

CANN  
7.0.0

# ATC 工具使用指南 (开放态)

文档版本	01
发布日期	2024-12-05



**版权所有 © 华为技术有限公司 2024。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暗示的声明或保证。

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。

# 安全声明

## 产品生命周期政策

华为公司对产品生命周期的规定以“产品生命周期终止政策”为准，该政策的详细内容请参见如下网址：  
<https://support.huawei.com/ecolumnsweb/zh/warranty-policy>

## 漏洞处理流程

华为公司对产品漏洞管理的规定以“漏洞处理流程”为准，该流程的详细内容请参见如下网址：  
<https://www.huawei.com/cn/psirt/vul-response-process>  
如企业客户须获取漏洞信息，请参见如下网址：  
<https://securitybulletin.huawei.com/enterprise/cn/security-advisory>

## 华为初始证书权责说明

华为公司对随设备出厂的初始数字证书，发布了“华为设备初始数字证书权责说明”，该说明的详细内容请参见如下网址：  
<https://support.huawei.com/enterprise/zh/bulletins-service/ENEWS2000015766>

## 华为企业业务最终用户许可协议(EULA)

本最终用户许可协议是最终用户（个人、公司或其他任何实体）与华为公司就华为软件的使用所缔结的协议。最终用户对华为软件的使用受本协议约束，该协议的详细内容请参见如下网址：  
<https://e.huawei.com/cn/about/eula>

## 产品资料生命周期策略

华为公司针对随产品版本发布的售后客户资料（产品资料），发布了“产品资料生命周期策略”，该策略的详细内容请参见如下网址：  
<https://support.huawei.com/enterprise/zh/bulletins-website/ENEWS2000017760>

目 录

1 学习向导.....	1
2 环境搭建.....	2
3 快速入门.....	5
4 ATC 简介.....	9
4.1 ATC 工具介绍.....	9
4.2 运行流程.....	13
4.3 关键概念.....	13
5 初级功能.....	16
5.1 原始模型文件或离线模型转成 json 文件.....	16
5.2 离线模型支持动态 BatchSize/动态分辨率.....	17
5.3 离线模型支持动态维度.....	17
5.4 自定义离线模型的输入输出数据类型.....	18
5.5 借助离线模型查看软件基础版本号.....	19
6 高级功能.....	21
6.1 AIPP 使能.....	21
6.1.1 什么是 AIPP.....	21
6.1.2 如何使能 AIPP.....	22
6.1.3 AIPP 配置示例.....	29
6.1.3.1 静态 AIPP 配置示例.....	29
6.1.3.2 动态 AIPP 配置示例.....	31
6.1.4 色域转换配置说明.....	33
6.1.5 归一化配置说明.....	58
6.1.6 Crop/Padding 配置说明.....	58
6.1.7 AIPP 对模型输入大小的校验说明.....	59
6.1.8 配置文件模板.....	60
6.1.9 典型场景样例参考.....	66
6.1.9.1 YUV400_U8 转 GRAY 格式.....	66
6.1.9.2 YUV420SP_U8 转 BGR 格式.....	66
6.1.9.3 RGB888_U8 转 RGB（或 BGR）格式.....	67
6.2 单算子模型转换.....	68
6.2.1 什么是单算子描述文件.....	68

6.2.2 如何将算子描述文件转成离线模型.....	68
6.2.3 配置文件样例.....	68
6.2.3.1 单算子描述文件配置.....	68
6.2.3.2 多组算子描述文件配置.....	73
6.2.3.3 动态 Shape 单算子描述文件配置.....	74
6.2.4 描述文件参数说明.....	76
<b>7 参数说明.....</b>	<b>83</b>
7.1 参数概览.....	83
7.2 基础功能参数.....	89
7.2.1 总体选项.....	89
7.2.1.1 --help 或--h.....	89
7.2.1.2 --mode.....	90
7.2.2 输入选项.....	92
7.2.2.1 --model.....	92
7.2.2.2 --weight.....	93
7.2.2.3 --om.....	93
7.2.2.4 --framework.....	95
7.2.2.5 --input_format.....	96
7.2.2.6 --input_shape.....	98
7.2.2.7 --input_shape_range.....	100
7.2.2.8 --dynamic_batch_size.....	101
7.2.2.9 --dynamic_image_size.....	103
7.2.2.10 --dynamic_dims.....	104
7.2.2.11 --singleop.....	106
7.2.2.12 --distributed_cluster_build.....	107
7.2.2.13 --cluster_config.....	109
7.2.2.14 --enable_graph_parallel.....	111
7.2.2.15 --graph_parallel_option_path.....	112
7.2.2.16 --shard_model_dir.....	113
7.2.2.17 --model_relation_config.....	114
7.2.3 输出选项.....	116
7.2.3.1 --output.....	116
7.2.3.2 --output_type.....	117
7.2.3.3 --check_report.....	118
7.2.3.4 --json.....	119
7.2.3.5 --host_env_os.....	120
7.2.3.6 --host_env_cpu.....	121
7.2.4 目标芯片选项.....	122
7.2.4.1 --soc_version.....	122
7.2.4.2 --core_type.....	122
7.2.4.3 --aicore_num.....	123
7.2.4.4 --virtual_type.....	124

7.3 高级功能参数.....	125
7.3.1 功能配置选项.....	125
7.3.1.1 --out_nodes.....	125
7.3.1.2 --input_fp16_nodes.....	126
7.3.1.3 --insert_op_conf.....	127
7.3.1.4 --external_weight.....	128
7.3.1.5 --op_name_map.....	129
7.3.1.6 --is_input_adjust_hw_layout.....	130
7.3.1.7 --is_output_adjust_hw_layout.....	131
7.3.2 模型调优选项.....	131
7.3.2.1 --disable_reuse_memory.....	132
7.3.2.2 --fusion_switch_file.....	132
7.3.2.3 --enable_scope_fusion_passes.....	136
7.3.2.4 --enable_small_channel.....	137
7.3.2.5 --ac_parallel_enable.....	138
7.3.2.6 --compression_optimize_conf.....	138
7.3.2.7 --buffer_optimize.....	141
7.3.2.8 --mdl_bank_path.....	142
7.3.3 算子调优选项.....	143
7.3.3.1 --precision_mode.....	143
7.3.3.2 --precision_mode_v2.....	146
7.3.3.3 --op_precision_mode.....	148
7.3.3.4 --modify_mixlist.....	149
7.3.3.5 --op_select_implmode.....	151
7.3.3.6 --optypelist_for_implmode.....	153
7.3.3.7 --keep_dtype.....	154
7.3.3.8 --customize_dtypes.....	156
7.3.3.9 --op_bank_path.....	158
7.3.3.10 --op_debug_level.....	158
7.3.4 调试选项.....	160
7.3.4.1 --dump_mode.....	160
7.3.4.2 --log.....	161
7.3.4.3 --debug_dir.....	163
7.3.4.4 --op_compiler_cache_mode.....	164
7.3.4.5 --op_compiler_cache_dir.....	166
7.3.4.6 --display_model_info.....	167
7.3.4.7 --shape_generalized_build_mode.....	168
7.3.4.8 --status_check.....	169
7.3.4.9 --op_debug_config.....	172
7.3.4.10 --atomic_clean_policy.....	174
7.3.4.11 --deterministic.....	175
<b>8 专题.....</b>	<b>176</b>

8.1 定制网络修改 ( Caffe ) .....	176
8.1.1 简介.....	176
8.1.2 扩展算子列表.....	177
8.1.3 扩展算子规则.....	178
8.1.4 样例参考.....	191
8.1.4.1 FasterRCNN 网络模型 prototxt 修改.....	191
8.1.4.2 YOLOv3 网络模型 prototxt 修改.....	192
8.1.4.3 YOLOv2 网络模型 prototxt 修改.....	195
8.1.4.4 SSD 网络模型 prototxt 修改.....	196
8.1.4.5 BatchedMatMul 算子 prototxt 修改.....	198
8.1.4.6 SNet 网络模型 prototxt 修改.....	198
8.2 定制网络修改 ( TensorFlow ) .....	199
8.2.1 概述.....	200
8.2.2 编译可执行文件.....	201
8.2.3 转换模型.....	202
8.2.4 FAQ.....	203
8.2.4.1 使用 bazel 编译工具编译时提示 “An error occurred during the fetch of repository 'io_bazel_rules_docker'”，编译失败.....	203
8.2.4.2 使用 pip3.7.5 install 软件时提示" Could not find a version that satisfies the requirement xxx".....	205
8.2.4.3 获取 Switch_v1.pb 网络模型.....	206
8.2.4.4 获取 xlacompile.patch 补丁文件.....	206
<b>9 参考.....</b>	<b>211</b>
9.1 dump 图详细信息.....	211
9.2 算子规格参考.....	217
9.2.1 Caffe 框架算子规格.....	217
9.2.2 TensorFlow 框架算子规格.....	217
9.2.3 ONNX 算子规格.....	217
9.3 开启 AI CPU Cast 算子自动插入特性.....	218
9.4 支持量化的层及约束.....	219
9.5 简易配置文件.....	222
<b>10 FAQ.....</b>	<b>230</b>
10.1 开发环境架构为 Arm ( aarch64 ) 时模型转换耗时较长.....	230
10.2 使用 AIPP 色域转换模型时如何判断视频流的格式标准.....	231
10.3 如何确定原始框架网络模型中的算子与昇腾 AI 处理器支持的算子的对应关系.....	231
10.4 模型中存在不支持量化的层，量化模型失败.....	232
10.5 量化模型时模型输入大小过大，AI Core 执行任务失败，量化模型失败.....	233
10.6 量化模型时校准集数据大小与模型输入大小不匹配，量化模型失败.....	234

# 1 学习向导

本文档用于指导开发者如何使用ATC（Ascend Tensor Compiler，简称ATC）工具进行模型转换，得到适配昇腾AI处理器的离线模型。通过本文档您可以达成以下目标：

- 了解不同框架原始网络模型转成昇腾AI处理器离线模型的方法。
- 能够基于本文档中的参数，转成满足不同定制要求的离线模型。
- 掌握AIPP配置文件的配置方法，以及如何使能AIPP功能。

熟悉Linux基本命令，对机器学习、深度学习有一定了解的人员，可以更好地理解本文档。

## 针对新手

快速入门	ATC简介	初级功能
本章节以ATC工具支持的所有框架网络模型为例，简单介绍如何进行基础功能的模型转换。	介绍ATC工具功能架构、运行流程以及关键概念。	简单介绍其他功能点场景下，比如将模型转成json文件，离线模型支持动态batch，动态分辨率等场景，如何组合各种ATC参数转换成满足要求的离线模型。

## 适合专家

AIPP使能	单算子模型转换	专题
介绍什么是AIPP、模型转换时如何使能AIPP、根据配置文件模板如何构造AIPP配置文件，以及根据色域转换功能如何输出满足要求的图片数据等功能，并给出典型场景下的配置示例。	介绍什么是单算子描述文件、如何构造单算子描述文件，以及如何将该文件转成适配昇腾AI处理器的离线模型，用于验证单算子功能。	如果网络包含了一些没有定义的算子结构、或者含有控制流算子等场景时，则不能直接使用ATC工具转换模型，需要先按照本章的指导定制网络。



# 2 环境搭建

## 获取 ATC 工具

参见《[CANN 软件安装指南](#)》进行开发环境搭建，并确保开发套件包Ascend-cann-toolkit安装完成。该场景下ATC工具安装在“*Ascend-cann-toolkit安装目录/ascend-toolkit/latest/bin*”下。

## 设置环境变量

### 须知

- 若开发环境架构为Arm（aarch64），模型转换耗时较长，则可以参考[10.1 开发环境架构为Arm（aarch64）时模型转换耗时较长](#)解决。
- 该工具对Python版本的支持请参见《[CANN 软件安装指南](#)》中的“安装开发环境>安装OS依赖>依赖列表”章节，本手册以Python3.7.5为例进行介绍，相应环境变量和安装命令以实际安装Python版本为准。
- 使用ATC工具进行模型转换的过程中，会自动将ATC工具所在位置“../python/site-packages”目录下算子编译依赖的TBE Python库写入PYTHONPATH环境变量。  
若算子实现时用户引入了TBE模块外的其他Python依赖，请自行添加PYTHONPATH的环境变量，配置引入的Python依赖所在路径，如下所示：  
export PYTHONPATH=xxx:\$PYTHONPATH

### 1. 必选环境变量

#### – 设置公共环境变量

##### ■ 以root用户安装Ascend-cann-toolkit包

```
./usr/local/Ascend/ascend-toolkit/set_env.sh  
#若开发套件包Ascend-cann-toolkit在非昇腾设备上安装，则如下环境变量必须执行，用于设置动态链接库所在路径，否则无需执行  
export LD_LIBRARY_PATH=/usr/local/Ascend/ascend-toolkit/latest/<arch>-linux/  
devlib:$LD_LIBRARY_PATH
```

##### ■ 以非root用户安装Ascend-cann-toolkit包

```
./${HOME}/Ascend/ascend-toolkit/set_env.sh  
#若开发套件包Ascend-cann-toolkit在非昇腾设备上安装，则如下环境变量必须执行，用于设置动态链接库所在路径，否则无需执行  
export LD_LIBRARY_PATH=${HOME}/Ascend/ascend-toolkit/latest/<arch>-linux/  
devlib:$LD_LIBRARY_PATH
```

<arch>请替换为操作系统具体架构。

- 设置Python相关环境变量

模型编译依赖Python，以Python3.7.5为例，请以CANN软件包运行用户执行如下命令设置Python3.7.5相关环境变量。

```
#如果用户环境存在多个python3版本，则指定使用python3.7.5版本
export PATH=/usr/local/python3.7.5/bin:$PATH
#设置python3.7.5库文件路径
export LD_LIBRARY_PATH=/usr/local/python3.7.5/lib:$LD_LIBRARY_PATH
```

上述环境变量只在当前窗口生效，用户可以将上述命令写入~/.bashrc文件，使其永久生效，方法如下：

- 以安装用户在任意目录下执行`vi ~/.bashrc`，在该文件最后添加上述内容。
- 执行`wq!`命令保存文件并退出。
- 执行`source ~/.bashrc`使环境变量生效。

## 2. 可选环境变量

a. 日志落盘、打屏与重定向。

■ 日志落盘：

atc命令执行过程中，日志默认落盘到如下路径，由于**7.3.4.2 --log**默认值为null，即不输出日志，若下述路径存在日志信息，则为atc进程之外的其他日志信息，比如依赖Python相关信息；若想要日志体现atc进程相关信息，则模型转换时，需要设置**7.3.4.2 --log**参数（不能设置为null）。

- \$HOME/ascend/log/**debug**/plog/plog-pid\_\*.log：调试日志。
  - \$HOME/ascend/log/**run**/plog/plog-pid\_\*.log：运行日志。
  - \$HOME/ascend/log/**security**/plog/plog-pid\_\*.log：安全日志。
- pid*代表进程ID，“\*”表示该日志文件创建时的时间戳。

■ 日志打屏：

atc命令执行过程中，日志默认不打屏，如需打屏显示，则请在执行atc命令的当前窗口设置如下环境变量，然后再执行atc命令：

```
export ASCEND_SLOG_PRINT_TO_STDOUT=1
```

关于日志的更多信息请参见《[日志参考（开放态）](#)》。若设置上述环境变量后，仍旧未打屏有效信息，则请在atc命令设置**7.3.4.2 --log**参数（不能设置为null）显示相应的日志级别。

■ 日志重定向：

如果不想日志落盘，而是重定向到文件，则模型转换前需要设置上述的日志打屏环境变量，并且atc命令需要设置**7.3.4.2 --log**参数（不能设置为null），样例如下：

```
atc xxx --log=debug >log.txt
```

b. 开启**算子并行编译**功能。

若网络模型较大，模型转换过程中，可设置如下环境变量，开启算子的并行编译功能。

```
export TE_PARALLEL_COMPILER=xx
```

TE\_PARALLEL\_COMPILER的值代表算子编译进程数（配置为整数），取值范围为1~32，默认值为8，当取值大于1时开启算子的并行编译功能。建议不超过：CPU核数\*80%/昇腾AI处理器个数。

c. 打印模型转换过程中**各阶段的图描述信息**。

```
export DUMP_GE_GRAPH=1
```

上述环境变量控制dump图的内容多少，取值如下：

- 1：全量dump。
- 2：不含有权重等数据的基本版dump。
- 3：只显示节点关系的精简版dump。

设置上述环境变量后，还可以设置如下环境变量，控制dump图的个数。

```
export DUMP_GRAPH_LEVEL=1
```

- 1：dump所有图。
- 2：dump除子图外的所有图。
- 3：dump最后的生成图，即经过GE优化、编译后的图。
- 4：dump最早的生成图，即经过GE解析映射算子后，最早下沉到Device上的整图，此时图结构尚未经过GE的编译优化。

该环境变量只有在DUMP\_GE\_GRAPH开启时才生效，并且默认为2。设置上述变量后，在执行atc命令的当前路径会生成如下文件：

- ge\_onnx\*.pbtxt：基于ONNX的开源模型描述结构，可以使用Netron等可视化软件打开。
- ge\_proto\*.txt：protobuf格式存储的文本文件，该文件可以转成json格式文件方便用户定位问题。该文件与ge\_onnx\*.pbtxt成对出现，但是比ge\_onnx\*.pbtxt文件会多string类型的属性信息，用户选择其中一种文件打开即可。

上述每个文件对应模型编译过程中的一个步骤，比如以ge\_onnx\_00000001\_graph\_0\_PreRunBegin.pbtxt开始，以ge\_onnx\_00000078\_graph\_0\_PreRunAfterBuild.pbtxt结尾。每个文件中包括完成该步骤所涉及的所有算子，关于dump图的详细信息请参见[9.1 dump图详细信息](#)。

- d. 更多可选环境变量请参见《[环境变量参考](#)》。

# 3 快速入门

本章节以各框架下模型转换为例，演示如何快速转换一个模型。

## 须知

- 使用高版本ATC工具转换的模型，在低版本环境上使用可能会出现不兼容问题，建议使用匹配的版本重新进行模型转换，如果用于想查看已有离线模型使用的ATC工具等基础版本信息，则请参见[7.2.3.4 --json](#)。
- 如果用户使用的网络模型中有自定义算子，则请优先参见《[TBE&AI CPU算子开发指南（开放态）](#)》手册开发部署好自定义算子，模型转换时会优先去查找自定义算子库匹配模型文件中的算子；若匹配失败，则会去查找内置算子库。
- 如果用户使用Faster RCNN、YOLOv3、YOLOv2、SSD等Caffe框架网络模型进行模型转换，由于此类网络中包含了一些原始Caffe框架中没有定义的算子结构，如ROI Pooling、Normalize、PSROI Pooling和Upsample等。为了使昇腾AI处理器能支持这些网络，需要对原始的Caffe框架网络模型进行扩展，降低开发者开发自定义算子/开发后处理代码的工作量，详细扩展方法请参见[8.1 定制网络修改（Caffe）](#)。
- 针对TensorFlow原始网络模型，如果存在控制流算子，该类网络模型不能直接使用ATC工具进行模型转换，需要先将控制流算子的网络模型转成函数类算子的网络模型，然后利用ATC工具转换成适配昇腾AI处理器的离线模型，详细转换方式请参见[8.2 定制网络修改（TensorFlow）](#)。

## 开源框架的 TensorFlow 网络模型转换成离线模型

**步骤1** 获取TensorFlow网络模型。

单击[Link](#)，根据页面提示获取ResNet50网络的模型文件（\*.pb），并以CANN软件包运行用户将获取的文件上传至开发环境任意目录，例如上传到\$HOME/module/目录下。

**步骤2** 执行如下命令生成离线模型。（如下命令中使用的目录以及文件均为样例，请以实际为准）

```
atc --model=$HOME/module/resnet50_tensorflow*.pb --framework=3 --output=$HOME/module/out/tf_resnet50 --soc_version=<soc_version>
```

关于参数的详细解释以及使用方法请参见[参数说明](#)，请使用与芯片名相对应的 `<soc_version>` 取值进行模型转换，然后再进行推理，具体使用芯片查询方法请参见 [7.2.4.1 --soc\\_version](#)。

- 步骤3** 若提示如下信息，则说明模型转换成功，若模型转换失败，则请参见《[故障处理（开放态）](#)》进行定位。

```
ATC run success
```

成功执行命令后，在 `--output` 参数指定的路径下，可查看离线模型（如：`tf_resnet50.om`）。

模型编译时，若遇到AI CPU算子不支持某种数据类型导致编译失败的场景，可通过启用Cast算子自动插入特性快速将输入转换为算子支持的数据类型，从而实现网络的快速打通，详细流程请参见[9.3 开启AI CPU Cast算子自动插入特性](#)。

- 步骤4** （后续处理）如果想快速体验直接使用转换后的om离线模型文件进行推理，请准备好环境、om模型文件、符合模型输入要求的\*.bin格式的输入数据，单击[Link](#)，获取msame工具，参考该工具配套的README，进行体验。

----结束

## ONNX 网络模型转换成离线模型

- 步骤1** 获取ONNX网络模型。

单击[Link](#)进入ModelZoo页面，查看README.md中“快速上手>模型推理”章节获取.onnx模型文件，再以CANN软件包运行用户将onnx模型文件上传至开发环境任意目录，例如上传到 `$HOME/module/` 目录下。

- 步骤2** 执行如下命令生成离线模型。（如下命令中使用的目录以及文件均为样例，请以实际为准）

```
atc --model=$HOME/module/resnet50*.onnx --framework=5 --output=$HOME/module/out/onnx_resnet50
--soc_version=<soc_version>
```

关于参数的详细解释以及使用方法请参见[参数说明](#)，请使用与芯片名相对应的 `<soc_version>` 取值进行模型转换，然后再进行推理，具体使用芯片查询方法请参见 [7.2.4.1 --soc\\_version](#)。

- 步骤3** 若提示如下信息，则说明模型转换成功，若模型转换失败，则请参见《[故障处理（开放态）](#)》进行定位。

```
ATC run success
```

成功执行命令后，在 `--output` 参数指定的路径下，可查看离线模型（如：`onnx_resnet50.om`）。

模型编译时，若遇到AI CPU算子不支持某种数据类型导致编译失败的场景，可通过启用Cast算子自动插入特性快速将输入转换为算子支持的数据类型，从而实现网络的快速打通，详细流程请参见[9.3 开启AI CPU Cast算子自动插入特性](#)。

- 步骤4** （后续处理）如果想快速体验直接使用转换后的om离线模型文件进行推理，请准备好环境、om模型文件、符合模型输入要求的\*.bin格式的输入数据，单击[Link](#)，获取msame工具，参考该工具配套的README，进行体验。

----结束

## 开源框架的 Caffe 网络模型转换成离线模型

### 步骤1 获取Caffe网络模型。

您可以从以下链接中获取ResNet-50网络的模型文件 (\*.prototxt)、权重文件 (\*.caffemodel)，并以CANN软件包运行用户将获取的文件上传至开发环境任意目录，例如上传到\$HOME/module/目录下。

- ResNet-50网络的模型文件 (\*.prototxt)：单击[Link](#)下载该文件。
- ResNet-50网络的权重文件 (\*.caffemodel)：单击[Link](#)下载该文件。

### 步骤2 执行如下命令生成离线模型。（如下命令中使用的目录以及文件均为样例，请以实际为准）

```
atc --model=$HOME/module/resnet50.prototxt --weight=$HOME/module/resnet50.caffemodel --framework=0 --output=$HOME/module/out/caffe_resnet50 --soc_version=<soc_version>
```

关于参数的详细解释以及使用方法请参见[参数说明](#)，请使用与芯片名相对应的<soc\_version>取值进行模型转换，然后再进行推理，具体使用芯片查询方法请参见[7.2.4.1 --soc\\_version](#)。

### 步骤3 若提示如下信息，则说明模型转换成功，若模型转换失败，则请参见《[故障处理（开放态）](#)》进行定位。

```
ATC run success
```

成功执行命令后，在--output参数指定的路径下，可查看离线模型（如：caffe\_resnet50.om）。

模型编译时，若遇到AI CPU算子不支持某种数据类型导致编译失败的场景，可通过启用Cast算子自动插入特性快速将输入转换为算子支持的数据类型，从而实现网络的快速打通，详细流程请参见[9.3 开启AI CPU Cast算子自动插入特性](#)。

### 步骤4 （后续处理）如果想快速体验直接使用转换后的om离线模型文件进行推理，请准备好环境、om模型文件、符合模型输入要求的\*.bin格式的输入数据，单击[Link](#)，获取msame工具，参考该工具配套的README，进行体验。

----结束

## MindSpore 框架的网络模型转换成离线模型

### 步骤1 获取MindSpore框架的网络模型。

单击[Link](#)，获取ResNet-50网络的模型文件（仅支持\*.air格式的模型文件进行模型转换），并以CANN软件包运行用户将获取的文件上传到开发环境任意路径，例如\$HOME/module/目录下。

### 步骤2 执行如下命令生成离线模型。（如下命令中使用的目录以及文件均为样例，请以实际为准）

```
atc --model=$HOME/module/ResNet50.air --framework=1 --output=$HOME/module/out/ResNet50_mindspore --soc_version=<soc_version>
```

关于参数的详细解释以及使用方法请参见[参数说明](#)，请使用与芯片名相对应的<soc\_version>取值进行模型转换，然后再进行推理，具体使用芯片查询方法请参见[7.2.4.1 --soc\\_version](#)。

### 步骤3 若提示如下信息，则说明模型转换成功，若模型转换失败，则请参见《[故障处理（开放态）](#)》进行定位。

```
ATC run success
```

成功执行命令后，在--output参数指定的路径下，可查看离线模型（如：*ResNet50\_mindspore.om*）。

模型编译时，若遇到AI CPU算子不支持某种数据类型导致编译失败的场景，可通过启用Cast算子自动插入特性快速将输入转换为算子支持的数据类型，从而实现网络的快速打通，详细流程请参见[9.3 开启AI CPU Cast算子自动插入特性](#)。

**步骤4** （后续处理）如果想快速体验直接使用转换后的om离线模型文件进行推理，请准备好环境、om模型文件、符合模型输入要求的\*.bin格式的输入数据，单击[Link](#)，获取msame工具，参考该工具配套的README，进行体验。

----结束



# 4 ATC 简介

## ATC工具介绍

介绍ATC工具的功能架构以及模型转换过程中，各组件的交互流程。

## 运行流程

ATC工具运行前需要准备环境或者模型，本节给出ATC工具的运行流程以及每个流程中包括的必选或者可选操作。

## 关键概念

解释使用ATC工具过程中遇到的一些术语或者缩略语。

## 4.1 ATC 工具介绍

介绍ATC工具的功能架构以及模型转换过程中，各组件的交互流程。

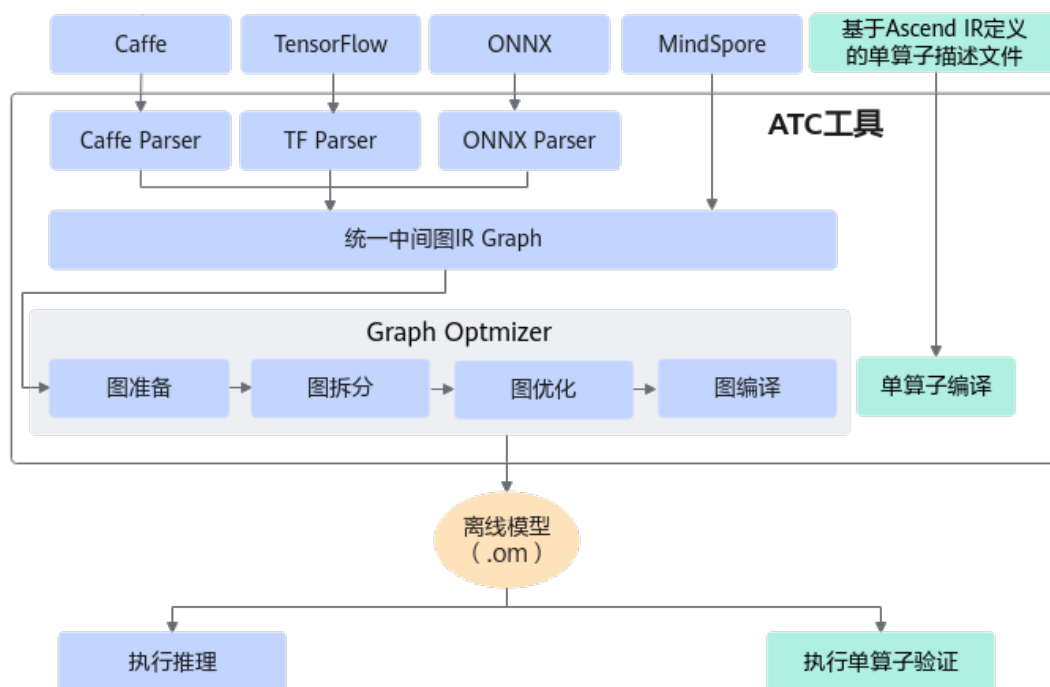
### ATC 简介

昇腾张量编译器（Ascend Tensor Compiler，简称ATC）是异构计算架构CANN体系下的模型转换工具，它可以将开源框架的网络模型以及Ascend IR定义的单算子描述文件（json格式）转换为昇腾AI处理器支持的.om格式离线模型。其功能架构如[图4-1](#)所示。

模型转换过程中，ATC会进行算子调度优化、权重数据重排、内存使用优化等具体操作，对原始的深度学习模型进行进一步的调优，从而满足部署场景下的高性能需求，使其能够高效执行在昇腾AI处理器上。



图 4-1 ATC 工具功能架构



其中：

- 开源框架网络模型场景：
  - a. 开源框架网络模型经过 Parser 解析后，转换为中间态 IR Graph。
  - b. 中间态 IR 经过图准备、图拆分、图优化、图编译等一系列操作后，转成适配昇腾 AI 处理器的离线模型（此处图指网络模型拓扑图）。
  - c. 转换后的离线模型上传到板端环境，通过 AscendCL 接口加载模型文件实现推理过程。

用户也可以将开源框架网络模型转换后的离线模型转成 json 文件，方便查看相关参数；也可以直接将开源框架网络模型通过 ATC 工具转成 json 文件，查看相关参数。

- 单算子描述文件场景下：

Ascend IR 定义的单算子描述文件（json 格式）通过 ATC 工具进行单算子编译后，转成适配昇腾 AI 处理器的单算子离线模型，然后上传到板端环境，通过 AscendCL 接口加载单算子模型文件用于验证单算子功能。

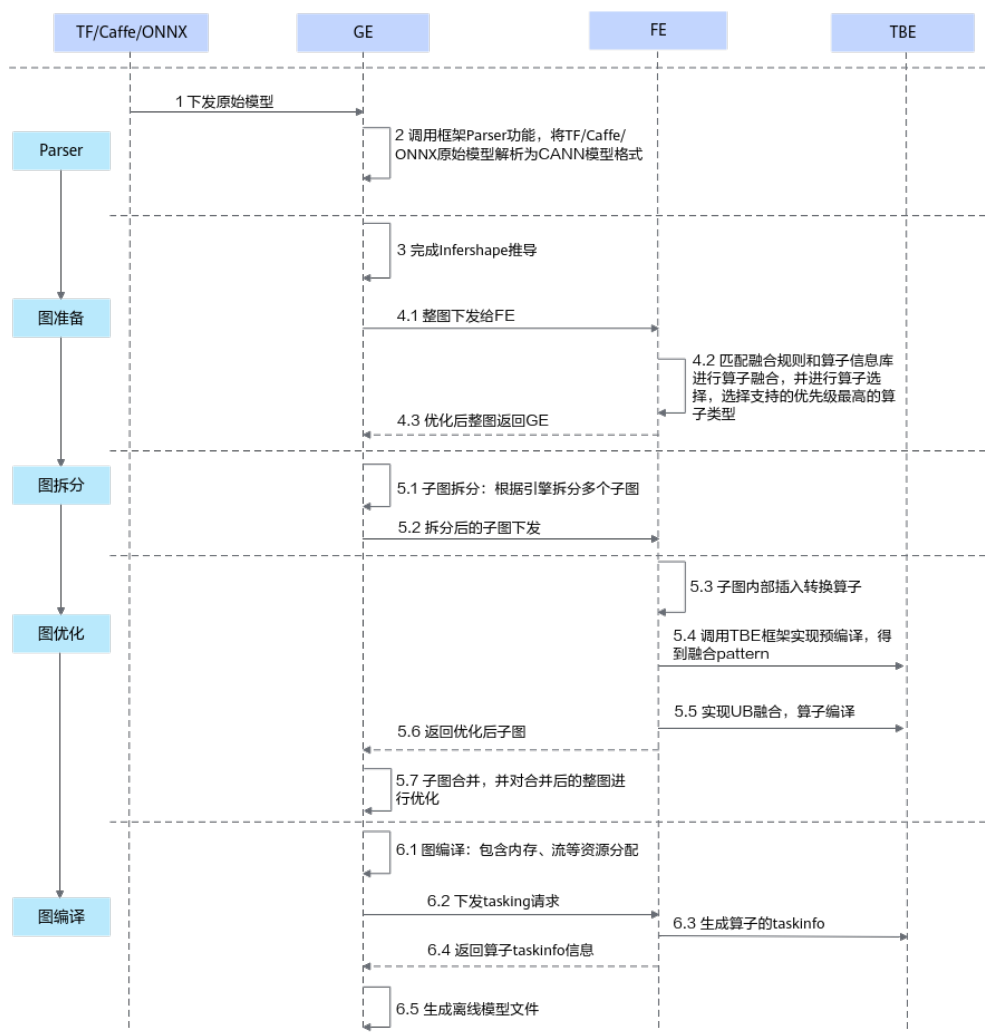
## 模型转换交互流程

下面以开源框架网络模型转换为 .om 离线模型为例，详细介绍模型转换过程中与周边模块的交互流程。

根据网络模型中算子计算单元的不同，分为 TBE（Tensor Boost Engine）算子、AI CPU 算子，TBE 算子在 AI Core 上运行，AI CPU 算子在 AI CPU 上运行。在 TBE 算子、AI CPU 算子的模型转换交互流程中，虽然都涉及图准备、图拆分、图优化、图编译等节点，但由于两者的计算单元不同，因此涉及交互的内部模块也有所不同，请参见下图。关于算子类型、基本概念等详细介绍请参见《[TBE&AI CPU 算子开发指南（开放态）](#)》。

- TBE算子模型转换交互流程

图 4-2 TBE 算子模型转换交互流程



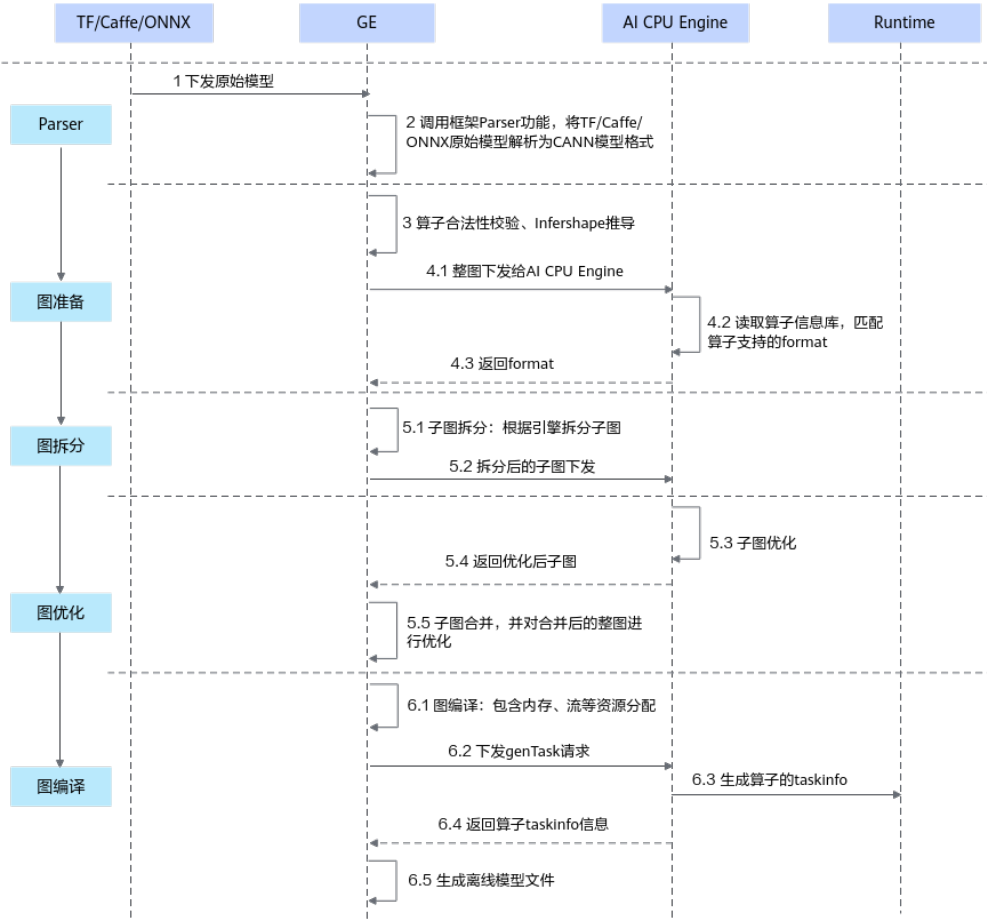
- 调用框架Parser功能，将主流框架的模型格式转换成CANN模型格式。
- 图准备阶段：该阶段会完成原图优化以及Infershape推导（设置算子输出的shape和dtype）等功能。

