum8268CPU@2.90GHz", "Stepping": "5CPU", "MHz": "1000.0000", "BogoMIPS":
"5800.00", "Virtualization": "VT-xL1d", " cache": "33792KNUMAnode0"}
:::ABK 1.0.0 resnet50 2020-09-07 13:04:59.893396 (resnet_tf_r1_benchmark.py:236) npu_info: "Ascend910"
:::ABK 1.0.0 resnet50 2020-09-07 13:04:59.893668 (resnet_tf_r1_benchmark.py:237) os_info: "Linux version
4.19.28 (root@debian) (gcc version 6.3.0 20170516 (Debian 6.3.0-18+deb9 u1)) #1 SMP Sat Jun 27
20:20:14 EDT 2020"
:::ABK 1.0.0 resnet50 2020-09-07 13:04:59.893928 (resnet_tf_r1_benchmark.py:238) framework_info:
"tensorflow 1.15.0"
:::ABK 1.0.0 resnet50 2020-09-07 13:04:59.894183 (resnet_tf_r1_benchmark.py:239) benchmark_version:
"v1.0.0"
:::ABK 1.0.0 resnet50 2020-09-07 13:04:59.894450 (resnet_tf_r1_benchmark.py:240) config_info: {"data_url":
"/home/data/imagenet_TF/", "batch_size": 32, "epoche": 90, "max_train_steps": 1000,
"epochs_between_evals": 5, "iterations_per_loop": 100, "save_checkpoints_steps": 115200,
"device_group_1p": 0, "device_group_2p": "0 1", "device_group_4p": "0 1 2 3"}
:::ABK 1.0.0 resnet50 2020-09-07 13:04:59.894716 (resnet_tf_r1_benchmark.py:241) base_lr: 0.128
:::ABK 1.0.0 resnet50 2020-09-07 13:04:59.894973 (resnet_tf_r1_benchmark.py:242) dataset: "imagenet1024"
:::ABK 1.0.0 resnet50 2020-09-07 13:04:59.895226 (resnet_tf_r1_benchmark.py:243) opt_name: "SGD"
:::ABK 1.0.0 resnet50 2020-09-07 13:04:59.895483 (resnet_tf_r1_benchmark.py:244) loss_scale: 512
:::ABK 1.0.0 resnet50 2020-09-07 13:04:59.895739 (resnet_tf_r1_benchmark.py:245) input_batch_size: 32
:::ABK 1.0.0 resnet50 2020-09-07 13:05:33.782576 (resnet_run_loop.py:817) current_epoch: 5
:::ABK 1.0.0 resnet50 2020-09-07 13:06:54.945292 (hooks.py:146) fps: 1544.8800666121388
:::ABK 1.0.0 resnet50 2020-09-07 13:06:57.017517 (hooks.py:146) fps: 1415.8806648316802
:::ABK 1.0.0 resnet50 2020-09-07 13:06:59.087315 (hooks.py:146) fps: 1547.1837712407075
:::ABK 1.0.0 resnet50 2020-09-07 13:07:07.474523 (hooks.py:146) fps: 381.5333808130886
:::ABK 1.0.0 resnet50 2020-09-07 13:07:15.689626 (hooks.py:146) fps: 389.54121878487666
:::ABK 1.0.0 resnet50 2020-09-07 13:07:24.280087 (hooks.py:146) fps: 372.47881660849606
:::ABK 1.0.0 resnet50 2020-09-07 13:07:32.729384 (hooks.py:146) fps: 378.74249606196514
:::ABK 1.0.0 resnet50 2020-09-07 13:07:41.472380 (hooks.py:146) fps: 365.9875884968988
:::ABK 1.0.0 resnet50 2020-09-07 13:07:49.544516 (hooks.py:146) fps: 396.39799197479834
:::ABK 1.0.0 resnet50 2020-09-07 13:10:47.113701 (resnet_run_loop.py:858) eval_accuracy_top1:
0.0013795585837215185
:::ABK 1.0.0 resnet50 2020-09-07 13:10:47.114221 (resnet_run_loop.py:859) eval_accuracy_top5:
0.005578214768320322
```

----结束

以Docker单机多卡场景下启动框架为TensorFlow、模型为Bert-Base训练为例：

步骤1 修改 “/home/train/yaml/” 目录下Bert-Base.yaml文件中的**tensorflow_config**参数。

tensorflow_config:

- **input_files_dir:** /home/BertData/cn-clue-256/training/
- **eval_files_dir:** /home/BertData/cn-clue-256/test/
- **docker_image:** ubuntu:b030
- **device_group_2p:** 0 1

步骤2 在train目录下执行如下命令运行benchmark工具，启动训练。

```
bash benckmark.sh -e Bert-Base -hw 2p -docker
```

若显示如下所示的信息，表示Bert-Base模型训练成功。

```
find script path success
run train script file path is /home/train/atlas_benchmark-master/nlp/Bert-Base/tensorflow/scripts/run.sh
[20200922-16:24:32] [INFO] see more config info in /home/train/atlas_benchmark-master/nlp/Bert-Base/
tensorflow/config
[20200922-16:24:32] [INFO] train result in /home/train/result/tf_bert_base/training_job_20200922162432/
[20200922-16:24:32] [INFO] train job is working, wait more 5s
[20200922-16:24:37] [INFO] train job is working, wait more 5s
[20200922-16:24:42] [INFO] train job is working, wait more 5s
...
[20200907-08:18:13] [INFO] train job is working, wait more 5s
:::ABK 1.0.0 bert train success
:::ABK 1.0.0 bert train total time 0:10:54
:::ABK 1.0.0 bert train success
:::ABK 1.0.0 bert train total time 0:10:54
[20200907-08:18:18] [INFO] process end
```

通过查看 “/home/train/result/tf_bert_base/training_job_20200922162432” 下的日志文件hw_bert.log，可以看到训练结果、训练时长、吞吐率（throwout）和精度（masked_lm_accuracy）等信息。显示示例如下：

