CANN
V100R020C10

推理 benchmark 工具用户指南

文档版本 01

发布日期 2020-10-31

华为技术有限公司

商标声明


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。

目 录

1 工具介绍

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（型号 3000,3010）
操作系统	Ubuntu 18.04 CentOS 7.6 EulerOS 2.8

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。

```
├── 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.{arch}
```

表 1-1 目录说明

目录名称	目录说明
scripts	工具脚本目录。
Sample	工程样例。
bert_metric.py	Bert模型精度统计脚本。
bin_to_predict.py	YoloV3 TensorFlow模型后处理脚本。
get_info.py	创建数据集脚本。

目录名称	目录说明
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.*{arch}*	benchmark工具运行脚本。{arch}为CPU架构，取值为aarch64或x86_64。

2 获取工具包

步骤1 登录**软件获取**页面。

步骤2 在“软件下载 > CANN-Benchmark”下，根据CPU架构单击“软件包下载”获取推理benchmark工具包。

----结束

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。

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**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:${install_path}/atc/ccec_compiler/bin:${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3-1。

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3-2。

```
[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 3-1 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 3-2 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。

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**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:${install_path}/atc/ccec_compiler/bin:${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3-3**。

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（文件名格式为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3-4**。

```
[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 3-3 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： ● 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。

参数	说明	是否必填
[-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。
[-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： ● 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： ● 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 3-4 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] [preprocess] [post]	throughputRate	当前模块的吞吐率。
	moduleLatency	执行一次当前模块的时延。
[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。

4 使用实例

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

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] 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] DeInit SUCCESS
[INFO][Preprocess] DeInit SUCCESS
[INFO][Inference] DeInit SUCCESS
[INFO][DataManager] DeInit 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] DeInit SUCCESS
[INFO][DeepPreprocess] DeInit SUCCESS
[INFO][Preprocess] DeInit SUCCESS
[INFO][Inference] DeInit SUCCESS
[INFO][DataManager] DeInit 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] 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] 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）。

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] 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 =  3.17
BLEU_cased =  2.98
```

----结束

5 新模型适配 benchmark 工具方法

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。

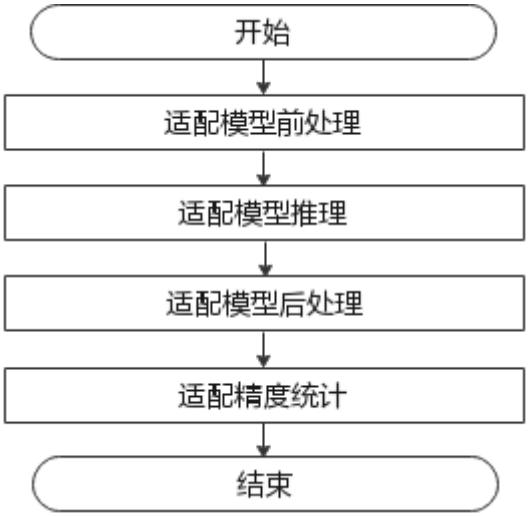
新模型适配开发工作主要涉及模型前处理、模型推理、模型后处理等三个方面。

- 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。

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5-1**所示：

图 5-1 总体流程



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

A 修订记录

发布日期	修改说明
2020-10-15	第一次正式发布。