原图优化时：GE（Graph Engine，基于昇腾AI软件栈对不同的机器学习框架提供统一的IR接口，对接上层网络模型框架，例如Tensorflow、PyTorch等，GE的主要功能包括图准备、图拆分、图优化、图编译、图加载、图执行和图管理等。）向FE发送图优化请求，并将图下发给FE，FE匹配融合规则进行图融合，并进行算子选择，选择优先级最高的算子类型进行算子匹配，最后将优化后的整图返回给GE。

- 图拆分阶段：GE根据图中数据将图拆分为多个子图。
- 图优化阶段：GE将拆分后的子图下发给FE，FE首先在子图内部插入转换算子，然后按照当前子图流程进行TBE算子预编译，对TBE算子进行UB（Unified Buffer）融合，并根据算子信息库中算子信息找到算子实现将其编译成算子kernel（算子的\*.o与\*.json），最后将优化后子图返回给GE。优化后的子图合并为整图，再进行整图优化。

- e. 图编译阶段：GE进行图编译，包含内存分配、流资源分配等，并向FE发送tasking请求，FE返回算子的taskinfo信息给GE，图编译完成之后生成适配昇腾AI处理器的离线模型文件（\*.om）。
- AI CPU算子模型转换交互流程

图 4-3 AI CPU 算子模型转换交互流程



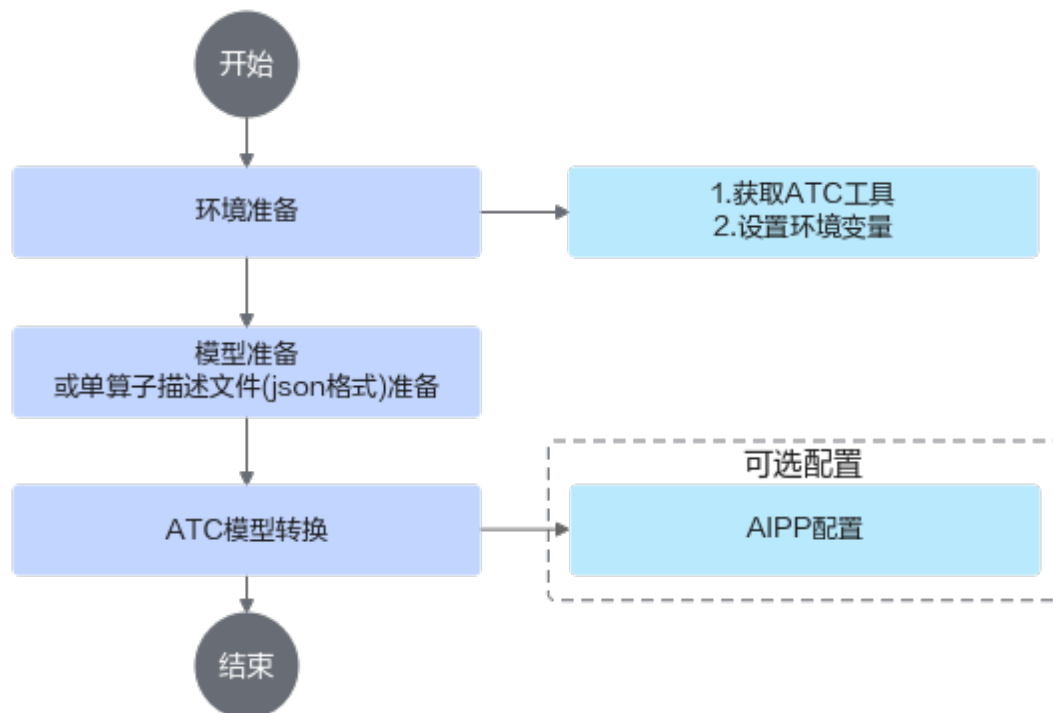
- a. 调用框架Parser功能，将主流框架的模型格式转换成CANN模型格式。
- b. 图准备阶段：该阶段会完成算子基本参数校验以及Infershape推导（设置算子输出的shape和dtype）等功能。  
另外，GE将整图下发给AI CPU Engine，AI CPU Engine读取算子信息库，匹配算子支持的format，并将format返回给GE。
- c. 图拆分阶段：GE根据图中数据将图拆分为多个子图。
- d. 图优化阶段：GE将拆分后的子图下发给AI CPU Engine，AI CPU Engine进行子图优化，并将优化后子图返回给GE。  
优化后的子图合并为整图，再进行整图优化。
- e. 图编译阶段：GE进行图编译，包含内存分配、流资源分配等，并向AI CPU Engine发送genTask请求，AI CPU Engine返回算子的taskinfo信息给GE，图编译完成之后生成适配昇腾AI处理器的离线模型文件（\*.om）。

## 4.2 运行流程

ATC工具运行前需要准备环境或者模型，本节给出ATC工具的运行流程以及每个流程中包括的必选或者可选操作。

使用ATC工具进行模型转换的运行流程如图4-4所示。

图 4-4 运行流程



1. 使用ATC工具之前，请先在开发环境安装CANN软件包，获取相关路径下的ATC工具，详细说明请参见[获取ATC工具](#)。
2. 准备要进行转换的模型或单算子描述文件，并上传到开发环境。
3. 使用ATC工具进行模型转换，在配置相关参数时，根据实际情况选择是否进行[AIPP配置](#)。

## 4.3 关键概念

解释使用ATC工具过程中遇到的一些术语或者缩略语。

- AIPP

AIPP ( Artificial Intelligence Pre-Processing ) AI预处理，是昇腾AI处理器提供的硬件图像预处理模块，包括色域转换，图像归一化（减均值/乘系数）和抠图（指定抠图起始点，抠出神经网络需要大小的图片）等功能。

Atlas 200/300/500 推理产品、Atlas 训练系列产品场景：DVPP模块输出的图片多为对齐后的YUV420SP类型，不支持输出RGB图片；如果模型需要RGB格式的图片，业务流需要使用AIPP模块转换对齐后YUV420SP类型图片的格式，并抠出模型需要的输入图片。

- YUV420SP  
有损图像颜色编码格式，常用为YUV420SP\_UV、YUV420SP\_VU两种格式。
- 知识库  
存储调优后的Schedule，在后续算子编译中直接使用。
- cost model  
评估器，调优过程中如果没有命中知识库，则通过cost model评估tiling空间中tiling的优劣，选择最优tiling数据。
- 数据排布格式（Format）  
Format为数据的物理排布格式，定义了解读数据的维度，比如1D、2D、3D、4D、5D等。
  - NCHW和NHWC  
在深度学习框架中，多维数据通过多维数组存储，比如卷积神经网络的特征图（Feature Map）通常用四维数组保存，即4D，4D格式解释如下：
    - N：Batch数量，例如图像的数目。
    - H：Height，特征图高度，即垂直高度方向的像素个数。
    - W：Width，特征图宽度，即水平宽度方向的像素个数。
    - C：Channels，特征图通道，例如彩色RGB图像的Channels为3。由于数据只能线性存储，因此这四个维度有对应的顺序。不同深度学习框架会按照不同的顺序存储特征图数据，比如Caffe，排列顺序为[Batch, Channels, Height, Width]，即NCHW；TensorFlow中，排列顺序为[Batch, Height, Width, Channels]，即NHWC。  
如图4-5所示，以一张格式为RGB的图片为例，NCHW中，C排列在外层，每个通道内，像素紧挨在一起，实际存储的是“RRRRRRGGGGGGBBBBBB”，即同一通道的所有像素值顺序存储在一起；而NHWC中C排列在最内层，每个通道内，像素间隔挨在一起，实际存储的则是“RGBRGBRGBRGBRGB”，即多个通道的同一位置的像素值顺序存储在一起。
  - NC1HWC0  
昇腾AI处理器中，为了提高通用矩阵乘法（GEMM）运算数据块的访问效率，所有张量数据统一采用NC1HWC0的五维数据格式，如下图所示。

图 4-5 NCHW 和 NHWC

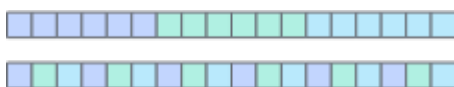
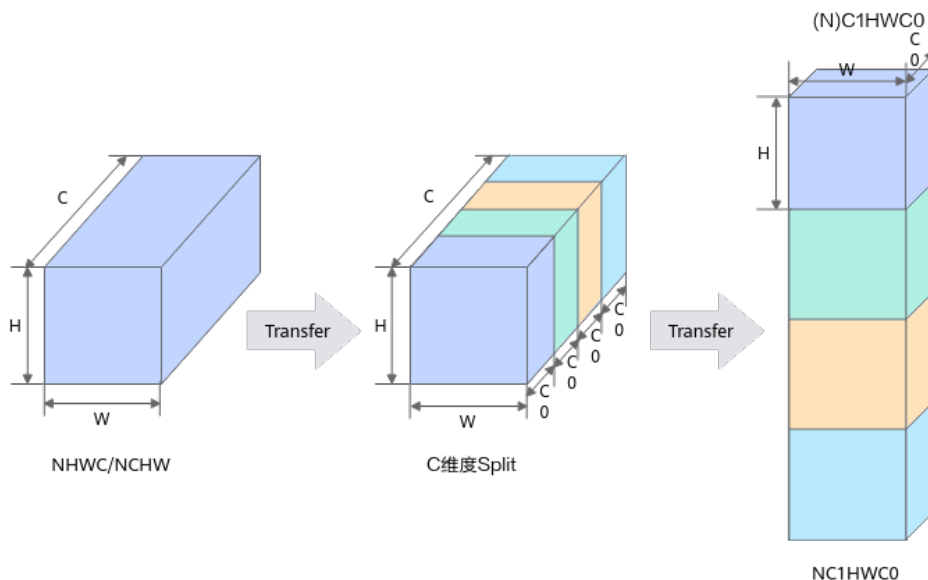


图 4-6 NC1HWC0



其中C0与微架构强相关，等于AI Core中矩阵计算单元的大小； $C1 = (C + C0 - 1) / C0$ ，如果结果不整除，向上取整。

例如：NHWC -> NC1HWC0的转换过程为：

- 将NHWC数据在C维度进行分割，变成C1份NHWC0。
- 将C1份NHWC0在内存中连续排列，由此变成NC1HWC0。

NHWC->NC1HWC0的转换场景示例：

- 首层RGB图像通过AIPP转换为NC1HWC0格式。
- 中间层Feature Map每层输出为NC1HWC0格式，在搬运过程中需要重排。

# 5 初级功能

原始模型文件或离线模型转成json文件  
离线模型支持动态BatchSize/动态分辨率  
离线模型支持动态维度  
自定义离线模型的输入输出数据类型  
借助离线模型查看软件基础版本号

## 5.1 原始模型文件或离线模型转成 json 文件

### 场景介绍

如果用户不方便查看原始模型或离线模型的参数信息时，可以将原始模型或离线模型转成json文件进行查看。

### 转换方法

本章节以TensorFlow框架ResNet-50网络模型为例进行演示，参见[步骤1](#)获取原始模型文件。

- 原始模型文件转json文件

该场景下[7.2.2.3 --om](#)参数需要指定为原始模型文件，命令示例如下：

```
atc --mode=1 --om=$HOME/module/resnet50_tensorflow*.pb --json=$HOME/module/out/tf_resnet50.json --framework=3
```

- 离线模型转json文件

该场景下需要先将原始模型转成离线模型，然后再执行离线模型转成json的操作。

- 原始模型转成离线模型，命令示例如下：

```
atc --model=$HOME/module/resnet50_tensorflow*.pb --framework=3 --output=$HOME/module/out/tf_resnet50 --soc_version=<soc_version>
```

- 离线模型转成json文件，命令示例如下：

```
atc --mode=1 --om=$HOME/module/out/tf_resnet50.om --json=$HOME/module/out/tf_resnet50.json
```

关于参数的详细解释以及使用方法请参见[参数说明](#)。若提示如下信息，则说明转换成功，若模型转换失败，则请参见《[故障处理（开放态）](#)》进行定位。



ATC run success

成功执行命令后，在--json参数指定的路径下，可查看转换后的json文件信息。

## 5.2 离线模型支持动态 BatchSize/动态分辨率

### 场景介绍

某些推理场景，如检测出目标后再执行目标识别网络，由于目标个数不固定导致目标识别网络输入BatchSize不固定。如果每次推理都按照最大的BatchSize或最大分辨率进行计算，会造成计算资源浪费。因此，模型转换需要支持动态BatchSize和动态分辨率的设置，使用ATC工具时，通过7.2.2.8 --dynamic\_batch\_size参数设置支持的BatchSize档位，通过7.2.2.9 --dynamic\_image\_size参数设置支持的分辨率档位。

### 转换方法

如下转换示例以TensorFlow框架ResNet50网络模型为例进行演示，参见步骤1获取原始模型文件。

- 步骤1** 以CANN软件包运行用户登录开发环境，并将模型转换过程中用到的模型文件 (\*.pb) 上传到开发环境任意路径，例如上传到\$HOME/module/目录下。
- 步骤2** atc命令中加入7.2.2.8 --dynamic\_batch\_size参数或者7.2.2.9 --dynamic\_image\_size，执行如下命令生成离线模型。（如下命令中使用的目录以及文件均为样例，请以实际为准）

- 动态BatchSize  
atc --model=\$HOME/module/resnet50\_tensorflow\*.pb --framework=3 --output=\$HOME/module/out/tf\_resnet50 --soc\_version=<soc\_version> --input\_shape="Placeholder:-1,224,224,3" --dynamic\_batch\_size="1,2,4,8"
- 动态分辨率  
atc --model=\$HOME/module/resnet50\_tensorflow\*.pb --framework=3 --output=\$HOME/module/out/tf\_resnet50 --soc\_version=<soc\_version> --input\_shape="Placeholder:1,-1,-1,3" --dynamic\_image\_size="224,224,448,448"

关于参数的详细解释以及使用方法请参见参数说明。若提示如下信息，则说明模型转换成功，若模型转换失败，则请参见《故障处理（开放态）》进行定位。

ATC run success

成功执行命令后，在--output参数指定的路径下，可查看离线模型（如：tf\_resnet50.om）。

模型转换完成后，在生成的om模型中，会新增一个输入，在模型推理时通过该新增的输入提供具体的Batch值（或分辨率值）。例如，a输入的BatchSize是动态的（或分辨率是动态的），在om模型中，会有与a对应的b输入来描述a的BatchSize（或分辨率取值）。

----结束

## 5.3 离线模型支持动态维度

### 场景介绍

为支持Transformer等网络在输入格式的维度不确定的场景，通过7.2.2.10 --dynamic\_dims参数实现ND格式下任意维度的档位设置。



ND表示支持任意格式，当前N<=4。

## 转换方法

本章节以TensorFlow框架ResNet50网络模型为例进行演示，参见[步骤1](#)获取原始模型文件。

**步骤1** 以CANN软件包运行用户登录开发环境，并将支持设置动态维度的模型文件上传到开发环境任意路径，例如上传到\$HOME/module/目录下。

**步骤2** atc命令中加入[7.2.2.10 --dynamic\\_dims](#)等相关参数，执行如下命令生成离线模型。（如下命令中使用的目录以及文件均为样例，请以实际为准）

```
atc --model=$HOME/module/resnet50_tensorflow*.pb --framework=3 --output=$HOME/module/out/
tf_resnet50 --soc_version=<soc_version> --input_shape="Placeholder:-1,-1,-1,3" --
dynamic_dims="1,224,224,8,448,448" --input_format=ND
```

关于参数的详细解释以及使用方法请参见[参数说明](#)。若提示如下信息，则说明模型转换成功，若模型转换失败，则请参见《[故障处理（开放态）](#)》进行定位。

```
ATC run success
```

成功执行命令后，在--output参数指定的路径下，可查看离线模型。

模型转换完成后，在生成的om模型中，会新增一个输入，在模型推理时通过该新增的输入提供具体的维度值。例如，a输入的维度为动态的，在om模型中，会有与a对应的b输入来描述a的维度值。

----结束

## 5.4 自定义离线模型的输入输出数据类型

### 场景介绍

模型转换时支持指定网络的输入节点、输出节点的Datatype、Format、模型转换支持精度选择等关键参数。

假如，针对TensorFlow框架ResNet-50网络模型，要求转换后离线模型的输入数据为FP16类型，指定MaxPoolWithArgmax算子作为输出算子（对应的节点名称为fp32\_vars/MaxPoolWithArgmax），并且指定该输出节点的数据类型为FP16。该场景下就需要分别使用[7.3.1.2 --input\\_fp16\\_nodes](#)、[7.3.1.1 --out\\_nodes](#)、[7.2.3.2 --output\\_type](#)等参数来实现上述功能。

### 转换方法

本章节以TensorFlow框架ResNet-50网络模型为例进行演示，参见[步骤1](#)获取原始模型文件。

**步骤1** 以CANN软件包运行用户登录开发环境，并将模型转换过程中用到的模型文件（\*.pb）上传到开发环境任意路径，例如上传到\$HOME/module/目录下。

**步骤2** atc命令中加入[7.3.1.2 --input\\_fp16\\_nodes](#)、[7.3.1.1 --out\\_nodes](#)、[7.2.3.2 --output\\_type](#)参数，执行如下命令生成离线模型。（如下命令中使用的目录以及文件均为样例，请以实际为准）

```
atc --model=$HOME/module/resnet50_tensorflow_1.7.pb --framework=3 --output=$HOME/module/out/
tf_resnet50 --soc_version=<soc_version> --input_fp16_nodes="Placeholder" --out_nodes="fp32_vars/
MaxPoolWithArgmax:0" --output_type="fp32_vars/MaxPoolWithArgmax:0:FP16"
```

关于参数的详细解释以及使用方法请参见[参数说明](#)。若提示如下信息，则说明模型转换成功，若模型转换失败，则请参见《[故障处理（开放态）](#)》进行定位。

```
ATC run success
```

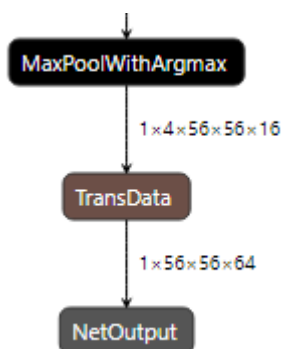
成功执行命令后，在output参数指定的路径下，可查看离线模型（如：tf\_resnet50.om）。

**步骤3** （可选）如果用户想查看转换后离线模型中上述指定节点以及指定数据类型相关信息，则可以将上述离线模型转成json文件查看，命令如下：

```
atc --mode=1 --om=$HOME/module/out/tf_resnet50.om --json=$HOME/module/out/tf_resnet50.json
```

图5-1为MaxPoolWithArgmax算子作为模型输出算子的示意图。

图 5-1 指定某个算子为离线模型输出



----结束

## 5.5 借助离线模型查看软件基础版本号

### 场景介绍

不同软件基础版本号，由于软件功能差异，所转换出的离线模型功能也有差异，该场景下建议用户使用匹配软件版本的ATC工具重新进行模型转换。假如用户已有转换好的离线模型，想查看使用的软件基础版本号，则可以参见该章节完成。

### 查看方法

**步骤1** 获取已经转换好的离线模型，例如tf\_resnet50.om，并以CANN软件包运行用户将其上传至开发环境任意目录，例如上传到\$HOME/module/目录下。

**步骤2** 将离线模型转成json文件：

```
atc --mode=1 --om=$HOME/module/tf_resnet50.om --json=$HOME/module/out/tf_resnet50.json
```

在转换后的json文件中，可以查看原始模型转换为该离线模型时，使用的基础版本号，示例如下：（如下示例中的版本号都为样例，请以实际查询的为准）

说明：离线模型转换为json文件可以查看基础版本号，必须保证使用的软件为1.77.22.6.220及之后的版本，之前版本无法查阅该信息；atc\_cmdline参数必须为1.78.23.34.230及之后的软件版本方可查阅。

```
{
  "key": "opp_version",
  "value": {
    "s": "<version>"
  }
}
```

```
    }  
  },  
  ...  
  {  
    "key": "atc_version",  
    "value": {  
      "s": "<version>"  
    }  
  },  
  ...  
  
  {  
    "key": "atc_cmdline",  
    "value": {  
      "s": "xxx/atc.bin --model ./resnet50_tensorflow*.pb --framework 3 --output ./out/tf_resnet50 --  
soc_version <soc_version>"  
    }  
  },  
  ...  
  {  
    "key": "soc_version",  
    "value": {  
      "s": "<soc_version>"  
    }  
  },  
  ...  
}
```

----结束

# 6 高级功能

AIPP使能

单算子模型转换

## 6.1 AIPP 使能

### 6.1.1 什么是 AIPP

本节介绍什么是AIPP，AIPP分类以及包括的特性。

AIPP ( Artificial Intelligence Pre-Processing ) 人工智能预处理，用于在AI Core上完成数据预处理，包括改变图像尺寸、色域转换（转换图像格式）、减均值/乘系数（改变图像像素），数据预处理之后再行真正的模型推理。

该模块功能与DVPP相似，都可以实现媒体数据处理的功能，两者的功能范围有重合，比如改变图像尺寸、转换图像格式，但功能范围也有不同的，比如DVPP可以做图像编解码、视频编解码，AIPP可以做归一化配置。与DVPP不同的是，AIPP主要用于在AI Core上完成数据预处理，DVPP是昇腾AI处理器内置的图像处理单元，通过AscendCL媒体数据处理接口提供强大的媒体处理硬加速能力。

AIPP、DVPP可以分开独立使用，也可以组合使用。组合使用场景下，一般先使用DVPP对图片/视频进行解码、抠图、缩放等基本处理，但由于DVPP硬件上的约束，DVPP处理后的图片格式、分辨率有可能不满足模型的要求，因此还需要再经过AIPP进一步做色域转换、抠图、填充等处理。

AIPP根据配置方式不同，分为静态AIPP和动态AIPP；如果要将原始图片输出为满足推理要求的图片格式，则需要使用色域转换功能；如果要输出固定大小的图片，则需要使用AIPP提供的Crop（抠图）、Padding（补边）功能。

### 静态 AIPP 和动态 AIPP

在使能AIPP功能时，您只能选择静态AIPP或动态AIPP方式来处理图片，不能同时配置静态AIPP和动态AIPP两种方式，使能AIPP时可以通过[aipp\\_mode](#)参数控制。具体配置示例请参见[6.1.3 AIPP配置示例](#)，关于参数解释请参见[6.1.8 配置文件模板](#)。

- 静态AIPP：模型转换时设置AIPP模式为静态，同时设置AIPP参数，模型生成后，AIPP参数值被保存在离线模型中，每次模型推理过程采用固定的AIPP预处理参数进行处理，而且在之后的推理过程中无法通过业务代码进行直接的修改。

如果使用静态AIPP方式，多batch情况下共用同一份AIPP参数。

- 动态AIPP：模型转换时设置AIPP模式为动态，每次在执行推理前，根据需求动态修改AIPP参数值，然后在模型执行时可使用不同的AIPP参数。动态AIPP参数值会根据需求在不同的业务场景下选用合适的参数（如不同摄像头采用不同的归一化参数，输入图片格式需要兼容YUV420和RGB等）。

如果模型转换时设置了动态AIPP，则使用应用工程进行模型推理时，需要在CANN AscendCL应用软件开发指南（C&C++）（开放态）》手册中的“AscendCL API参考>模型加载与执行>aclmdlSetInputAIPP”章节。

如果使用动态AIPP方式，多batch使用不同的参数，体现在动态参数结构体中，每个batch可以配置不同的crop等参数。关于动态参数结构体，请参见[动态AIPP的参数输入结构](#)。

AIPP支持的图像输入格式包括：YUV420SP\_U8、RGB888\_U8、XRGB8888\_U8、YUV400\_U8。

## 色域转换

色域转换，用于将输入的图片格式，转换为模型需要的图片格式，在使能AIPP功能时，通过[csc\\_switch](#)参数控制色域转换功能是否开启，参数解释请参见[6.1.8 配置文件模板](#)。

一旦确认了AIPP处理前与AIPP处理后的图片格式，即可确定色域转换其他相关的参数值，本文提供相关模板可以供用户使用，无需再次修改，配置示例请参见[6.1.4 色域转换配置说明](#)。

## 改变图像尺寸

AIPP功能中的改变图像尺寸操作由Crop（抠图）、Padding（补边）完成，分别对应配置模板中的crop、padding参数。参数解释请参见[6.1.8 配置文件模板](#)。

关于该功能的详细说明以及AIPP参数配置示例请参见[6.1.6 Crop/Padding配置说明](#)。

### 6.1.2 如何使能 AIPP

通过在模型转换过程中开启AIPP功能，可以在推理之前就完成所有的数据处理；由于用的是专门的加速模块实现并保证性能，从而可以不让图像处理成为推理阶段的瓶颈，图像处理方式比较灵活。本章节给出如何在模型转换阶段开启AIPP功能。

本章节以TensorFlow框架ResNet50网络模型为例，演示如何通过模型转换使能静态AIPP功能，使能AIPP功能后，若实际提供给模型推理的测试图片不满足要求（包括图片格式，图片尺寸等），经过模型转换后，会输出满足模型要求的图片，并将该信息固化到转换后的离线模型中（模型转换后AIPP功能会以aipp算子形式插入离线模型中）。

ResNet50网络模型要求的图片格式为RGB，图片尺寸为224\*224，另外，假设提供给模型推理的测试图片尺寸为250\*250，图片格式为YUV420SP，有效数据区域从左上角(0, 0)像素开始，使能AIPP过程中所需操作如[表6-1](#)分析所示。

表 6-1 场景分析

分类	ResNet50网络模型要求	实际提供给模型推理的测试图片	所需操作
图片格式	RGB	YUV420SP	该场景下需要开启AIPP的色域转换功能，将YUV420SP格式转成模型要求的RGB格式，关于色域转换功能详细说明请参见6.1.4 色域转换配置说明。
图片尺寸	224*224	250*250	提供的测试图片尺寸250*250大于224*224，该场景下需要开启AIPP抠图功能，并且抠图起始位置水平、垂直方向坐标load_start_pos_h、load_start_pos_w为0，执行推理时，将从(0, 0)点开始选取224*224区域的数据。

详细实现步骤如下：

步骤1 获取TensorFlow网络模型。

单击[Link](#)，根据页面提示获取ResNet50网络的模型文件（\*.pb），并以CANN软件包运行用户将获取的文件上传至开发环境任意目录，例如上传到\$HOME/module/目录下。

步骤2 构造AIPP配置文件insert\_op.cfg。

静态AIPP配置模板主要由如下几部分组成：AIPP配置模式（静态AIPP或者动态AIPP），原始图片信息（包括图片格式，以及图片尺寸），改变图片尺寸（抠图，补边）、色域转换功能等，如下分别介绍如何进行配置。

1. AIPP配置模式由aipp\_mode参数决定，静态场景下的配置示例如下：

aipp\_mode : static #static表示配置为静态AIPP
2. 配置原始图片信息。

input\_format : YUV420SP\_U8 #输入给AIPP的原始图片格式  
src\_image\_size\_w : 250 #输入给AIPP的原始图片宽高  
src\_image\_size\_h : 250
3. 改变图片尺寸。

改变图片尺寸由抠图和补边等功能完成，本示例需要配置抠图起始位置，抠图后的图片大小等信息，若抠图后图片尺寸仍旧不满足模型要求，还需要配置补边功能。

而AIPP提供了更为方便的配置方式，就是若开启抠图功能，并且不配置补边功能，抠图大小可以取值为0或者不配置，此时抠图大小的宽和高来自模型--input\_shape中的宽和高。本示例中我们不配置抠图大小，配置示例如下：

crop: true #抠图开关  
load\_start\_pos\_h: 0 #抠图起始位置水平、垂直方向坐标  
load\_start\_pos\_w: 0
4. 色域转换功能。

色域转换功能由csc\_switch参数控制，并通过色域转换系数matrix\_r\*c\*、通道交换rbuv\_swap\_switch等参数配合使用。AIPP提供了一个比较方便的功能，就是一旦确认了AIPP处理前与AIPP处理后的图片格式，即可确定色域转换相关的参数值，**用户无需修改**，即上述参数都可以直接从模板中进行复制，模板示例以及更多配置模板请参见6.1.4 色域转换配置说明。如下为该场景下的配置示例：



```
csc_switch : true          #色域转换开关, true表示开启色域转换
rbuv_swap_switch : false   #通道交换开关 ( R通道与B通道交换开关/U通道与V通道交换 ), 本例
                           #不涉及两个通道的交换, 故设置为false, 默认为false
matrix_r0c0 : 256          #色域转换系数
matrix_r0c1 : 0
matrix_r0c2 : 359
matrix_r1c0 : 256
matrix_r1c1 : -88
matrix_r1c2 : -183
matrix_r2c0 : 256
matrix_r2c1 : 454
matrix_r2c2 : 0
input_bias_0 : 0
input_bias_1 : 128
input_bias_2 : 128
```

将上述所有的参数组合到`insert_op.cfg`文件中, 即为我们需要构造的AIPP配置文件, 完整示例如下:

```
aipp_op {
  aipp_mode : static      #AIPP配置模式
  input_format : YUV420SP_U8 #输入给AIPP的原始图片格式
  src_image_size_w : 250   #输入给AIPP的原始图片宽高
  src_image_size_h : 250
  crop : true             #抠图开关, 用于改变图片尺寸
  load_start_pos_h : 0     #抠图起始位置水平、垂直方向坐标
  load_start_pos_w : 0
  csc_switch : true        #色域转换开关
  matrix_r0c0 : 256        #色域转换系数, 用户无需修改
  matrix_r0c1 : 0
  matrix_r0c2 : 359
  matrix_r1c0 : 256
  matrix_r1c1 : -88
  matrix_r1c2 : -183
  matrix_r2c0 : 256
  matrix_r2c1 : 454
  matrix_r2c2 : 0
  input_bias_0 : 0
  input_bias_1 : 128
  input_bias_2 : 128
}
```

您可以根据[6.1.3 AIPP配置示例](#)或[6.1.9 典型场景样例参考](#)章节获取更多场景AIPP配置示例, 如果上述示例仍旧无法满足要求, 则需要参见[6.1.8 配置文件模板](#)自行构造配置文件。将上述`insert_op.cfg`文件上传到ATC工具所在Linux服务器。

**步骤3** atc命令中加入[7.3.1.3 --insert\\_op\\_conf](#)参数, 用于插入aipp预处理算子, 执行如下命令生成离线模型。(如下命令中使用的目录以及文件均为样例, 请以实际为准)

```
atc --model=$HOME/module/resnet50_tensorflow*.pb --framework=3 --output=$HOME/module/out/
tf_resnet50 --soc_version=<soc_version> --insert_op_conf=$HOME/module/insert_op.cfg
```

关于参数的详细解释以及使用方法请参见[参数说明](#)。若提示如下信息, 则说明模型转换成功。

```
ATC run success
```

成功执行命令后, 在`--output`参数指定的路径下, 可查看离线模型(如:`tf_resnet50.om`)。

**步骤4** (可选) 如果用户想查看转换后离线模型中aipp算子的相关信息, 则可以将上述离线模型转成json文件查看, 命令如下:

```
atc --mode=1 --om=$HOME/module/out/tf_resnet50.om --json=$HOME/module/out/tf_resnet50.json
```

如下为json文件中带有aipp信息的样例(如下样例中所有aipp属性值都为样例, 请以用户实际构造的配置文件为准):

```
{
  "key": "aipp",
```

```
"value": {
  "func": {
    "attr": [
      {
        "key": "mean_chn_0",
        "value": {
          "i": 0
        }
      },
      {
        "key": "mean_chn_1",
        "value": {
          "i": 0
        }
      },
      {
        "key": "mean_chn_2",
        "value": {
          "i": 0
        }
      },
      {
        "key": "mean_chn_3",
        "value": {
          "i": 0
        }
      },
      {
        "key": "csc_switch",
        "value": {
          "b": true
        }
      },
      {
        "key": "input_format",
        "value": {
          "i": 1
        }
      },
      {
        "key": "input_bias_0",
        "value": {
          "i": 0
        }
      },
      {
        "key": "input_bias_1",
        "value": {
          "i": 128
        }
      },
      {
        "key": "input_bias_2",
        "value": {
          "i": 128
        }
      },
      {
        "key": "aipp_mode",
        "value": {
          "i": 1
        }
      },
      {
        "key": "src_image_size_h",
        "value": {
          "i": 0
        }
      }
    ]
  }
}
```



```
{
  "key": "crop_size_h",
  "value": {
    "i": 0
  }
},
{
  "key": "matrix_r0c0",
  "value": {
    "i": 256
  }
},
{
  "key": "matrix_r0c1",
  "value": {
    "i": 443
  }
},
{
  "key": "matrix_r0c2",
  "value": {
    "i": 0
  }
},
{
  "key": "src_image_size_w",
  "value": {
    "i": 0
  }
},
{
  "key": "crop_size_w",
  "value": {
    "i": 0
  }
},
{
  "key": "rbuv_swap_switch",
  "value": {
    "b": false
  }
},
{
  "key": "padding",
  "value": {
    "b": false
  }
},
{
  "key": "ax_swap_switch",
  "value": {
    "b": false
  }
},
{
  "key": "top_padding_size",
  "value": {
    "i": 0
  }
},
{
  "key": "matrix_r1c0",
  "value": {
    "i": 256
  }
},
{
  "key": "matrix_r1c1",
  "value": {
```

```
        "i": -86
      }
    },
    {
      "key": "matrix_r1c2",
      "value": {
        "i": -178
      }
    },
    {
      "key": "resize",
      "value": {
        "b": false
      }
    },
    {
      "key": "resize_output_h",
      "value": {
        "i": 0
      }
    },
    {
      "key": "related_input_rank",
      "value": {
        "i": 0
      }
    },
    {
      "key": "load_start_pos_h",
      "value": {
        "i": 0
      }
    },
    {
      "key": "matrix_r2c0",
      "value": {
        "i": 256
      }
    },
    {
      "key": "matrix_r2c1",
      "value": {
        "i": 0
      }
    },
    {
      "key": "matrix_r2c2",
      "value": {
        "i": 350
      }
    },
    {
      "key": "resize_output_w",
      "value": {
        "i": 0
      }
    },
    {
      "key": "var_reci_chn_0",
      "value": {
        "f": "1"
      }
    },
    {
      "key": "var_reci_chn_1",
      "value": {
        "f": "1"
      }
    },
  ],
```

```
{
  "key": "var_recn_chn_2",
  "value": {
    "f": "1"
  }
},
{
  "key": "load_start_pos_w",
  "value": {
    "i": 0
  }
},
{
  "key": "var_recn_chn_3",
  "value": {
    "f": "1"
  }
},
{
  "key": "single_line_mode",
  "value": {
    "b": false
  }
},
{
  "key": "output_bias_0",
  "value": {
    "i": 16
  }
},
{
  "key": "output_bias_1",
  "value": {
    "i": 128
  }
},
{
  "key": "output_bias_2",
  "value": {
    "i": 128
  }
},
{
  "key": "right_padding_size",
  "value": {
    "i": 0
  }
},
{
  "key": "bottom_padding_size",
  "value": {
    "i": 0
  }
},
{
  "key": "min_chn_0",
  "value": {
    "f": "0"
  }
},
{
  "key": "min_chn_1",
  "value": {
    "f": "0"
  }
},
{
  "key": "min_chn_2",
  "value": {
```

```
        "f": "0"
      }
    },
    {
      "key": "min_chn_3",
      "value": {
        "f": "0"
      }
    },
    {
      "key": "crop",
      "value": {
        "b": false
      }
    },
    {
      "key": "cpadding_value",
      "value": {
        "f": "0"
      }
    },
    {
      "key": "left_padding_size",
      "value": {
        "i": 0
      }
    }
  ]
}
```

----结束

## 6.1.3 AIPP 配置示例

### 6.1.3.1 静态 AIPP 配置示例

AIPP配置文件支持定义多组AIPP配置，对不同的模型输入进行不同的AIPP处理，配置多组AIPP参数时，将一组AIPP配置放到一个aipp\_op配置项里；如果模型只有一个输入，则只需要配置第一组aipp\_op即可。

如下示例以网络模型为多输入时进行说明：

#### 须知

- 如果模型转换时，用户设置了[7.2.2.9 --dynamic\\_image\\_size](#)动态分辨率参数，既输入图片的宽和高不确定，同时又通过[7.3.1.3 --insert\\_op\\_conf](#)参数设置了静态AIPP功能：该场景下，AIPP配置文件中不能开启Crop和Padding功能，并且需要将配置文件中的src\_image\_size\_w和src\_image\_size\_h取值设置为0。
- 如果模型转换时，用户设置了[7.2.2.6 --input\\_shape](#)动态shape范围参数，同时又通过[7.3.1.3 --insert\\_op\\_conf](#)参数配置了AIPP功能，则AIPP输出的宽和高要在[7.2.2.6 --input\\_shape](#)所设置的范围内。