```
:::ABK 1.0.0 bert 2020-09-22 16:24:37.084792 (run_pretraining.py:769) cpu_info: {"Architecture":
"x86_64CPU", "op-mode(s)": "32-bit,64-bitByte", "Order": "LittleEndian", "CPU(s)":
"0-19,40-59NUMANode1", "list": "0-79Thread(s)per", "core": "2Core(s)per", "socket": "20", "Socket(s)":
"2NUMA", "node(s)": "2Vendor", "ID": "GenuineIntelCPU", "family": "6", "Model": "85Model", "name":
"Intel(R)Xeon(R)Gold6148CPU@2.40GHz", "Stepping": "4CPU", "MHz": "1000.0000", "BogoMIPS":
"4800.00", "Virtualization": "VT-xL1d", "cache": "28160KNUMANode0"}
:::ABK 1.0.0 bert 2020-09-22 16:24:37.085350 (run_pretraining.py:770) npu_info: "Ascend910"
:::ABK 1.0.0 bert 2020-09-22 16:24:37.085667 (run_pretraining.py:771) os_info: "Linux version
3.10.0-1127.13.1.el7.x86_64 (mockbuild@kbuilder.bsys.centos.org) (gcc version 4.8.5 20150623 (Red Hat
4.8.5-39) (GCC) ) #1 SMP Tue Jun 23 15:46:38 UTC 2020"
:::ABK 1.0.0 bert 2020-09-22 16:24:37.085966 (run_pretraining.py:772) framework_info: "tensorflow 1.15.0"
:::ABK 1.0.0 bert 2020-09-22 16:24:37.086266 (run_pretraining.py:773) benchmark_version: "v1.0.0"
:::ABK 1.0.0 bert 2020-09-22 16:24:37.086601 (run_pretraining.py:774) config_info: {"bert_config_file":
"bert_base_layer6_cn.json", "max_seq_length": 128, "max_predictions_per_seq": 20, "train_batch_size": 160,
"learning_rate": 3.125e-05, "num_warmup_steps": 100, "num_train_steps": 1000, "optimizer_type": "adam",
"manual_fp16": true, "use_fp16_cls": true, "input_files_dir": "/home/BertData/cn-wiki-128/", "eval_files_dir":
"/home/BertData/cn-wiki-128/", "npu_bert_debug": false, "npu_bert_use_tdt": true, "distributed": true,
"do_train": true, "do_eval": true, "num_accumulation_steps": 1, "iterations_per_loop": 100,
"npu_bert_loss_scale": 0, "save_checkpoints_steps": 1000, "npu_bert_clip_by_global_norm": false,
"device_group_1p": 0, "device_group_2p": "0 1", "device_group_4p": "0 1 2 3", "PROFILING_MODE": false,
"AICPU_PROFILING_MODE": false, "PROFILING_OPTIONS": "training_trace", "FP_POINT": "bert/embeddings/
GatherV2", "BP_POINT": "gradients/bert/embeddings/IdentityN_1_grad/UnsortedSegmentSum"}
:::ABK 1.0.0 bert 2020-09-18 16:43:26.839171 (run_pretraining.py:775) base_lr: 0.01
:::ABK 1.0.0 bert 2020-09-18 16:43:26.839409 (run_pretraining.py:776) dataset: "cn-clue/en-wiki"
:::ABK 1.0.0 bert 2020-09-18 16:43:26.839633 (run_pretraining.py:777) opt_name: "Adam"
:::ABK 1.0.0 bert 2020-09-18 16:43:26.839862 (run_pretraining.py:778) loss_scale: 512
:::ABK 1.0.0 bert 2020-09-18 16:45:09.928856 (run_pretraining.py:237) current_step: " 79"
:::ABK 1.0.0 bert 2020-09-18 16:45:09.930988 (run_pretraining.py:238) throwout: " 2589.4"
:::ABK 1.0.0 bert 2020-09-18 16:45:20.475623 (run_pretraining.py:237) current_step: " 179"
:::ABK 1.0.0 bert 2020-09-18 16:45:20.477533 (run_pretraining.py:238) throwout: " 12146.9"
:::ABK 1.0.0 bert 2020-09-18 16:45:31.015402 (run_pretraining.py:237) current_step: " 279"
:::ABK 1.0.0 bert 2020-09-18 16:45:31.016302 (run_pretraining.py:238) throwout: " 12151.5"
:::ABK 1.0.0 bert 2020-09-18 16:45:41.583725 (run_pretraining.py:237) current_step: " 379"
:::ABK 1.0.0 bert 2020-09-18 16:45:41.585642 (run_pretraining.py:238) throwout: " 12118.8"
:::ABK 1.0.0 bert 2020-09-18 16:45:52.133378 (run_pretraining.py:237) current_step: " 479"
:::ABK 1.0.0 bert 2020-09-18 16:45:52.134726 (run_pretraining.py:238) throwout: " 12140.9"
:::ABK 1.0.0 bert 2020-09-18 16:46:02.689664 (run_pretraining.py:237) current_step: " 579"
:::ABK 1.0.0 bert 2020-09-18 16:46:02.690570 (run_pretraining.py:238) throwout: " 12131.7"
:::ABK 1.0.0 bert 2020-09-18 16:46:13.252272 (run_pretraining.py:237) current_step: " 679"
:::ABK 1.0.0 bert 2020-09-18 16:46:13.254195 (run_pretraining.py:238) throwout: " 12125.2"
:::ABK 1.0.0 bert 2020-09-18 16:46:23.798265 (run_pretraining.py:237) current_step: " 779"
:::ABK 1.0.0 bert 2020-09-18 16:46:23.800281 (run_pretraining.py:238) throwout: " 12146.1"
:::ABK 1.0.0 bert 2020-09-18 16:46:34.356962 (run_pretraining.py:237) current_step: " 879"
```

```
...ABK 1.0.0 bert 2020-09-18 16:46:34.358889 (run_pretraining.py:238) throwout: " 12132.1"
...ABK 1.0.0 bert 2020-09-18 16:46:44.910931 (run_pretraining.py:237) current_step: " 979"
...ABK 1.0.0 bert 2020-09-18 16:46:44.912744 (run_pretraining.py:238) throwout: " 12136.0"
...ABK 1.0.0 bert 2020-09-18 16:46:55.463024 (run_pretraining.py:237) current_step: " 1077"
...ABK 1.0.0 bert 2020-09-18 16:46:55.464920 (run_pretraining.py:238) throwout: " 12138.5"
...ABK 1.0.0 bert 2020-09-18 16:47:33.190724 (run_pretraining.py:748) global_step: "1076"
...ABK 1.0.0 bert 2020-09-18 16:47:33.191398 (run_pretraining.py:750) loss: "7.067054"
...ABK 1.0.0 bert 2020-09-18 16:47:33.191961 (run_pretraining.py:744) masked_lm_accuracy: "0.11443632"
...ABK 1.0.0 bert 2020-09-18 16:47:33.192522 (run_pretraining.py:752) masked_lm_loss: "6.4371023"
...ABK 1.0.0 bert train success
...ABK 1.0.0 bert train total time 0:4:12
```

----结束

3.4.3 Host 集群训练场景（TensorFlow）

3.4.3.1 部署前准备

部署Host集群训练环境需要准备如下工作：

- 多台训练服务器Device使用交换机连接。
- 多台训练服务器的Device网卡IP地址在同一网段内。
- 保证多台训练服务器环境配置相同（例如操作系统、CPU架构和CANN软件包版本等），且benchmark运行目录绝对路径相同。
- 在多台训练服务器上分别参照[3.3 运行工具](#)章节完成运行工具前的准备工作。包括：
 - 安装运行环境，具体安装方法请参见《CANN 软件安装指南》中的“安装运行环境（训练）> 物理机安装”章节。
 - 参照benchmark工具中各模型的README文件做好训练前准备工作。
 - 获取测试模型对应的数据集并上传到服务器。
 - 获取benchmark工具包并上传到服务器，解压benchmark工具包，并增加权限。
 - 确保环境变量与环境中安装的CANN软件包路径一致。

3.4.3.2 安装 OpenMPI 开源库

多机多卡环境下分布式训练部署依赖于OpenMPI开源库，参与模型训练的每台服务器都需要安装OpenMPI开源库。目前OpenMPI开源库的版本要求为4.0.1、4.0.2或4.0.3。

执行**mpirun --version**命令检查是否已安装OpenMPI。如果返回如下信息，说明已经安装。如果已安装，且版本为4.0.1、4.0.2或4.0.3其中一个，则无需再安装。

```
root@ubuntu:~# mpirun --version
mpirun (Open MPI) 4.0.2
```

Report bugs to <http://www.open-mpi.org/community/help/>

否则请按照如下步骤安装OpenMPI：

步骤1 访问如下链接下载OpenMPI软件包。例如**openmpi-4.0.2.tar.bz2**。

<https://www.open-mpi.org/software/ompi/v4.0/>

步骤2 以root用户登录安装环境。

步骤3 将下载的OpenMPI软件包上传到安装环境的某一目录下。

步骤4 进入软件包所在目录，执行如下命令解压软件包。

```
tar -jxvf openmpi-4.0.2.tar.bz2
```

步骤5 进入解压后的目录，执行如下命令配置、编译和安装。

```
./configure --prefix=/usr/local/mpirun4.0
```

```
make
```

```
make install
```

其中“--prefix”参数用于指定OpenMPI安装路径，用户根据实际情况进行修改。

步骤6 执行`vi ~/.bashrc`命令，打开`.bashrc`文件，在文件的最后添加如下环境变量。

```
export OPENMPI=/usr/local/mpirun4.0
export LD_LIBRARY_PATH=$OPENMPI/lib
export PATH=$OPENMPI/bin:$PATH
```

其中“/usr/local/mpirun4.0”为OpenMPI安装路径，用户需要根据实际进行修改。

执行`wq!`命令保存文件并退出。

步骤7 执行如下命令使环境变量生效。

```
source ~/.bashrc
```

步骤8 安装完成之后，执行如下命令查看安装版本，如果返回正确的版本信息，则说明安装成功。

```
mpirun --version
```

----结束

3.4.3.3 配置服务器 SSH 免密登录

如果使用OpenMPI在多机多卡环境下分布式训练部署，需要在每两台服务器之间配置SSH免密登录，确保服务器之间可以互通。具体操作步骤如下：

步骤1 以root用户登录每台服务器。

步骤2 在每台服务器上分别执行如下命令打开`/etc/hosts`文件，并在该文件中添加本服务器对应的IP地址和主机名。如果文件中已添加，则跳过此步骤。

```
vi /etc/hosts
```

添加内容示例如下：

```
10.90.140.199 ubuntu
```

其中10.90.140.199为该服务器的IP地址，ubuntu为主机名。

步骤3 在第一台服务器执行如下命令生成公钥（例如第一台服务器IP为10.90.140.199）。

```
cd ~/.ssh/ # 若没有该目录，请先执行一次ssh localhost
ssh-keygen -t rsa # 生成秘钥，会出现提示，连续点击3次"Enter"即可
mv id_rsa.pub authorized_keys # 将生成的秘钥id_rsa.pub重命名为公钥authorized_keys
```

步骤4 在其他每台服务器上生成秘钥，并复制到第一台服务器的`authorized_keys`文件中。

1. 在其他每台服务器上执行如下命令生成秘钥。

```
cd ~/.ssh/
ssh-keygen -t rsa
```

2. 将其他每台服务器上生成的密钥文件id_rsa.pub下载到本地，并复制该文件中的密钥。
3. 在第一台服务器内执行如下命令打开公钥文件authorized_keys，将步骤4.2复制的其他每台服务器的密钥粘贴到第一台服务器的公钥后面。

```
vi ~/.ssh/authorized_keys
```

执行:wq!保存该文件。

步骤5 在其他每台服务器上执行如下命令将第一台服务器制作好的公钥复制到其他每台服务器内。

```
cd ~/.ssh/  
scp root@10.90.140.199:~/.ssh/authorized_keys ./
```

步骤6 在每台服务器执行如下命令测试免密登录。

ssh 用户名@IP地址

例如：在第一台服务器10.90.140.199免密登录服务器10.90.140.231，执行**ssh root@10.90.140.231**命令。

若显示类似如下信息，说明已免密登录服务器10.90.140.231。

```
root@ubuntu:~/.ssh# ssh root@10.90.140.231  
Linux ubuntu 4.19.28 #1 SMP Tue Jun 23 19:05:23 EDT 2020 x86_64  
  
The programs included with the ubuntu GNU/Linux system are free software;  
the exact distribution terms for each program are described in the  
individual files in /usr/share/doc/*/copyright.  
  
ubuntu GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent  
permitted by applicable law.  
Last login: Tue Sep 15 14:37:21 2020 from 10.254.88.75  
root@ubuntu:~#
```

执行**exit**命令可以退出登录，若显示类似如下信息，说明已退出登录。