- 使用related\_input\_rank参数标识，对模型第几个输入进行AIPP处理，如下配置定义了两组AIPP参数，分别对模型第一个和第二个输入进行AIPP处理：

```
aipp_op {
  aipp_mode : static
  related_input_rank : 0 # 标识对第1个输入进行AIPP处理
```

```
src_image_size_w : 608
src_image_size_h : 608
crop : false
input_format : YUV420SP_U8
csc_switch : true
rbuv_swap_switch : false
matrix_r0c0 : 298
matrix_r0c1 : 0
matrix_r0c2 : 409
matrix_r1c0 : 298
matrix_r1c1 : -100
matrix_r1c2 : -208
matrix_r2c0 : 298
matrix_r2c1 : 516
matrix_r2c2 : 0
input_bias_0 : 16
input_bias_1 : 128
input_bias_2 : 128
mean_chn_0 : 104
mean_chn_1 : 117
mean_chn_2 : 123
}
aipp_op {
  aipp_mode : static
  related_input_rank : 1 # 标识对第2个输入进行AIPP处理
  src_image_size_w : 608
  src_image_size_h : 608
  crop : false
  input_format : YUV420SP_U8
  csc_switch : true
  rbuv_swap_switch : false
  matrix_r0c0 : 298
  matrix_r0c1 : 0
  matrix_r0c2 : 409
  matrix_r1c0 : 298
  matrix_r1c1 : -100
  matrix_r1c2 : -208
  matrix_r2c0 : 298
  matrix_r2c1 : 516
  matrix_r2c2 : 0
  input_bias_0 : 16
  input_bias_1 : 128
  input_bias_2 : 128
  mean_chn_0 : 104
  mean_chn_1 : 117
  mean_chn_2 : 123
}
```

- 使用related\_input\_name参数标识，对模型第几个输入进行AIPP处理，此处需要填写为模型输入的name（input对应的值）或者模型首层节点的输出（top参数对应的取值），该参数只适用于Caffe网络模型，且不能与related\_input\_rank参数同时使用。

如下配置定义了两组AIPP参数，分别对模型第一个和第二个输入进行AIPP处理：

```
aipp_op {
  aipp_mode : static
  related_input_name : "data" # 标识对第1个输入进行AIPP处理
  src_image_size_w : 608
  src_image_size_h : 608
  crop : false
  input_format : YUV420SP_U8
  csc_switch : true
  rbuv_swap_switch : false
  matrix_r0c0 : 298
  matrix_r0c1 : 0
  matrix_r0c2 : 409
  matrix_r1c0 : 298
  matrix_r1c1 : -100
  matrix_r1c2 : -208
}
```

```
matrix_r2c0 : 298
matrix_r2c1 : 516
matrix_r2c2 : 0
input_bias_0 : 16
input_bias_1 : 128
input_bias_2 : 128
mean_chn_0 : 104
mean_chn_1 : 117
mean_chn_2 : 123
}
aipp_op {
  aipp_mode : static
  related_input_name : "im_info" # 标识对第2个输入进行AIPP处理
  src_image_size_w : 608
  src_image_size_h : 608
  crop : false
  input_format : YUV420SP_U8
  csc_switch : true
  rbuv_swap_switch : false
  matrix_r0c0 : 298
  matrix_r0c1 : 0
  matrix_r0c2 : 409
  matrix_r1c0 : 298
  matrix_r1c1 : -100
  matrix_r1c2 : -208
  matrix_r2c0 : 298
  matrix_r2c1 : 516
  matrix_r2c2 : 0
  input_bias_0 : 16
  input_bias_1 : 128
  input_bias_2 : 128
  mean_chn_0 : 104
  mean_chn_1 : 117
  mean_chn_2 : 123
}
```

### 6.1.3.2 动态 AIPP 配置示例

AIPP配置文件支持定义多组AIPP配置，对不同的模型输入进行不同的AIPP处理，配置多组AIPP参数时，将一组AIPP配置放到一个aipp\_op配置项里；如果模型只有一个输入，则只需要配置第一组aipp\_op即可。

如下示例以网络模型为多输入时进行说明。

## 配置示例

### 须知

- 如果模型转换时，用户设置了7.2.2.8 `--dynamic_batch_size`动态Batch档位参数，同时又通过7.3.1.3 `--insert_op_conf`参数配置了动态AIPP功能：  
实际推理时，调用《[CANN AscendCL应用软件开发指南（C&C++）（开放态）](#)》手册中的“AscendCL API参考>模型加载与执行>`aclmdlSetInputAIPP`”接口，设置动态AIPP相关参数值时，需确保batchSize要设置为最大Batch数。
- 如果模型转换时，用户设置了7.2.2.9 `--dynamic_image_size`动态分辨率参数，同时又通过7.3.1.3 `--insert_op_conf`参数配置了动态AIPP功能：  
实际推理时，调用《[CANN AscendCL应用软件开发指南（C&C++）（开放态）](#)》手册中的“AscendCL API参考>模型加载与执行>`aclmdlSetInputAIPP`”接口，设置动态AIPP相关参数值时，不能开启Crop和Padding功能。该场景下，还需要确保通过CANN AscendCL应用软件开发指南（C&C++）（开放态）》手册中的“AscendCL API参考>模型加载与执行>`aclmdlSetDynamicHWSIZE`”接口设置的宽、高相等，都必须设置成动态分辨率最大档位的宽、高。
- 如果模型转换时，用户设置了7.2.2.6 `--input_shape`动态shape范围参数，同时又通过7.3.1.3 `--insert_op_conf`参数配置了AIPP功能，则AIPP输出的宽和高要在7.2.2.6 `--input_shape`所设置的范围内。

动态AIPP场景下，用户无需手动配置csc\_switch、rbuv\_swap\_switch等参数，根据如下配置文件配置好相关参数后，模型转换时，ATC会为动态AIPP新增一个模型输入（以下简称AippData）。

实际推理时，需要调用《[CANN AscendCL应用软件开发指南（C&C++）（开放态）](#)》手册中的“AscendCL API参考>模型加载与执行>`aclmdlSetInputAIPP`”接口，设置动态AIPP相关参数值，然后传给上述新增的AippData，AippData根据传入的参数值构造的结构体为[动态AIPP的参数输入结构](#)，该结构体无需用户手动处理。

```
aipp_op
{
    aipp_mode: dynamic
    related_input_rank: 0    # 标识对第1个输入进行AIPP处理
    max_src_image_size: 752640 # 输入图像最大的size，参数必填
}
aipp_op
{
    aipp_mode: dynamic
    related_input_rank: 1    # 标识对第2个输入进行AIPP处理
    max_src_image_size: 752640 # 输入图像最大的size，参数必填
}
```

## 动态 AIPP 的参数输入结构

根据[配置示例](#)配置好动态AIPP文件后，模型推理时为动态AIPP新增模型输入（AippData）传入参数值后，自动形成的结构体如下，该结构体无需用户手动处理：

```
typedef struct tagAippDynamicBatchPara
{
    int8_t cropSwitch;        //crop switch
    int8_t scfSwitch;        //resize switch
    int8_t paddingSwitch;    // 0: unable padding,
                            // 1: padding config value,sfr_filling_hblank_ch0 ~ sfr_filling_hblank_ch2
                            // 2: padding source picture data, single row/collumn copy
```

```

        // 3: padding source picture data, block copy
        // 4: padding source picture data, mirror copy
    int8_t rotateSwitch; //rotate switch, 0: non-rotate, 1: rotate 90°clockwise, 2: rotate 180°clockwise, 3:
    rotate 270° clockwise
    int8_t reserve[4];
    int32_t cropStartPosW; //the start horizontal position of cropping
    int32_t cropStartPosH; //the start vertical position of cropping
    int32_t cropSizeW; //crop width
    int32_t cropSizeH; //crop height
    int32_t scfInputSizeW; //input width of scf
    int32_t scfInputSizeH; //input height of scf
    int32_t scfOutputSizeW; //output width of scf
    int32_t scfOutputSizeH; //output height of scf
    int32_t paddingSizeTop; //top padding size
    int32_t paddingSizeBottom; //bottom padding size
    int32_t paddingSizeLeft; //left padding size
    int32_t paddingSizeRight; //right padding size
    int16_t dtcPixelMeanChn0; //mean value of channel 0
    int16_t dtcPixelMeanChn1; //mean value of channel 1
    int16_t dtcPixelMeanChn2; //mean value of channel 2
    int16_t dtcPixelMeanChn3; //mean value of channel 3
    uint16_t dtcPixelMinChn0; //min value of channel 0
    uint16_t dtcPixelMinChn1; //min value of channel 1
    uint16_t dtcPixelMinChn2; //min value of channel 2
    uint16_t dtcPixelMinChn3; //min value of channel 3
    uint16_t dtcPixelVarReciChn0; //sfr_dtc_pixel_variance_reci_ch0
    uint16_t dtcPixelVarReciChn1; //sfr_dtc_pixel_variance_reci_ch1
    uint16_t dtcPixelVarReciChn2; //sfr_dtc_pixel_variance_reci_ch2
    uint16_t dtcPixelVarReciChn3; //sfr_dtc_pixel_variance_reci_ch3
    int8_t reserve1[16]; //32B assign, for ub copy
}kAippDynamicBatchPara;
typedef struct tagAippDynamicPara
{
    uint8_t inputFormat; //input format: YUV420SP_U8/XRGB8888_U8/RGB888_U8
    //uint8_t outDataType; //output data type: CC_DATA_HALF,CC_DATA_INT8, CC_DATA_UINT8
    int8_t cscSwitch; //csc switch
    int8_t rbuvSwapSwitch; //rb/ub swap switch
    int8_t axSwapSwitch; //RGBA->ARGB, YUVA->AYUV swap switch
    int8_t batchNum; //batch parameter number
    int8_t reserve1[3];
    int32_t srcImageSizeW; //source image width
    int32_t srcImageSizeH; //source image height
    int16_t cscMatrixR0C0; //csc_matrix_r0_c0
    int16_t cscMatrixR0C1; //csc_matrix_r0_c1
    int16_t cscMatrixR0C2; //csc_matrix_r0_c2
    int16_t cscMatrixR1C0; //csc_matrix_r1_c0
    int16_t cscMatrixR1C1; //csc_matrix_r1_c1
    int16_t cscMatrixR1C2; //csc_matrix_r1_c2
    int16_t cscMatrixR2C0; //csc_matrix_r2_c0
    int16_t cscMatrixR2C1; //csc_matrix_r2_c1
    int16_t cscMatrixR2C2; //csc_matrix_r2_c2
    int16_t reserve2[3];
    uint8_t cscOutputBiasR0; //output bias for RGB to YUV, element of row 0, unsigned number
    uint8_t cscOutputBiasR1; //output bias for RGB to YUV, element of row 1, unsigned number
    uint8_t cscOutputBiasR2; //output bias for RGB to YUV, element of row 2, unsigned number
    uint8_t cscInputBiasR0; //input bias for YUV to RGB, element of row 0, unsigned number
    uint8_t cscInputBiasR1; //input bias for YUV to RGB, element of row 1, unsigned number
    uint8_t cscInputBiasR2; //input bias for YUV to RGB, element of row 2, unsigned number
    uint8_t reserve3[2];
    int8_t reserve4[16]; //32B assign, for ub copy
    kAippDynamicBatchPara aippBatchPara; //allow transfer several batch para.
} kAippDynamicPara;

```

## 6.1.4 色域转换配置说明

AIPP提供了更为方便的图像格式转换方式：色域转换，用于将输入的图片格式，转换为模型需要的图片格式，一旦确认了AIPP处理前与AIPP处理后的图片格式，即可确定色域转换相关的参数值（**matrix\_r\*c\*配置项的值是固定的，不需要调整**）。



例如：将JPEG格式的图像文件（如后缀为jpg、jpeg、JPG、JPEG的图像文件）转为RGB格式；将视频解码后的YUV格式数据转为RGB格式。而根据不同的彩色视频数字化标准又可以将视频格式分为BT-601标准清晰度视频格式（定义于SDTV标准中）和BT-709高清晰度视频格式（定义于HDTV标准中）。两种视频格式又分为NARROW和WIDE，其中：

Ranges:  
Y[16 ... 235]  
Cb/Cr[16 ... 240]  
R/G/B[0 ... 255]

NARROW取值范围为：

Ranges:  
Y/Cb/Cr[0 ... 255]  
R/G/B[0 ... 255]

WIDE取值范围为：

关于如何判断输入数据的标准，请参见[10.2 使用AIPP色域转换模型时如何判断视频流的格式标准](#)。

YUV格式的数据转为RGB格式可以视作如下公式展示的矩阵乘法，这其中的转换矩阵就是待配置的参数和偏移量。

```
# YUV转BGR:
| B | | matrix_r0c0 matrix_r0c1 matrix_r0c2 | | Y - input_bias_0 |
| G | = | matrix_r1c0 matrix_r1c1 matrix_r1c2 | | U - input_bias_1 | >> 8
| R | | matrix_r2c0 matrix_r2c1 matrix_r2c2 | | V - input_bias_2 |
```

在AIPP处理前，针对模型输入的图片或视频（各颜色编码方式，如YUV420SP\_U8、RGB888\_U8等），当前给出JPEG、BT-601NARROW、BT-601WIDE、BT-709NARROW、BT-709WIDE几种典型场景下的色域转换配置。

说明

DVPP处理后的数据，不会对色域配置产生影响，例如DVPP处理前的数据为BT-709格式，经过DVPP处理后，输入给AIPP，此时数据仍为BT-709格式。

色域转换概览

支持的色域转换配置如[表6-2](#)所示。

表 6-2 色域转换概览表

支持的色域转换列表	支持的色域转换列表	支持的色域转换列表	支持的色域转换列表
YUV420SP_U8转YUV444	RGB888_U8转BGR	XRGB8888_U8转BGR	RGB888_U8转FP32 RGB
YUV420SP_U8转YVU444	RGB888_U8转YUV444	XRGB8888_U8转YUV444	-
YUV420SP_U8转RGB	RGB888_U8转YVU444	XRGB8888_U8转YVU444	-
YUV420SP_U8转BGR	RGB888_U8转GRAY	XRGB8888_U8转GRAY	-
YUV420SP_U8转GRAY	BGR888_U8转GRAY	XBGR8888_U8转GRAY	-

支持的色域转换列表	支持的色域转换列表	支持的色域转换列表	支持的色域转换列表
YVU420SP_U8转RGB	BGR888_U8转RGB	RGBX8888_U8转GRAY	-
YVU420SP_U8转BGR	BGR888_U8转BGR	BGRX8888_U8转GRAY	-
RGB888_U8转RGB	XRGB8888_U8转RGB	YUV400_U8转GRAY	-

YUV420SP\_U8 转 YUV444

```
aipp_op {  
  aipp_mode: static  
  input_format : YUV420SP_U8  
  csc_switch : false  
  rbuv_swap_switch : false  
}
```

YUV420SP\_U8 转 YVU444

```
aipp_op {  
  aipp_mode: static  
  input_format : YUV420SP_U8  
  csc_switch : false  
  rbuv_swap_switch : true  
}
```

YUV420SP\_U8 转 RGB

- 输入数据为JPEG图像

JPEG

```
aipp_op {  
  aipp_mode: static  
  input_format : YUV420SP_U8  
  csc_switch : true  
  rbuv_swap_switch : false  
  matrix_r0c0 : 256  
  matrix_r0c1 : 0  
  matrix_r0c2 : 359  
  matrix_r1c0 : 256  
  matrix_r1c1 : -88  
  matrix_r1c2 : -183  
  matrix_r2c0 : 256  
  matrix_r2c1 : 454  
  matrix_r2c2 : 0  
  input_bias_0 : 0  
  input_bias_1 : 128  
  input_bias_2 : 128  
}
```

- 输入数据为BT-601NARROW视频

#### BT-601NARROW

```
aipp_op {  
  aipp_mode: static  
  input_format : YUV420SP_U8  
  csc_switch : true  
  rbuv_swap_switch : false  
  matrix_r0c0 : 298  
  matrix_r0c1 : 0  
  matrix_r0c2 : 409  
  matrix_r1c0 : 298  
  matrix_r1c1 : -100  
  matrix_r1c2 : -208  
  matrix_r2c0 : 298  
  matrix_r2c1 : 516  
  matrix_r2c2 : 0  
  input_bias_0 : 16  
  input_bias_1 : 128  
  input_bias_2 : 128  
}
```

- 输入数据为BT-601WIDE视频

#### BT-601WIDE

```
aipp_op {  
  aipp_mode: static  
  input_format : YUV420SP_U8  
  csc_switch : true  
  rbuv_swap_switch : false  
  matrix_r0c0 : 256  
  matrix_r0c1 : 0  
  matrix_r0c2 : 359  
  matrix_r1c0 : 256  
  matrix_r1c1 : -88  
  matrix_r1c2 : -183  
  matrix_r2c0 : 256  
  matrix_r2c1 : 454  
  matrix_r2c2 : 0  
  input_bias_0 : 0  
  input_bias_1 : 128  
  input_bias_2 : 128  
}
```

- 输入数据为BT-709NARROW视频

#### BT-709NARROW

```
aipp_op {  
  aipp_mode: static  
  input_format : YUV420SP_U8  
  csc_switch : true  
  rbuv_swap_switch : false  
  matrix_r0c0 : 298  
  matrix_r0c1 : 0  
  matrix_r0c2 : 459  
  matrix_r1c0 : 298  
  matrix_r1c1 : -55  
  matrix_r1c2 : -136  
  matrix_r2c0 : 298  
  matrix_r2c1 : 541  
  matrix_r2c2 : 0  
  input_bias_0 : 16  
  input_bias_1 : 128  
  input_bias_2 : 128  
}
```

- 输入数据为BT-709WIDE视频

#### BT-709WIDE

```
aipp_op {  
  aipp_mode: static  
  input_format : YUV420SP_U8  
  csc_switch : true  
  rbuv_swap_switch : false  
  matrix_r0c0 : 256  
  matrix_r0c1 : 0  
  matrix_r0c2 : 403  
  matrix_r1c0 : 256  
  matrix_r1c1 : -48  
  matrix_r1c2 : -120  
  matrix_r2c0 : 256  
  matrix_r2c1 : 475  
  matrix_r2c2 : 0  
  input_bias_0 : 0  
  input_bias_1 : 128  
  input_bias_2 : 128  
}
```

## YUV420SP\_U8 转 BGR

- 输入数据为JPEG图像

### JPEG

```
aipp_op {  
  aipp_mode: static  
  input_format : YUV420SP_U8  
  csc_switch : true  
  rbuv_swap_switch : false  
  matrix_r0c0 : 256  
  matrix_r0c1 : 454  
  matrix_r0c2 : 0  
  matrix_r1c0 : 256  
  matrix_r1c1 : -88  
  matrix_r1c2 : -183  
  matrix_r2c0 : 256  
  matrix_r2c1 : 0  
  matrix_r2c2 : 359  
  input_bias_0 : 0  
  input_bias_1 : 128  
  input_bias_2 : 128  
}
```

- 输入数据为BT-601NARROW视频

### BT-601NARROW

```
aipp_op {  
  aipp_mode: static  
  input_format : YUV420SP_U8  
  csc_switch : true  
  rbuv_swap_switch : false  
  matrix_r0c0 : 298  
  matrix_r0c1 : 516  
  matrix_r0c2 : 0  
  matrix_r1c0 : 298  
  matrix_r1c1 : -100  
  matrix_r1c2 : -208  
  matrix_r2c0 : 298  
  matrix_r2c1 : 0  
  matrix_r2c2 : 409  
  input_bias_0 : 16  
  input_bias_1 : 128  
  input_bias_2 : 128  
}
```

- 输入数据为BT-601WIDE视频

#### BT-601WIDE

```
aipp_op {  
  aipp_mode: static  
  input_format : YUV420SP_U8  
  csc_switch : true  
  rbuv_swap_switch : false  
  matrix_r0c0 : 256  
  matrix_r0c1 : 454  
  matrix_r0c2 : 0  
  matrix_r1c0 : 256  
  matrix_r1c1 : -88  
  matrix_r1c2 : -183  
  matrix_r2c0 : 256  
  matrix_r2c1 : 0  
  matrix_r2c2 : 359  
  input_bias_0 : 0  
  input_bias_1 : 128  
  input_bias_2 : 128  
}
```

- 输入数据为BT-709NARROW视频

#### BT-709NARROW

```
aipp_op {  
  aipp_mode: static  
  input_format : YUV420SP_U8  
  csc_switch : true  
  rbuv_swap_switch : false  
  matrix_r0c0 : 298  
  matrix_r0c1 : 541  
  matrix_r0c2 : 0  
  matrix_r1c0 : 298  
  matrix_r1c1 : -55  
  matrix_r1c2 : -136  
  matrix_r2c0 : 298  
  matrix_r2c1 : 0  
  matrix_r2c2 : 459  
  input_bias_0 : 16  
  input_bias_1 : 128  
  input_bias_2 : 128  
}
```

- 输入数据为BT-709WIDE视频

#### BT-709WIDE

```
aipp_op {  
  aipp_mode: static  
  input_format : YUV420SP_U8  
  csc_switch : true  
  rbuv_swap_switch : false  
  matrix_r0c0 : 256  
  matrix_r0c1 : 475  
  matrix_r0c2 : 0  
  matrix_r1c0 : 256  
  matrix_r1c1 : -48  
  matrix_r1c2 : -120  
  matrix_r2c0 : 256  
  matrix_r2c1 : 0  
  matrix_r2c2 : 403  
  input_bias_0 : 0  
  input_bias_1 : 128  
  input_bias_2 : 128  
}
```

### YUV420SP\_U8 转 GRAY

```
aipp_op {  
  aipp_mode: static  
  input_format : YUV420SP_U8  
  csc_switch : true  
  rbuv_swap_switch : false  
  matrix_r0c0 : 256  
  matrix_r0c1 : 0  
  matrix_r0c2 : 0  
  matrix_r1c0 : 0  
  matrix_r1c1 : 0  
  matrix_r1c2 : 0  
  matrix_r2c0 : 0  
  matrix_r2c1 : 0  
  matrix_r2c2 : 0  
  input_bias_0 : 0  
  input_bias_1 : 0  
  input_bias_2 : 0  
}
```

### YVU420SP\_U8 转 RGB

- 输入数据为JPEG图像

### JPEG

```
aipp_op {  
  aipp_mode: static  
  input_format : YUV420SP_U8  
  csc_switch : true  
  rbuv_swap_switch : true  
  matrix_r0c0 : 256  
  matrix_r0c1 : 0  
  matrix_r0c2 : 359  
  matrix_r1c0 : 256  
  matrix_r1c1 : -88  
  matrix_r1c2 : -183  
  matrix_r2c0 : 256  
  matrix_r2c1 : 454  
  matrix_r2c2 : 0  
  input_bias_0 : 0  
  input_bias_1 : 128  
  input_bias_2 : 128  
}
```

- 输入数据为BT-601NARROW视频

### BT-601NARROW

```
aipp_op {  
  aipp_mode: static  
  input_format : YUV420SP_U8  
  csc_switch : true  
  rbuv_swap_switch : true  
  matrix_r0c0 : 298  
  matrix_r0c1 : 0  
  matrix_r0c2 : 409  
  matrix_r1c0 : 298  
  matrix_r1c1 : -100  
  matrix_r1c2 : -208  
  matrix_r2c0 : 298  
  matrix_r2c1 : 516  
  matrix_r2c2 : 0  
  input_bias_0 : 16  
  input_bias_1 : 128  
  input_bias_2 : 128  
}
```

- 输入数据为BT-601WIDE视频



#### BT-601WIDE

```
aipp_op {  
  aipp_mode: static  
  input_format : YUV420SP_U8  
  csc_switch : true  
  rbuv_swap_switch : true  
  matrix_r0c0 : 256  
  matrix_r0c1 : 0  
  matrix_r0c2 : 359  
  matrix_r1c0 : 256  
  matrix_r1c1 : -88  
  matrix_r1c2 : -183  
  matrix_r2c0 : 256  
  matrix_r2c1 : 454  
  matrix_r2c2 : 0  
  input_bias_0 : 0  
  input_bias_1 : 128  
  input_bias_2 : 128  
}
```

- 输入数据为BT-709NARROW视频

#### BT-709NARROW

```
aipp_op {  
  aipp_mode: static  
  input_format : YUV420SP_U8  
  csc_switch : true  
  rbuv_swap_switch : true  
  matrix_r0c0 : 298  
  matrix_r0c1 : 0  
  matrix_r0c2 : 459  
  matrix_r1c0 : 298  
  matrix_r1c1 : -55  
  matrix_r1c2 : -136  
  matrix_r2c0 : 298  
  matrix_r2c1 : 541  
  matrix_r2c2 : 0  
  input_bias_0 : 16  
  input_bias_1 : 128  
  input_bias_2 : 128  
}
```

- 输入数据为BT-709WIDE视频

#### BT-709WIDE

```
aipp_op {  
  aipp_mode: static  
  input_format : YUV420SP_U8  
  csc_switch : true  
  rbuv_swap_switch : true  
  matrix_r0c0 : 256  
  matrix_r0c1 : 0  
  matrix_r0c2 : 403  
  matrix_r1c0 : 256  
  matrix_r1c1 : -48  
  matrix_r1c2 : -120  
  matrix_r2c0 : 256  
  matrix_r2c1 : 475  
  matrix_r2c2 : 0  
  input_bias_0 : 0  
  input_bias_1 : 128  
  input_bias_2 : 128  
}
```

### YVU420SP\_U8 转 BGR

- 输入数据为JPEG图像

#### JPEG

```
aipp_op {  
  aipp_mode: static  
  input_format : YUV420SP_U8  
  csc_switch : true  
  rbuv_swap_switch : true  
  matrix_r0c0 : 256  
  matrix_r0c1 : 454  
  matrix_r0c2 : 0  
  matrix_r1c0 : 256  
  matrix_r1c1 : -88  
  matrix_r1c2 : -183  
  matrix_r2c0 : 256  
  matrix_r2c1 : 0  
  matrix_r2c2 : 359  
  input_bias_0 : 0  
  input_bias_1 : 128  
  input_bias_2 : 128  
}
```

- 输入数据为BT-601NARROW视频

#### BT-601NARROW

```
aipp_op {  
  aipp_mode: static  
  input_format : YUV420SP_U8  
  csc_switch : true  
  rbuv_swap_switch : true  
  matrix_r0c0 : 298  
  matrix_r0c1 : 516  
  matrix_r0c2 : 0  
  matrix_r1c0 : 298  
  matrix_r1c1 : -100  
  matrix_r1c2 : -208  
  matrix_r2c0 : 298  
  matrix_r2c1 : 0  
  matrix_r2c2 : 409  
  input_bias_0 : 16  
  input_bias_1 : 128  
  input_bias_2 : 128  
}
```

- 输入数据为BT-601WIDE视频

#### BT-601WIDE

```
aipp_op {  
  aipp_mode: static  
  input_format : YUV420SP_U8  
  csc_switch : true  
  rbuv_swap_switch : true  
  matrix_r0c0 : 256  
  matrix_r0c1 : 454  
  matrix_r0c2 : 0  
  matrix_r1c0 : 256  
  matrix_r1c1 : -88  
  matrix_r1c2 : -183  
  matrix_r2c0 : 256  
  matrix_r2c1 : 0  
  matrix_r2c2 : 359  
  input_bias_0 : 0  
  input_bias_1 : 128  
  input_bias_2 : 128  
}
```

- 输入数据为BT-709NARROW视频

#### BT-709NARROW

```
aipp_op {  
  aipp_mode: static  
  input_format : YUV420SP_U8  
  csc_switch : true  
  rbuv_swap_switch : true  
  matrix_r0c0 : 298  
  matrix_r0c1 : 541  
  matrix_r0c2 : 0  
  matrix_r1c0 : 298  
  matrix_r1c1 : -55  
  matrix_r1c2 : -136  
  matrix_r2c0 : 298  
  matrix_r2c1 : 0  
  matrix_r2c2 : 459  
  input_bias_0 : 16  
  input_bias_1 : 128  
  input_bias_2 : 128  
}
```

- 输入数据为BT-709WIDE视频

#### BT-709WIDE

```
aipp_op {  
  aipp_mode: static  
  input_format : YUV420SP_U8  
  csc_switch : true  
  rbuv_swap_switch : true  
  matrix_r0c0 : 256  
  matrix_r0c1 : 475  
  matrix_r0c2 : 0  
  matrix_r1c0 : 256  
  matrix_r1c1 : -48  
  matrix_r1c2 : -120  
  matrix_r2c0 : 256  
  matrix_r2c1 : 0  
  matrix_r2c2 : 403  
  input_bias_0 : 0  
  input_bias_1 : 128  
  input_bias_2 : 128  
}
```

## RGB888\_U8 转 RGB

```
aipp_op {  
  aipp_mode: static  
  input_format : RGB888_U8  
  csc_switch : false  
  rbuv_swap_switch : false  
}
```

## RGB888\_U8 转 BGR

```
aipp_op {  
  aipp_mode : static  
  input_format : RGB888_U8  
  csc_switch : false  
  rbuv_swap_switch : true  
}
```

## RGB888\_U8 转 YUV444

- 输入数据为JPEG图像

### JPEG

```
aipp_op {  
  aipp_mode: static  
  input_format : RGB888_U8  
  csc_switch : true  
  rbuv_swap_switch : false  
  matrix_r0c0 : 77  
  matrix_r0c1 : 150  
  matrix_r0c2 : 29  
  matrix_r1c0 : -43  
  matrix_r1c1 : -85  
  matrix_r1c2 : 128  
  matrix_r2c0 : 128  
  matrix_r2c1 : -107  
  matrix_r2c2 : -21  
  output_bias_0 : 0  
  output_bias_1 : 128  
  output_bias_2 : 128  
}
```

- 输入数据为BT-601NARROW视频

### BT-601NARROW

```
aipp_op {  
  aipp_mode: static  
  input_format : RGB888_U8  
  csc_switch : true  
  rbuv_swap_switch : false  
  matrix_r0c0 : 66  
  matrix_r0c1 : 129  
  matrix_r0c2 : 25  
  matrix_r1c0 : -38  
  matrix_r1c1 : -74  
  matrix_r1c2 : 112  
  matrix_r2c0 : 112  
  matrix_r2c1 : -94  
  matrix_r2c2 : -18  
  output_bias_0 : 16  
  output_bias_1 : 128  
  output_bias_2 : 128  
}
```

- 输入数据为BT-601WIDE视频

#### BT-601WIDE

```
aipp_op {  
  aipp_mode: static  
  input_format : RGB888_U8  
  csc_switch : true  
  rbuv_swap_switch : false  
  matrix_r0c0 : 77  
  matrix_r0c1 : 150  
  matrix_r0c2 : 29  
  matrix_r1c0 : -43  
  matrix_r1c1 : -85  
  matrix_r1c2 : 128  
  matrix_r2c0 : 128  
  matrix_r2c1 : -107  
  matrix_r2c2 : -21  
  output_bias_0 : 0  
  output_bias_1 : 128  
  output_bias_2 : 128  
}
```

- 输入数据为BT-709NARROW视频

#### BT-709NARROW

```
aipp_op {  
  aipp_mode: static  
  input_format : RGB888_U8  
  csc_switch : true  
  rbuv_swap_switch : false  
  matrix_r0c0 : 47  
  matrix_r0c1 : 157  
  matrix_r0c2 : 16  
  matrix_r1c0 : -26  
  matrix_r1c1 : -87  
  matrix_r1c2 : 112  
  matrix_r2c0 : 112  
  matrix_r2c1 : -102  
  matrix_r2c2 : -10  
  output_bias_0 : 16  
  output_bias_1 : 128  
  output_bias_2 : 128  
}
```

- 输入数据为BT-709WIDE视频

#### BT-709WIDE

```
aipp_op {  
  aipp_mode: static  
  input_format : RGB888_U8  
  csc_switch : true  
  rbuv_swap_switch : false  
  matrix_r0c0 : 55  
  matrix_r0c1 : 183  
  matrix_r0c2 : 19  
  matrix_r1c0 : -29  
  matrix_r1c1 : -99  
  matrix_r1c2 : 128  
  matrix_r2c0 : 128  
  matrix_r2c1 : -116  
  matrix_r2c2 : -12  
  output_bias_0 : 0  
  output_bias_1 : 128  
  output_bias_2 : 128  
}
```

## RGB888\_U8 转 YVU444

- 输入数据为JPEG图像

#### JPEG

```
aipp_op {  
  aipp_mode: static  
  input_format : RGB888_U8  
  csc_switch : true  
  rbuv_swap_switch : false  
  matrix_r0c0 : 77  
  matrix_r0c1 : 150  
  matrix_r0c2 : 29  
  matrix_r1c0 : 128  
  matrix_r1c1 : -107  
  matrix_r1c2 : -21  
  matrix_r2c0 : -43  
  matrix_r2c1 : -85  
  matrix_r2c2 : 128  
  output_bias_0 : 0  
  output_bias_1 : 128  
  output_bias_2 : 128  
}
```

- 输入数据为BT-601NARROW视频

#### BT-601NARROW

```
aipp_op {  
  aipp_mode: static  
  input_format : RGB888_U8  
  csc_switch : true  
  rbuv_swap_switch : false  
  matrix_r0c0 : 66  
  matrix_r0c1 : 129  
  matrix_r0c2 : 25  
  matrix_r1c0 : 112  
  matrix_r1c1 : -94  
  matrix_r1c2 : -18  
  matrix_r2c0 : -38  
  matrix_r2c1 : -74  
  matrix_r2c2 : 112  
  output_bias_0 : 16  
  output_bias_1 : 128  
  output_bias_2 : 128  
}
```

- 输入数据为BT-601WIDE视频

#### BT-601WIDE

```
aipp_op {  
  aipp_mode: static  
  input_format : RGB888_U8  
  csc_switch : true  
  rbuv_swap_switch : false  
  matrix_r0c0 : 77  
  matrix_r0c1 : 150  
  matrix_r0c2 : 29  
  matrix_r1c0 : 128  
  matrix_r1c1 : -107  
  matrix_r1c2 : -21  
  matrix_r2c0 : -43  
  matrix_r2c1 : -85  
  matrix_r2c2 : 128  
  output_bias_0 : 0  
  output_bias_1 : 128  
  output_bias_2 : 128  
}
```

- 输入数据为BT-709NARROW视频



**BT-709NARROW**

```
aipp_op {  
  aipp_mode: static  
  input_format : RGB888_U8  
  csc_switch : true  
  rbuv_swap_switch : false  
  matrix_r0c0 : 47  
  matrix_r0c1 : 157  
  matrix_r0c2 : 16  
  matrix_r1c0 : 112  
  matrix_r1c1 : -102  
  matrix_r1c2 : -10  
  matrix_r2c0 : -26  
  matrix_r2c1 : -87  
  matrix_r2c2 : 112  
  output_bias_0 : 16  
  output_bias_1 : 128  
  output_bias_2 : 128  
}
```

- 输入数据为BT-709WIDE视频

**BT-709WIDE**

```
aipp_op {  
  aipp_mode: static  
  input_format : RGB888_U8  
  csc_switch : true  
  rbuv_swap_switch : false  
  matrix_r0c0 : 55  
  matrix_r0c1 : 183  
  matrix_r0c2 : 19  
  matrix_r1c0 : 128  
  matrix_r1c1 : -116  
  matrix_r1c2 : -12  
  matrix_r2c0 : -29  
  matrix_r2c1 : -99  
  matrix_r2c2 : 128  
  output_bias_0 : 0  
  output_bias_1 : 128  
  output_bias_2 : 128  
}
```

**RGB888\_U8 转 GRAY**

```
aipp_op {  
  aipp_mode: static  
  input_format : RGB888_U8  
  csc_switch : true  
  rbuv_swap_switch : false  
  matrix_r0c0 : 76  
  matrix_r0c1 : 150  
  matrix_r0c2 : 30  
  matrix_r1c0 : 0  
  matrix_r1c1 : 0  
  matrix_r1c2 : 0  
  matrix_r2c0 : 0  
  matrix_r2c1 : 0  
  matrix_r2c2 : 0  
  output_bias_0 : 0  
  output_bias_1 : 0  
  output_bias_2 : 0  
}
```

## BGR888\_U8 转 GRAY

```
aipp_op {  
  aipp_mode: static  
  input_format : RGB888_U8  
  csc_switch : true  
  rbuv_swap_switch : true  
  matrix_r0c0 : 76  
  matrix_r0c1 : 150  
  matrix_r0c2 : 30  
  matrix_r1c0 : 0  
  matrix_r1c1 : 0  
  matrix_r1c2 : 0  
  matrix_r2c0 : 0  
  matrix_r2c1 : 0  
  matrix_r2c2 : 0  
  output_bias_0 : 0  
  output_bias_1 : 0  
  output_bias_2 : 0  
}
```

## BGR888\_U8 转 RGB

```
aipp_op {  
  aipp_mode: static  
  input_format : RGB888_U8  
  csc_switch : false  
  rbuv_swap_switch : true  
}
```

## BGR888\_U8 转 BGR

```
aipp_op {  
  aipp_mode: static  
  input_format : RGB888_U8  
  csc_switch : false  
  rbuv_swap_switch : false  
}
```

## XRGB8888\_U8 转 RGB

```
aipp_op {  
  aipp_mode: static  
  input_format : XRGB8888_U8  
  csc_switch : false  
  rbuv_swap_switch : false  
  ax_swap_switch : true  
}
```

## XRGB8888\_U8 转 BGR

```
aipp_op {  
  aipp_mode : static  
  input_format : XRGB8888_U8  
  csc_switch : false  
  rbuv_swap_switch : true  
  ax_swap_switch : true  
}
```