```
root@ubuntu:~# exit  
logout  
Connection to 10.90.140.231 closed.
```

----结束

3.4.3.4 测试运行环境

测试一台服务器的运行环境即可，具体操作步骤如下：

步骤1 进入如下链接下载源码，例如AtlasBoost-master.tar.gz。

<https://gitee.com/ascend/atlasboost>

步骤2 以root用户登录服务器。

步骤3 将获取的源码上传到服务器的某一目录下。

步骤4 进入源码所在目录，执行如下命令解压源码。

tar -zxvf AtlasBoost-master.tar.gz

步骤5 执行如下命令将解压后的AtlasBoost-master文件夹下的所有文件拷贝到“*benchmark工具所在路径/train/atlas_benchmark-master/*utils/atlasboost/”目录下。

cp -r AtlasBoost-master/* *benchmark工具所在路径/train/atlas_benchmark-master/*utils/atlasboost/

步骤6 进入build目录，执行如下命令重新编译AtlasBoost。

须知

- 编译AtlasBoost要求cmake版本为3.13及以上。
- 在每台服务器上都需要重新编译AtlasBoost。

```
cd benchmark工具所在路径/train/atlas_benchmark-master/utils/atlasboost/  
build
```

```
./compile.sh
```

若出现如下显示信息，说明编译成功。

```
-- The C compiler identification is GNU 8.3.0  
-- The CXX compiler identification is GNU 8.3.0  
-- Check for working C compiler: /usr/bin/cc  
-- Check for working C compiler: /usr/bin/cc -- works  
-- Detecting C compiler ABI info  
-- Detecting C compiler ABI info - done  
-- Detecting C compile features  
-- Detecting C compile features - done  
-- Check for working CXX compiler: /usr/bin/c++  
-- Check for working CXX compiler: /usr/bin/c++ -- works  
-- Detecting CXX compiler ABI info  
-- Detecting CXX compiler ABI info - done  
-- Detecting CXX compile features  
-- Detecting CXX compile features - done  
-- Configuring done  
-- Generating done  
-- Build files have been written to: /home/zhangmei/AtlasBoost-master/build  
Scanning dependencies of target atlasboost  
[ 20%] Building CXX object CMakeFiles/atlasboost.dir/Json.cpp.o  
[ 40%] Building CXX object CMakeFiles/atlasboost.dir/MpiControl.cpp.o  
[ 60%] Building CXX object CMakeFiles/atlasboost.dir/MpiEnv.cpp.o  
[ 80%] Building CXX object CMakeFiles/atlasboost.dir/Operations.cpp.o  
[100%] Linking CXX shared library bin/libatlasboost.so  
[100%] Built target atlasboost
```

编译成功后，在“*benchmark工具所在路径/train/atlas_benchmark-master/utils/atlasboost/lib*”目录下生成新的libatlasboost.so文件。

步骤7 进入“*benchmark工具所在路径/train/atlas_benchmark-master/utils/atlasboost/test*”目录。

步骤8 执行如下命令打开mpi.sh脚本，并修改其中的-H参数。

```
vi mpi.sh
```

-H参数指定了启动哪些服务器上的test_tensorflow.py脚本以及每台服务器上启动几条进程（冒号后的数值即为在该服务器上启动的进程数），请用户根据实际环境进行配置。

例如修改为：

```
mpirun -H 10.90.140.199:2,10.90.140.231:4
```

步骤9 执行如下命令运行test目录下的mpi.sh脚本。

```
./mpi.sh
```

若出现类似如下执行结果，说明运行环境正常。

```
deviceid = 1 phyid= 1
deviceid = 0 phyid= 0
{
  "server_count": "2",
  "server_list": [
    {
      "device": [
        {
          "device_id": "0",
          "device_ip": "192.168.100.101",
          "rank_id": "0"
        },
        {
          "device_id": "1",
          "device_ip": "192.168.101.101",
          "rank_id": "1"
        }
      ],
      "server_id": "10.90.140.199"
    },
    {
      "device": [
        {
          "device_id": "0",
          "device_ip": "192.168.190.100",
          "rank_id": "2"
        },
        {
          "device_id": "1",
          "device_ip": "192.168.191.100",
          "rank_id": "3"
        },
        {
          "device_id": "2",
          "device_ip": "192.168.192.100",
          "rank_id": "4"
        },
        {
          "device_id": "3",
          "device_ip": "192.168.193.100",
          "rank_id": "5"
        }
      ],
      "server_id": "10.90.140.231"
    }
  ],
  "status": "completed",
  "version": "1.0"
}
```

同时在10.90.140.199和10.90.140.231服务器的“*benchmark工具所在路径*/train/atlas_benchmark-master/utils/atlasboost/test”目录下，分别生成了一份名为*_atlasboost_hccl.json的RANK_TABLE_FILE文件，两份文件内容一致，都记录了每条进程的信息。

----结束

3.4.3.5 运行 benchmark 工具

以Host集群场景下启动框架为TensorFlow、模型为ResNet50训练为例：

步骤1 修改“/home/train/yaml/”目录下ResNet50.yaml文件中的**tensorflow_config**参数。

tensorflow_config:

- **data_url**: /home/dataset/imagenet_TF/
- **mpirun_ip**: 10.90.140.199:8,10.90.140.231:8

步骤2 在train目录下执行如下命令运行benchmark工具，启动训练。

```
bash benckmark.sh -e ResNet50 -hw ct
```

----结束

3.4.4 Docker 集群训练场景（TensorFlow）

3.4.4.1 部署前准备

部署Docker集群训练环境需要准备如下工作：

- 多台训练服务器Device使用交换机连接。
- 多台训练服务器的Device网卡IP地址在同一网段内。
- 保证多台训练服务器环境配置相同（例如操作系统、CPU架构和CANN软件包版本等），且benchmark运行目录绝对路径相同。
- 在多台训练服务器上分别参照[3.3 运行工具](#)章节完成运行工具前的准备工作。包括：
 - 参照benchmark工具中各模型的README文件做好训练前准备工作。
 - 获取测试模型对应的数据集并上传到服务器。
 - 获取benchmark工具包并上传到服务器，解压benchmark工具包，并增加权限。
 - 确保环境变量与环境中安装的CANN软件包路径一致。
- 在多台服务器的宿主机上安装驱动、固件和实用工具包。具体请参见《CANN 软件安装指南》中的“安装运行环境（训练）> 容器安装”章节。

3.4.4.2 构建容器镜像

前提条件

请按照[表3-4](#)所示，获取对应操作系统的软件包与打包镜像所需Dockerfile文件与脚本文件。

其中{version}表示版本号，{arch}表示操作系统。

表 3-4 所需软件

软件包	说明	获取方法
Ascend-cann-nnae_{version}_linux-{arch}.run	深度学习加速引擎包	获取链接
Ascend-fwk-tfplugin_{version}_linux-{arch}.run	框架插件包	

软件包	说明	获取方法
tensorflow-1.15.0-cp37-cp37m- {version}.whl	tensorflow框架whl包	aarch64架构请参见 3.7.2 安装TensorFlow 自行编译tensorflow安装包。x86架构请忽略。
ascend_install.info	软件包安装日志文件	从host拷贝“/etc/ascend_install.info”文件。
version.info	driver包版本信息文件	从host拷贝“/usr/local/Ascend/driver/version.info”文件。
libstdc++.so.6.0.24	动态库文件	仅当容器镜像OS为CentOS时需要准备libstdc++.so.6.0.24文件。 可以通过find命令查询libstdc++.so.6.0.24文件所在路径，然后从host拷贝。
Dockerfile	制作镜像需要	用户根据业务自行准备。
prebuild.sh	执行训练运行环境安装准备工作，例如配置代理等。	
install_ascend_pkgs.sh	昇腾软件包安装脚本	
postbuild.sh	清除不需要保留在容器中的安装包、脚本、代理配置等	
openmpi-{version}.tar.bz2	OpenMPI开源库，版本要求为4.0.1、4.0.2或4.0.3。	从host拷贝openmpi-{version}.tar.bz2软件包。如果host上没有该软件包，请从 获取链接 获取。
cmake-{version}.tar.gz	cmake版本要求为3.13及以上。	从host拷贝cmake-{version}.tar.gz软件包。

操作步骤

步骤1 以root用户登录服务器。

步骤2 创建软件包上传路径（以“/home/test”为例）。

mkdir -p /home/test

步骤3 将准备的软件包、深度学习框架（x86架构请忽略）、OpenMPI开源库、cmake、host昇腾软件包安装信息文件、driver包版本信息文件、pip源文件和source源文件上传至服务器“/home/test”目录下。

- Ascend-cann-nnae_{version}_linux-{arch}.run
- Ascend-fwk-tfplugin_{version}_linux-{arch}.run
- tensorflow-1.15.0-cp37-cp37m-linux_aarch64.whl（x86架构请忽略）
- openmpi-{version}.tar.bz2
- cmake-{version}.tar.gz
- ascend_install.info
- version.info
- libstdc++.so.6.0.24（仅当容器镜像OS为CentOS时需要）
- pip.conf
- sources.list

步骤4 准备Dockerfile、prebuild.sh、install_ascend_pkgs.sh和postbuild.sh文件。

当前benchmark工具仅提供了Ubuntu Arm64操作系统的Dockerfile文件，其他操作系统的Dockerfile文件请用户参照Ubuntu Arm64操作系统的Dockerfile文件进行修改。

可将Dockerfile文件等拷贝到“/home/test”目录下。命令示例如下：

cp benchmark工具所在路径/train/atlas_benchmark-master/utils/Dockerfile/tf_ubuntu_arm64/文件名 /home/test

其中文件名为Dockerfile、prebuild.sh、install_ascend_pkgs.sh或postbuild.sh。注意各文件仅提供基础示例或提示，需用户自行修改编写。

说明

- 在创建Dockerfile等文件后，可自行修改Dockerfile等文件权限。
- 为获取镜像“ubuntu:18.04”，用户也可以通过执行**docker pull ubuntu:18.04**命令从Docker Hub拉取。
- 由于Dockerfile文件里有安装第三方软件包，如果第三方软件出现漏洞，请用户自行修复。

步骤5 进入软件包所在目录，执行以下命令构建容器镜像。

docker build -t image-name --build-arg http_proxy=\$http_proxy --build-arg https_proxy=\$https_proxy .

命令解释如表3-5所示。

表 3-5 命令参数说明

参数	说明
<i>image-name</i>	指定镜像名称与标签，请用户根据实际情况替换。
--build-arg	指定Dockerfile文件内的参数。
http_proxy、https_proxy	指定网络代理，用户自行配置。如果用户服务器能直接连接外部网络，则无需配置此选项。

当出现“Successfully built xxx”表示镜像构建成功，注意不要遗漏命令结尾的“.”。

步骤6 构建完成后，执行以下命令查看镜像信息。

docker images

显示示例：

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
mpirun	latest	1e9dff52caa9	2 days ago	5.09GB

步骤7 执行以下命令将新生成的镜像保存为文件。

docker save -o mpirun.tar mpirun:latest

mpirun.tar即为生成的镜像文件，文件名用户可自行定义。

步骤8 执行以下命令将**步骤7**生成的镜像文件导入到部署Docker集群的其他服务器上。

docker load -i mpirun.tar

至此部署Docker集群的多台服务器上的容器镜像构建完成。

----结束

3.4.4.3 启动训练容器

请在每台服务器的宿主机上分别进行如下操作：

步骤1 进入“*benchmark工具所在路径/train/yaml*”目录，修改cluster_info.yaml文件中的参数。

说明

建议将以下数据集路径全部映射到容器内。

```
tensorflow_config:
  imagenet_data: /home/imagenet_TF/          # ImageNet数据集路径，请用户根据实际路径替换
  yolo_data: /opt/npu/dataset/                # YoloV3数据集路径，请用户根据实际路径替换
  bert_data: /home/BertData/                 # Bert数据集路径，请用户根据实际路径替换
  ssd_data: /home/data/raw_data              # SSD数据集路径，请用户根据实际路径替换
  docker_images: mpirun:latest               # 构建好的容器镜像，例如mpirun:latest
  ip: 10.90.176.110                          # 容器的IP地址，需选择与服务器同网段且没有启用的真实IP地址
  epcount: 24                               # 掩码
```

步骤2 进入“*benchmark工具所在路径/train*”目录，执行以下命令启动训练容器。

./docker_ct_build.sh

如果显示容器ID（例如“ca4792c6f5bb”），则表示已进入该容器。

步骤3 执行以下命令进行容器间SSH远程互通。