## XRGB8888\_U8 转 YUV444

- 输入数据为JPEG图像

### JPEG

```
aipp_op {
  aipp_mode: static
  input_format : XRGB8888_U8
  csc_switch : true
  rbuv_swap_switch : false
  ax_swap_switch : true
  matrix_r0c0 : 77
  matrix_r0c1 : 150
  matrix_r0c2 : 29
  matrix_r1c0 : -43
  matrix_r1c1 : -85
  matrix_r1c2 : 128
  matrix_r2c0 : 128
  matrix_r2c1 : -107
  matrix_r2c2 : -21
  output_bias_0 : 0
  output_bias_1 : 128
  output_bias_2 : 128
}
```

- 输入数据为BT-601NARROW视频

### BT-601NARROW

```
aipp_op {
  aipp_mode: static
  input_format : XRGB8888_U8
  csc_switch : true
  rbuv_swap_switch : false
  ax_swap_switch : true
  matrix_r0c0 : 66
  matrix_r0c1 : 129
  matrix_r0c2 : 25
  matrix_r1c0 : -38
  matrix_r1c1 : -74
  matrix_r1c2 : 112
  matrix_r2c0 : 112
  matrix_r2c1 : -94
  matrix_r2c2 : -18
  output_bias_0 : 16
  output_bias_1 : 128
  output_bias_2 : 128
}
```

- 输入数据为BT-601WIDE视频

#### BT-601WIDE

```
aipp_op {  
  aipp_mode: static  
  input_format : XRGB8888_U8  
  csc_switch : true  
  rbuv_swap_switch : false  
  ax_swap_switch : true  
  matrix_r0c0 : 77  
  matrix_r0c1 : 150  
  matrix_r0c2 : 29  
  matrix_r1c0 : -43  
  matrix_r1c1 : -85  
  matrix_r1c2 : 128  
  matrix_r2c0 : 128  
  matrix_r2c1 : -107  
  matrix_r2c2 : -21  
  output_bias_0 : 0  
  output_bias_1 : 128  
  output_bias_2 : 128  
}
```

- 输入数据为BT-709NARROW视频

#### BT-709NARROW

```
aipp_op {  
  aipp_mode: static  
  input_format : XRGB8888_U8  
  csc_switch : true  
  rbuv_swap_switch : false  
  ax_swap_switch : true  
  matrix_r0c0 : 47  
  matrix_r0c1 : 157  
  matrix_r0c2 : 16  
  matrix_r1c0 : -26  
  matrix_r1c1 : -87  
  matrix_r1c2 : 112  
  matrix_r2c0 : 112  
  matrix_r2c1 : -102  
  matrix_r2c2 : -10  
  output_bias_0 : 16  
  output_bias_1 : 128  
  output_bias_2 : 128  
}
```

- 输入数据为BT-709WIDE视频

#### BT-709WIDE

```
aipp_op {  
  aipp_mode: static  
  input_format : XRGB8888_U8  
  csc_switch : true  
  rbuv_swap_switch : false  
  ax_swap_switch : true  
  matrix_r0c0 : 55  
  matrix_r0c1 : 183  
  matrix_r0c2 : 19  
  matrix_r1c0 : -29  
  matrix_r1c1 : -99  
  matrix_r1c2 : 128  
  matrix_r2c0 : 128  
  matrix_r2c1 : -116  
  matrix_r2c2 : -12  
  output_bias_0 : 0  
  output_bias_1 : 128  
  output_bias_2 : 128  
}
```

## XRGB8888\_U8 转 YVU444

- 输入数据为JPEG图像

#### JPEG

```
aipp_op {  
  aipp_mode: static  
  input_format : XRGB8888_U8  
  csc_switch : true  
  rbuv_swap_switch : false  
  ax_swap_switch : true  
  matrix_r0c0 : 77  
  matrix_r0c1 : 150  
  matrix_r0c2 : 29  
  matrix_r1c0 : 128  
  matrix_r1c1 : -107  
  matrix_r1c2 : -21  
  matrix_r2c0 : -43  
  matrix_r2c1 : -85  
  matrix_r2c2 : 128  
  output_bias_0 : 0  
  output_bias_1 : 128  
  output_bias_2 : 128  
}
```

- 输入数据为BT-601NARROW视频

#### BT-601NARROW

```
aipp_op {  
  aipp_mode: static  
  input_format : XRGB8888_U8  
  csc_switch : true  
  rbuv_swap_switch : false  
  ax_swap_switch : true  
  matrix_r0c0 : 66  
  matrix_r0c1 : 129  
  matrix_r0c2 : 25  
  matrix_r1c0 : 112  
  matrix_r1c1 : -94  
  matrix_r1c2 : -18  
  matrix_r2c0 : -38  
  matrix_r2c1 : -74  
  matrix_r2c2 : 112  
  output_bias_0 : 16  
  output_bias_1 : 128  
  output_bias_2 : 128  
}
```

- 输入数据为BT-601WIDE视频

#### BT-601WIDE

```
aipp_op {  
  aipp_mode: static  
  input_format : XRGB8888_U8  
  csc_switch : true  
  rbuv_swap_switch : false  
  ax_swap_switch : true  
  matrix_r0c0 : 77  
  matrix_r0c1 : 150  
  matrix_r0c2 : 29  
  matrix_r1c0 : 128  
  matrix_r1c1 : -107  
  matrix_r1c2 : -21  
  matrix_r2c0 : -43  
  matrix_r2c1 : -85  
  matrix_r2c2 : 128  
  output_bias_0 : 0  
  output_bias_1 : 128  
  output_bias_2 : 128  
}
```

- 输入数据为BT-709NARROW视频

#### BT-709NARROW

```
aipp_op {  
  aipp_mode: static  
  input_format : XRGB8888_U8  
  csc_switch : true  
  rbuv_swap_switch : false  
  ax_swap_switch : true  
  matrix_r0c0 : 47  
  matrix_r0c1 : 157  
  matrix_r0c2 : 16  
  matrix_r1c0 : 112  
  matrix_r1c1 : -102  
  matrix_r1c2 : -10  
  matrix_r2c0 : -26  
  matrix_r2c1 : -87  
  matrix_r2c2 : 112  
  output_bias_0 : 16  
  output_bias_1 : 128  
  output_bias_2 : 128  
}
```

- 输入数据为BT-709WIDE视频

#### BT-709WIDE

```
aipp_op {  
  aipp_mode: static  
  input_format : XRGB8888_U8  
  csc_switch : true  
  rbuv_swap_switch : false  
  ax_swap_switch : true  
  matrix_r0c0 : 55  
  matrix_r0c1 : 183  
  matrix_r0c2 : 19  
  matrix_r1c0 : 128  
  matrix_r1c1 : -116  
  matrix_r1c2 : -12  
  matrix_r2c0 : -29  
  matrix_r2c1 : -99  
  matrix_r2c2 : 128  
  output_bias_0 : 0  
  output_bias_1 : 128  
  output_bias_2 : 128  
}
```

## XRGB8888\_U8 转 GRAY

```
aipp_op {  
  aipp_mode: static  
  input_format : XRGB8888_U8  
  csc_switch : true  
  rbuv_swap_switch : false  
  ax_swap_switch : true  
  matrix_r0c0 : 76  
  matrix_r0c1 : 150  
  matrix_r0c2 : 30  
  matrix_r1c0 : 0  
  matrix_r1c1 : 0  
  matrix_r1c2 : 0  
  matrix_r2c0 : 0  
  matrix_r2c1 : 0  
  matrix_r2c2 : 0  
  output_bias_0 : 0
```

```
    output_bias_1 : 0  
    output_bias_2 : 0  
}
```

## XBGR8888\_U8 转 GRAY

```
aipp_op {  
  aipp_mode: static  
  input_format : XRGB8888_U8  
  csc_switch : true  
  rbuv_swap_switch : true  
  ax_swap_switch : true  
  matrix_r0c0 : 76  
  matrix_r0c1 : 150  
  matrix_r0c2 : 30  
  matrix_r1c0 : 0  
  matrix_r1c1 : 0  
  matrix_r1c2 : 0  
  matrix_r2c0 : 0  
  matrix_r2c1 : 0  
  matrix_r2c2 : 0  
  output_bias_0 : 0  
  output_bias_1 : 0  
  output_bias_2 : 0  
}
```

## RGBX8888\_U8 转 GRAY

```
aipp_op {  
  aipp_mode: static  
  input_format : XRGB8888_U8  
  csc_switch : true  
  rbuv_swap_switch : false  
  ax_swap_switch : false  
  matrix_r0c0 : 76  
  matrix_r0c1 : 150  
  matrix_r0c2 : 30  
  matrix_r1c0 : 0  
  matrix_r1c1 : 0  
  matrix_r1c2 : 0  
  matrix_r2c0 : 0  
  matrix_r2c1 : 0  
  matrix_r2c2 : 0  
  output_bias_0 : 0  
  output_bias_1 : 0  
  output_bias_2 : 0  
}
```

## BGRX8888\_U8 转 GRAY

```
aipp_op {  
  aipp_mode: static  
  input_format : XRGB8888_U8  
  csc_switch : true  
  rbuv_swap_switch : true  
  ax_swap_switch : false  
  matrix_r0c0 : 76  
  matrix_r0c1 : 150  
  matrix_r0c2 : 30  
  matrix_r1c0 : 0  
  matrix_r1c1 : 0  
  matrix_r1c2 : 0  
  matrix_r2c0 : 0  
  matrix_r2c1 : 0  
  matrix_r2c2 : 0  
  output_bias_0 : 0  
  output_bias_1 : 0  
  output_bias_2 : 0  
}
```



## YUV400\_U8 转 GRAY

```
aipp_op {  
  aipp_mode: static  
  input_format : YUV400_U8  
  csc_switch : false  
}
```

## RGB888\_U8 转 FP32 RGB

```
aipp_op {  
  aipp_mode: static  
  related_input_rank: 0  
  input_format : RGB888_U8  
  src_image_size_w : 640  
  src_image_size_h : 640  
  mean_chn_0 : 0  
  mean_chn_1 : 0  
  mean_chn_2 : 0  
  var_reci_chn_0 : 1.0  
  var_reci_chn_1 : 1.0  
  var_reci_chn_2 : 1.0  
}
```

### 6.1.5 归一化配置说明

归一化就是要把需要处理的数据经过处理后限制在一定范围内，方便后面数据的处理。AIPP支持的归一化设置，通过减均值和乘系数的操作完成，这样的能力不仅能用于常规的归一化，还能用于不同数据格式的转化。

比如在由unit8转为fp16时，其转换可以视作如下公式。其中，mean\_chn\_i表示每个通道的均值，min\_chn\_i表示每个通道的最小值，var\_reci\_chn\_i表示每个通道方差的倒数，各通路的这三个值都是需要进行配置的参数。

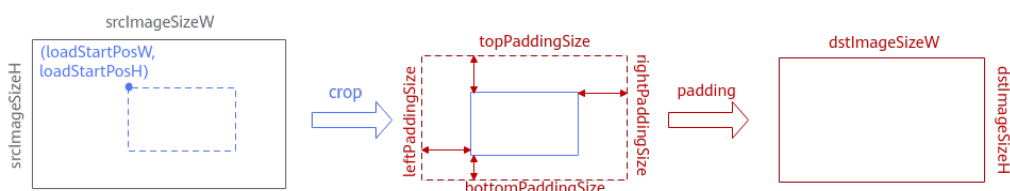
$$\text{pixel\_out\_chx}(i) = [\text{pixel\_in\_chx}(i) - \text{mean\_chn\_i} - \text{min\_chn\_i}] * \text{var\_reci\_chn\_i}$$

### 6.1.6 Crop/Padding 配置说明

AIPP改变图片尺寸需要遵守如下图中的顺序，即先Crop再Padding，每个操作仅能执行一次。

原图大小为srcImageSizeW、srcImageSizeH的图像经过图像预处理后变为模型预期的dstImageSizeW、dstImageSizeH图像尺寸。

图 6-1 改变图像尺寸



#### 说明

图中实线框表示当前图片size，虚线框表示经过右侧箭头上的AIPP操作处理后的图片size。

从执行角度看，我们需要在配置文件中指出裁剪的起始位置左上点坐标loadStartPosW、loadStartPosH以及裁剪后的图像大小crop\_size\_w, crop\_size\_h。在

padding环节，我们需要指明在裁剪后的图像四周padding的尺寸，即 left\_padding\_size、right\_padding\_size、top\_padding\_size和bottom\_padding\_size。而经过图像尺寸改变之后最终图片大小，需要跟模型文件输入的图像大小即--input\_shape中的宽和高相等。

对于YUV420SP\_U8图片类型，load\_start\_pos\_w、load\_start\_pos\_h参数必须配置为偶数。配置样例如下：

```
aipp_op {
  aipp_mode: static
  input_format: YUV420SP_U8

  src_image_size_w: 320
  src_image_size_h: 240

  crop: true
  load_start_pos_w: 10
  load_start_pos_h: 20
  crop_size_w: 50
  crop_size_h: 60

  padding: true
  left_padding_size: 20
  right_padding_size: 15
  top_padding_size: 20
  bottom_padding_size: 15
  padding_value: 0
}
```

6.1.7 AIPP 对模型输入大小的校验说明

如果有配置AIPP，无论静态AIPP还是动态AIPP，最终生成离线模型的输入大小（即input\_size）均会被Crop、Padding等操作影响。本节给出对模型输入大小的约束说明。

假设模型的Batch数量为N（如果为动态batch场景，N为最大档位数的取值），模型输入图片的宽为src\_image\_size\_w，高为src\_image\_size\_h，最后模型输入的Size的计算公式如下所示。

静态 AIPP 对模型输入大小的校验

表 6-3 input\_size 校验公式

input_format	input_size	备注
YUV400_U8	$N * src\_image\_size\_w * src\_image\_size\_h * 1$	-
YUV420SP_U8	$N * src\_image\_size\_w * src\_image\_size\_h * 1.5$	-
XRGB8888_U8	$N * src\_image\_size\_w * src\_image\_size\_h * 4$	-
RGB888_U8	$N * src\_image\_size\_w * src\_image\_size\_h * 3$	-

动态 AIPP 对模型输入大小的校验

如果为动态AIPP，模型转换时，ATC会为动态AIPP新增一个模型输入，用于接收模型推理阶段通过调用《[CANN AscendCL应用软件开发指南（C&C++）（开放态）](#)》手

册中的“AscendCL API参考>模型加载与执行>aclmdlSetInputAIPP”接口后传入的AIPP参数，该场景下新增输入节点大小计算公式为：

```
sizeof(kAippDynamicPara) - sizeof(kAippDynamicBatchPara) + batch_count *  
sizeof(kAippDynamicBatchPara)
```

kAippDynamicPara以及kAippDynamicBatchPara参数解释请参见[动态AIPP的参数输入结构](#)。

## 6.1.8 配置文件模板

AIPP配置文件通过本章节给出的模板进行配置，内容需要满足prototxt格式，用户根据场景决定配置哪些参数，修改为合适的取值另存后供模型转换使用；使用配置模板之前需要先查看相关约束。

### 模板使用整体约束

1. 使用配置文件模板时，请将需要配置的参数去注释，并改为合适的值。
2. 模板中参数取值都为默认值，实际使用时，如果配置文件中某些参数未配置，则模型转换时自动设置成该模板中相应参数的默认值。
3. 静态AIPP场景下，input\_format属性为必选属性，其余属性均为可选配置，如果未配置，则模型转换时自动设置成该模板中相应参数的默认值。
4. 由于硬件处理逻辑的限制，配置文件中的参数有如下处理顺序要求：通道交换（rbuv\_swap\_switch）>图像裁剪（crop）>色域转换（通道交换）>数据减均值/归一化>图像边缘填充（padding）。
5. AIPP当前支持色域转换、图像裁剪、减均值、乘系数、通道交换、单行模式的能力，输入图片的类型仅支持RAW和UINT8格式。
6. 若输入图片为RGB（由R、G、B三个分量组成的图片），其对应的输入、输出通道顺序，从高地址到低地址依次为：{R,G,B}。
7. 动态AIPP的参数每次推理需要计算，计算需要耗时，所以动态AIPP的性能比静态AIPP性能要差。
8. 经过AIPP处理后的图片，统一采用NC1HWC0的五维数据格式进行存储：  
以原始模型要求的图片为RGB（由R、G、B三个分量组成的图片）为例进行说明，配置了AIPP功能场景下：
  - Caffe/ONNX框架数据格式只能设置为NCHW（数据存储格式为RRRRRRGGGGGGBBBBBB）
  - TensorFlow框架数据格式只能设置为NHWC（数据存储格式为RGBRGBRGBRGBRGB）或NCHW（数据存储格式为RRRRRRGGGGGGBBBBBB）。实际提供的图片经过AIPP色域转换功能处理后，输出的离线模型中图片为RGB，并以NC1HWC0五维数据格式进行存储（关于NC1HWC0详细介绍请参见[4.3 关键概念](#)）：若AIPP输出数据类型为FP16，则C0=16，从高位到低位依次为R,G,B，其余位数补0；C1=1。
9. 模型转换是否开启AIPP功能，执行推理业务时，对输入图片数据的要求：
  - 模型转换时开启AIPP：在进行推理业务时，输入图片数据要求为NHWC排布，该场景下最终与AIPP连接的输入节点的格式被强制改成NHWC，可能与模型转换命令中[7.2.2.5 --input\\_format](#)参数指定的格式不一致。
  - 模型转换时没有开启AIPP：在进行推理业务时，模型的Format需与输入图片的Format保持一致。例如，输入图片的Format为NHWC，但某模型默认的Format为NCHW，此时输入图片和模型的Format不一致，用户可在模型转换

时指定**7.2.2.5 --input\_format**调整模型的Format，也可以选择符合模型要求的输入图片。

10. 对于输入图像格式YUV420SP，根据UV分量顺序不同，YUV420SP又分为YUV420SP\_UV(NV12)和YUV420SP\_VU(NV21)，分别对应**6.1.4 色域转换配置说明**中的YUV420SP\_U8、YVU420SP\_U8，默认为YUV420SP\_UV(NV12)。  
对于AIPP配置文件中的input\_format参数，需始终配置为NV12格式（YUV420SP\_U8），通过rbuv\_swap\_switch参数控制实际提供给AIPP的图片格式：
  - 若rbuv\_swap\_switch设置为false，则实际提供的图片格式为YUV420SP\_U8。
  - 若rbuv\_swap\_switch设置为true，则实际提供的图片格式为YVU420SP\_U8。
11. AIPP针对各种图像格式的典型参数配置如下表所示。

表 6-4 各种图像格式的典型参数配置

aipp的输入图像格式 (input_format)	输入图像内存排布格式	对应原始输入图像格式	AIPP输出格式	关于AIPP配置文件中相关参数的说明
RGB888_U8	NHWC	RGB package	NC1HWC0	rbuv_swap_switch通常设置为false
RGB888_U8	NHWC	BGR package	NC1HWC0	rbuv_swap_switch通常设置为true，内部先转为RGB package再做后续处理
YUV420SP_U8	/	YUV420sp NV12 8bit	NC1HWC0	rbuv_swap_switch通常设置为false
YUV420SP_U8	/	YUV420sp NV21 8bit	NC1HWC0	rbuv_swap_switch通常设置为true，内部先转为NV12格式再做后续处理
YUV400_U8	NHWC	灰度图	NC1HWC0	/

aipp的输入图像格式 (input_format)	输入图像内存排布格式	对应原始输入图像格式	AIPP 输出格式	关于AIPP配置文件中相关参数的说明
XRGB8888_U8	NHWC	<ul style="list-style-type: none"><li>• XRGB package</li><li>• XBGR package</li><li>• RGBX package</li><li>• BGRX package</li></ul>	NC1HWC0	<ul style="list-style-type: none"><li>• XRGB package rbuv_swap_switch通常设置为false, ax_swap_switch通常设置为true</li><li>• XBGR package rbuv_swap_switch通常设置为true, ax_swap_switch通常设置为true, 内部先转为XRGB package再做后续处理</li><li>• RGBX package rbuv_swap_switch通常设置为false, ax_swap_switch通常设置为false, 内部先转为XRGB package再做后续处理</li><li>• BGRX package rbuv_swap_switch通常设置为true, ax_swap_switch通常设置为false, 内部先转为XRGB package再做后续处理</li></ul>

配置文件模板

```
# AIPP的配置以aipp_op开始，标识这是一个AIPP算子的配置，aipp_op支持配置多个
aipp_op {

===== 全局设置 ( start )
=====

# aipp_mode指定了AIPP的模式，必须配置
# 类型: enum
# 取值范围: dynamic/static, dynamic 表示动态AIPP, static 表示静态AIPP
aipp_mode:

# related_input_rank参数为可选，标识对模型的第几个输入做AIPP处理，从0开始，默认为0。例如模型有两个输入，需要对第2个输入做AIPP，则配置related_input_rank为1。
# 类型: 整型
# 配置范围 >= 0
related_input_rank: 0

# related_input_name参数为可选，标识对模型的第几个输入做AIPP处理，此处需要填写为模型输入的名称（input对应的值）或者模型首层节点的输出（top参数对应的取值）。该参数只适用于Caffe网络模型，且不能与related_input_rank参数同时使用。
# 例如模型有两个输入，且输入name分别为data、im_info，需要对第二个输入做AIPP，则配置related_input_name为im_info。
# 类型: string
# 配置范围: 无
related_input_name: ""

===== 全局设置 ( end )
=====
}
```

```
#===== 动态AIPP需设置，静态AIPP无需设置（start）
=====
# 输入图像最大的size，动态AIPP必须配置（如果为动态batch场景，N为最大档位数的取值）
# 类型: int
max_src_image_size: 0
# 若输入图像格式为YUV400_U8，则max_src_image_size>=N * src_image_size_w * src_image_size_h * 1。
# 若输入图像格式为YUV420SP_U8，则max_src_image_size>=N * src_image_size_w * src_image_size_h * 1.5。
# 若输入图像格式为XRGB8888_U8，则max_src_image_size>=N * src_image_size_w * src_image_size_h * 4。
# 若输入图像格式为RGB888_U8，则max_src_image_size>=N * src_image_size_w * src_image_size_h * 3。

# 是否支持旋转，保留字段，暂不支持该功能
# 类型: bool
# 取值范围: true/false，true表示支持旋转，false表示不支持旋转
support_rotation: false
#===== 动态AIPP需设置，静态AIPP无需设置（end）
=====

#===== 静态AIPP需设置，动态AIPP无需设置（start）
=====
# 输入图像格式，必选
# 类型: enum
# 取值范围: YUV420SP_U8、XRGB8888_U8、RGB888_U8、YUV400_U8
input_format:
# 说明: 模型转换完毕后，在对应的*.om模型文件中，上述参数分别以1、2、3、4枚举值呈现。

# 原始图像的宽度、高度
# 类型: int32
# 取值范围 & 约束: 宽度取值范围为[2,4096]或0；高度取值范围为[1,4096]或0，对于YUV420SP_U8类型的图像，要求原始图像的宽和高取值是偶数
src_image_size_w: 0
src_image_size_h: 0
# 说明: 请根据实际图片的宽、高配置src_image_size_w和src_image_size_h；只有crop，padding功能都没有开启的场景，src_image_size_w和src_image_size_h才能取值为0或不配置，该场景下会取网络模型输入定义的w和h，并且网络模型输入定义的w取值范围为[2,4096]，h取值范围为[1,4096]。
# C方向的填充值，保留字段，暂不支持该功能
# 类型: float16
# 取值范围: [-65504, 65504]
padding_value: 0.0

#===== crop参数设置（配置样例请参见AIPP配置 > Crop/Padding配置说明）=====
# AIPP处理图片时是否支持抠图
# 类型: bool
# 取值范围: true/false，true表示支持，false表示不支持
crop: false

# 抠图起始位置水平、垂直方向坐标，抠图大小为网络输入定义的w和h
# 类型: int32
# 取值范围 & 约束: [0,4095]、对于YUV420SP_U8类型的图像，要求取值是偶数
# 说明: load_start_pos_w<src_image_size_w，load_start_pos_h<src_image_size_h
load_start_pos_w: 0
load_start_pos_h: 0

# 抠图后的图像size
# 类型: int32
# 取值范围 & 约束: [0,4096]、load_start_pos_w + crop_size_w <= src_image_size_w、load_start_pos_h + crop_size_h <= src_image_size_h
crop_size_w: 0
crop_size_h: 0
说明: 若开启抠图功能，并且没有配置padding，该场景下crop_size_w和crop_size_h才能取值为0或不配置，此时抠图大小（crop_size[W|H]）的宽和高取值来自模型文件--input_shape中的宽和高，并且--input_shape中的宽和高取值范围为[1,4096]。

# 抠图约束如下:
# 若input_format取值为YUV420SP_U8，则load_start_pos_w、load_start_pos_h必须为偶数。
# 若input_format取值为其他值，对load_start_pos_w、load_start_pos_h无约束。
# 若开启抠图功能，则src_image_size[W|H] >= crop_size[W|H]+load_start_pos[W|H]。
```

```
#===== resize参数设置 =====
# AIPP处理图片时是否支持缩放，保留字段，暂不支持该功能
# 类型: bool
# 取值范围: true/false, true表示支持, false表示不支持
resize: false

# 缩放后图像的宽度和高度，保留字段，暂不支持该功能
# 类型: int32
# 取值范围 & 约束: resize_output_h: [16,4096]或0; resize_output_w: [16,1920]或0; resize_output_w/
resize_input_w∈[1/16,16]、resize_output_h/resize_input_h∈[1/16,16]
resize_output_w: 0
resize_output_h: 0
# 说明: 若开启了缩放功能，并且没有配置padding，该场景下resize_output_w和resize_output_h才能取值为0
或不配置，此时缩放后图像的宽和高取值来自模型文件--input_shape中的宽和高，并且--input_shape中的高取
值范围为[16,4096]，宽取值范围为[16,1920]。

#===== padding参数设置（配置样例请参见AIPP配置 > Crop/Padding配置说明） =====
# AIPP处理图片时padding使能开关
# 类型: bool
# 取值范围: true/false, true表示支持, false表示不支持
padding: false

# H和W的填充值，静态AIPP配置
# 类型: int32
# 取值范围: [0,32]
left_padding_size: 0
right_padding_size: 0
top_padding_size: 0
bottom_padding_size: 0
# 说明: AIPP经过padding后，输出的H和W要与模型需要的H和W保持一致
# 针对Atlas 200/300/500 推理产品、Atlas 推理系列产品、Atlas 训练系列产品，W取值要<=1080。

# 上下左右方向上padding的像素取值，静态AIPP配置
# 类型: uint8/int8/float16
# 取值范围分别为: [0,255]、[-128, 127]、[-65504, 65504]
padding_value: 0
# 说明: 该参数取值需要与最终AIPP输出图片的数据类型保持一致。

#===== rotation参数设置 =====
# AIPP处理图片时的旋转角度，保留字段，暂不支持该功能
# 类型: uint8
# 范围: {0, 1, 2, 3} 0不旋转, 1顺时针90°, 2顺时针180°, 3顺时针270°
rotation_angle: 0

#===== 色域转换参数设置（配置样例请参见AIPP配置 > 色域转换配置说明） =====
# 色域转换开关，静态AIPP配置
# 类型: bool
# 取值范围: true/false, true表示开启色域转换, false表示关闭
csc_switch: false

# R通道与B通道交换开关/U通道与V通道交换开关
# 类型: bool
# 取值范围: true/false, true表示开启通道交换, false表示关闭
rbuv_swap_switch: false

# RGBA->ARGB, YUVA->AYUV交换开关
# 类型: bool
# 取值范围: true/false, true表示开启, false表示关闭
ax_swap_switch: false

# 单行处理模式（只处理抠图后的第一行）开关，保留字段，暂不支持该功能
# 类型: bool
# 取值范围: true/false, true表示开启单行处理模式, false表示关闭
single_line_mode: false
```

```
# 若色域转换开关为false, 则本功能不起作用。
# 若输入图片通道数为4, 则忽略A通道或X通道。
# YUV转BGR:
# | B | | matrix_r0c0 matrix_r0c1 matrix_r0c2 || Y - input_bias_0 |
# | G | | matrix_r1c0 matrix_r1c1 matrix_r1c2 || U - input_bias_1 | >> 8
# | R | | matrix_r2c0 matrix_r2c1 matrix_r2c2 || V - input_bias_2 |
# BGR转YUV:
# | Y | | matrix_r0c0 matrix_r0c1 matrix_r0c2 || B | | output_bias_0 |
# | U | | matrix_r1c0 matrix_r1c1 matrix_r1c2 || G | >> 8 + | output_bias_1 |
# | V | | matrix_r2c0 matrix_r2c1 matrix_r2c2 || R | | output_bias_2 |

# 3*3 CSC矩阵元素
# 类型: int16
# 取值范围: [-32677, 32676]
matrix_r0c0: 298
matrix_r0c1: 516
matrix_r0c2: 0
matrix_r1c0: 298
matrix_r1c1: -100
matrix_r1c2: -208
matrix_r2c0: 298
matrix_r2c1: 0
matrix_r2c2: 409

# RGB转YUV时的输出偏移
# 类型: uint8
# 取值范围: [0, 255]
output_bias_0: 16
output_bias_1: 128
output_bias_2: 128

# YUV转RGB时的输入偏移
# 类型: uint8
# 取值范围: [0, 255]
input_bias_0: 16
input_bias_1: 128
input_bias_2: 128

#===== 减均值、乘系数设置 =====
# 计算规则如下:
# 当uint8->uint8时, 本功能不起作用
# 当uint8->fp16时, pixel_out_chx(i) = [pixel_in_chx(i) - mean_chn_i - min_chn_i] * var_reci_chn_i

# 每个通道的均值
# 类型: uint8
# 取值范围: [0, 255]
mean_chn_0: 0
mean_chn_1: 0
mean_chn_2: 0
mean_chn_3: 0

# 每个通道的最小值
# 类型: float16
# 取值范围: [0, 255]
min_chn_0: 0.0
min_chn_1: 0.0
min_chn_2: 0.0
min_chn_3: 0.0

# 每个通道方差的倒数
# 类型: float16
# 取值范围: [-65504, 65504]
var_reci_chn_0: 1.0
var_reci_chn_1: 1.0
var_reci_chn_2: 1.0
var_reci_chn_3: 1.0
```



```
#===== 静态AIPP需设置，动态AIPP无需设置（end）  
=====
```

## 6.1.9 典型场景样例参考

### 6.1.9.1 YUV400\_U8 转 GRAY 格式

- **场景说明：**  
AIPP输入图像格式为YUV400\_U8、输出图像格式为GRAY，输入图像尺寸为224\*224，有效数据区域从左上角(0, 0)像素开始；原始网络模型的C=1，H和W均为220。
- **该场景下涉及以下AIPP配置：**
  - 开启抠图功能参数crop；
  - 抠图起始位置水平、垂直方向坐标load\_start\_pos\_h、load\_start\_pos\_w为0；
  - 无需配置crop\_size\_w和crop\_size\_h参数，此时抠图大小（crop\_size[W|H]）的宽和高取值来自模型转换时--input\_shape参数中的宽和高，将从(0, 0)点开始选取220\*220区域的数据；
  - 无需配置色域转换开关csc\_switch，并且对于同一个原始网络模型，如果AIPP输入的是YUV420SP\_U8图像，则可以使用同一套AIPP配置，即只取了Y通道的数据。

- **AIPP配置文件示例如下：**

```
aipp_op{  
  aipp_mode: static  
  csc_switch: false  
  crop: true  
  input_format: YUV400_U8  
  load_start_pos_h: 0  
  load_start_pos_w: 0  
  src_image_size_w: 224  
  src_image_size_h: 224  
  # 归一化系数需要根据用户模型实际需求配置，如下所取常见值仅作为示例  
  mean_chn_0: 128  
  min_chn_0: 0.0  
  var_reci_chn_0: 0.00390625  
}
```

### 6.1.9.2 YUV420SP\_U8 转 BGR 格式

- **场景说明：**  
AIPP输入图像格式为YUV420SP\_U8（NV12）、输出图像格式为BGR，输入图像尺寸为256\*256；原始网络模型的C=3，H和W与AIPP输入图像尺寸相同。
- **该场景涉及以下AIPP配置：**
  - 无需配置抠图功能参数crop；
  - 需要配置色域转换开关csc\_switch和相应的CSC矩阵参数。

- **AIPP配置文件示例如下：**

```
aipp_op {  
  aipp_mode: static  
  input_format: YUV420SP_U8  
  csc_switch: true  
  # 如果输入的是YVU420SP_U8（NV21）图像，则需要将rbuv_swap_switch参数设置为true  
  rbuv_swap_switch: false  
}
```

```
related_input_rank: 0
src_image_size_w: 256
src_image_size_h: 256
crop: false
matrix_r0c0: 298
matrix_r0c1: 516
matrix_r0c2: 0
matrix_r1c0: 298
matrix_r1c1: -100
matrix_r1c2: -208
matrix_r2c0: 298
matrix_r2c1: 0
matrix_r2c2: 409
input_bias_0: 16
input_bias_1: 128
input_bias_2: 128
# 归一化系数需要根据用户模型实际需求配置，如下所取常见值仅作为示例
# 归一化系数应用于色域转换和通道交换之后的通道
mean_chn_0: 104
mean_chn_1: 117
mean_chn_2: 123
min_chn_0: 0.0
min_chn_1: 0.0
min_chn_2: 0.0
var_reci_chn_0: 1.0
var_reci_chn_1: 1.0
var_reci_chn_2: 1.0
}
```

### 6.1.9.3 RGB888\_U8 转 RGB（或 BGR）格式

- **场景说明：**

AIPP输入图像格式为RGB888\_U8、输出图像格式为RGB，输入图像尺寸为250\*250，有效数据区域从左上角(0, 0)像素开始；原始网络模型的C=3，H和W均为240。

- **该场景下涉及以下AIPP配置：**

- 开启抠图功能参数crop；
- 抠图起始位置水平、垂直方向坐标load\_start\_pos\_h、load\_start\_pos\_w为0；
- 无需配置crop\_size\_w和crop\_size\_h参数，此时抠图大小（crop\_size[W|H]）的宽和高取值来自模型转换时--input\_shape参数中的宽和高，将从(0, 0)点开始选取240\*240区域的数据；
- 无需配置通道交换开关参数rbuv\_swap\_switch、色域转换开关参数csc\_switch和CSC矩阵参数。

- **AIPP配置文件示例如下：**

```
aipp_op {
  aipp_mode: static
  input_format: RGB888_U8
  csc_switch: false
  related_input_rank: 0
  src_image_size_w: 250
  src_image_size_h: 250
  crop: true
  load_start_pos_w: 0
  load_start_pos_h: 0
  # 如果原始模型需要的是BGR格式，则需要将rbuv_swap_switch参数设置为true
  rbuv_swap_switch: false
  # 归一化系数需要根据用户模型实际需求配置，此处取默认值，即不改变像素的值
  # 若配置归一化系数，将应用于通道交换之后的通道
}
```

## 6.2 单算子模型转换

### 6.2.1 什么是单算子描述文件

单算子描述文件是基于Ascend IR定义的单个算子的定义文件，包括算子的输入、输出及属性信息；借助该文件转换成适配昇腾AI处理器的离线模型后，可以验证单算子的功能。

### 6.2.2 如何将算子描述文件转成离线模型

本节给出单算子描述文件转成离线模型的详细步骤。

**步骤1** 参见[6.2.3 配置文件样例](#)和[6.2.4 描述文件参数说明](#)构造单算子描述文件。本章节以构造format为ND的Add单算子为例进行说明。

**步骤2** 以CANN软件包运行用户，将[步骤1](#)构造的单算子描述文件上传到开发环境任意目录，例如`$HOME/singleop/`目录下。

**步骤3** 执行如下命令生成离线模型。（如下命令中使用的目录以及文件均为样例，请以实际为准）

```
atc --singleop=$HOME/singleop/add.json --output=$HOME/singleop/out/op_model --  
soc_version=<soc_version>
```

关于参数的详细解释以及使用方法请参见[7.2.2.11 --singleop](#)。

**步骤4** 若提示如下信息，则说明模型转换成功。

```
ATC run success
```

成功执行命令后，在output参数指定的路径下，可查看离线模型文件\*.om。

----结束

### 6.2.3 配置文件样例

#### 6.2.3.1 单算子描述文件配置

不同输入或者不同format场景，单算子描述文件配置不同，本章节给出各场景的配置示例。

本章节中的单算子是基于Ascend IR定义的，描述文件为json格式。关于json描述文件中各参数的解释请参见[6.2.4 描述文件参数说明](#)，关于单算子的定义请参见《[算子清单（开放态）](#)》。

- **format为ND:**

该示例中的单算子转换后的离线模型为：add.om

```
[  
  {  
    "op": "Add",  
    "name": "add",  
    "input_desc": [  
      {  
        "format": "ND",  
        "shape": [3,3],  
        "type": "int32"
```

```
    },  
    {  
      "format": "ND",  
      "shape": [3,3],  
      "type": "int32"  
    }  
  ],  
  "output_desc": [  
    {  
      "format": "ND",  
      "shape": [3,3],  
      "type": "int32"  
    }  
  ]  
}
```