```
ssh root@容器IP
```

例如两台容器，互相进行SSH登录：

在10.90.176.158上，执行ssh root@10.90.176.157。

在10.90.176.157上，执行ssh root@10.90.176.158。

----结束

3.4.4 运行 benchmark 工具

前提条件

运行benchmark工具前，需要先执行**service docker restart**命令重启Docker。

操作步骤

以Docker集群场景下启动框架为TensorFlow、模型为ResNet50训练为例：

步骤1 修改“/home/train/yaml/”目录下ResNet50.yaml文件中的**tensorflow_config**参数。

tensorflow_config:

- **data_url:** /home/dataset/imagenet_TF/
- **mpirun_ip:** 10.90.176.152:8,10.90.176.154:8
- **docker_image:** mpirun:latest

步骤2 执行如下命令运行benchmark工具，启动训练。

```
./benckmark.sh -e ResNet50 -hw ct -docker
```

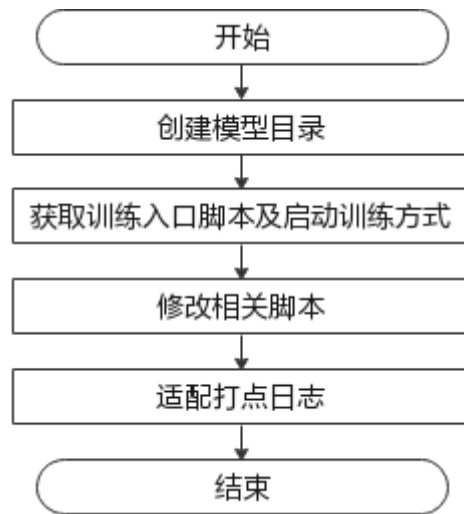
----结束

3.5 新模型适配 benchmark 工具方法

3.5.1 总体流程

新模型适配benchmark工具的总体流程如[图3-2](#)所示：

图 3-2 总体流程



3.5.2 创建模型目录

步骤1 从3.2 获取工具包章节获取benchmark工具包。

步骤2 在工具包所在目录下执行如下命令解压工具包。

```
tar -zxvf train_benchmark.tar.gz
```

步骤3 在模型类型对应的目录下创建以模型名称命名的目录（如“\train\atlas_benchmark-master\image_classification”）。

步骤4 在创建的模型目录下根据框架不同，创建框架目录。

目前支持的框架有tensorflow、pytorch和mindspore。

步骤5 每个框架目录下创建code、config、result和scripts目录。

步骤6 在config目录下创建npu_set_env.sh文件，创建方法可以参考其他模型。

步骤7 在scripts目录下创建run.sh和train.sh两个文件，创建方法可以参考其他模型。

步骤8 在“\train\yaml”目录下创建以模型名称命名的yaml文件，创建方法可以参考其他模型。

----结束

3.5.3 获取训练入口脚本及启动训练方式

步骤1 将用户写好的训练入口脚本（例如train.py）拷贝到对应框架目录的code目录下。

步骤2 确定启动训练方式。

启动训练方式一般分为以下两种：

- 命令行传参方式启动：启动方法可以参考“Bert-Large/tensorflow/scripts/train.sh”。
- 配置文件传参方式启动：启动方法可以参考“ResNet50/tensorflow/scripts/train.sh”。

----结束

3.5.4 修改相关脚本

修改环境变量配置脚本

可以参考“ResNet50/tensorflow/config/npu_set_env.sh”修改环境变量配置脚本。

修改 run.sh 脚本

- 如果模型训练的启动方式为配置文件传参方式启动，在run.sh脚本调用train.sh之前，需要在run.sh脚本里修改配置文件，该配置文件一般在config目录下。具体修改方式可以参考“ResNet50/tensorflow/scripts/run.sh”。
- 如果模型训练的启动方式为命令行传参方式启动，run.sh脚本负责调用train.sh脚本即可，不需要修改配置文件。

修改 train.sh 脚本

train.sh脚本需要修改的内容如下：

- 在train.sh中增加模型训练时相关的环境变量，具体可以参考“ResNet50/tensorflow/scripts/train.sh”。
- 修改模型训练入口脚本的启动命令，该命令通常以python3.7执行，模型训练入口脚本路径尽量写绝对路径。启动命令的参数需要分别根据单机单卡、单机多卡、多机多卡进行修改。
- 在train.sh中增加日志路径，例如hw_resnet50.log的路径，具体可以参考“ResNet50/tensorflow/scripts/train.sh”。
- 如果要进行集群训练，需要在train.sh中增加如下代码调用AtlasBoost，获取训练时所需的Device信息，具体可以参考“ResNet50/tensorflow/scripts/train.sh”。

```
device_id_mo=$(python3.7 -c "import src.tensorflow.mpi_ops as
atlasboost;atlasboost.init(); \ device_id = atlasboost.local_rank();cluster_device_id =
str(device_id); \ atlasboost.set_device_id(device_id);print(atlasboost.rank())")
```

3.5.5 适配打点日志

3.5.5.1 添加打点日志步骤

- 在train.sh中加入hw_benchmark模块环境变量与声明日志文件名称。

```
benchmark_log_path=${currentDir%atlas_benchmark-master*}/atlas_benchmark-master/
utils
export PYTHONPATH=$PYTHONPATH:${benchmark_log_path}
export REMARK_LOG_FILE=hw_resnet50.log #每个模型日志名称不一致，请根据实际模型名称替换
```

将模型运行时间添加到打点日志文件，即算出训练时间输入到打点日志文件，具体可参考ResNet50的train.sh脚本。

- 修改模型训练入口脚本，通常为train.py。
 - 导入打点日志模块。

```
from benchmark_log import hwlog
```
 - 若要打印硬件与环境参数、训练初始化参数、模型配置参数等信息，建议在模型训练入口脚本处打印，具体可参考ResNet101模型的训练入口脚本。

示例如下：

```
from benchmark_log import hwlog
from benchmark_log.basic_utils import get_environment_info
```

```
from benchmark_log.basic_utils import get_model_parameter
cpu_info, npu_info, framework_info, os_info, benchmark_version =
get_environment_info("tensorflow") #获取硬件与环境参数
config_info = get_model_parameter("ResNet101") #获取模型配置参数
initinal_data = {"base_lr": 0.128, "dataset": "imagenet1024", "optimizer": "SGD",
"loss_scale": 512, "batchsize": 32} #训练初始化参数
hwlog.remark_print(key=hwlog.CPU_INFO, value=cpu_info)
hwlog.remark_print(key=hwlog.NPU_INFO, value=npu_info)
hwlog.remark_print(key=hwlog.OS_INFO, value=os_info)
hwlog.remark_print(key=hwlog.FRAMEWORK_INFO, value=framework_info)
hwlog.remark_print(key=hwlog.BENCHMARK_VERSION, value=benchmark_version)
hwlog.remark_print(key=hwlog.CONFIG_INFO, value=config_info)
hwlog.remark_print(key=hwlog.BASE_LR, value=initinal_data.get("base_lr"))
hwlog.remark_print(key=hwlog.DATASET, value=initinal_data.get("dataset"))
hwlog.remark_print(key=hwlog.OPT_NAME, value=initinal_data.get("optimizer"))
hwlog.remark_print(key=hwlog.LOSS_SCALE, value=initinal_data.get("loss_scale"))
```

3.5.5.2 修改训练过程打点日志

修改训练脚本（可能多个），在这些脚本中增加以下代码：

- 在脚本中找到参数fps和steps，并在其下方添加以下代码：
hwlog.remark_print(key=hwlog.FPS, value=float(current_examples_per_sec))
- 在脚本中找到参数accuracy和accuracy_top_5，并在其下方添加以下代码：
hwlog.remark_print(key=hwlog.EVAL_ACCURACY_TOP1, value=float(eval_results.get("accuracy")))
hwlog.remark_print(key=hwlog.EVAL_ACCURACY_TOP5, value=float(eval_results.get("accuracy_top_5")))

打点日志通常生成在“/train/result/”目录下。打点日志的格式可参考ResNet50模型生成的打点日志hw_resnet50.log。

3.6 常见问题

3.6.1 模型训练时报错“Open TsdClient failed, tdt error code: *****, error message: tdt client open failed”

问题描述

模型训练失败，通过查看日志报错：“Open TsdClient failed, tdt error code: *****, error message: tdt client open failed”。

可能原因

Device侧存在aicpu和hccp残余进程。

解决方法

方法一：

登录Device侧，执行如下命令将aicpu和hccp残余进程kill掉。

ifconfig endvnic ip

ssh HwHiAiUser@192.168.1.199


```
ps -ef|grep aicpu
```

```
Kill -9 pid
```

```
ps -ef|grep hccp
```

```
Kill -9 pid
```

方法二：

执行reboot命令重启设备。

3.6.2 YoloV3 模型训练时缺少预训练数据集

问题描述

YoloV3模型执行训练时失败，通过查看日志，报如下错误：

```
tensorflow.python.framework.errors_impl.NotFoundError: /home/wx5318057/benchmark-benchmark_alpha/train/atlas_benchmark-master/object_detection/yolov3/tensorflow/code/data/darknet_weights; No such file or directory
2020-08-19 03:57:31.174087: I tf_adapter/util/ge_plugin.cc:39] [GePlugin] destroy constructor begin
2020-08-19 03:57:31.174116: I tf_adapter/util/ge_plugin.cc:160] [GePlugin] ge has already finalized.
2020-08-19 03:57:31.174209: I tf_adapter/util/ge_plugin.cc:41] [GePlugin] destroy constructor end
:::ABK 1.0.0 yolov3 train failed
```

可能原因

缺少预训练数据集。

解决方法

在“*benchmark工具所在路径*/train/atlas_benchmark-master/object_detection/yolov3/tensorflow/code/data/”路径下需要有预训练好的数据集。具体请参考YoLoV3模型的README文件（路径为“*benchmark工具所在路径*/train/atlas_benchmark-master/object_detection/yolov3/tensorflow/README.md”）。

3.6.3 搭建训练环境时依赖 numpy 安装报错

问题描述

在搭建训练环境过程中，依赖numpy安装报错。

可能原因

numpy版本过高。

解决方法

查看numpy版本，如果numpy版本高于1.16.1，降低版本到1.16.1即可。

3.6.4 ResNet50 模型训练时报错 “ValueError: Data formats other than 'NHWC' are not yet supported”

问题描述

ResNet50模型执行训练时报错：“ValueError: Data formats other than 'NHWC' are not yet supported”。

在每台服务器上执行`ps -ef |grep python`命令查看进程，若每台服务器上的进程都被拉起且正常输出日志，说明集群环境测试正常。

----结束

3.7.2 安装 TensorFlow

安装前准备

- 对于x86架构，请跳过安装前准备。
- 对于aarch64架构，由于tensorflow依赖h5py，而h5py依赖HDF5，需要先编译安装HDF5，否则使用pip安装h5py会报错，以下步骤以root用户操作。

步骤1 编译安装HDF5。

1. 使用wget下载HDF5源码包，可以下载到安装环境的任意目录，命令为：

```
wget https://support.hdfgroup.org/ftp/HDF5/releases/hdf5-1.10/hdf5-1.10.5/src/hdf5-1.10.5.tar.gz --no-check-certificate
```
2. 进入下载后的目录，解压源码包，命令为：

```
tar -zxvf hdf5-1.10.5.tar.gz
```


进入解压后的文件夹，执行配置、编译和安装命令：

```
cd hdf5-1.10.5/  
./configure --prefix=/usr/include/hdf5  
make  
make install
```

步骤2 配置环境变量并建立动态链接库软连接。

1. 配置环境变量。

```
export CPATH="/usr/include/hdf5/include:/usr/include/hdf5/lib/"
```
2. root用户建立动态链接库软连接命令如下，非root用户需要在以下命令前添加sudo。

```
ln -s /usr/include/hdf5/lib/libhdf5.so /usr/lib/libhdf5.so  
ln -s /usr/include/hdf5/lib/libhdf5_hl.so /usr/lib/libhdf5_hl.so
```

步骤3 安装h5py。

root用户下安装h5py依赖包命令如下。