- **format为NCHW:**

该示例中的单算子转换后的离线模型为: conv2d.om

```
[  
{  
  "op": "Conv2D",  
  "name": "conv2d",  
  "input_desc": [  
    {  
      "format": "NCHW",  
      "shape": [1, 3, 16, 16],  
      "type": "float16"  
    },  
    {  
      "format": "NCHW",  
      "shape": [3, 3, 3, 3],  
      "type": "float16"  
    }  
  ],  
  "output_desc": [  
    {  
      "format": "NCHW",  
      "shape": [1, 3, 16, 16],  
      "type": "float16"  
    }  
  ],  
  "attr": [  
    {  
      "name": "strides",  
      "type": "list_int",  
      "value": [1, 1, 1, 1]  
    },  
    {  
      "name": "pads",  
      "type": "list_int",  
      "value": [1, 1, 1, 1]  
    },  
    {  
      "name": "dilations",  
      "type": "list_int",  
      "value": [1, 1, 1, 1]  
    }  
  ]  
}  
]
```

- **Tensor的实现format与原始format不同**

ATC模型转换时, 会将origin\_format与origin\_shape转成离线模型需要的format与shape。

该示例中的单算子转换后的离线模型为: add.om

```
[  
{
```

```
"op": "Add",
"name": "add",
"input_desc": [
  {
    "format": "NC1HWC0",
    "origin_format": "NCHW",
    "shape": [8, 1, 16, 4, 16],
    "origin_shape": [8, 16, 16, 4],
    "type": "float16"
  },
  {
    "format": "NC1HWC0",
    "origin_format": "NCHW",
    "shape": [8, 1, 16, 4, 16],
    "origin_shape": [8, 16, 16, 4],
    "type": "float16"
  }
],
"output_desc": [
  {
    "format": "NC1HWC0",
    "origin_format": "NCHW",
    "shape": [8, 1, 16, 4, 16],
    "origin_shape": [8, 16, 16, 4],
    "type": "float16"
  }
]
}
```

- **输入指定为常量**

该场景下，支持设置为常量的输入，新增**is\_const**和**const\_value**两个参数，分别表示是否为常量以及常量取值，**const\_value**当前仅支持一维list配置，具体配置个数由shape取值决定，例如，如下样例中shape为2，则**const\_value**中列表个数为2；**const\_value**中取值类型由type决定，假设type取值为float16，则单算子编译时会自动将**const\_value**中的取值转换为float16格式的取值。

该示例中的单算子转换后的离线模型为：resizeBilinearV2.om

```
[
  {
    "op": "ResizeBilinearV2",
    "name": "resizeBilinearV2",
    "input_desc": [
      {
        "format": "NHWC",
        "name": "x",
        "shape": [
          4,
          16,
          16,
          16
        ],
        "type": "float16"
      },
      {
        "format": "NHWC",
        "is_const": true,
        "const_value": [49, 49],
        "name": "size",
        "shape": [
          2
        ],
        "type": "int32",
        "test": [7, 7.0]
      }
    ],
    "output_desc": [
      {

```

```
    "format": "NHWC",
    "name": "y",
    "shape": [
      4,
      48,
      48,
      16
    ],
    "type": "float"
  }
],
"attr": [
  {
    "name": "align_corners",
    "type": "bool",
    "value": false
  },
  {
    "name": "half_pixel_centers",
    "type": "bool",
    "value": false
  }
]
}
```

- **可选输入 ( optional input ) :**

当存在可选输入，且可选输入没有输入数据时，则必须将可选输入的format配置为RESERVED，同时将type配置为UNDEFINED；若可选输入有输入数据时，则按其输入数据的format、type配置即可。

该示例中的单算子转换后的离线模型为：matMulV2.om

```
[
  {
    "op": "MatMulV2",
    "name": "matMulV2",
    "input_desc": [
      {
        "format": "ND",
        "shape": [16, 16],
        "type": "float"
      },
      {
        "format": "ND",
        "shape": [16, 16],
        "type": "float"
      },
      {
        "format": "RESERVED",
        "shape": [],
        "type": "UNDEFINED"
      },
      {
        "format": "RESERVED",
        "shape": [],
        "type": "UNDEFINED"
      }
    ],
    "attr": [
      {
        "name": "transpose_x1",
        "type": "bool",
        "value": false
      },
      {
        "name": "transpose_x2",
        "type": "bool",
        "value": false
      }
    ]
  }
]
```

```
],  
"output_desc": [  
  {  
    "format": "ND",  
    "shape": [16, 16],  
    "type": "float"  
  }  
]  
}  
]
```

- **输入个数不确定（动态输入场景）：**

该场景下，单算子的输入个数不确定。此处以AddN单算子为例。

该示例中的单算子转换后的离线模型为：addN.om

- 构造的单算子json文件使用动态输入dynamic\_input参数，而不使用Tensor的名称name参数，构造的描述文件为：

该场景下算子的dynamic\_input取值必须和算子信息库中该算子定义的输入name的取值相同。

具体设置几个输入，由AddN单算子描述文件属性参数中N的取值决定，用户可以自行修改输入的个数，但是必须和属性中N的取值匹配。（该说明仅针对AddN算子生效，其他动态输入算子的约束以具体算子为准。）

```
[  
  {  
    "op": "AddN",  
    "name": "addN",  
    "input_desc": [  
      {  
        "dynamic_input": "x",  
        "format": "NCHW",  
        "shape": [1,3,166,166],  
        "type": "float32"  
      },  
      {  
        "dynamic_input": "x",  
        "format": "NCHW",  
        "shape": [1,3,166,166],  
        "type": "int32"  
      },  
      {  
        "dynamic_input": "x",  
        "format": "NCHW",  
        "shape": [1,3,166,166],  
        "type": "float32"  
      }  
    ],  
    "output_desc": [  
      {  
        "format": "NCHW",  
        "shape": [1,3,166,166],  
        "type": "float32"  
      }  
    ],  
    "attr": [  
      {  
        "name": "N",  
        "type": "int",  
        "value": 3  
      }  
    ]  
  }  
]
```

- 构造的单算子json文件使用Tensor的名称name参数，而不使用动态输入dynamic\_input参数，构造的描述文件为：

该场景下算子的name取值必须和算子原型定义中算子的输入名称相同，根据输入的个数自动生成x0、x1、x2……。

具体设置几个Tensor名称，由AddN单算子描述文件属性参数中N的取值决定，用户可以自行修改Tensor名称的个数，但是必须和属性中N的取值匹配，例如N取值为3，则name取值分别设置为x0、x1、x2。（该说明仅针对AddN算子生效，其他动态输入算子的约束以具体算子为准。）

```
[
  {
    "op": "AddN",
    "name": "addN",
    "input_desc": [
      {
        "name": "x0",
        "format": "NCHW",
        "shape": [1,3,166,166],
        "type": "float32"
      },
      {
        "name": "x1",
        "format": "NCHW",
        "shape": [1,3,166,166],
        "type": "int32"
      },
      {
        "name": "x2",
        "format": "NCHW",
        "shape": [1,3,166,166],
        "type": "float32",
      }
    ],
    "output_desc": [
      {
        "format": "NCHW",
        "shape": [1,3,166,166],
        "type": "float32"
      }
    ],
    "attr": [
      {
        "name": "N",
        "type": "int",
        "value": 3
      }
    ]
  }
]
```

### 6.2.3.2 多组算子描述文件配置

描述文件支持定义多组算子Json文件配置，一组配置包括算子类型、算子输入和输出信息、视算子情况决定是否包括属性信息。

如果Json文件配置了多组算子，则模型转换完成后，会生成多组算子对应的.om离线模型文件。如下配置文件只是样例，请根据实际情况进行修改。

```
[
  {
    "op": "MatMul",
    "name": "matMul01",
    "input_desc": [
      {
        "format": "ND",
        "shape": [
          16,
          16
        ]
      }
    ],
```



```
        "type": "float16"
      },
      ...
    ],
    "output_desc": [
      {
        "format": "ND",
        "shape": [
          16,
          16
        ],
        "type": "float16"
      }
    ],
    "attr": [
      {
        "name": "alpha",
        "type": "float",
        "value": 1.0
      },
      ...
    ]
  },
  {
    "op": "MatMul",
    "name": "matMul02",
    "input_desc": [
      {
        "format": "ND",
        "shape": [
          256,
          256
        ],
        "type": "float16"
      },
      ...
    ],
    "output_desc": [
      {
        "format": "ND",
        "shape": [
          256,
          256
        ],
        "type": "float16"
      }
    ],
    "attr": [
      {
        "name": "alpha",
        "type": "float",
        "value": 1.0
      },
      ...
    ]
  }
]
```

### 6.2.3.3 动态 Shape 单算子描述文件配置

动态shape场景，单算子描述文件根据场景不同，内容也有差异，本章节就给出不同场景下的配置样例。

- 模型编译时不指定Shape，模型执行时根据输入固定Shape，能推导出具体输出Shape：

```
[
  {
    "op": "Add",
```

```
"name": "add",
"input_desc": [
  {
    "format": "ND",
    "shape": [-1,16],
    "shape_range": [[0, 32]],
    "type": "int64"
  },
  {
    "format": "ND",
    "shape": [-1,16],
    "shape_range": [[0, 32]],
    "type": "int64"
  }
],
"output_desc": [
  {
    "format": "ND",
    "shape": [-1,16],
    "shape_range": [[0,32]],
    "type": "int64"
  }
]
}
```

- 模型编译时不指定Shape，模型执行时根据输入固定Shape和常量，能推导出具体输出Shape：

```
[
  {
    "op": "TopK",
    "name": "topK",
    "input_desc": [
      {
        "format": "ND",
        "shape": [-1],
        "shape_range": [[1,-1]],
        "type": "int32"
      },
      {
        "format": "ND",
        "shape": [], #推理时会传入常量
        "type": "int32"
      }
    ],
    "output_desc": [
      {
        "format": "ND",
        "shape": [-1],
        "shape_range": [[1,-1]],
        "type": "int32"
      },
      {
        "format": "ND",
        "shape": [-1],
        "shape_range": [[1,-1]],
        "type": "int32"
      }
    ],
    "attr": [
      {
        "name": "sorted",
        "type": "bool",
        "value": true
      }
    ]
  }
]
```

- 模型编译时不指定Shape，模型执行时根据输入固定Shape，无法得到算子的准确输出Shape，但可以得到输出Shape的范围。

该场景下在输出参数output\_desc中将算子输出TensorDesc中Shape为动态维度的纬度值记为“-1”，并对其“-1”的维度给出shape\_range取值范围：

```
[
  {
    "op": "Where",
    "name": "where",
    "input_desc": [
      {
        "format": "ND",
        "shape": [-1],
        "shape_range": [[1,-1]],
        "type": "int32"
      }
    ],
    "output_desc": [
      {
        "format": "ND",
        "shape": [-1, 1],
        "shape_range": [[1,-1]],
        "type": "int64"
      }
    ]
  }
]
```

6.2.4 描述文件参数说明

单算子描述文件是由OpDesc的数组构成的json文件，本章节详细介绍该文件中各参数的含义。

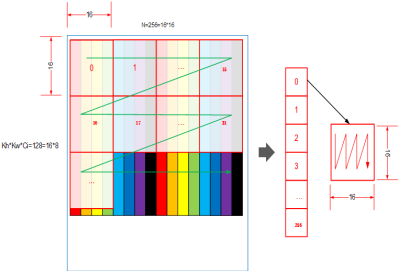
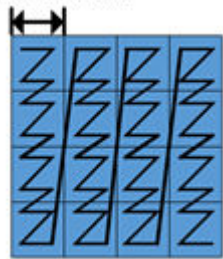
表 6-5 OpDesc 参数说明

属性名	类型	说明	是否必填
compile_flag	INT32	<p><b>该参数废弃，不建议使用，后续版本将会删除。</b></p> <p>编译类型。取值如下：</p> <ul style="list-style-type: none"><li>0：表示进行精确编译。精确编译是指按照用户指定的维度信息、在编译时系统内部不做任何转义直接编译，其中，AI CPU算子不受该标记影响。</li><li>1：表示进行模糊编译。模糊编译是指对于支持动态Shape的算子，在编译时系统内部对可变维度做了泛化后再进行编译。如果用户无法获取算子的Shape范围，又想编译一次达到多次执行推理的目的时，可以使用模糊编译特性。</li></ul> <p>默认值为0。</p> <p>使用约束：当前仅支持transformer网络模型涉及的算子。</p>	否
op	string	算子类型。	是

属性名	类型	说明	是否必填
name	string	单算子模型文件的名称。 如果不设置name参数，则模型文件名的命名规则默认为：序号_opType_输入的描述(dtype_format_shape)_输出的描述(dtype_format_shape)，例如，0_Add_3_2_3_3_3_2_3_3_2_3_3.om。 dtype以及format对应枚举值请从\${INSTALL_DIR}/include/graph/types.h文件中查看，枚举值依次递增。其中，\${INSTALL_DIR}请替换为CANN软件安装后文件存储路径。例如，若安装的Ascend-cann-toolkit软件包，则安装后文件存储路径为：\$HOME/Ascend/ascend-toolkit/latest。	否
input_desc	TensorDesc数组	算子输入描述。	是
output_desc	TensorDesc数组	算子输出描述。	是
attr	Attr数组	算子属性。	否

表 6-6 TensorDesc 数组参数说明

属性名	类型	说明
dynamic_input	string	可选。动态输入，取值必须和算子信息库中该算子定义的输入name相同。 该参数用于设置算子动态输入的分组与动态输入的个数，例如算子原型定义中某算子的动态输入为： .DYNAMIC_INPUT(x,...) .DYNAMIC_INPUT(y,...) 则表示动态输入有两组，分别为x，y。每一组的输入个数，根据dynamic_input的个数确定。具体设置原则可以参见TensorDesc数组中name参数的说明。 <ul style="list-style-type: none"><li>如果构造的单算子描述文件中已经设置过name参数，则该参数可选。</li><li>如果构造的单算子描述文件中没有name参数，则该参数必填。</li><li>如果同时存在dynamic_input和name参数，则以name参数设置的为准。</li></ul>

属性名	类型	说明
format	string	<p>必填。Tensor的排布格式，配置为算子原始框架支持的Format。</p> <p>当前支持的Format格式以及对应的枚举如下：</p> <ul style="list-style-type: none"><li>• NCHW: 0</li><li>• NHWC: 1</li><li>• ND: 2，表示支持任意格式。</li><li>• NC1HWC0: 3，5维数据格式。其中，C0与微架构强相关，该值等于cube单元的size，例如16；C1是将C维度按照C0切分：C1=C/C0。如果结果不整除，向上取整。</li><li>• FRACTAL_Z: 4，用于定义卷积权重数据格式，由FT Matrix（FT：Filter，卷积核）变换得到。FRACTAL_Z是送往Cube的最终数据格式，采用“C1HW,N1,N0,C0”的4维数据排布。数据有两层Tiling，如下图所示：</li></ul> <div></div> <p>第一层与Cube的Size相关，数据按照列的方向连续（小n）；第二层与矩阵的Size相关，数据按照行的方向连续（大Z）。</p> <ul style="list-style-type: none"><li>• FRACTAL_NZ: 29，分形格式，如Feature Map的数据存储，在cube单元计算时，输出矩阵的数据格式为NW1H1H0W0。整个矩阵被分为（H1*W1）个分形，按照column major排布，形状如N字形；每个分形内部有（H0*W0）个元素，按照row major排布，形状如z字形。考虑到数据排布格式，将NW1H1H0W0数据格式称为Nz格式。其中，H0,W0表示一个分形的大小，示意图如下所示：</li></ul> <div><p>Fractal Matrix Size</p><p>Matrix C</p></div> <ul style="list-style-type: none"><li>• RESERVED: 40，当存在可选输入，且可选输入没有输入数据时，则必须将可选输入的format配置为RESERVED，同时将type配置为UNDEFINED；若可选</li></ul>

属性名	类型	说明
		<p>输入有输入数据时，则按其输入数据的format、type配置即可。</p> <p>模型转换完毕，上述format在对应om文件名中以对应的枚举呈现，例如若输入为NHWC格式，则展示为1。</p>
origin_format	string	<p>可选。Tensor的原始Format。</p> <p>不带此字段时，默认Tensor的实现Format与原始Format一致。</p>
name	string	<p>可选。Tensor的名称。算子的输入为动态输入时，需要设置该字段。</p> <p>该参数用于设置每一组动态输入中，具体输入的名称，每一个输入名称为算子原型中定义的输入名称+编号，编号根据dynamic_input的个数确定，从0开始依次递增。</p> <ul style="list-style-type: none"><li>如果构造的单算子描述文件中已经设置过dynamic_input参数，则该参数可选。</li><li>如果构造的单算子描述文件中没有dynamic_input参数，则该参数必填。</li><li>如果同时存在dynamic_input和name参数，则以name参数设置的为准。</li></ul>
shape	int数组	<p>必填。Tensor的Shape，例如[1, 224, 224, 3]，实际Shape乘积不能超过int32最大值（2147483647）。</p> <ul style="list-style-type: none"><li>静态Shape场景： Shape维度以及取值都为固定值，该场景下不需要再配置shape_range参数。</li><li>Shape为常量场景： 如果希望指定算子输入、输出Shape为标量，则该参数需要设置为"[]"形式，比如"shape": []。该场景下不需要再配置shape_range参数。</li><li>动态Shape场景，Shape取值有如下场景：<ul style="list-style-type: none"><li>Shape维度确定，但是某一维度的取值不确定，则该不确定的维度取值设置为“-1”，例如[16,-1,20,-1]，该场景下还需要与shape_range参数配合使用，用于给出“-1”维度的取值范围。例如：<pre>"shape": [-1,16], "shape_range": [[0,32]],</pre></li><li>Shape维度也不确定，该场景下Shape取值为“-2”，例如"shape": [-2]，该场景下不需要配置shape_range参数（当前版本暂不支持）。</li></ul></li></ul>
origin_shape	string	<p>可选。Tensor的原始Shape。</p> <p>不带此字段时，默认Tensor的实现Shape与原始Shape一致。</p>

属性名	类型	说明
type	string	<p>必填。Tensor的数据格式，支持的type以及对应的枚举如下：</p> <ul style="list-style-type: none"><li>• bool: 12</li><li>• int8: 2</li><li>• uint8: 4</li><li>• int16: 6</li><li>• uint16: 7</li><li>• int32: 3</li><li>• uint32: 8</li><li>• int64: 9</li><li>• uint64: 10</li><li>• float16/fp16/half: 1</li><li>• float/float32: 0</li><li>• double: 11</li><li>• complex32: 33</li><li>• complex64: 16</li><li>• complex128: 17</li><li>• uint1: 30</li><li>• bfloat16: 27</li><li>• UNDEFINED: 28，当存在可选输入，且可选输入没有输入数据时，则必须将可选输入的type配置为UNDEFINED，同时将format配置为RESERVED；若可选输入有输入数据时，则按其输入数据的format、type配置即可。</li></ul> <p>模型转换完毕，上述type在对应om文件名中以对应的枚举呈现，例如若输入为int8类型，则展示为2。</p>
shape_range	int[2]数组	<p>可选。Shape为动态时（不包括-2场景），unknown shape的取值范围。</p> <p>例如，若Shape取值为[16,-1,20,-1]：其中的-1表示unknown shape。</p> <p>shape_range取值为[1,128],[1,-1]：[1,128]表示从1到128的取值范围，对应Shape参数中第一个-1；[1,-1]表示从1到无穷大的取值范围，对应Shape参数中第二个-1。</p>
is_const	bool	<p>可选，表示输入是否为常量：</p> <ul style="list-style-type: none"><li>• true：常量。</li><li>• false：非常量。</li></ul>

属性名	类型	说明
const_value	list	可选，常量取值。 当前仅支持一维list配置，list中具体配置个数由Shape取值决定。例如，Shape取值为2，则const_value中列表个数为2。 取值类型由type决定，假设type取值为float16，则单算子编译时会自动将const_value中的取值转换为float16格式的取值。

表 6-7 Attr 数组参数说明

属性名	类型	说明
name	string	必填。属性名。
type	string	必填。属性值的类型，支持的类型有： <ul style="list-style-type: none"><li>bool</li><li>int</li><li>float</li><li>string</li><li>list_bool</li><li>list_int</li><li>list_float</li><li>list_string</li><li>list_list_int</li><li>data_type</li></ul>



属性名	类型	说明
value	由type的取值决定	<p>必填。属性值，根据type不同，属性值不同，举例如下：</p> <ul style="list-style-type: none"><li>• bool: true/false</li><li>• int: 10</li><li>• float: 1.0</li><li>• string: “NCHW”</li><li>• list_bool: [false, true]</li><li>• list_int: [1, 224, 224, 3]</li><li>• list_float: [1.0, 0.0]</li><li>• list_string: ["str1","str2"]</li><li>• list_list_int: [[1, 3, 5, 7], [2, 4, 6, 8]]</li><li>• data_type: "DT_FLOAT"或该枚举值对应的数字，例如0。</li></ul> <p>其他取值请参见\${INSTALL_DIR}/include/graph/types.h中DataType的枚举值或枚举值对应的数字。其中，\${INSTALL_DIR}请替换为CANN软件安装后文件存储路径。例如，若安装的Ascend-cann-toolkit软件包，则安装后文件存储路径为：\$HOME/Ascend/ascend-toolkit/latest。</p>

# 7 参数说明

## 参数概览

使用ATC工具转换模型之前，首先查看使用工具过程中的一些限制，然后借助本章节提供的参数概览功能，可以快速预览相关参数。

## 基础功能参数

## 高级功能参数

## 7.1 参数概览

使用ATC工具转换模型之前，首先查看使用工具过程中的一些限制，然后借助本章节提供的参数概览功能，可以快速预览相关参数。

### 总体约束

在进行模型转换前，请务必查看如下约束要求：

- 如果要将Faster RCNN等网络模型转成适配昇腾AI处理器的离线模型，则务必参见 [8.1 定制网络修改 \(Caffe\)](#) 先修改prototxt模型文件。
- 支持原始框架类型为Caffe、TensorFlow、MindSpore、ONNX的模型转换：
  - 当原始框架类型为Caffe、MindSpore、ONNX时，输入数据类型为FP32、FP16、UINT8（通过配置数据预处理[7.3.1.3 --insert\\_op\\_conf](#)实现）。
  - 当原始框架类型为TensorFlow时，输入数据类型为FP16、FP32、UINT8、INT32、INT64、BOOL。
- 当原始框架类型为Caffe时，模型文件（.prototxt）和权重文件（.caffemodel）的op name、op type必须保持名称一致（包括大小写）。
- 当原始框架类型为TensorFlow时，只支持FrozenGraphDef格式。
- 对于Caffe框架网络模型：输入数据最大支持四维，转维算子（reshape、expanddim等）不能输出五维。
- 模型中的所有层算子除const算子外，输入和输出需要满足dim!=0。
- 只支持[9.2 算子规格参考](#)中的算子，并需满足算子限制条件。
- 由于软件约束（动态shape场景下暂不支持输入数据为DT\_INT8），量化后的部署模型使用ATC工具进行模型转换时，不能使用动态shape相关参数，例如[7.2.2.8 --](#)

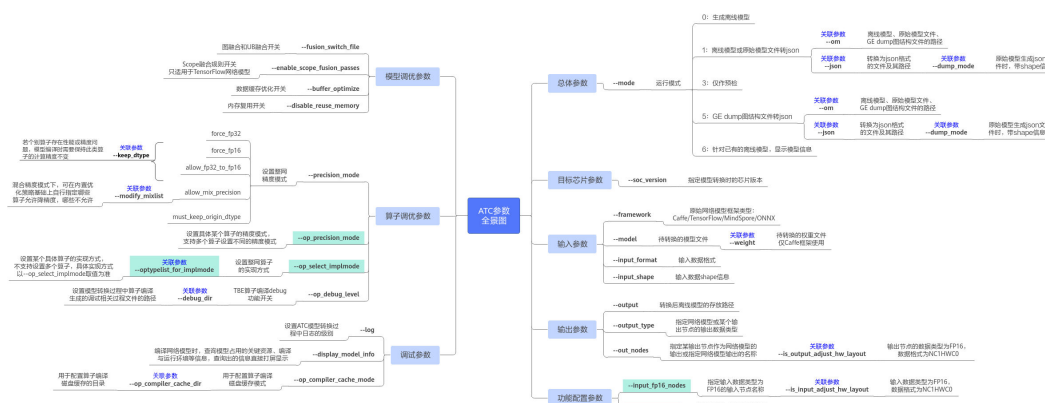
**dynamic\_batch\_size**和**7.2.2.9 --dynamic\_image\_size**等，否则模型转换会失败。

- 使用AMCT工具量化后的部署模型，使用ATC工具进行模型转换时，不能再使用高精度特性，比如不能再通过**7.3.3.1 --precision\_mode**参数配置**force\_fp32**或**must\_keep\_origin\_dtype**（原图fp32输入）；不能再通过**7.3.3.2 --precision\_mode\_v2**参数配置**origin**；不能通过**7.3.3.3 --op\_precision\_mode**配置**high\_precision**参数等。在高精度模式下设置量化参数，既拿不到量化的性能收益，也拿不到高精度模式的精度收益。

## 参数概览

图7-1列出了所有芯片共用的ATC参数（参数只在某些芯片下使用的未列出），其中表示参数互斥，不能同时使用；关联参数表示需要相互配合或者某些场景下需要配合使用。通过该图，您可以快速了解ATC中的部分参数，更多详细参数请参见表7-1。

图 7-1 ATC 参数全景图



## 须知

- 如果通过**atc --help**命令查询出的参数未解释在表7-1，则说明该参数预留或适用于其他芯片版本，用户无需关注。
- 使用atc命令进行模型转换时，命令有两种方式，用户根据实际情况进行选择：
  - atc param1=value1 param2=value2 ...**（value值前面不能有空格，否则会导致截断，param取的value值为空）
  - atc param1 value1 param2 value2 ...**
- 参数是否必选以--mode为0和3为准。
- 使用ATC参数时，参数名支持以--作为前缀（例如--help），也支持以-作为前缀（例如-help），当使用-作为前缀时，在执行atc命令时，会自动转换为--。本文的参数名均以--前缀为例。
- 使用ATC参数时，参数名称支持以下划线连接两个字符串（例如soc\_version），也支持以中划线连接两个字符串（例如soc-version）。本文的参数名称均以下划线连接两个字符串（例如soc\_version）为例。

表 7-1 ATC 参数概览

ATC参数名称	参数简述（具体说明见参数描述章节）	是否 必选	默认值
<a href="#">7.2.1.1 --help或--h</a>	显示帮助信息。	否	不涉及
<a href="#">7.2.1.2 --mode</a>	运行模式。	否	0
<a href="#">7.2.2.1 --model</a>	原始模型文件路径与文件名。	是	不涉及
<a href="#">7.2.2.2 --weight</a>	权重文件路径与文件名。	否	不涉及
<a href="#">7.2.2.3 --om</a>	需要转换为json格式的离线模型或原始模型文件的路径和文件名。	否	不涉及
<a href="#">7.2.2.4 --framework</a>	原始框架类型。	是	不涉及
<a href="#">7.2.2.5 --input_format</a>	输入数据格式。	否	Caffe、ONNX默认为NCHW；TensorFlow默认为NHWC。
<a href="#">7.2.2.6 --input_shape</a>	模型输入数据的shape。	否	不涉及
<a href="#">7.2.2.7 --input_shape_range</a>	指定模型输入数据的shape范围。 <b>该参数已废弃，请勿使用。</b>	否	不涉及
<a href="#">7.2.2.8 --dynamic_batch_size</a>	设置动态batch档位参数，适用于执行推理时，每次处理图片数量不固定的场景。	否	不涉及
<a href="#">7.2.2.9 --dynamic_image_size</a>	设置输入图片的动态分辨率参数。适用于执行推理时，每次处理图片宽和高不固定的场景。	否	不涉及
<a href="#">7.2.2.10 --dynamic_dims</a>	设置ND格式下动态维度的档位。适用于执行推理时，每次处理任意维度的场景。	否	不涉及
<a href="#">7.2.2.11 --singleop</a>	单算子定义文件，将单个算子json文件转换成适配昇腾AI处理器的离线模型。	否	不涉及
<a href="#">7.2.2.12 --distributed_cluster_build</a>	用于分布式部署的模型编译开关，使能该参数后，生成的离线模型将用于分布式部署。	否	不涉及
<a href="#">7.2.2.13 --cluster_config</a>	指定目标执行环境的逻辑拓扑结构配置文件。	否	不涉及

ATC参数名称	参数简述（具体说明见参数描述章节）	是否 必选	默认值
<a href="#">7.2.2.14 --enable_graph_parallel</a>	是否对原始模型进行自动切分。	否	不涉及
<a href="#">7.2.2.15 --graph_parallel_option_path</a>	对原始模型进行切分时，切分策略配置文件路径。	否	不涉及
<a href="#">7.2.2.16 --shard_model_dir</a>	指定切片模型文件所在路径。	否	不涉及
<a href="#">7.2.2.17 --model_relation_config</a>	表达多个切片模型间的数据关联和分布式通信组关系的配置文件。	否	不涉及
<a href="#">7.2.3.1 --output</a>	<ul style="list-style-type: none"><li>如果是开源框架的网络模型，存放转换后的离线模型的路径以及文件名。</li><li>如果是单算子描述文件，存放转换后的单算子模型的路径。</li></ul>	是	不涉及
<a href="#">7.2.3.2 --output_type</a>	指定网络输出数据类型或指定某个输出节点的输出类型。	否	不涉及
<a href="#">7.2.3.3 --check_report</a>	预检结果保存文件路径和文件名。	否	check_result.json
<a href="#">7.2.3.4 --json</a>	离线模型或原始模型文件转换为json格式文件的路径和文件名。	否	不涉及
<a href="#">7.2.3.5 --host_env_os</a>	若模型编译环境的操作系统及其架构与模型运行环境不一致时，则需使用本参数设置模型运行环境的操作系统类型。	否	不涉及
<a href="#">7.2.3.6 --host_env_cpu</a>	若模型编译环境的操作系统及其架构与模型运行环境不一致时，则需使用本参数设置模型运行环境的操作系统架构。	否	不涉及
<a href="#">7.2.4.1 --soc_version</a>	模型转换时指定芯片版本。	是	不涉及
<a href="#">7.2.4.2 --core_type</a>	设置网络模型使用的Core类型，若网络模型中包括Cube算子，则只能使用AiCore。	否	AiCore
<a href="#">7.2.4.3 --aicore_num</a>	设置模型编译时使用的aicore数目。	否	默认值为最大值
<a href="#">7.2.4.4 --virtual_type</a>	是否支持离线模型在算力分组生成的虚拟设备上运行。	否	0
<a href="#">7.3.1.1 --out_nodes</a>	指定输出节点。	否	不涉及

ATC参数名称	参数简述（具体说明见参数描述章节）	是否 必选	默认值
<a href="#">7.3.1.2 -- input_fp16_nodes</a>	指定输入数据类型为FP16的输入节点名称。	否	不涉及
<a href="#">7.3.1.3 -- insert_op_conf</a>	插入算子的配置文件路径与文件名，例如aipp预处理算子。	否	不涉及
<a href="#">7.3.1.4 -- external_weight</a>	生成om模型文件时，是否将原始网络中的Const/Constant节点的权重保存在单独的文件中，同时将节点类型转换为FileConstant类型。	否	0
<a href="#">7.3.1.5 -- op_name_map</a>	扩展算子（非标准算子）映射配置文件路径和文件名，不同的网络中某扩展算子的功能不同，可以指定该扩展算子到具体网络中实际运行的扩展算子的映射。	否	不涉及
<a href="#">7.3.1.6 -- is_input_adjust_hw_layout</a>	用于指定网络输入数据类型是否为FP16，数据格式是否为NC1HWC0。	否	false
<a href="#">7.3.1.7 -- is_output_adjust_hw_layout</a>	用于指定网络输出的数据类型是否为FP16，数据格式是否为NC1HWC0。	否	false
<a href="#">7.3.2.1 -- disable_reuse_memory</a>	内存复用开关。	否	0
<a href="#">7.3.2.2 -- fusion_switch_file</a>	融合开关配置文件路径以及文件名。	否	不涉及
<a href="#">7.3.2.3 -- enable_scope_fusion_passes</a>	指定编译时需要生效的融合规则列表。	否	不涉及
<a href="#">7.3.2.4 -- enable_small_channel</a>	是否使能small channel的优化，使能后在channel<=4的首层卷积会有性能收益。	否	0
<a href="#">7.3.2.5 -- ac_parallel_enable</a>	动态shape图中，是否允许AI CPU算子和AI Core算子并行运行。	否	0
<a href="#">7.3.2.6 -- compression_optimize_conf</a>	压缩优化功能配置文件路径以及文件名。	否	不涉及
<a href="#">7.3.2.7 -- buffer_optimize</a>	是否开启数据缓存优化。	否	l2_optimize

ATC参数名称	参数简述（具体说明见参数描述章节）	是否 必选	默认值
<a href="#">7.3.2.8 --mdl_bank_path</a>	加载模型调优后自定义知识库的路径。	否	\${HOME}/Ascend/latest/data/aoe/custom/graph/<soc_version>
<a href="#">7.3.3.1 --precision_mode</a>	设置网络模型的精度模式。	否	force_fp16
<a href="#">7.3.3.2 --precision_mode_v2</a>	设置网络模型的精度模式。	否	fp16
<a href="#">7.3.3.3 --op_precision_mode</a>	设置具体某个算子的精度模式，通过该参数可以为不同的算子设置不同的精度模式。	否	不涉及
<a href="#">7.3.3.4 --modify_mixlist</a>	混合精度场景下，修改算子使用混合精度名单。	否	不涉及
<a href="#">7.3.3.5 --op_select_implmode</a>	设置网络模型中算子是高精度实现模式还是高性能实现模式。	否	high_performance
<a href="#">7.3.3.6 --optypelist_for_implmode</a>	设置optype列表中算子的实现模式，算子实现模式包括high_precision、high_performance两种。	否	不涉及
<a href="#">7.3.3.7 --keep_dtype</a>	保持原始模型编译时个别算子的计算精度不变。	否	不涉及
<a href="#">7.3.3.8 --customize_dtypes</a>	模型编译时自定义算子的计算精度。	否	不涉及
<a href="#">7.3.3.9 --op_bank_path</a>	加载AOE调优后自定义知识库的路径。	否	默认自定义知识库路径 \$HOME/Ascend/latest/data/aoe/custom/op
<a href="#">7.3.3.10 --op_debug_level</a>	TBE算子编译debug功能开关。	否	0
<a href="#">7.3.4.1 --dump_mode</a>	是否生成带shape信息的json文件。	否	0
<a href="#">7.3.4.2 --log</a>	设置ATC模型转换过程中显示日志的级别。	否	null

ATC参数名称	参数简述（具体说明见参数描述章节）	是否 必选	默认值
<a href="#">7.3.4.3 --debug_dir</a>	用于配置保存模型转换、网络迁移过程中算子编译生成的调试相关过程文件的路径，包括算子.o/.json/.cce等文件。	否	./ kernel_meta
<a href="#">7.3.4.4 --op_compiler_cache_mode</a>	用于配置算子编译磁盘缓存模式。	否	disable
<a href="#">7.3.4.5 --op_compiler_cache_dir</a>	用于配置算子编译磁盘缓存的目录。	否	\$HOME/ atc_data
<a href="#">7.3.4.6 --display_model_info</a>	模型编译时或对已有的离线模型，查询模型占用的关键资源信息、编译与运行环境等信息。	否	0
<a href="#">7.3.4.7 --shape_generalized_build_mode</a>	图编译时Shape的编译方式。 该参数后续版本会废弃，请勿使用。	否	shape_precise
<a href="#">7.3.4.8 --status_check</a>	控制编译算子时是否添加溢出检测逻辑。	否	0
<a href="#">7.3.4.9 --op_debug_config</a>	使能Global Memory（DDR）内存检测功能的配置文件路径及文件名。	否	不涉及
<a href="#">7.3.4.10 --atomic_clean_policy</a>	是否集中清理网络中所有atomic算子（含有atomic属性的算子都是atomic算子）占用的内存。	否	0
<a href="#">7.3.4.11 --deterministic</a>	是否开启确定性计算。	否	0

## 7.2 基础功能参数

### 7.2.1 总体选项

#### 7.2.1.1 --help 或--h

##### 功能说明

显示帮助信息。

##### 关联参数

无。



参数取值

无。

推荐配置及收益

无。

示例

```
atc --help
```

支持的型号

Atlas 200/300/500 推理产品  
Atlas 推理系列产品

依赖约束

无。

7.2.1.2 --mode

功能说明

运行模式。

关联参数

- 若7.2.1.2 --mode取值为1或5，则需要与7.2.2.3 --om、7.2.3.4 --json参数配合使用。如果将原始模型文件转换成带shape信息的json文件，则还需要与7.3.4.1 --dump\_mode参数配合使用。
- 若7.2.1.2 --mode取值为6，则只需要与7.2.2.3 --om参数配合使用。

参数取值

**参数值：**

- 0：生成适配昇腾AI处理器的离线模型，模型文件格式为\*.om。
- 1：离线模型或原始模型文件转json，方便查看模型中的参数信息。
- 3：仅做预检，检查模型文件的内容是否合法。
- 5：GE（Graph Engine）dump图结构文件转json，用于解析GE图编码过程中产生的dump图结构（ge\_proto\*.txt格式文件，ge\_onnx\*.pbtxt暂不支持），然后将dump图结构转换成json文件，方便用户定位。
- 6：针对已有的**离线模型**，显示模型信息，包括模型占用的关键资源信息、编译与运行环境等信息。