```
pip3.7 install Cython
```

安装h5py命令如下：

```
pip3.7 install h5py==2.8.0
```

----结束

安装 Tensorflow 1.15.0

需要安装Tensorflow 1.15才可以进行算子开发验证、训练业务开发。

- 对于x86架构：直接从pip源下载即可，具体步骤请参考<https://www.tensorflow.org/install/pip?lang=python3>。需要注意tensorflow官网提供的指导描述有错误，从pip源下载cpu版本需要显式指定tensorflow-cpu，如果不指定cpu，默认下载的是gpu版本。即官网的“Tensorflow==1.15: 仅支持 CPU 的版本”需要修改成“Tensorflow-cpu==1.15: 仅支持 CPU 的版本”。另外，官网描述的安装命令“pip3 install --user --upgrade tensorflow”需要在root更改为“pip3.7 install Tensorflow-cpu==1.15”，非root用户下更改为“pip3.7 install Tensorflow-cpu==1.15 --user”。

- 对于aarch64架构：由于pip源未提供对应的版本，所以需要用户使用官网要求的gcc7.3.0编译器编译tensorflow1.15.0，编译步骤参考官网<https://www.tensorflow.org/install/source>。特别注意点如下：

在下载完tensorflow tag v1.15.0源码后执行需要如下步骤。

步骤1 下载“nsync-1.22.0.tar.gz”源码包。

- 进入tensorflow tag v1.15.0源码目录，打开“tensorflow/workspace.bzl”文件，找到其中name为nsync的“tf_http_archive”定义。

```
tf_http_archive(  
    name = "nsync",  
    sha256 = "caf32e6b3d478b78cff6c2ba009c3400f8251f646804bcb65465666a9cea93c4",  
    strip_prefix = "nsync-1.22.0",  
    system_build_file = clean_dep("//third_party/systemlibs:nsync.BUILD"),  
    urls = [  
        "https://storage.googleapis.com/mirror.tensorflow.org/github.com/google/nsync/  
archive/1.22.0.tar.gz",  
        "https://github.com/google/nsync/archive/1.22.0.tar.gz",  
    ],  
)
```

- 从urls中的任一路径下载nsync-1.22.0.tar.gz的源码包，保存到任意路径。

步骤2 修改“nsync-1.22.0.tar.gz”源码包。

- 切换到nsync-1.22.0.tar.gz所在路径，解压缩该源码包。解压缩后存在“nsync-1.22.0”文件夹和“pax_global_header”文件。
- 编辑“nsync-1.22.0/platform/c++11/atomic.h”。

在NSYNC_CPP_START_内容后添加如下加粗字体内容。

```
#include "nsync_cpp.h"  
#include "nsync_atomic.h"  
  
NSYNC_CPP_START_  
  
#define ATM_CB_() __sync_synchronize()  
  
static INLINE int atm_cas_nomb_u32_ (nsync_atomic_uint32_ *p, uint32_t o, uint32_t n) {  
    int result = (std::atomic_compare_exchange_strong_explicit (NSYNC_ATOMIC_UINT32_PTR_ (p),  
&o, n, std::memory_order_relaxed, std::memory_order_relaxed));  
    ATM_CB_();  
    return result;  
}  
  
static INLINE int atm_cas_acq_u32_ (nsync_atomic_uint32_ *p, uint32_t o, uint32_t n) {  
    int result = (std::atomic_compare_exchange_strong_explicit (NSYNC_ATOMIC_UINT32_PTR_ (p),  
&o, n, std::memory_order_acquire, std::memory_order_relaxed));  
    ATM_CB_();  
    return result;  
}  
  
static INLINE int atm_cas_rel_u32_ (nsync_atomic_uint32_ *p, uint32_t o, uint32_t n) {  
    int result = (std::atomic_compare_exchange_strong_explicit (NSYNC_ATOMIC_UINT32_PTR_ (p),  
&o, n, std::memory_order_release, std::memory_order_relaxed));  
    ATM_CB_();  
    return result;  
}  
  
static INLINE int atm_cas_relacq_u32_ (nsync_atomic_uint32_ *p, uint32_t o, uint32_t n) {  
    int result = (std::atomic_compare_exchange_strong_explicit (NSYNC_ATOMIC_UINT32_PTR_ (p),  
&o, n, std::memory_order_acq_rel, std::memory_order_relaxed));  
    ATM_CB_();  
    return result;  
}
```

步骤3 重新压缩“nsync-1.22.0.tar.gz”源码包。

将解压出的“nsync-1.22.0”文件夹和“pax_global_header”压缩为一个新的“nsync-1.22.0.tar.gz”源码包，保存（假如保存在“/tmp/nsync-1.22.0.tar.gz”）。

步骤4 重新生成“nsync-1.22.0.tar.gz”源码包的sha256sum校验码。

```
sha256sum /tmp/nsync-1.22.0.tar.gz
```

执行如上命令后得到sha256sum校验码（一串数字和字母的组合）

步骤5 修改sha256sum校验码和urls。

进入tensorflow tag v1.15.0源码目录，打开“tensorflow/workspace.bzl”文件，找到其中name为nsync的“tf_http_archive”定义，其中“sha256=”后面的数字填写**步骤4**得到的校验码，“urls=”后面的列表第一行，填写存放“nsync-1.22.0.tar.gz”的file://索引。

```
tf_http_archive(
    name = "nsync",
    sha256 = "caf32e6b3d478b78cff6c2ba009c3400f8251f646804bcb65465666a9cea93c4",
    strip_prefix = "nsync-1.22.0",
    system_build_file = clean_dep("//third_party/systemlibs:nsync.BUILD"),
    urls = [
        "file:///tmp/nsync-1.22.0.tar.gz ",
        "https://storage.googleapis.com/mirror.tensorflow.org/
github.com/google/nsync/archive/1.22.0.tar.gz",
        "https://github.com/google/nsync/archive/1.22.0.tar.gz",
    ],
)
```

步骤6 继续从官方的“配置build”（<https://www.tensorflow.org/install/source>）执行编译。

执行完./configure之后，需要修改 .tf_configure.bazelrc 配置文件，添加如下一行build编译选项：

```
build:opt --cxxopt=-D_GLIBCXX_USE_CXX11_ABI=0
```

删除以下两行：

```
build:opt --copt=-march=native
build:opt --host_copt=-march=native
```

步骤7 继续执行官方的编译指导步骤（<https://www.tensorflow.org/install/source>）即可。

步骤8 安装编译好的Tensorflow。

以上步骤执行完后会打包Tensorflow到指定目录，进入指定目录后root用户执行如下命令安装Tensorflow1.15：

```
pip3.7 install tensorflow-1.15.0-cp37m-linux_aarch64.whl
```

非root用户执行如下命令：

```
pip3.7 install tensorflow-1.15.0-cp37m-linux_aarch64.whl --user
```

----结束