**参数值约束：**

若7.2.1.2 --mode取值为5，需要先设置如下环境变量，生成dump图结构文件，然后才能进行下一步的转换：

打印模型转换过程中各个阶段的图描述信息。

```
export DUMP_GE_GRAPH=1
```

上述环境变量控制dump图的内容多少：取值为1，全量dump；取值为2：不含有权重等数据的基本版dump；取值为3：只显示节点关系的精简版dump。

设置上述环境变量后，还可以设置如下环境变量，控制dump图的个数。

```
export DUMP_GRAPH_LEVEL=1
```

取值为1，dump所有图；取值为2：dump除子图外的所有图；取值为3：dump最后的生成图；默认值为2。

设置上述变量后，在执行atc命令的当前路径会生成相应的图文件。

**参数默认值：0**

## 推荐配置及收益

无。

## 示例

### 说明

使用atc命令进行模型转换时，命令有两种方式，用户根据实际情况进行选择，本章节以选择第一种方式为例进行说明：

- **atc param1=value1 param2=value2 ...**（value值前面不能有空格，否则会导致截断，param取的value值为空）
- **atc param1 value1 param2 value2 ...**
- 参数值取值为0：  

```
atc --mode=0 --framework=3 --model=$HOME/module/resnet50_tensorflow*.pb --output=$HOME/module/out/tf_resnet50 --soc_version=<soc_version>
```
- 参数值取值为1：
  - 离线模型转换为json  

```
--mode=1 --om=$HOME/module/out/tf_resnet50.om --json=$HOME/module/out/tf_resnet50.json
```
  - 原始模型文件转换为json  

```
--mode=1 --om=$HOME/module/resnet50_tensorflow*.pb --json=$HOME/module/out/tf_resnet50.json --framework=3
```
- 参数值取值为5：  
GE dump图结构文件转json  

```
--mode=5 --om=$HOME/module/ge_proto_00000000_PreRunBegin.txt --json=$HOME/module/out/ge_proto.json
```
- 参数值取值为6：  

```
atc --mode=6 --om=$HOME/module/out/tf_resnet50.om
```

命令执行完毕，屏幕会打印类似如下信息：

```
===== Display Model Info start =====
# 模型转换使用的atc命令
Original Atc command line: ${INSTALL_DIR}/bin/atc.bin --model=$HOME/module/resnet50_tensorflow*.pb --framework=3 --output=$HOME/module/out/tf_resnet50 --soc_version=<soc_version> --display_model_info=1
# ATC软件版本信息、soc_version版本信息、原始框架信息
system info: atc_version[xxx], soc_version[xxx], framework_type[xxx].
# 运行时的占用内存、权重大小、逻辑stream数目、event数目
resource info: memory_size[xxx B], weight_size[xxx B], stream_num[xxx], event_num[xxx].
# 离线模型文件中各分区大小、包括ModelDef、权重、tbekernels、taskinfo占用的大小等
om info: modeldef_size[xxx B], weight_data_size[xxx B], tbe_kernels_size[xxx B],
```

```
cust_aicpu_kernel_store_size[xxx B], task_info_size[xxx B].  
===== Display Model Info end =====
```

## 支持的型号

Atlas 200/300/500 推理产品

Atlas 推理系列产品

## 依赖约束

无。

## 7.2.2 输入选项

### 7.2.2.1 --model

#### 功能说明

原始网络模型文件路径与文件名。

#### 关联参数

当原始模型为Caffe框架时，需要和[7.2.2.2 --weight](#)参数配合使用。

#### 参数取值

**参数值：**模型文件路径与文件名。

**参数值格式：**路径和文件名：支持大小写字母（a-z，A-Z）、数字（0-9）、下划线（\_）、短横线（-）、句点（.）、中文汉字。

#### 推荐配置及收益

无。

#### 示例

```
--model=$HOME/module/resnet50_tensorflow*.pb
```

## 支持的型号

Atlas 200/300/500 推理产品

Atlas 推理系列产品

## 依赖约束

无。

## 7.2.2.2 --weight

### 功能说明

原始网络模型权重文件路径与文件名，当原始网络模型是Caffe时需要指定。

### 关联参数

当原始模型为Caffe框架时，需要和[7.2.2.1 --model](#)参数配合使用。

### 参数取值

**参数值：**权重文件路径与文件名。

**参数值格式：**路径和文件名：支持大小写字母（a-z，A-Z）、数字（0-9）、下划线（\_）、短横线（-）、句点（.）、中文汉字。

### 推荐配置及收益

无。

### 示例

```
--model=$HOME/module/resnet50.prototxt --weight=$HOME/module/resnet50.caffemodel
```

### 支持的型号

Atlas 200/300/500 推理产品

Atlas 推理系列产品

### 依赖约束

无。

## 7.2.2.3 --om

### 功能说明

离线模型（.om）、原始模型文件（例如Caffe框架的.prototxt，TensorFlow框架的.pb等）、GE dump图结构文件（.txt）的路径和文件名。

### 关联参数

- 若[7.2.1.2 --mode](#)取值为1：
  - 离线模型转换为json  
该参数需要与[7.2.1.2 --mode](#)=1、[7.2.3.4 --json](#)参数配合使用。
  - 原始模型文件转换为json  
该参数需要与[7.2.1.2 --mode](#)=1、[7.2.3.4 --json](#)参数、[7.2.2.4 --framework](#)配合使用。
- 若[7.2.1.2 --mode](#)取值为5：

GE dump图结构文件转json，该参数需要与7.2.1.2 --mode=5、7.2.3.4 --json参数配合使用。

- 若7.2.1.2 --mode取值为6：  
则只需要与7.2.1.2 --mode参数配合使用。

## 参数取值

**参数值：**离线模型（.om）、原始模型文件（例如Caffe框架的.prototxt，TensorFlow框架的.pb）或GE dump图结构文件（.txt）的路径。

**参数值格式：**路径和文件名：支持大小写字母（a-z，A-Z）、数字（0-9）、下划线（\_）、短横线（-）、句点（.）、中文汉字。

## 推荐配置及收益

无。

## 示例

- 若7.2.1.2 --mode取值为1

- 离线模型转换为json：

```
--mode=1 --om=$HOME/module/out/tf_resnet50.om --json=$HOME/module/out/tf_resnet50.json
```

在转换后的json文件中，可以查看原始模型转换为该离线模型时，使用的基础版本号，以及当时转换使用的atc命令，示例如下：

```
{
  "key": "opp_version",
  "value": {
    "s": "<version>"
  }
},
...
{
  "key": "atc_version",
  "value": {
    "s": "<version>"
  }
},
...
{
  "key": "atc_cmdline",
  "value": {
    "s": "xxx/atc.bin --model ./resnet50_tensorflow*.pb --framework 3 --output ./out/tf_resnet50 --soc_version <soc_version>"
  }
},
...
{
  "key": "soc_version",
  "value": {
    "s": "<soc_version>"
  }
},
...
```

- 原始模型文件转换为json

```
--mode=1 --om=$HOME/module/resnet50_tensorflow*.pb --json=$HOME/module/out/tf_resnet50.json --framework=3
```

- 若7.2.1.2 --mode取值为5

GE dump图结构文件转json:  
--mode=5 --om=\$HOME/module/ge\_proto\_00000000\_PreRunBegin.txt --json=\$HOME/module/out/ge\_proto.json

- 若7.2.1.2 --mode取值为6  
atc --mode=6 --om=\$HOME/module/out/tf\_resnet50.om

支持的型号

Atlas 200/300/500 推理产品  
Atlas 推理系列产品

依赖约束

无。

7.2.2.4 --framework

功能说明

原始网络模型框架类型。

关联参数

无。

参数取值

- 参数值：
- 0：Caffe
  - 1：MindSpore
  - 3：TensorFlow
  - 5：ONNX
- 参数值约束：
- 当7.2.1.2 --mode为1时，该参数可选，可以指定Caffe、TensorFlow、ONNX原始模型转成json，不指定时默认为离线模型转json，如果指定时需要保证7.2.2.3 --om模型和7.2.2.4 --framework类型对应一致，例如：  
--mode=1 --framework=0 --om=\$HOME/module/resnet50.prototxt  
--mode=1 --framework=3 --om=\$HOME/module/resnet50\_tensorflow\*.pb  
--mode=1 --framework=5 --om=\$HOME/module/resnet50.onnx
  - 当7.2.1.2 --mode为0或3时，该参数必选，可以指定Caffe、TensorFlow、MindSpore或ONNX。
  - 当取值为0时，即为Caffe框架网络模型，模型包括后缀为prototxt的模型文件和后缀为caffemodel的权重文件，并且此两个文件的op name和op type必须保持名称一致（包括大小写）。Atlas A2训练系列产品不支持该框架。
  - 当取值为3时，即为TensorFlow框架网络模型，只支持FrozenGraphDef格式，即尾缀为pb的模型文件，pb文件采用protobuf格式存储，网络模型和权值数据都储存在同一个文件中。

- 当取值为5时，即为ONNX格式网络模型，仅支持ai.onnx算子域中opset v9~v18版本的算子；PyTorch框架的pth模型需要转化为ONNX格式的模型，才能进行模型转换。
- 当取值为1时，即为MindSpore框架网络模型时，请务必查看如下限制：
  - 模型转换时，仅支持后缀为\*.air的模型文件；
  - **7.2.1.2 --mode**只支持配置为0；
  - **7.2.2.5 --input\_format**只支持配置为NCHW，配置其它值无效，但模型转换成功；
  - 在MindSpore框架下，使用**7.3.1.1 --out\_nodes**、**7.3.1.7 --is\_output\_adjust\_hw\_layout**、**7.3.1.2 --input\_fp16\_nodes**、**7.3.1.6 --is\_input\_adjust\_hw\_layout**、**7.3.1.5 --op\_name\_map**参数不生效，但模型转换成功；
  - 当模型大小超过2G时，在MindSpore框架中保存模型时会同时生成\*.air文件、weight文件夹及其中的权重文件，在模型转换时，需要将weight文件夹与\*.air文件存放在同级目录下，否则模型转换报错。

## 推荐配置及收益

无。

## 示例

- Caffe框架：  
`--mode=0 --framework=0 --model=$HOME/module/resnet50.prototxt --weight=$HOME/module/resnet50.caffemodel --output=$HOME/module/out/caffe_resnet50 --soc_version=<soc_version>`
- MindSpore框架：  
`--mode=0 --framework=1 --model=$HOME/module/ResNet50.air --output=$HOME/module/out/ResNet50_mindspore --soc_version=<soc_version>`
- TensorFlow框架：  
`--mode=0 --framework=3 --model=$HOME/module/resnet50_tensorflow*.pb --output=$HOME/module/out/tf_resnet50 --soc_version=<soc_version>`
- ONNX网络模型：  
`--mode=0 --framework=5 --model=$HOME/module/resnet50.onnx --output=$HOME/module/out/onnx_resnet50 --soc_version=<soc_version>`

## 支持的型号

Atlas 200/300/500 推理产品

Atlas 推理系列产品

## 依赖约束

无。

## 7.2.2.5 --input\_format

### 功能说明

指定模型输入数据的格式。

## 关联参数

无。

## 参数取值

参数值：

- 当原始框架为Caffe时，支持NCHW、ND（表示支持任意维度格式， $N \leq 4$ ）两种格式，默认为NCHW。
- 当原始框架为ONNX时，支持NCHW、NCDHW、ND（表示支持任意维度格式， $N \leq 4$ ）三种格式，默认为NCHW。
- 当原始框架是TensorFlow时，支持NCHW、NHWC、ND、NCDHW、NDHWC五种输入格式，默认为NHWC。
  - 如果TensorFlow模型是通过ONNX模型转换工具输出的，则该参数必填，且值为NCHW。
  - 如果原始模型中含有带data\_format入参的算子，则该参数必填，推荐取值为ND，模型转换过程中会根据data\_format属性的算子，推导出具体的format。若用户无法确定输入数据格式，则推荐指定为ND。
- 当原始框架为MindSpore时，只支持配置为NCHW，设置为其它值无效，但模型转换成功。

**参数默认值：**Caffe、MindSpore、ONNX默认为NCHW；TensorFlow默认为NHWC。

**参数值约束：**

- 如果模型转换时开启AIPP，在进行推理业务时，输入图片数据要求为NHWC排布，该场景下最终与AIPP连接的输入节点的格式被强制改成NHWC，可能与atc模型转换命令中**7.2.2.5 --input\_format**参数指定的格式不一致。
- 如果同时配置了**7.3.1.3 --insert\_op\_conf**参数，则**7.2.2.5 --input\_format**参数只能配置为NCHW、NHWC。
- 该参数不支持模型多输入且不同输入存在不同数据格式的设置场景，即该参数只支持设置为单个取值，例如TensorFlow框架的模型，只支持设置为NCHW、NHWC、ND、NCDHW、NDHWC五种输入格式中的一种。

## 推荐配置及收益

无。

## 示例

```
--input_format=NCHW
```

## 支持的型号

Atlas 200/300/500 推理产品  
Atlas 推理系列产品

## 依赖约束

无。



## 7.2.2.6 --input\_shape

### 功能说明

指定模型输入数据的shape。

### 关联参数

设置shape分档场景，需要配合使用[7.2.2.8 --dynamic\\_batch\\_size](#)（设置BatchSize档位）或[7.2.2.9 --dynamic\\_image\\_size](#)（设置分辨率档位）或[7.2.2.10 --dynamic\\_dims](#)（设置指定维度档位）参数。

### 参数取值

#### 参数值：

- 静态shape，--input\_shape参数可选配置
  - 若模型为单个输入，则shape信息为"input\_name:n,c,h,w"；指定的节点必须放在双引号中。
  - 若模型有多个输入，则shape信息--input\_shape="input\_name1:n1,c1,h1,w1;input\_name2:n2,c2,h2,w2"；不同输入之间使用英文分号分隔，input\_name必须是转换前的网络模型中的节点名称。
- 动态shape，--input\_shape参数必须配置

若原始模型中输入数据的某个或某些维度值不固定，当前支持通过设置shape分档或设置shape范围两种方式转换模型：

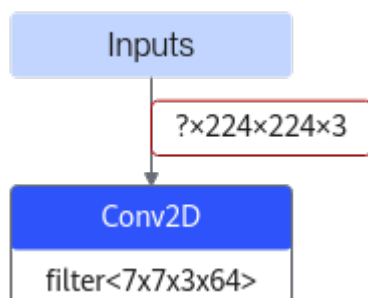
  - 设置shape分档，包括设置BatchSize档位、设置分辨率档位、设置指定维度档位（最多4维）。

设置--input\_shape参数时，将对应维度值设置为-1，同时配合使用[7.2.2.8 --dynamic\\_batch\\_size](#)（设置BatchSize档位）或[7.2.2.9 --dynamic\\_image\\_size](#)（设置分辨率档位）或[7.2.2.10 --dynamic\\_dims](#)（设置指定维度档位）参数。
  - 设置shape范围。Atlas 200/300/500 推理产品、Atlas 200I/500 A2推理产品不支持设置shape范围。

设置--input\_shape参数时，可将对应维度的值设置为范围，例如1~10。  
如果用户不想指定维度的范围或具体取值，则可以将其设置为-1，模型转换时该维度被解析为>=0的任意取值。

#### 参数值约束：

- 如果模型输入数据为标量（例如BOOL类型），该场景下无需配置对应节点的[7.2.2.6 --input\\_shape](#)参数。
- 若原始模型中输入数据的某个维度值不固定（例如input\_name1:?,h,w,c），通过Netron等可视化软件打开模型之后，输入信息样例如下：



该场景下--input\_shape参数必填，并可以进行如下操作：

- 固定shape，将维度值设置为固定取值，例如，input\_name1:1,h,w,c，用于将输入数据某个维度不固定的原始模型转换为固定维度的离线模型。
- 设置shape分档，例如设置为“-1”，与7.2.2.8 --dynamic\_batch\_size参数配合使用。
- 设置shape范围时，若设置为-1，表示此维度可以使用 $\geq 0$ 的任意取值，该场景下取值上限为int64数据类型表达范围，但受限于host和device侧物理内存的大小，用户可以通过增大内存来支持。
- 若使用该参数时，同时通过7.3.1.3 --insert\_op\_conf设置了AIPP功能，则AIPP输出图片的宽和高要在本参数所设置的范围内。

## 推荐配置及收益

无。

## 示例

- 静态shape，--input\_shape可选配置  
例如某网络的输入shape信息，输入1：input\_0\_0 [16,32,208,208]，输入2：input\_1\_0 [16,64,208,208]，则--input\_shape的配置信息为：  
`--input_shape="input_0_0:16,32,208,208;input_1_0:16,64,208,208"`
- 动态shape，--input\_shape必须配置
  - 设置BatchSize档位的示例，请参见7.2.2.8 --dynamic\_batch\_size。
  - 设置分辨率档位的示例，请参见7.2.2.9 --dynamic\_image\_size。
  - 设置指定维度档位的示例，请参见7.2.2.10 --dynamic\_dims。
  - 设置shape范围的示例：  
`--input_shape="input_0_0:1~10,32,208,208;input_1_0:16,64,100~208,100~208"`

## 支持的型号

Atlas 200/300/500 推理产品

Atlas 推理系列产品

## 依赖约束

- 使用约束：  
如果用户通过7.2.2.6 --input\_shape设置了动态shape范围参数，同时又通过7.3.1.3 --insert\_op\_conf参数配置了AIPP功能，则AIPP输出的宽和高要在7.2.2.6 --input\_shape所设置的范围内。

- **接口约束:**

如果模型转换时通过该参数设置了shape的范围，使用应用工程进行模型推理时，需在接口之前，调用接口，用于设置真实的输入Tensor描述信息（输入shape范围）；模型执行之后，调用接口获取模型动态输出的Tensor描述信息；再进一步调用下的操作接口获取输出Tensor数据占用的内存大小、Tensor的Format信息、Tensor的维度信息等。

关于、等接口的具体使用方法，请参见《[CANN AscendCL应用软件开发指南（C&C++）（开放态）](#)》手册“AscendCL API参考”。

### 7.2.2.7 --input\_shape\_range

#### 功能说明

指定模型输入数据的shape范围。

#### 须知

该参数已废弃，请勿使用。若涉及指定模型输入数据的shape范围，请使用[7.2.2.6 --input\\_shape](#)。

#### 关联参数

该参数不能与[7.2.2.8 --dynamic\\_batch\\_size](#)、[7.2.2.9 --dynamic\\_image\\_size](#)、[7.2.2.10 --dynamic\\_dims](#)同时使用。

#### 参数取值

**参数值：**模型输入数据的shape范围信息，例如："input\_name1:[n1~n2,c1,h1,w1];input\_name2:[n2,c2,h2,w2]"。指定的节点必须放在双引号中，节点中间使用英文分号分隔。input\_name必须是转换前的网络模型中的节点名称。

**参数值约束：**

- shape范围信息必须放在英文[]中。
- 该参数不限定维度，维度中的任一值都可以由用户指定取值范围。
- 如果用户不想指定维度的取值，则可以将其设置为-1，表示此维度可以使用 $\geq 1$ 的任意取值。

#### 推荐配置及收益

无。

#### 示例

```
--input_shape_range="input1:[8~20,3,5,-1];input2:[5,3~9,10,-1]"
```

#### 支持的型号

Atlas 推理系列产品

## 依赖约束

- **使用约束:**
  - 该参数只适用于TensorFlow和ONNX网络模型。
  - 若使用该参数时,同时通过[7.3.1.3 --insert\\_op\\_conf](#)设置了AIPP功能,则AIPP输出图片的宽和高要在[7.2.2.7 --input\\_shape\\_range](#)所设置的范围内。
- **接口约束:**

如果模型转换时通过该参数设置了shape的范围,则使用应用工程进行模型推理时,需要在[aclmdlExecute](#)接口之前,调用[aclmdlSetDatasetTensorDesc](#)接口,用于设置真实的输入Tensor描述信息(输入shape范围);模型执行之后,调用[aclmdlGetDatasetTensorDesc](#)接口获取模型动态输出的Tensor描述信息;再进一步调用[aclTensorDesc](#)下的操作接口获取输出Tensor数据占用的内存大小、Tensor的Format信息、Tensor的维度信息等)。

关于[aclmdlSetDatasetTensorDesc](#)、[aclmdlGetDatasetTensorDesc](#)等接口的具体使用方法,请参见《[CANN AscendCL应用软件开发指南 \(C&C++\) \(开放态\)](#)》手册“AscendCL API参考”。

### 7.2.2.8 --dynamic\_batch\_size

## 功能说明

设置动态BatchSize参数,适用于执行推理时,每次处理图片或者句子数量不固定的场景。

在某些推理场景,如检测出目标后再执行目标识别网络,由于目标个数不固定导致目标识别网络输入BatchSize不固定。如果每次推理都按照最大的BatchSize或最大分辨率进行计算,会造成计算资源浪费。因此,推理需要支持动态BatchSize和动态分辨率的场景,使用ATC工具时,通过该参数设置支持的BatchSize,通过[7.2.2.9 --dynamic\\_image\\_size](#)参数设置支持的分辨率档位。

模型转换完成后,在生成的om模型中,会新增一个输入,在模型推理时通过该新增的输入提供具体的BatchSize值。例如,a输入的BatchSize是动态的,在om模型中,会有与a对应的b输入来描述a的具体BatchSize。

## 关联参数

- 该参数需要与[7.2.2.6 --input\\_shape](#)配合使用,不能与[7.2.2.9 --dynamic\\_image\\_size](#)、[7.2.2.10 --dynamic\\_dims](#)、[7.2.2.7 --input\\_shape\\_range](#)同时使用。且只支持N在shape首位的场景,即shape的第一位设置为"-1"。如果N在非首位场景下,请使用[7.2.2.10 --dynamic\\_dims](#)参数进行设置。
- 该参数不支持与[7.2.2.4 --framework=1](#)同时使用,\*air的模型不支持动态分档特性。

## 参数取值

**参数值:** 档位数,例如"1,2,4,8"。

**参数值格式:** 指定的参数必须放在双引号中,档位之间使用英文逗号分隔。

**参数值约束:** 档位数取值范围为(1,100],即必须设置至少2个档位,最多支持100档配置;每个档位数建议限制为:[1~2048]。

## 推荐配置及收益

- 如果用户设置的档位数值过大或档位过多，可能会导致模型转换失败，此时建议用户减少档位或调低档位数值。
- CV（计算机视觉）类的网络，[7.2.2.8 --dynamic\\_batch\\_size](#)建议取值为8、16档位，该场景下的网络性能比单个BatchSize更优（8、16档位只是建议取值，实际使用时还请以实际测试结果为准）。
- OCR/NLP（文字识别/自然语言处理）类网络，[7.2.2.8 --dynamic\\_batch\\_size](#)档位取值建议为16的整数倍（该档位值只是建议取值，实际使用时还请以实际测试结果为准）。
- 如果用户设置的档位数值过大或档位过多，在运行环境执行推理时，建议执行 **swapoff -a**命令关闭swap交换区间作为内存的功能，防止出现由于内存不足，将swap交换空间作为内存继续调用，导致运行环境异常缓慢的情况。

## 示例

```
--input_shape="data:-1,3,416,416;img_info:-1,4" --dynamic_batch_size="1,2,4,8"
```

其中，“--input\_shape”中的“-1”表示设置动态BatchSize。则ATC在模型编译时，支持的输入组合档数分别为：

第0档：data(1,3,416,416)+img\_info(1,4)

第1档：data(2,3,416,416)+img\_info(2,4)

第2档：data(4,3,416,416)+img\_info(4,4)

第3档：data(8,3,416,416)+img\_info(8,4)

## 支持的型号

Atlas 200/300/500 推理产品

Atlas 推理系列产品

## 依赖约束

- 使用约束：
  - 不支持含有过程动态shape算子（网络中间层shape不固定）的网络。
  - 若用户执行推理业务时，每次处理的图片数量不固定，则可以通过配置该参数来动态分配每次处理的图片数量。例如用户执行推理业务时需要每次处理2张、4张、8张图片，则可以配置为2,4,8，申请了档位后，模型推理时会根据实际档位申请内存。
  - 如果用户设置了动态BatchSize，同时又通过[7.3.1.3 --insert\\_op\\_conf](#)参数设置了动态AIPP功能：  
实际推理时，调用《[CANN AscendCL应用软件开发指南（C&C++）（开放态）](#)》手册中的“AscendCL API参考>模型加载与执行>[aclmdlSetInputAIPP](#)”接口设置动态AIPP相关参数值时，需确保batchSize要设置为最大Batch数。
  - 某些场景下，通过该参数设置动态BatchSize特性后，生成的离线模型网络结构会与固定BatchSize场景下的不同，推理性能可能存在差异。
- 接口约束：

如果模型转换时通过该参数设置了动态BatchSize，则使用应用工程进行推理时，需在接口之前，增加接口，用于设置真实的BatchSize档位。

关于接口的具体使用方法，请参见《[CANN AscendCL应用软件开发指南（C&C++）（开放态）](#)》手册“AscendCL API参考>模型加载与执行”章节。

### 7.2.2.9 --dynamic\_image\_size

#### 功能说明

设置输入图片的动态分辨率参数。适用于执行推理时，每次处理图片宽和高不固定的场景。

#### 关联参数

- 该参数需要与[7.2.2.6 --input\\_shape](#)配合使用，不能与[7.2.2.8 --dynamic\\_batch\\_size](#)、[7.2.2.10 --dynamic\\_dims](#)、[7.2.2.7 --input\\_shape\\_range](#)同时使用。
- 使用该参数设置动态分辨率时，[7.2.2.5 --input\\_format](#)参数只支持配置为NCHW、NHWC；其他format场景下，设置分辨率请使用[7.2.2.10 --dynamic\\_dims](#)参数。
- 该参数不支持与[7.2.2.4 --framework=1](#)同时使用，\*.air的模型不支持动态分档特性。

#### 参数取值

**参数值：**动态分辨率参数，例如  
"imagesize1\_height,imagesize1\_width;imagesize2\_height,imagesize2\_width"。

**参数值格式：**指定的参数必须放在双引号中，档位之间英文分号分隔，每档内参数使用英文逗号分隔。

**参数值约束：**档位数取值范围为(1,100]，即必须设置至少2个档位，最多支持100档配置。

#### 推荐配置及收益

- 如果用户设置的分辨率数值过大或档位过多，可能会导致模型转换失败，此时建议用户减少档位或调低档位数。
- 如果用户设置的分辨率数值过大或档位过多，在运行环境执行推理时，建议执行[swapoff -a](#)命令关闭swap交换区间作为内存的功能，防止出现由于内存不足，将swap交换空间作为内存继续调用，导致运行环境异常缓慢的情况。

#### 示例

```
--input_shape="data:8,3,-1,-1;img_info:8,4,-1,-1" --dynamic_image_size="416,416;832,832"
```

其中，“--input\_shape”中的“-1”表示设置动态分辨率。则ATC在编译模型时，支持的输入组合档数分别为：

第0档：data(8,3,416,416)+img\_info(8,4,416,416)

第1档：data(8,3,832,832)+img\_info(8,4,832,832)



## 支持的型号

Atlas 200/300/500 推理产品

Atlas 推理系列产品

## 依赖约束

- 使用约束:

- 不支持含有过程动态shape算子（网络中间层shape不固定）的网络。
- 如果用户设置了动态分辨率，则请确保不同档位的分辨率能在原生框架下正常推理。
- 如果用户设置了动态分辨率，实际推理时，使用的数据集图片大小需要与具体使用的分辨率相匹配。
- 如果用户设置了动态分辨率，既输入图片的宽和高不确定，同时又通过 [7.3.1.3 --insert\\_op\\_conf](#) 参数设置了静态AIPP功能：该场景下，AIPP配置文件中不能开启Crop和Padding功能，并且需要将配置文件中的 `src_image_size_w` 和 `src_image_size_h` 取值设置为0。
- 如果用户设置了动态分辨率，同时又通过 [7.3.1.3 --insert\\_op\\_conf](#) 参数设置了动态AIPP功能：

实际推理时，调用《[CANN AscendCL应用软件开发指南（C&C++）（开放态）](#)》手册中的“AscendCL API参考>模型加载与执行>`aclmdlSetInputAIPP`”接口，设置动态AIPP相关参数值时，不能开启Crop和Padding功能。该场景下，还需要确保通过CANN AscendCL应用软件开发指南（C&C++）（开放态）》手册“AscendCL API参考>模型加载与执行>`aclmdlSetDynamicHWSIZE`”接口设置的宽、高相等，都必须设置成动态分辨率最大档位的宽、高。

- 某些场景下，通过该参数设置动态分辨率特性后，生成的离线模型网络结构会与固定分辨率场景下的不同，推理性能可能存在差异。

- 接口约束:

如果模型转换时通过该参数设置了动态分辨率，则使用应用工程进行模型推理时，需要在

关于CANN AscendCL应用软件开发指南（C&C++）（开放态）》手册“AscendCL API参考>模型加载与执行”章节。

### 7.2.2.10 --dynamic\_dims

## 功能说明

设置ND格式下动态维度的档位。适用于执行推理时，每次处理任意维度的场景。

为支持Transformer等网络在输入格式的维度不确定的场景，通过该参数实现ND格式下任意维度的档位设置。ND表示支持任意格式，当前 $N \leq 4$ 。

## 关联参数

- 该参数需要与[7.2.2.6 --input\\_shape](#)、[7.2.2.5 --input\\_format](#)配合使用，不能与[7.2.2.8 --dynamic\\_batch\\_size](#)、[7.2.2.9 --dynamic\\_image\\_size](#)、[7.2.2.7 --input\\_shape\\_range](#)、[7.3.1.3 --insert\\_op\\_conf](#)同时使用。

- 该参数不支持与7.2.2.4 --framework=1同时使用，\*.air的模型不支持动态分档特性。

参数取值

**参数值：**通过"dim1,dim2,dim3;dim4,dim5,dim6;dim7,dim8,dim9"的形式设置。

**参数值格式：**所有档位必须放在双引号中，档位之间使用英文分号分隔，每档内的dim值与7.2.2.6 --input\_shape参数中的-1标识的参数依次对应，7.2.2.6 --input\_shape参数中有几个-1，则每档必须设置几个维度。

**参数值约束：**

- 档位数取值范围为(1,100]，即必须设置至少2个档位，最多支持100档配置，建议配置为3~4档；每档最多支持任意指定4个维度。

推荐配置及收益

无。

示例

- 若网络模型只有一个输入：  
每档中的dim值与7.2.2.6 --input\_shape参数中的-1标识的参数依次对应，7.2.2.6 --input\_shape参数中有几个-1，则每档必须设置几个维度。例如：  
ATC参数取值为：  

```
--input_shape="data:1,-1" --dynamic_dims="4;8;16;64" --input_format=ND
```

  
则ATC在编译模型时，支持的data算子的shape为1,4; 1,8; 1,16; 1,64。  
ATC参数取值为：  

```
--input_shape="data:1,-1,-1" --dynamic_dims="1,2;3,4;5,6;7,8" --input_format=ND
```

  
则ATC在编译模型时，支持的data算子的shape为1,1,2; 1,3,4; 1,5,6; 1,7,8。
- 若网络模型有多个输入：  
每档中的dim值与网络模型输入参数中的-1标识的参数依次对应，网络模型输入参数中有几个-1，则每档必须设置几个维度。例如网络模型有三个输入，分别为data(1,1,40,T)，label(1,T)，mask(T,T)，其中T为动态可变。则配置示例为：  

```
--input_shape="data:1,1,40,-1;label:1,-1;mask:-1,-1" --dynamic_dims="20,20,1,1;40,40,2,2;80,60,4,4" --input_format=ND
```

  
在ATC编译模型时，支持的输入dims组合档数分别为：  
第0档：data(1,1,40,20)+label(1,20)+mask(1,1)  
第1档：data(1,1,40,40)+label(1,40)+mask(2,2)  
第2档：data(1,1,40,80)+label(1,60)+mask(4,4)

支持的型号

Atlas 200/300/500 推理产品  
Atlas 推理系列产品

依赖约束

- 使用约束



不支持含有过程动态shape算子（网络中间层shape不固定）的网络。

- 接口约束

如果模型转换时通过该参数设置了动态维度，则使用应用工程进行模型推理时，需要在[aclmdlExecute](#)接口之前，增加[aclmdlSetInputDynamicDims](#)接口，用于设置真实的维度。

关于[aclmdlSetInputDynamicDims](#)接口的具体使用方法，请参见《[CANN AscendCL应用软件开发指南（C&C++）（开放态）](#)》手册“AscendCL API参考>模型加载与执行”章节。

### 7.2.2.11 --singleop

#### 功能说明

单算子描述文件，将单个算子描述文件（json格式）转换成适配昇腾AI处理器的离线模型，以便进行后续的单算子功能验证。

#### 关联参数

使用该参数时，只有如下参数可以配合使用，其中[7.2.3.1 --output](#)、[7.2.4.1 --soc\\_version](#)为必填。

- Atlas 200/300/500 推理产品可配合使用的参数：
  - [7.2.3.1 --output](#)
  - [7.2.4.1 --soc\\_version](#)
  - [7.3.3.1 --precision\\_mode](#)
  - [7.3.3.5 --op\\_select\\_implmode](#)
  - [7.3.3.6 --optypelist\\_for\\_implmode](#)
  - [7.3.2.4 --enable\\_small\\_channel](#)
  - [7.3.4.2 --log](#)
  - [7.3.4.3 --debug\\_dir](#)
  - [7.3.4.4 --op\\_compiler\\_cache\\_mode](#)
  - [7.3.4.5 --op\\_compiler\\_cache\\_dir](#)
  - [7.3.4.10 --atomic\\_clean\\_policy](#)
  - [7.3.3.8 --customize\\_dtypes](#)
- Atlas 推理系列产品可配合使用的参数：
  - [7.2.3.1 --output](#)
  - [7.2.4.1 --soc\\_version](#)
  - [7.2.4.2 --core\\_type](#)
  - [7.2.4.3 --aicore\\_num](#)
  - [7.3.3.1 --precision\\_mode](#)
  - [7.3.3.5 --op\\_select\\_implmode](#)
  - [7.3.3.6 --optypelist\\_for\\_implmode](#)
  - [7.3.4.2 --log](#)
  - [7.3.4.3 --debug\\_dir](#)

- [7.3.4.4 --op\\_compiler\\_cache\\_mode](#)
- [7.3.4.5 --op\\_compiler\\_cache\\_dir](#)
- [7.3.4.10 --atomic\\_clean\\_policy](#)
- [7.3.3.8 --customize\\_dtypes](#)

## 参数取值

**参数值：**单算子描述文件（json格式）格式以及参数配置请参见[6.2 单算子模型转换](#)。

**参数值约束：**该参数指定的单算子都是基于Ascend IR定义的，关于单算子的详细定义请参见《[算子清单（开放态）](#)》手册。

## 推荐配置及收益

无。

## 示例

下面以Add单算子为例进行说明，该单算子对应的描述文件为`add.json`，将该文件上传到ATC工具所在服务器任意目录，例如上传到`$HOME/singleop`，使用示例如下：

```
--singleop=$HOME/singleop/add.json --output=$HOME/singleop/out/op_model --  
soc_version=<soc_version>
```

## 支持的型号

Atlas 200/300/500 推理产品

Atlas 推理系列产品

## 依赖约束

- 使用约束

单算子json文件转换成离线模型场景，如果希望模型转换时只使用TBE算子（不查找AI CPU算子，找不到TBE算子则报错），还需设置如下环境变量：

```
export ASCEND_ENGINE_PATH=${INSTALL_DIR}/lib64/plugin/opskernel/libfe.so:${INSTALL_DIR}/lib64/  
plugin/opskernel/libge_local_engine.so
```

执行上述命令后，如果用户想要执行其他操作，需要删除上述环境变量：执行 **unset ASCEND\_ENGINE\_PATH** 命令，使其失效。

- 接口约束

单算子描述文件转换后的om模型文件，使用应用工程进行模型推理时，需调用AscendCL接口加载算子模型（例如**aclopSetModelDir**接口），最后调用AscendCL接口执行算子（例如**aclopExecuteV2**接口）。

接口详细说明请参见《[CANN AscendCL应用软件开发指南（C&C++）（开放态）](#)》手册“AscendCL API参考>模型加载与执行”章节。

### 7.2.2.12 --distributed\_cluster\_build

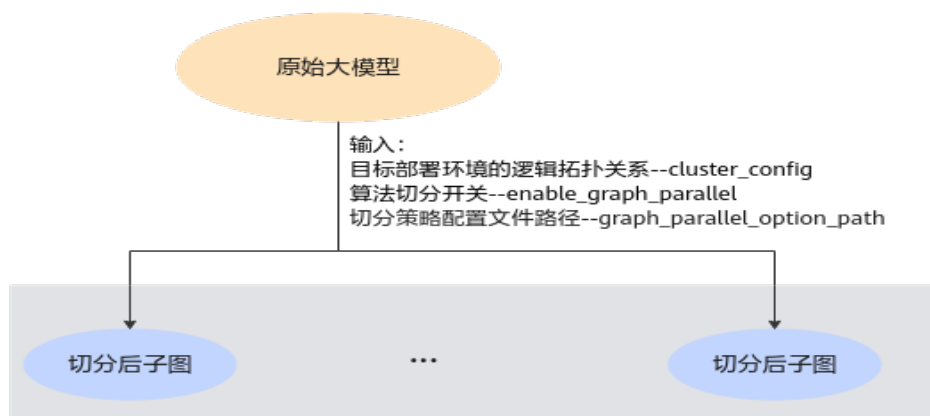
## 功能说明

大模型分布式编译切分开关，使能该参数后，生成的离线模型将用于分布式部署。

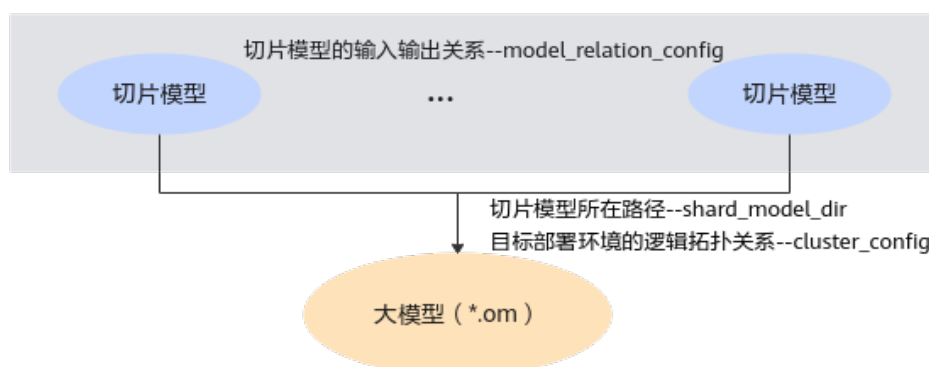
## 关联参数

当该参数设置为1时，如下参数可以同步设置：

- 输入是完整大模型：
  - 若输入的原始大模型开启了算法切分，则如下参数必须配置：  
**7.2.2.13 --cluster\_config**参数必须配置，并且通过**7.2.2.14 --enable\_graph\_parallel**参数开启算法切分，通过**7.2.2.15 --graph\_parallel\_option\_path**参数配置切分策略配置文件路径。  
算法切分场景，模型编译阶段会自动插入通信算子。



- 输入是切片模型，模型中包括通信算子，将切片模型编译为om离线模型：  
**7.2.2.13 --cluster\_config**参数必须配置，并且通过**7.2.2.16 --shard\_model\_dir**参数设置多个切片模型所在路径，通过**7.2.2.17 --model\_relation\_config**参数设置多个切片模型之间的输入输出关系。



## 参数取值

参数值：

1：打开分布式编译切分开关。

参数默认值：无。

## 推荐配置及收益

无。

## 示例

- 分布式部署模型，输入是单个原始大模型（无算法切分，未指定部署设备，负载均衡将模型部署到所有设备的场景）  

```
atc --model=xxx.air --framework=1 --soc_version=<soc_version> --output=xxx --cluster_config=./numa_config.json --distributed_cluster_build=1
```
- 分布式部署模型，输入是单个原始大模型，开启算法切分  

```
atc --model=./matmul2.pb --distributed_cluster_build=1 --cluster_config=./numa_config_2p.json --enable_graph_parallel="1" --graph_parallel_option_path=./parallel_option.json --soc_version=<soc_version> --output=test_parallel --framework=3 --log=debug
```

大模型算法切分后会将子模型要部署的逻辑 device id 存储在子模型的属性中，重新加载部署时，部署模块根据该属性进行分布式部署。
- 分布式部署模型，输入是切片模型，模型中包含通信算子（shared\_dir 目录下包含多个切片模型）  

```
atc --distributed_cluster_build=1 --cluster_config=./numa_config_4p.json --shard_model_dir=./1_air --model_relation_config=./model_relation_config.json --output=1_increase_4p --framework=1 --log=debug --soc_version=<soc_version>
```

## 支持的型号

Atlas 推理系列产品

## 使用约束

分布式编译切分场景编译出来的离线模型，进行模型推理时，需要使用 Ascend IR 构图中的 **LoadGraph** 接口加载模型，最后使用 **RunGraph** 运行加载了该模型的 Graph，得到图的执行结果。详情请参见《[Ascend Graph 开发指南](#)》手册“编译 Graph 为离线模型并运行 Graph（大模型分布式编译切分）”章节。

### 7.2.2.13 --cluster\_config

#### 功能说明

指定目标部署环境逻辑拓扑关系的配置文件，用于生成 hcom group 和 rankid 信息。

只要原始大模型含有通信算子，无论是否分布式部署都需要配置该参数，否则通信算子执行时可能会报错。

#### 关联参数

若模型中包含通信算子，或开启了算法切分（[7.2.2.14 --enable\\_graph\\_parallel=1](#)），该参数必填。

#### 参数取值

**参数值：**逻辑拓扑文件的路径和文件名。

**参数值格式：**路径和文件名：支持大小写字母（a-z, A-Z）、数字（0-9）、下划线（\_）、短横线（-）、句点（.）、中文汉字。

**参数值约束：**配置文件中的内容必须为 json 格式。

#### 推荐配置及收益

无。

示例

将配置文件（文件名为举例为：*numa\_config.json*）上传到ATC工具所在服务器，例如上传到*\$HOME/conf*，使用示例如下：

```
atc --model=xxx.air --framework=1 --soc_version=<soc_version> --output=$HOME/out --cluster_config=$HOME/conf/numa_config.json
```

逻辑拓扑文件示例如下：

- Atlas 推理系列产品场景：具体使用的device芯片数量为item\_list\*item\_def。

```
{
  "cluster": [{
    "cluster_nodes": [{           //必填，Array of Cluster_node类型，集群资源信息描述。
      "is_local": true,         //非必填，BOOL类型，多个Node组成集群时，此文件此节点是否是本机。
      "item_list": [{          //必填，Array of item_info类型，云资源管理编排的执行该JOB的加速卡。
        "item_id": 0           //必填，Integer类型，Node内加速卡id。
      }],
      "node_id": 0,             //必填，Integer类型，集群内节点编号，一般0作为主节点。
      "node_type": "ATLAS300"   //必填，String类型，节点类型，如ATLAS800。
    }],
    "item_def": [{              //Node内同种类型的加速卡的公共属性。
      "device_list": [{         //非必填，Array of device_info类型，整芯片内包含几个物理device。
        "device_id": 0          //必填，Integer类型，整芯片内物理device id。
      }],
      {
        "device_id": 1
      },
      "item_type": "<soc_version>" //必填，String类型，节点内加速卡类型。
    }],
  }
}
```

参数解释如下：

表 7-2 参数解释

参数			类型	是否必填	描述
cluster	-				集群配置。
	cluster_nodes	-	Array of Cluster_node	是	集群资源信息描述。
		node_id	Integer	是	集群内节点编号，一般0作为主节点。
		node_type	String	是	节点类型，如ATLAS800。
		ipaddr	String	是	节点控制面通信的IP，如训练服务器为HOST IP、SOC服务器为头节点IP。
		port	Integer	是	节点控制面通信的端口。
		is_local	BOOL	否	多个Node组成集群时，此文件此节点是否是本机。

参数				类型	是否必填	描述
		item_list	-	Array of item_info	是	云资源管理编排的执行该JOB的加速卡。
		-	item_id	Integer	是	Node内加速卡id。
item_def	-					Node内同种类型的加速卡的公共属性。
	device_list	-		Array of device_info	否	整芯片内包含几个物理device。Atlas 训练系列产品不需要填写该配置项。
		device_id	-	Integer	是	整芯片内物理device id。
	item_type	-	-	String	是	节点内加速卡类型。
node_def	-					集群内同种类型Node的公共属性。
	item	item_type	-	String	是	节点内加速卡类型。

支持的型号

Atlas 推理系列产品

依赖约束

无。

7.2.2.14 --enable\_graph\_parallel

功能说明

是否对原始大模型进行自动切分。

关联参数

- 7.2.2.12 --distributed\_cluster\_build参数开启大模型分布式编译后，才支持开启自动切分功能，原始大模型会按照7.2.2.15 --graph\_parallel\_option\_path文件中的要求进行自动切分。
- 算法切分场景7.2.2.13 --cluster\_config必须配置。

参数取值

参数值：

1: 开启自动切分。

参数默认值: 无

## 推荐配置及收益

无。

## 示例

```
atc --distributed_cluster_build=1 --cluster_config=./numa_config_2p.json --model=./matmul2.pb --  
enable_graph_parallel="1" --graph_parallel_option_path=./parallel_option.json --  
soc_version=<soc_version> --output=test_parallel --framework=3 --log=debug
```

## 支持的型号

Atlas 推理系列产品

## 依赖约束

无。

### 7.2.2.15 --graph\_parallel\_option\_path

## 功能说明

对原始大模型进行切分时，算法切分策略配置文件路径。

## 关联参数

- 7.2.2.12 --distributed\_cluster\_build 参数开启分布式编译，且 7.2.2.14 --enable\_graph\_parallel 参数开启切分功能后，才支持配置切分策略配置文件路径。
- 算法切分场景 7.2.2.13 --cluster\_config 必须配置。

## 参数取值

参数值: 切分策略配置文件路径和文件名。

参数值格式: 路径和文件名: 支持大小写字母 (a-z, A-Z)、数字 (0-9)、下划线 ( \_ )、短横线 ( - )、句点 ( . )、中文汉字。

## 推荐配置及收益

无。

## 示例

```
atc --distributed_cluster_build=1 --cluster_config=./numa_config_2p.json --model=./matmul2.pb --  
enable_graph_parallel="1" --graph_parallel_option_path=./parallel_option.json --  
soc_version=<soc_version> --output=test_parallel --framework=3 --log=debug
```

切分策略配置文件示例如下:

- 半自动切分

```
{
  "graph_parallel_option": {
    "auto": false,
    "opt_level": "O1"
    "tensor_parallel_option": {
      "tensor_parallel_size": 2
    },
    "tensor_sharding": {
      "optimizer_state_sharding": true,
      "gradient_sharding": true,
      "model_weight_sharding": true,
      "model_weight_prefetch": true,
      "model_weight_prefetch_buffer_size": 50
    }
  }
}
```

- 全自动切分

```
{
  "graph_parallel_option": {
    "auto": true
  }
}
```

参数解释如下：

- auto：配置为true表示全自动切分，配置为false表示半自动切分。
- opt\_level：指Tensor Parallel求解算法，支持配置为O2和O1，O2使用的是ILP算法，O1使用的是DP算法，若不配置，默认使用O2。
- tensor\_parallel\_option：配置该option表示使能TP切分。  
TP切分：Tensor Parallel也称为算子内并行（Intra-Op Parallel），将计算图中各个Operators的Tensor沿一个或多个轴（batch/Non-batch）进行切分，切分后的partition分发给各个Device分别计算。
- tensor\_parallel\_size：表示TP切分的份数，即需要配置的device芯片数量。该参数取值必须和[7.2.2.13 --cluster\\_config](#)拓扑文件中具体使用的device芯片数量相等。
- optimizer\_state\_sharding：是否开启优化器切片，true表示开启；false表示不开启。
- gradient\_sharding：是否开启梯度切片，true表示开启；false表示不开启。
- model\_weight\_sharding：是否开启权重切片，true表示开启；false表示不开启。
- model\_weight\_prefetch：是否开启权重预取，true表示开启；false表示不开启。
- model\_weight\_prefetch\_buffer\_size：配置权重预取的缓存大小。

## 支持的型号

Atlas 推理系列产品

## 依赖约束

无。

### 7.2.2.16 --shard\_model\_dir

## 功能说明

指定切片模型文件所在路径。



该参数适用于原始大模型已经面向分布式推理环境切片分好，且切片模型内含通信算子的场景。

## 关联参数

- 该参数只有[7.2.2.12 --distributed\\_cluster\\_build](#)开启大模型分布式编译后才生效，且必须与[7.2.2.17 --model\\_relation\\_config](#)参数同时使用，通过relation表达多个切片模型之间的数据关联和分布式通信组关系。
- 模型中包含通信算子场景，[7.2.2.13 --cluster\\_config](#)必填。

## 参数取值

**参数值：**切片模型所在路径和文件名。

**参数值格式：**路径和文件名：支持大小写字母（a-z，A-Z）、数字（0-9）、下划线（\_）、短横线（-）、句点（.）、中文汉字。

## 推荐配置及收益

无。

## 示例

```
atc --distributed_cluster_build=1 --cluster_config=./numa_config_4p.json --output=1_increase_4p --framework=1 --log=debug --shard_model_dir=./1_air --model_relation_config=./model_relation_config.json --soc_version=<soc_version>
```

## 支持的型号

Atlas 推理系列产品

## 依赖约束

无。

### 7.2.2.17 --model\_relation\_config

## 功能说明

表达多个切片模型间的数据关联和分布式通信组关系的配置文件。适用于原始大模型为切片模型，且切片模型内含通信算子的场景。

## 关联参数

- 该参数只有[7.2.2.12 --distributed\\_cluster\\_build](#)使能分布式编译后才生效，且必须与[7.2.2.16 --shard\\_model\\_dir](#)参数同时使用，用来指定切片模型所在路径。
- 模型中包含通信算子场景，[7.2.2.13 --cluster\\_config](#)必填。

## 参数取值

**参数值：**配置文件路径和文件名。

**参数值格式：**路径和文件名：支持大小写字母（a-z，A-Z）、数字（0-9）、下划线（\_）、短横线（-）、句点（.）、中文汉字。

**参数值约束：**配置文件中的内容必须为json格式。

## 推荐配置及收益

无。

## 示例

将配置文件上传到ATC工具所在服务器，例如上传到`$HOME/conf`，使用示例如下：

```
atc --distributed_cluster_build=1 --cluster_config=$HOME/conf/numa_config_4p.json --  
output=1_increase_4p --framework=1 --log=debug --shard_model_dir=../1_air --model_relation_config=  
$HOME/conf/model_relation_config.json --soc_version=<soc_version>
```

配置文件示例如下，对于TP切分后的模型，配置文件只存在deploy\_config节点：

```
{  
  "deploy_config" :[           //必选，部署模型与目标部署节点的映射关系  
    {  
      "submodel_name":"submodel1.air", // 前端切分后的文件名称，要和7.2.2.16 --shard_model_dir中前端切  
分后的模型名称保持一致  
      "deploy_device_id_list":"0:0:0" // 该模型要部署的目标设备cluster:0 node:0 item:0  
    },  
    {  
      "submodel_name":"submodel2.air",  
      "deploy_device_id_list":"0:0:1"  
    }  
  ],  
  "model_name_to_instance_id":[ // 必选  
    {  
      "submodel_name":"submodel1.air", // 模型对应的id，文件中由用户指定，不同文件对应不同id值即可  
      "model_instance_id":0  
    },  
    {  
      "submodel_name":"submodel2.air",  
      "model_instance_id":1  
    }  
  ],  
  "comm_group":[           // 非必选，若前端切分的模型包含通信算子，此处应是切分后模型通信算子的  
通信域相关信息  
    {  
      "group_name":"tp_group_name_0", // 前端切分模型通信算子的子通信域  
      "group_rank_list":["0,1]" // 前端切分模型通信算子的子rank列表  
    },  
    "rank_table":[  
      {  
        "rank_id":0,           // rankid与模型id的映射关系  
        "model_instance_id":0  
      },  
      {  
        "rank_id":1,  
        "model_instance_id":1  
      }  
    ]  
  }  
}
```

## 支持的型号

Atlas 推理系列产品

## 依赖约束

无。

## 7.2.3 输出选项

### 7.2.3.1 --output

#### 功能说明

- 如果是开源框架的网络模型：  
存放转换后的离线模型的路径以及文件名，例如：*\$HOME/module/out/tf\_resnet50*，转换后的模型文件名以指定的为准，自动以.om后缀结尾，例如：*tf\_resnet50.om*或*tf\_resnet50\_linux\_x86\_64.om*，若om文件名中包含操作系统及架构，则该om文件只能在该操作系统及架构的运行环境中使用。
- 如果是单算子描述文件（json格式）：  
存放转换后的单算子模型的路径，例如：*\$HOME/singleop/out/op\_model*。转换后的模型文件命名规则默认为：序号\_opType\_输入的描述 (dataType\_format\_shape)\_输出的描述 (dataType\_format\_shape)，如果不采用默认命名规则，可以通过单算子描述文件中的name属性指定模型文件名。

#### 关联参数

若使用atc命令转换出来的om模型文件名中含操作系统及架构，但操作系统及其架构与模型运行环境不一致时，则需要与--host\_env\_os、--host\_env\_cpu参数配合使用，设置模型运行环境的操作系统类型及架构。

#### 参数取值

##### 参数值：

- 如果是开源框架的网络模型：存放转换后的离线模型的路径以及文件名。
- 如果是单算子描述文件（json格式）：存放转换后的单算子模型的路径。

**参数值格式：**路径和文件名：支持大小写字母（a-z, A-Z）、数字（0-9）、下划线（\_）、短横线（-）、句点（.）、中文汉字。

#### 推荐配置及收益

无。

#### 示例

- Caffe框架网络模型：  
`--output=$HOME/module/out/caffe_resnet50`
- TF框架网络模型：  
`--output=$HOME/module/out/tf_resnet50`
- ONNX网络模型：  
`--output=$HOME/module/out/onnx_resnet50`
- 单算子描述文件：  
`--output=$HOME/singleop/out/op_model`

#### 支持的型号

Atlas 200/300/500 推理产品

Atlas 推理系列产品

## 依赖约束

无。

### 7.2.3.2 --output\_type

## 功能说明

指定网络输出数据类型或指定某个输出节点的输出类型。

## 关联参数

若指定某个输出节点的输出类型，则需要和[7.3.1.1 --out\\_nodes](#)参数配合使用。

## 参数取值

参数值：

- FP32：推荐分类网络、检测网络使用。
- UINT8：图像超分辨率网络，推荐使用，推理性能更好。
- FP16：推荐分类网络、检测网络使用。通常用于一个网络输出作为另一个网络输入场景。
- INT8

参数值约束：

模型转换完毕，在对应的\*.om模型文件中，数据类型以DT\_FLOAT或DT\_UINT8或DT\_FLOAT16或DT\_INT8值呈现。

若在模型转换时不指定网络具体输出数据类型，则以原始网络模型最后一层输出的算子数据类型为准；若指定了类型，则以该参数指定的类型为准，此时[7.3.1.7 --is\\_output\\_adjust\\_hw\\_layout](#)参数指定的类型不生效。

## 推荐配置及收益

无。

## 示例

- 指定网络输出数据类型  
`--output_type=FP32`
- 指定某个输出节点的输出数据类型  
例如：`--output_type="node1:0:FP16;node2:0:FP32"`，表示node1节点第一个输出设置为FP16，node2第一个节点输出设置为FP32。指定的节点必须放在双引号中，节点中间使用英文分号分隔。  
该场景下，该参数需要与[7.3.1.1 --out\\_nodes](#)参数配合使用。  
`--model=$HOME/module/resnet50_tensorflow*.pb --framework=3 --output=$HOME/module/out/tf_resnet50 --soc_version=<soc_version> --output_type="conv1:0:FP16" --out_nodes="conv1:0"`

## 支持的型号

Atlas 200/300/500 推理产品

Atlas 推理系列产品

## 依赖约束

无。

### 7.2.3.3 --check\_report

## 功能说明

预检结果保存文件路径和文件名。

## 关联参数

**7.2.1.2 --mode**：当**7.2.1.2 --mode**=0时解析图失败或**7.2.1.2 --mode**=3仅做预检时，用于生成check\_result.json预检结果文件。

## 参数取值

**参数值**：预检结果文件路径和文件名。

**参数值格式**：路径和文件名：支持大小写字母（a-z，A-Z）、数字（0-9）、下划线（\_）、短横线（-）、句点（.）、中文汉字。

**参数默认值**：执行atc命令当前路径生成check\_result.json

## 推荐配置及收益

无。

## 示例

```
--check_report=$HOME/module/out/check_result.json
```

## 支持的型号

Atlas 200/300/500 推理产品

Atlas 推理系列产品

## 使用约束

预检结果文件存储路径，除**7.2.3.3 --check\_report**参数设置的方式外，还可以配置环境变量ASCEND\_WORK\_PATH，几种方式优先级为

配置参数 “--check\_report” > 环境变量ASCEND\_WORK\_PATH > 默认存储路径（执行atc命令当前路径）。

关于环境变量ASCEND\_WORK\_PATH的详细说明请参见《[环境变量参考](#)》。

### 7.2.3.4 --json

#### 功能说明

离线模型、原始模型文件、GE dump图结构文件转换为json格式文件的路径和文件名。

如果是已有的离线模型转换为json格式文件，在转换后的文件中还可以查看原始模型转换为该离线模型时，使用的基础版本号（比如ATC软件版本信息，OPP算子包版本信息等）以及当时模型转换使用的atc命令。

#### 关联参数

- 离线模型转换为json  
该参数需要与[7.2.1.2 --mode=1](#)、[7.2.2.3 --om](#)参数配合使用。
- 原始模型文件转换为json  
该参数需要与[7.2.1.2 --mode=1](#)、[7.2.2.3 --om](#)参数、[7.2.2.4 --framework](#)配合使用。  
原始模型为MindSpore框架时，即--framework=1时，不支持转换为json文件。
- GE dump图结构文件转json  
该参数需要与[7.2.1.2 --mode=5](#)、[7.2.2.3 --om](#)参数配合使用。  
仅支持dump出的ge\_proto\*.txt格式文件转成json。

#### 参数取值

**参数值：** json格式文件的路径和文件名。

**参数值格式：** 路径和文件名：支持大小写字母（a-z，A-Z）、数字（0-9）、下划线（\_）、短横线（-）、句点（.）、中文汉字。

#### 推荐配置及收益

无。

#### 示例

- 离线模型转换为json：  
`--mode=1 --om=$HOME/module/out/tf_resnet50.om --json=$HOME/module/out/tf_resnet50.json`

在转换后的json文件中，可以查看原始模型转换为该离线模型时，使用的基础版本号，以及当时转换使用的atc命令，示例如下：

```
{
  "key": "opp_version",
  "value": {
    "s": "<version>"
  }
},
...
{
  "key": "atc_version",
  "value": {
    "s": "<version>"
  }
},
...
```

```
{
  "key": "atc_cmdline",
  "value": {
    "s": "xxx/atc.bin --model ./resnet50_tensorflow*.pb --framework 3 --output ./out/tf_resnet50 --soc_version <soc_version>"
  }
},
...
{
  "key": "soc_version",
  "value": {
    "s": "<soc_version>"
  }
},
...
```

- 原始模型文件转换为json  
`--mode=1 --om=$HOME/module/resnet50_tensorflow*.pb --json=$HOME/module/out/tf_resnet50.json --framework=3`
- GE dump图结构文件转json  
`--mode=5 --om=$HOME/module/ge_proto_00000000_PreRunBegin.txt --json=$HOME/module/out/ge_proto.json`

## 支持的型号

Atlas 200/300/500 推理产品

Atlas 推理系列产品

## 依赖约束

无。

### 7.2.3.5 --host\_env\_os

## 功能说明

若模型编译环境的操作系统及其架构与模型运行环境不一致时，则需使用本参数设置模型运行环境的操作系统类型。

如果不设置，则默认取模型编译环境的操作系统类型，即atc工具所在环境的操作系统类型。

## 关联参数

与[--host\\_env\\_cpu](#)参数配合使用，通过[--host\\_env\\_os](#)参数设置操作系统类型、通过[host\\_env\\_cpu](#)参数设置操作系统架构。

## 参数取值

**参数值：**执行[atc --help](#)命令查看“[--host\\_env\\_os](#)”参数支持的取值。

**参数默认值：**执行[atc --help](#)命令查看“[--host\\_env\\_os](#)”参数的默认值或查看\${INSTALL\_DIR}/opp/scene.info文件中的取值。

\${INSTALL\_DIR}请替换为CANN软件安装后文件存储路径。例如，若安装的Ascend-cann-toolkit软件包，则安装后文件存储路径为：\$HOME/Ascend/ascend-toolkit/latest。

## 推荐配置及收益

无。

## 示例

```
--host_env_os=linux --host_env_cpu=x86_64
```

转换后的离线模型文件名称会包含操作系统类型、架构，例如：`xxx_linux_x86_64.om`

## 支持的型号

Atlas 200/300/500 推理产品

Atlas 推理系列产品

## 依赖约束

无。

### 7.2.3.6 --host\_env\_cpu

## 功能说明

若模型编译环境的操作系统及其架构与模型运行环境不一致时，则需使用本参数设置模型运行环境的操作系统架构。

如果不设置，则默认取模型编译环境的操作系统架构，即atc工具所在环境的操作系统架构。

## 关联参数

与`--host_env_os`参数配合使用，通过`--host_env_os`参数设置操作系统类型、通过`--host_env_cpu`参数设置操作系统架构。

## 参数取值

**参数值：**执行`atc --help`命令查看“`--host_env_cpu`”参数支持的取值。

**参数默认值：**执行`atc --help`命令查看“`--host_env_cpu`”参数的默认值或查看`{INSTALL_DIR}/opp/scene.info`文件中的取值。

`{INSTALL_DIR}`请替换为CANN软件安装后文件存储路径。例如，若安装的Ascend-cann-toolkit软件包，则安装后文件存储路径为：`$HOME/Ascend/ascend-toolkit/latest`。

## 推荐配置及收益

无。

## 示例

```
--host_env_os=linux --host_env_cpu=x86_64
```

转换后的离线模型文件名称会包含操作系统类型、架构，例如：`xxx_linux_x86_64.om`



## 支持的型号

Atlas 200/300/500 推理产品

Atlas 推理系列产品

## 依赖约束

无。

## 7.2.4 目标芯片选项

### 7.2.4.1 --soc\_version

#### 功能说明

指定模型转换时昇腾AI处理器的版本。

#### 关联参数

无。

#### 参数取值

**参数值：** <soc\_version>

如果无法确定具体的 <soc\_version>，则在安装昇腾AI处理器的服务器执行 **npu-smi info** 命令进行查询，在查询到的“Name”前增加Ascend信息，例如“Name”对应取值为 xxxyy，实际配置的 <soc\_version> 值为 Ascend xxxyy。

**参数值约束：** 请使用与芯片名相对应的 <soc\_version> 取值进行模型转换，然后再进行推理。

#### 推荐配置及收益

无。

## 支持的型号

Atlas 200/300/500 推理产品

Atlas 推理系列产品

## 依赖约束

无。

### 7.2.4.2 --core\_type

#### 功能说明

设置网络模型使用的Core类型。

该参数仅在Atlas 推理系列产品支持，其他产品型号配置后，模型转换成功，但是功能不生效。

关联参数

无。

参数取值

- 参数值：
- VectorCore
  - AiCore，默认为AiCore
- 参数值约束：若网络模型中包括Cube算子，则只能使用AiCore。

推荐配置及收益

无。

示例

```
--core_type=VectorCore
```

支持的型号

Atlas 推理系列产品

依赖约束

无。

7.2.4.3 --aicore\_num

功能说明

用于设置模型编译时使用的AI Core数目。

关联参数

无。

参数取值

参数值：（当前版本该参数预留，不建议配置；如需配置，则只能配置为默认值）

不同芯片版本该参数默认值不同，可从“\${INSTALL\_DIR}/x86\_64-linux/data/platform\_config/<soc\_version>.ini”文件中查看，该文件中如下参数的取值即为aicore\_num的默认取值：

```
[SoCInfo]
#aicore_num默认值，默认值即为最大值，实际模型转换时，xx请替换为具体取值
ai_core_cnt=xx
```

其中，`${INSTALL_DIR}`请替换为CANN软件安装后文件存储路径。例如，若安装的Ascend-cann-toolkit软件包，则安装后文件存储路径为：`$HOME/Ascend/ascend-toolkit/latest`。

**参数默认值：**默认值即为最大值

## 推荐配置及收益

无。

## 示例

```
--aicore_num=1
```

## 依赖约束

无。

### 7.2.4.4 --virtual\_type

## 功能说明

是否支持离线模型在昇腾虚拟化实例特性生成的虚拟设备上运行。

当前芯片算力比较大，云端用户或者小企业完全不需要使用这么大算力，昇腾虚拟化实例特性支持对芯片的算力进行切分，可满足用户按照自己的业务按需申请算力的诉求

虚拟设备是按照指定算力在芯片上申请的虚拟加速资源。

## 关联参数

当`--buffer_optimize`参数值设置为`l1_optimize`，则不能与`--virtual_type`参数同时使用，会出现报错，表示虚拟化场景不做l1融合，防止算子过大导致调度异常。

## 参数取值

- 0：默认值，离线模型不在昇腾虚拟化实例特性生成的虚拟设备上运行。
- 1：离线模型在不同算力的虚拟设备上运行。

## 推荐配置及收益

无。

## 示例

```
--virtual_type=1
```

## 支持的产品形态

Atlas 推理系列产品

## 使用约束

- 使用该参数时，请确保运行环境已经搭建好昇腾虚拟化实例特性环境。
- 若使用7.2.4.4 --virtual\_type=1进行模型转换，则转换后离线模型的NPU运行核数（blockdim）可能比实际aicore\_num核数大，为aicore\_num支持配置范围的最小公倍数：  
例如aicore\_num支持配置范围为{1,2,4,8}，则使用7.2.4.4 --virtual\_type=1参数转换后的离线模型，NPU运行核数可能为8。
- 7.2.4.4 --virtual\_type=1时，使用ATC工具转换后的模型，如果包括如下算子，会默认使用单核，该场景下，将会导致转换后的模型推理性能下降：
  - DynamicRNN
  - PadV2D
  - SquareSumV2
  - DynamicRNNV2
  - DynamicRNNV3
  - DynamicGRUV

## 7.3 高级功能参数

### 7.3.1 功能配置选项

#### 7.3.1.1 --out\_nodes

##### 功能说明

指定某层输出节点（算子）作为网络模型的输出或指定网络模型输出的名称。

如果不指定输出节点，则模型的输出默认为最后一层的算子信息，如果指定，则以指定的为准。某些情况下，用户想要查看某层算子参数是否合适，则需要将该层算子的参数输出，既可以在模型转换时通过该参数指定输出某层算子，模型转换后，在相应.om模型文件最后即可以看到指定输出算子的参数信息，如果通过.om模型文件无法查看，则可以将.om模型文件转换成json格式后查看。

##### 关联参数

当7.2.2.4 --framework取值为1，即为MindSpore框架网络模型时，设置本参数无效，但模型转换成功。

##### 参数取值

###### 参数值：

- 网络模型中的节点（node\_name）名称  
指定的输出节点必须放在双引号中，节点中间使用英文分号分隔。node\_name必须是模型转换前的网络模型中的节点名称，冒号后的数字表示第几个输出，例如node\_name1:0，表示节点名称为node\_name1的第1个输出。
- 某层输出节点的topname（该场景仅支持Caffe网络模型）

指定的输出节点必须放在双引号中，节点中间使用英文分号分隔。topname必须是模型编译前的Caffe网络模型中layer的某个top名称；若几个layer有相同的topname，则以最后一个layer的topname为准。

- 指定网络模型输出的名称（output的名称）（该场景仅支持ONNX网络模型）  
指定的output的名称必须放在双引号中，多个output的名称中间使用英文分号分隔。output必须是网络模型的输出。

#### 参数值约束：

- 参数值中的三种方式，使用时只能取其一，不能同时存在。
- 若参数值取值为某层输出节点的topname，该种方式仅限于Caffe网络模型。
- 若参数值取值为网络模型输出的名称（output的名称），该种方式仅限于ONNX网络模型。

## 推荐配置及收益

无。

## 示例

- 参数值取网络模型中某个节点（node\_name）的名称  
--out\_nodes="node\_name1:0;node\_name1:1;node\_name2:0"
- 参数值取某层输出节点的topname  
--out\_nodes="topname0;topname1;topname2"
- 参数值取网络模型输出的名称（output的名称）  
--out\_nodes="output1;output2;output3"

## 支持的型号

Atlas 200/300/500 推理产品

Atlas 推理系列产品

## 依赖约束

无。

### 7.3.1.2 --input\_fp16\_nodes

#### 功能说明

指定输入数据类型为float16的输入节点名称。

#### 关联参数

- 若配置了该参数，则不能对同一个输入节点同时使用[7.3.1.3 --insert\\_op\\_conf](#)参数。
- 当[7.2.2.4 --framework](#)取值为1，即为MindSpore框架网络模型时，设置本参数无效，但模型转换成功。

#### 参数取值

**参数值：**数据类型为float16的输入节点名称。

**参数值约束：**指定的节点必须放在双引号中，节点中间使用英文分号分隔。

## 推荐配置及收益

无。

## 示例

```
--input_fp16_nodes="node_name1;node_name2"
```

## 支持的型号

Atlas 200/300/500 推理产品

Atlas 推理系列产品

## 依赖约束

无。

### 7.3.1.3 --insert\_op\_conf

## 功能说明

插入算子的配置文件路径与文件名，例如AIPP预处理算子。

若使用该参数后，输入数据类型为UINT8。

## 关联参数

- 若配置了该参数，则不能对同一个输入节点同时使用[7.3.1.2 --input\\_fp16\\_nodes](#)参数。
- 该参数不能与[7.2.2.10 --dynamic\\_dims](#)同时使用。

## 参数取值

**参数值：**插入算子的配置文件路径与文件名。

**参数值格式：**路径和文件名：支持大小写字母（a-z，A-Z）、数字（0-9）、下划线（\_）、短横线（-）、句点（.）、中文汉字。

**参数值约束：**文件后缀不局限于.cfg格式，但是配置文件中的内容需要满足prototxt格式。

## 推荐配置及收益

无。

## 示例

下面以插入AIPP预处理算子为例进行说明，配置文件内容示例如下（文件名为举例为：*insert\_op.cfg*），关于AIPP预处理配置文件的详细配置说明，请查看[6.1 AIPP使能](#)章节。

```
aipp_op {  
  aipp_mode:static
```

```
input_format:YUV420SP_U8
csc_switch:true
var_rec_i_chn_0:0.00392157
var_rec_i_chn_1:0.00392157
var_rec_i_chn_2:0.00392157
}
```

将配置好的`insert_op.cfg`文件上传到ATC工具所在服务器任意目录，例如上传到`$HOME/module`，使用示例如下：

```
--insert_op_conf=$HOME/module/insert_op.cfg
```

## 支持的型号

Atlas 200/300/500 推理产品

Atlas 推理系列产品

## 依赖约束

- 如果用户设置了AIPP功能，同时又通过[7.2.2.6 --input\\_shape](#)设置了动态shape范围参数，则AIPP输出的宽和高要在[7.2.2.6 --input\\_shape](#)所设置的范围内。
- 如果用户设置了**静态AIPP**功能，同时又通过[7.2.2.9 --dynamic\\_image\\_size](#)设置了动态分辨率（输入图片的宽和高不确定）：  
该场景下，AIPP配置文件中不能开启Crop和Padding功能，并且需要将配置文件中的`src_image_size_w`和`src_image_size_h`取值设置为0。
- 如果用户设置了**动态AIPP**功能，同时又通过[7.2.2.8 --dynamic\\_batch\\_size](#)设置了动态BatchSize：  
实际推理时，调用《[CANN AscendCL应用软件开发指南（C&C++）（开放态）](#)》手册中的“AscendCL API参考>模型加载与执行>`aclmdlSetInputAIPP`”接口，设置动态AIPP相关参数值时，需确保batchSize要设置为最大Batch数。
- 如果用户设置了**动态AIPP**功能，同时又通过[7.2.2.9 --dynamic\\_image\\_size](#)设置了动态分辨率（输入图片的宽和高不确定）：  
实际推理时，调用《[CANN AscendCL应用软件开发指南（C&C++）（开放态）](#)》手册中的“AscendCL API参考>模型加载与执行>`aclmdlSetInputAIPP`”接口，设置动态AIPP相关参数值时，不能开启Crop和Padding功能。该场景下，还需要确保通过aclmdlSetInputAIPP接口设置的宽和高与《[CANN AscendCL应用软件开发指南（C&C++）（开放态）](#)》手册“AscendCL API参考>模型加载与执行>`aclmdlSetDynamicHWSIZE`”接口设置的宽、高相等，都必须设置成动态分辨率最大档位的宽、高。

### 7.3.1.4 --external\_weight

#### 功能说明

生成om模型文件时，是否将原始网络中的Const/Constant节点的权重外置，同时将节点类型转换为FileConstant类型。

若网络中的weight占用内存较大，且模型加载环境内存受限时，建议通过此配置项将网络中Const/Constant节点的权重外置，防止由于内存不足导致模型编译出错。

## 关联参数

需要和**7.2.3.1 --output**参数配合使用，生成的权重文件保存在与om文件同层级的weight目录下，权重文件以算子名称命名。

## 参数取值

- 0：权重不外置，直接保存在om模型文件中。**默认为0。**
- 1：权重外置，将网络中所有的Const/Constant节点的权重文件落盘，且该文件保存在与om文件同级的weight目录下，权重文件以算子名称命名。

## 推荐配置及收益

当网络中weight占用内存较大且对模型大小有限制时，建议将此配置项设置为1。

## 示例

以ONNX网络模型为例：

```
atc --framework=5 --model=$HOME/module/resnet50.onnx --output=$HOME/module/out/onnx_resnet50 --soc_version=<soc_version> --external_weight=1
```

## 支持的型号

Atlas 200/300/500 推理产品

Atlas 推理系列产品

## 依赖约束

在模型转换时，若将--external\_weight参数设置为1，则在使用AscendCL接口开发推理应用、加载模型时，仅支持从om模型文件加载模型，同时需将weight目录与om文件放在同级目录下，否则AscendCL无法加载weight目录下的权重文件。

### 7.3.1.5 --op\_name\_map

## 功能说明

扩展算子（非标准算子）映射配置文件路径和文件名，不同的网络中某扩展算子的功能不同，可以指定该扩展算子到具体网络中实际运行的扩展算子的映射。

## 关联参数

当**7.2.2.4 --framework**取值为1，即为MindSpore框架网络模型时，设置本参数无效，但模型转换成功。

## 参数取值

**参数值：**扩展算子映射配置文件路径和文件名。

**参数值格式：**路径和文件名：支持大小写字母（a-z, A-Z）、数字（0-9）、下划线（\_）、短横线（-）、句点（.）、中文汉字。



## 推荐配置及收益

无。

## 示例

扩展算子映射配置文件内容示例如下（文件名举例为：*opname\_map.cfg*）：

```
OpA:Network1OpA
```

将配置好的*opname\_map.cfg*上传到ATC工具所在服务器任意目录，例如上传到*\$HOME/module*，使用示例如下：

```
--op_name_map=$HOME/module/opname_map.cfg
```

## 支持的型号

Atlas 200/300/500 推理产品

Atlas 推理系列产品

## 依赖约束

无。

### 7.3.1.6 --is\_input\_adjust\_hw\_layout

## 功能说明

与7.3.1.2 [--input\\_fp16\\_nodes](#)参数配合使用，指定网络输入数据类型为float16、数据格式为NC1HWC0。单独配置本参数无效，但模型转换成功。

## 关联参数

- 将7.3.1.6 [--is\\_input\\_adjust\\_hw\\_layout](#)参数设置为true，同时与7.3.1.2 [--input\\_fp16\\_nodes](#)参数配合使用，通过7.3.1.2 [--input\\_fp16\\_nodes](#)参数指定的节点输入数据类型为float16、输出数据格式为NC1HWC0。
- 当7.2.2.4 [--framework](#)取值为1，既为MindSpore框架网络模型时，设置本参数无效，但模型转换成功。

## 参数取值

参数值：false或true

参数默认值：false

## 推荐配置及收益

无。

## 示例

```
atc --model=$HOME/module/resnet50_tensorflow*.pb --framework=3 --output=$HOME/module/out/tf_resnet50 --is_input_adjust_hw_layout=true --input_fp16_nodes="data" --soc_version=<soc_version>
```

## 支持的型号

Atlas 200/300/500 推理产品

Atlas 推理系列产品

## 依赖约束

无。

### 7.3.1.7 --is\_output\_adjust\_hw\_layout

## 功能说明

与7.3.1.1 --out\_nodes参数配合使用，指定网络输出数据类型为float16、数据格式为NC1HWC0。单独配置本参数无效，但模型转换成功。

## 关联参数

- 将7.3.1.7 --is\_output\_adjust\_hw\_layout参数设置为true，同时与7.3.1.1 --out\_nodes参数配合使用，通过7.3.1.1 --out\_nodes参数指定的节点输出数据类型为float16、输入数据格式为NC1HWC0。
- 当7.2.2.4 --framework取值为1，既为MindSpore框架网络模型时，设置本参数无效，但模型转换成功。

## 参数取值

参数值：false或true

参数默认值：false

## 推荐配置及收益

无。

## 示例

```
atc --model=$HOME/module/resnet50_tensorflow*.pb --framework=3 --output=$HOME/module/out/tf_resnet50 --is_output_adjust_hw_layout=true --out_nodes="prob:0" --soc_version=<soc_version>
```

## 支持的型号

Atlas 200/300/500 推理产品

Atlas 推理系列产品

## 依赖约束

无。

## 7.3.2 模型调优选项

### 7.3.2.1 --disable\_reuse\_memory

#### 功能说明

内存复用开关。

#### 关联参数

无。

#### 参数取值

- 1：关闭内存复用。如果网络模型较大，关闭内存复用开关，模型转换时可能会造成内存不足，导致模型编译失败。
- 0：开启内存复用。默认为0。

#### 推荐配置及收益

无。

#### 示例

```
--disable_reuse_memory=0
```

#### 支持的型号

Atlas 200/300/500 推理产品

Atlas 推理系列产品

#### 依赖约束

在内存复用场景下（默认开启内存复用），支持基于指定算子（节点名称/算子类型）单独分配内存。通过OP\_NO\_REUSE\_MEM环境变量指定要单独分配的一个或多个节点，支持混合配置。配置多个节点时，中间通过英文逗号（“,”）隔开。详细说明请参见《[环境变量参考](#)》。

- 基于节点名称配置  
节点名称需要配置为转换为CANN平台网络后的节点名称，节点名称可以通过设置DUMP\_GE\_GRAPH环境变量，在导出的ge\_onnx\_xxx\_Build.pbtxt最终图中查看“name”字段获取。  

```
export OP_NO_REUSE_MEM=gradients/logits/semantic/kernel/Regularizer/l2_regularizer_grad/Mul_1,resnet_v1_50/conv1_1/BatchNorm/AssignMovingAvg2
```
- 基于算子类型配置  

```
export OP_NO_REUSE_MEM=FusedMulAddN,BatchNorm
```
- 混合配置  

```
export OP_NO_REUSE_MEM=FusedMulAddN,resnet_v1_50/conv1_1/BatchNorm/AssignMovingAvg
```

### 7.3.2.2 --fusion\_switch\_file

#### 功能说明

融合规则（包括图融合和UB融合）开关配置文件路径以及文件名，通过该参数关闭配置文件中指定的融合规则。

- **图融合：**是FE根据融合规则进行改图的过程。图融合用融合后算子替换图中融合前算子，提升计算效率。图融合的场景如下：
  - 在某些算子的数学计算量可以进行优化的情况下，可以进行图融合，融合后可以节省计算时间。例如：conv+biasAdd，可以融合成一个算子，直接在l0c中完成累加，从而省去add的计算过程。
  - 在融合后的计算过程可以通过硬件指令加速的情况下，可以进行图融合，融合后能够加速。例如：conv+biasAdd的累加过程，就是通过l0c中的累加功能进行加速的，可以通过图融合完成。
- **UB融合：**UB即昇腾AI处理器上的Unified Buffer，UB融合指A算子的计算结果在Unified Buffer上，需要搬移到Global Memory。B算子再执行时，需要将A算子的输出由Global Memory再搬移到Unified Buffer，进行B的计算逻辑，计算完之后，又从Unified Buffer搬移回Global Memory。  
从这个过程会发现A的结果从Unified Buffer->Global Memory->Unified Buffer->Global Memory。这个经过Global Memory进行数据搬移的过程是浪费的，因此将A和B算子合并成一个算子，省去了数据搬移的过程叫UB融合。UB融合可以减少整网中数据搬移的时间（Global Memory>Unified Buffer,Unified Buffer->Global Memory），提高运算效率，有效降低带宽。

## 关联参数

无。

## 参数取值

**参数值：**配置文件路径以及文件名。

**参数值格式：**路径和文件名：支持大小写字母（a-z，A-Z）、数字（0-9）、下划线（\_）、短横线（-）、句点（.）、中文汉字。

**参数值约束：**

系统内置的图融合和UB融合规则，均为默认开启，用户可以根据需要通过该参数关闭指定的融合规则。当前可以关闭的融合规则请参见《[图融合和UB融合规则参考](#)》，由于系统机制，其他融合规则无法关闭。

## 推荐配置及收益

无。

## 示例

- **场景1：逐条配置待关闭融合规则**

配置文件样例如下，冒号前面为融合规则名，后面字段表示融合规则是否开启（融合规则开关配置文件名举例为 *fusion\_switch.cfg*）：

```
xxxFusionPass:off
yyyFusionPass:off
....
```

- **场景2：一键式关闭融合规则**

该参数支持用户一键式关闭融合规则，配置文件样例如下：

```
{
  "Switch":{
    "GraphFusion":{
      "ALL":"off"
    },

```

```
"UBFusion":{
  "ALL":"off"
}
}
```

说明：

- 关闭某些融合规则可能会导致功能问题，因此此处的一键式关闭仅关闭系统部分融合规则，而不是全部融合规则。
- 一键式关闭融合规则时，可以同时开启部分融合规则，样例如下：

```
{
  "Switch":{
    "GraphFusion":{
      "ALL":"off",
      "SoftmaxFusionPass":"on"
    },
    "UBFusion":{
      "ALL":"off",
      "ThePool2dQuantFusionPass":"on"
    }
  }
}
```

将上述配置好的 *fusion\_switch.cfg* 文件上传到 ATC 工具所在服务器任意目录，例如上传到 *\$HOME/module*，使用示例如下：

```
--fusion_switch_file=$HOME/module/fusion_switch.cfg
```

模型转换完毕，会生成 "fusion\_result.json" 文件：

- 若设置了 ASCEND\_WORK\_PATH 环境变量，则该文件生成路径为：  
*\$ASCEND\_WORK\_PATH/FE/\${进程号}/fusion\_result.json*。环境变量 ASCEND\_WORK\_PATH 详细说明请参见《[环境变量参考](#)》。
- 若未设置上述环境变量，则默认生成在执行 atc 命令的当前路径。

该文件用于记录模型转换过程中除去 *fusion\_switch.cfg* 文件中关闭的融合规则外，仍旧使用的融合规则，其中，"match\_times" 字段表示模型转换过程中匹配到的融合规则次数，"effect\_times" 字段表示实际生效的次数。如果未配置 [7.3.2.2 --fusion\\_switch\\_file](#) 参数，则生成的 "fusion\_result.json" 文件中记录模型转换过程中匹配到的所有融合规则。

## 支持的型号

Atlas 200/300/500 推理产品

Atlas 推理系列产品

## 使用约束

- 若网络模型中 Convolution 算子的 "group" 属性取值 == 模型文件 prototxt 中 "num\_output" 属性的取值，则上述配置文件中 *VxxxRequantFusionPass* 必须打开。
- AMCT 对原始框架模型进行量化时，会插入量化和反量化算子，而使用 ATC 工具进行模型转换过程中，会对插入的量化和反量化算子进行融合，此情况下再进行量化后模型 dump 结果与原始模型 dump 结果的比对可能不准确，因此如果用户想使用 AMCT 量化后的模型进行精度比对，则需要通过 [7.3.2.2 --fusion\\_switch\\_file](#) 参数关闭部分融合功能，该场景下需要关闭的融合规则如下：

Atlas 200/300/500 推理产品、Atlas 训练系列产品场景必须关闭的融合规则：

```
V100RequantFusionPass:off  
ConvConcatFusionPass:off  
SplitConvConcatFusionPass:off  
TbeEltwiseQuantFusionPass:off  
TbeConvDequantVaddReluQuantFusionPass:off  
TbeConvDequantVaddReluFusionPass:off  
TbeConvDequantQuantFusionPass:off  
TbeDepthwiseConvDequantFusionPass:off  
TbeFullyconnectionElemwiseDequantFusionPass:off  
TbeConv2DAddMulQuantPass:off  
TbePool2dQuantFusionPass:off  
TbeCommonRules0FusionPass:off  
TbeCommonRules2FusionPass:off
```

Atlas 推理系列产品必须关闭的融合规则:

```
V200RequantFusionPass:off  
ConvConcatFusionPass:off  
SplitConvConcatFusionPass:off  
TbeEltwiseQuantFusionPass:off  
TbeConvDequantVaddReluQuantFusionPass:off  
TbeConvDequantVaddReluFusionPass:off  
TbeConvDequantQuantFusionPass:off  
TbeDepthwiseConvDequantFusionPass:off  
TbeFullyconnectionElemwiseDequantFusionPass:off  
TbeConv2DAddMulQuantPass:off  
TbePool2dQuantFusionPass:off  
TbeCommonRules0FusionPass:off  
TbeCommonRules2FusionPass:off
```

融合规则解释如下:

- V100RequantFusionPass  
图融合规则。V100量化场景下, 满足反量化(dequant)和量化(quant)相关pattern时, 进行部署优化, 提升推理性能。
- V200RequantFusionPass  
图融合规则。V200量化场景下, 满足反量化(dequant)和量化(quant)相关pattern时, 进行部署优化, 提升推理性能。
- ConvConcatFusionPass  
图融合, 支持conv2d\*N+concat算子的图融合规则, conv2d后面可以连接dequant和Relu类算子。
- SplitConvConcatFusionPass  
图融合, 支持split+conv2d\*N+concat算子的融合规则, conv2d后面可以连接dequant和Relu类算子。
- TbeEltwiseQuantFusionPass  
UB融合, 支持elemwise+quant算子的UB融合, quant算子为可选节点。
- TbeConvDequantVaddReluQuantFusionPass  
UB融合规则。量化场景下, 对Conv-dequant-vadd-relu-quant连续的节点, 标记UB融合, 提升推理性能。
- TbeConvDequantVaddReluFusionPass  
UB融合, 支持conv2d+dequant+vadd+relu/conv2d+dequant+(leakyrelu)+vadd算子的融合节点。
- TbeConvDequantQuantFusionPass  
UB融合规则。量化场景下, 对Conv-dequant-quant连续的节点, 标记UB融合, 提升推理性能。
- TbeDepthwiseConvDequantFusionPass  
UB融合, 支持depthwiseConv2d+dequant+(relu/mul)+quant/  
depthwiseConv2d+dequant+(sigmoid)+mul/depthwiseConv2d+requant/

- depthwiseConv2d+(power+relu6+power)+elemwise+(quant)算子的融合节点。
- TbeFullyconnectionElemwiseDequantFusionPass  
UB融合，支持如下两种形式的融合：
    - i. 固定shape场景BatchMatMul/BatchMatMulV2 + elemwise的融合。
    - ii. 固定shape场景MatMul/MatMulV2/BatchMatMul/BatchMatMulV2 + AscendDequant + elemwise1(+ elemwise2)的融合。
  - TbeConv2DAddMulQuantPass  
UB融合，支持conv+dequant+add+quant融合，add算子除quant外还必须有另两路任意输出才可以进行融合。
  - TbePool2dQuantFusionPass  
UB融合规则。量化场景下，对Pool2d-quant连续的节点，标记UB融合，提升推理性能。
  - TbeCommonRules0FusionPass  
UB融合，支持StridedRead+Conv2D+dequant+elemwise+quant+StridedWrite算子的UB融合，除Conv2D外，其他节点都是可选节点。
  - TbeCommonRules2FusionPass  
UB融合，支持StridedRead+Conv2D+dequant+elemwise+quant+StridedWrite算子的UB融合，除Conv2D外，其他节点都是可选节点；elemwise支持多输出场景下的融合。

### 7.3.2.3 --enable\_scope\_fusion\_passes

#### 功能说明

指定编译时需要生效的Scope融合规则列表。

无论是内置还是用户自定义的Scope融合规则，都分为如下两类：

- 通用融合规则（General）：各网络通用的Scope融合规则；默认生效，不支持用户指定失效。
- 定制化融合规则（Non-General）：特定网络适用的Scope融合规则；默认不生效，用户可以通过[7.3.2.3 --enable\\_scope\\_fusion\\_passes](#)指定生效的融合规则列表。

当前支持的融合规则请参见《[TensorFlow Parser Scope融合规则参考](#)》。

#### 关联参数

无。

#### 参数取值

**参数值：**注册的融合规则名称。

**参数值格式：**允许传入多个规则列表，中间使用英文逗号分隔，例如ScopePass1,ScopePass2,...。

## 推荐配置及收益

无。

## 示例

```
--enable_scope_fusion_passes=ScopePass1,ScopePass2
```

## 支持的型号

Atlas 200/300/500 推理产品

Atlas 推理系列产品

## 使用约束

该参数只适用于TensorFlow网络模型。如果要查看模型转换过程中融合规则相关的日志信息，则[7.3.4.2 --log](#)至少要设置为warning级别。

### 7.3.2.4 --enable\_small\_channel

## 功能说明

是否使能small channel的优化，使能后在channel $\leq$ 4的卷积层会有性能收益。建议用户在推理场景下打开此开关。

## 关联参数

该参数使能后，建议与[7.3.1.3 --insert\\_op\\_conf](#)参数（AIPP功能）配合使用，可以获得更优的性能。

在配合使用时，由于软件约束，只能和静态AIPP配合使用，不能和动态AIPP配合使用。

## 参数取值

**参数值：**

- 0：关闭。默认为0。
- 1：使能。

**参数值约束：**如果模型Input的channel $\leq$ 4，建议开启该参数，并配合静态AIPP（[7.3.1.3 --insert\\_op\\_conf](#)）使用，可获得更优的性能；如果开启之后出现性能下降，建议进行Tiling调优。

## 推荐配置及收益

无。

## 示例

```
--enable_small_channel=1
```



## 支持的型号

Atlas 200/300/500 推理产品

Atlas 推理系列产品

## 使用约束

该参数使能后，建议与AIPP功能[7.3.1.3 --insert\\_op\\_conf](#)同时使用，否则可能没有收益。

### 7.3.2.5 --ac\_parallel\_enable

## 功能说明

动态shape图中，是否允许AI CPU算子和AI Core算子并行运行。**保留参数，暂未使用。**

动态shape图中，开关开启时，系统自动识别图中可以和AI Core并发的AI CPU算子，不同引擎的算子下发到不同流上，实现多引擎间的并行，从而提升资源利用效率和动态shape执行性能。

## 关联参数

无。

## 参数取值

**参数值：**

- 1：允许AI CPU和AI Core算子间的并行运行。
- 0：AI CPU算子不会单独分流。默认为0。

## 推荐配置及收益

无。

## 示例

```
--ac_parallel_enable=1
```

## 支持的型号

Atlas 推理系列产品

### 7.3.2.6 --compression\_optimize\_conf

## 功能说明

压缩优化功能配置文件路径以及文件名，通过该参数使能配置文件中指定的压缩优化特性，从而提升网络性能。

## 关联参数

若通过该参数配置了**calibration**量化特性，则不能再使用高精度特性，比如不能再通过**7.3.3.1 --precision\_mode**参数配置**force\_fp32**或**must\_keep\_origin\_dtype**（原图fp32输入）；不能再通过**7.3.3.2 --precision\_mode\_v2**参数配置**origin**；不能通过**7.3.3.3 --op\_precision\_mode**配置**high\_precision**参数等。在高精度模式下设置量化参数，既拿不到量化的性能收益，也拿不到高精度模式的精度收益。

## 参数取值

**参数值：**配置文件路径以及文件名。

**参数值格式：**路径和文件名：支持大小写字母（a-z，A-Z）、数字（0-9）、下划线（\_）、短横线（-）、句点（.）、中文汉字。

**参数值约束：**

当前仅支持配置如下两种压缩方式，用户根据实际情况决定配置哪种压缩方式：

```
enable_first_layer_quantization:true
calibration:
{
  input_data_dir: ./data.bin,d2.bin
  input_shape: in:16,16;in1:16,16
  config_file: simple_config.cfg
  infer_soc: xxxxxx
  infer_device_id: 0
  log: info
}
```

其中：

- **enable\_first\_layer\_quantization：**用于控制AIPP首层卷积是否进行优化（AIPP会与量化后模型首层卷积CONV2D前的Quant算子进行融合），配置文件中冒号前面表示压缩优化特性名称，冒号后面表示是否使能该特性，true表示使能，false表示不使能，默认不使能。

使能**enable\_first\_layer\_quantization**特性时，只有网络结构中存在AIPP+CONV2D结构，并且在atc命令中将**7.3.2.4 --enable\_small\_channel**参数设置为1时，才有可能获得性能收益。由于量化后的模型存在一定程度上的精度损失，用户根据实际情况决定是否使能该特性。

- **calibration：**训练后量化，是指在模型训练结束之后进行的量化，对训练后模型中的权重由浮点数（当前支持float32）量化到低比特整数（当前支持int8），并通过少量校准数据基于推理过程对数据（activation）进行校准量化，进而加速模型推理速度。训练后量化简单易用，只需少量校准数据，适用于追求高易用性和缺乏训练资源的场景。训练后量化的样例请单击[Link](#)获取。

各参数说明如下，

- **input\_data\_dir：**必选配置，模型输入校准数据的bin文件路径。若模型有多个输入，则多个输入的bin数据文件以英文逗号分隔。校准数据集用来计算量化参数，获取校准集时应该具有代表性，推荐使用测试集的子集作为校准数据集。校准数据的bin文件的生成方式可以参考[链接](#)。
- **input\_shape：**必选配置，模型输入校准数据的shape信息，例如：  
input\_name1:n1,c1,h1,w1;input\_name2:n2,c2,h2,w2，节点中间使用英文分号分隔。
- **config\_file：**可选配置，训练后量化简易配置文件，该文件配置示例以及参数解释请参见[简易配置文件](#)。

- infer\_soc: 必选配置, 进行训练后量化校准推理时, 所使用的芯片名称, 查询方法请参见[参数取值](#)。
- infer\_device\_id: 可选配置, 进行训练后量化校准推理时所使用昇腾AI处理器设备的ID, 默认为0。
- log: 可选配置, 设置训练后量化时的日志等级, 该参数只控制训练后量化过程中显示的日志级别, 默认显示info级别:
  - debug: 输出debug/info/warning/error/event级别的日志信息。
  - info: 输出info/warning/error/event级别的日志信息。
  - warning: 输出warning/error/event级别的日志信息。
  - error: 输出error/event级别的日志信息。

此外, 训练后量化过程中的日志打屏以及日志落盘信息由  
**AMCT\_LOG\_DUMP**环境变量进行控制:

- **export AMCT\_LOG\_DUMP=1**: 表示日志打印到屏幕, 不保存量化因子record文件和graph文件。
- **export AMCT\_LOG\_DUMP=2**: 将日志落盘到当前路径的amct\_log\_时间戳/amct\_acl.log文件中, 并保存量化因子record文件。
- **export AMCT\_LOG\_DUMP=3**: 将日志落盘到当前路径的amct\_log\_时间戳/amct\_acl.log文件中, 并保存量化因子record文件和graph文件。

为防止日志文件、record文件、graph文件持续落盘导致磁盘被写满, 请及时清理这些文件。

如果用户配置了**ASCEND\_WORK\_PATH**环境变量, 则上述日志、量化因子record文件和graph文件存储到该环境变量指定的路径下, 例如  
**ASCEND\_WORK\_PATH=/home/test**, 则存储路径为: **/home/test/amct\_acl/amct\_log\_{pid}\_时间戳**。其中, amct\_acl模型转换过程中会自动创建, {pid}为进程号。

**参数默认值:** 无。

## 推荐配置及收益

无

## 示例

假设压缩优化功能配置文件名称为*compression\_optimize.cfg*, 文件内容配置示例如下:

```
enable_first_layer_quantization:true
calibration:
{
  input_data_dir: ./data.bin;d2.bin
  input_shape: in:16,16;in1:16,16
  config_file: simple_config.cfg
  infer_soc: xxxxxxxx
  infer_device_id: 0
  infer_ip: x.x.x.x
  infer_port: 1000
  log: info
}
```

将该文件上传到ATC工具所在服务器，例如上传到 $\$HOME/module$ ，使用示例如下：

```
--compression_optimize_conf=$HOME/module/compression_optimize.cfg
```

## 支持的型号

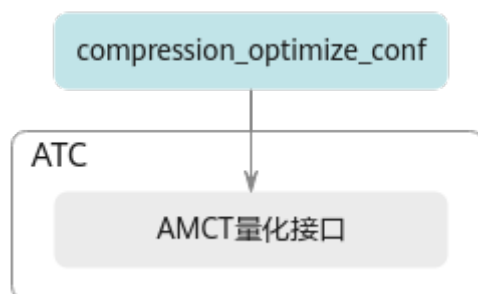
Atlas 200/300/500 推理产品

Atlas 推理系列产品

## 依赖约束

- 使用该参数中的**enable\_first\_layer\_quantization**特性时，请确保使用的模型是由AMCT（AMCT）进行量化操作后输出的部署模型。
- 使用配置文件中的**calibration**训练后量化功能时，只支持带NPU设备的安装场景，详细介绍请参见手册搭建对应产品环境。
- 使用配置文件中的calibration进行训练后量化功能时，ATC工具会调用AMCT量化接口执行相关操作，原理图如下：

图 7-2 训练后量化原理简图



### 7.3.2.7 --buffer\_optimize

#### 功能说明

数据缓存优化开关。

#### 关联参数

当--buffer\_optimize参数值设置为l1\_optimize，则不能与--virtual\_type参数同时使用，会出现报错，表示虚拟化场景不做l1融合，防止算子过大导致调度异常。

#### 参数取值

参数值：

- l1\_optimize：表示开启l1优化。当前版本该参数无效，等同于off\_optimize。
- l2\_optimize：表示开启l2优化。默认为l2\_optimize。
- off\_optimize：表示关闭数据缓存优化。

#### 推荐配置及收益

建议打开数据缓存优化功能：开启数据缓存优化可提高计算效率、提升性能，但由于部分算子在实现上可能存在未考虑的场景，导致影响精度，因此在出现精度问题时可

以尝试关闭数据缓存优化。如果关闭数据缓存优化功能后，精度达标，则需要识别出问题算子，反馈给技术支持进一步分析、解决算子问题，解决算子问题后，建议仍旧保持开启数据缓存优化功能。

## 示例

```
--buffer_optimize=l2_optimize
```

## 支持的型号

Atlas 200/300/500 推理产品

Atlas 推理系列产品

## 依赖约束

无。

### 7.3.2.8 --mdl\_bank\_path

## 功能说明

加载子图调优后自定义知识库的路径。

子图调优详情请参见《[AOE工具使用指南](#)》。

## 关联参数

该参数需要与[7.3.2.7 --buffer\\_optimize](#)参数配合使用，仅在数据缓存优化开关打开的情况下生效，通过利用高速缓存[7.3.2.7 --buffer\\_optimize](#)暂存数据的方式，达到提升性能的目的。

## 参数取值

**参数值：**子图调优后自定义知识库路径。

**参数值格式：**支持大小写字母（a-z, A-Z）、数字（0-9）、下划线（\_）、中划线（-）、句点（.）。

**参数默认值：**\$HOME/Ascend/latest/data/aoe/custom/graph/<soc\_version>

## 推荐配置及收益

无。

## 示例

例如子图调优后自定义知识库的路径为\$HOME/custom\_module\_bank，则使用示例为：

```
--mdl_bank_path=$HOME/custom_module_path --buffer_optimize=l2_optimize
```

## 依赖约束

加载子图调优后自定义知识库路径优先级：**7.3.2.8 --mdl\_bank\_path**参数加载路径>**TUNE\_BANK\_PATH**环境变量设置路径>默认子图调优后自定义知识库路径。

1. 如果模型转换前，通过**TUNE\_BANK\_PATH**环境变量指定了子图调优自定义知识库路径，模型转换时又通过**7.3.2.8 --mdl\_bank\_path**参数加载了自定义知识库路径，该场景下以**7.3.2.8 --mdl\_bank\_path**参数加载的路径为准，**TUNE\_BANK\_PATH**环境变量设置的路径不生效。
2. **7.3.2.8 --mdl\_bank\_path**参数和环境变量指定路径都不生效或无可用自定义知识库，则使用默认自定义知识库路径。
3. 如果上述路径下都无可用的自定义知识库，则atc工具会查找子图调优内置知识库，该路径为：`$(INSTALL_DIR)/compiler/data/fusion_strategy/built-in`

## 支持的型号

Atlas 200/300/500 推理产品

Atlas 推理系列产品

## 7.3.3 算子调优选项

### 7.3.3.1 --precision\_mode

#### 功能说明

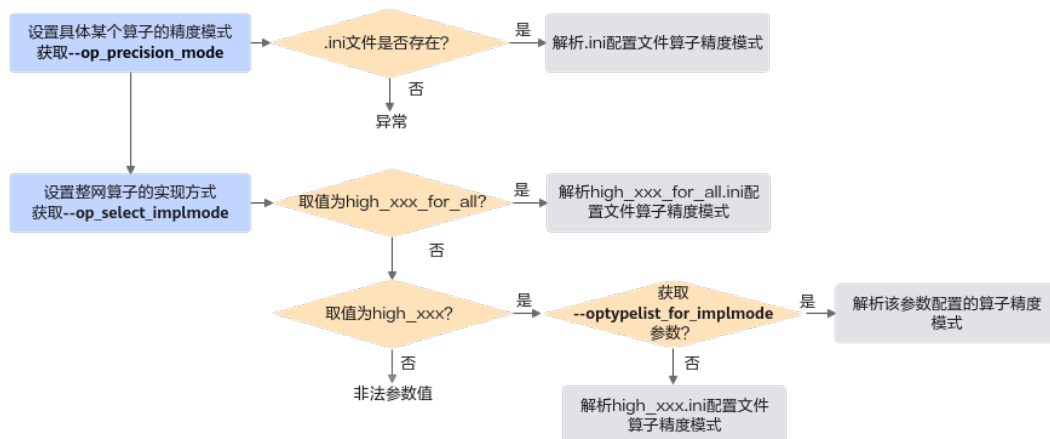
设置网络模型的精度模式。

#### 关联参数

- 当取值为**allow\_mix\_precision**时，如果用户想要在内置优化策略基础上进行调整，自行指定哪些算子允许降精度，哪些算子不允许降精度，则需要参见**7.3.3.4 --modify\_mixlist**参数设置。
- 推理场景下，使用**7.3.3.1 --precision\_mode**参数设置整个网络模型的精度模式，可能会有个别算子存在性能或精度问题，该场景下可以使用**7.3.3.7 --keep\_dtype**参数，使原始网络模型编译时保持个别算子的计算精度不变，但**7.3.3.1 --precision\_mode**参数取值为**must\_keep\_origin\_dtype**时，**7.3.3.7 --keep\_dtype**不生效。

关联参数示意图如图7-3所示。

图 7-3 关联参数示意图



设置具体算子精度模式场景下：

1. 首先读取7.3.3.3 --op\_precision\_mode参数，校验该参数的ini配置文件是否存在，若存在则解析文件并读取算子的精度模式，否则上报异常。
2. 7.3.3.3 --op\_precision\_mode不存在则读取7.3.3.5 --op\_select\_implmode参数：
  - a. 首先检测是否配置为high\_xxx\_for\_all参数，若是则解析high\_xxx\_for\_all.ini文件并读取算子的精度模式。
  - b. 若配置为high\_xxx参数，则检测是否配置7.3.3.6 --optypelist\_for\_implmode参数，若是，则读取该参数配置的算子精度模式；否则解析high\_xxx.ini文件并读取算子的精度模式。

## 参数取值

参数值：

- **force\_fp32/cube\_fp16in\_fp32out:**

配置为force\_fp32或cube\_fp16in\_fp32out，效果等同，系统内部都会根据矩阵类算子或矢量类算子，来选择不同的处理方式。cube\_fp16in\_fp32out为新版本中新增的，对于矩阵计算类算子，该选项语义更清晰。

- 对于矩阵计算类算子，系统内部会按算子实现的支持情况处理：
  - i. 优先选择输入数据类型为float16且输出数据类型为float32；
  - ii. 如果1中的场景不支持，则选择输入数据类型为float32且输出数据类型为float32；
  - iii. 如果2中的场景不支持，则选择输入数据类型为float16且输出数据类型为float16；
  - iv. 如果3中的场景不支持，则报错。
- 对于矢量计算类算子，表示网络模型中算子支持float16和float32时，强制选择float32，若原图精度为float16，也会强制转为float32。

如果网络模型中存在部分算子，并且该算子实现不支持float32，比如某算子仅支持float16类型，则该参数不生效，仍然使用支持的float16；如果该算子不支持float32，且又配置了黑名单（precision\_reduce = false），则会使用float32的AI CPU算子；如果AI CPU算子也不支持，则执行报错。

- **force\_fp16:**  
表示网络模型中算子支持float16和float32时，强制选择float16。
- **allow\_fp32\_to\_fp16:**
  - 对于矩阵类算子，使用float16。
  - 对于矢量类算子，优先保持原图精度，如果网络模型中算子支持float32，则保留原始精度float32，如果网络模型中算子不支持float32，则直接降低精度到float16。
- **must\_keep\_origin\_dtype:**  
保持原图精度。仅Atlas A2训练系列产品支持bfloat16类型。
  - 如果原图中某算子精度为float16，AI Core中该算子的实现不支持float16、仅支持float32和bfloat16，则系统内部会自动采用高精度float32。
  - 如果原图中某算子精度为float16，AI Core中该算子的实现不支持float16、仅支持bfloat16，则会使用float16的AI CPU算子；如果AI CPU算子也不支持，则执行报错。
  - 如果原图中某算子精度为float32，AI Core中该算子的实现不支持float32类型、仅支持float16类型，则会使用float32的AI CPU算子；如果AI CPU算子也不支持，则执行报错。
- **allow\_mix\_precision/allow\_mix\_precision\_fp16:**  
配置为allow\_mix\_precision或allow\_mix\_precision\_fp16，效果等同，均表示使用混合精度float16和float32数据类型来处理神经网络的过程。  
allow\_mix\_precision\_fp16为新版本中新增的，语义更清晰，便于理解。  
针对网络模型中float32数据类型的算子，按照内置的优化策略，自动将部分float32的算子降低精度到float16，从而在精度损失很小的情况下提升系统性能并减少内存使用。  
若配置了该种模式，则可以在OPP软件包安装路径\${INSTALL\_DIR}/opp/built-in/op\_impl/ai\_core/tbe/config/<soc\_version>/aic-<soc\_version>-ops-info.json内置优化策略文件中查看“precision\_reduce”参数的取值：
  - 若取值为true（白名单），则表示允许将当前float32类型的算子，降低精度到float16。
  - 若取值为false（黑名单），则不允许将当前float32类型的算子降低精度到float16，相应算子仍旧使用float32精度。
  - 若网络模型中算子没有配置该参数（灰名单），当前算子的混合精度处理机制和前一个算子保持一致，即如果前一个算子支持降精度处理，当前算子也支持降精度；如果前一个算子不允许降精度，当前算子也不支持降精度。

**参数默认值：**force\_fp16

## 推荐配置及收益

所配置的精度模式不同，网络模型精度以及性能有所不同，具体为：

精度高低排序：

force\_fp32>must\_keep\_origin\_dtype>allow\_fp32\_to\_fp16>allow\_mix\_precision>force\_fp16

性能优劣排序：

force\_fp16>=allow\_mix\_precision>allow\_fp32\_to\_fp16>must\_keep\_origin\_dtype>force\_fp32



## 示例

```
--precision_mode=force_fp16
```

## 支持的型号

Atlas 200/300/500 推理产品

Atlas 推理系列产品

## 依赖约束

无。

### 7.3.3.2 --precision\_mode\_v2

## 功能说明

设置网络模型的精度模式。

## 关联参数

- 该参数不能与[7.3.3.1 --precision\\_mode](#)参数同时使用，建议使用[7.3.3.2 --precision\\_mode\\_v2](#)参数。
- 当取值为**mixed\_float16**时，如果用户想要在内置优化策略基础上进行调整，自行指定哪些算子允许降精度，哪些算子不允许降精度，则需要参见[7.3.3.4 --modify\\_mixlist](#)参数设置。
- 推理场景下，使用[7.3.3.2 --precision\\_mode\\_v2](#)参数设置整个网络模型的精度模式，可能会有个别算子存在性能或精度问题，该场景下可以使用[7.3.3.7 --keep\\_dtype](#)参数，使原始网络模型编译时保持个别算子的计算精度不变，但[7.3.3.2 --precision\\_mode\\_v2](#)参数取值为**origin**时，[7.3.3.7 --keep\\_dtype](#)不生效。

## 参数取值

- fp16:**  
算子支持float16和float32数据类型时，强制选择float16。
- origin:**  
保持原图精度。仅Atlas A2训练系列产品支持bfloat16类型。
  - 如果原图中某算子精度为float16，AI Core中该算子的实现不支持float16、仅支持float32和bfloat16，则系统内部会自动采用高精度float32。
  - 如果原图中某算子精度为float16，AI Core中该算子的实现不支持float16、仅支持bfloat16，则会使用float16的AI CPU算子；如果AI CPU算子也不支持，则执行报错。
  - 如果原图中某算子精度为float32，AI Core中该算子的实现不支持float32类型、仅支持float16类型，则会使用float32的AI CPU算子；如果AI CPU算子也不支持，则执行报错。
- cube\_fp16in\_fp32out:**  
算子既支持float32又支持float16数据类型时，系统内部根据算子类型不同，选择不同的处理方式。

- 对于矩阵计算类算子，系统内部会按算子实现的支持情况处理：
  - i. 优先选择输入数据类型为float16且输出数据类型为float32；
  - ii. 如果1中的场景不支持，则选择输入数据类型为float32且输出数据类型为float32；
  - iii. 如果2中的场景不支持，则选择输入数据类型为float16且输出数据类型为float16；
  - iv. 如果3中的场景不支持，则报错。
- 对于矢量计算类算子，表示网络模型中算子支持float16和float32时，强制选择float32，若原图精度为float16，也会强制转为float32。  
如果网络模型中存在部分算子，并且该算子实现不支持float32，比如某算子仅支持float16类型，则该参数不生效，仍然使用支持的float16；如果该算子不支持float32，且又配置了黑名单（precision\_reduce = false），则会使用float32的AI CPU算子；如果AI CPU算子也不支持，则执行报错。
- **mixed\_float16:**  
表示使用混合精度float16和float32数据类型来处理神经网络。针对网络模型中float32数据类型的算子，按照内置的优化策略，自动将部分float32的算子降低精度到float16，从而在精度损失很小的情况下提升系统性能并减少内存使用。  
若配置了该种模式，则可以在OPP软件包安装路径\${INSTALL\_DIR}/opp/built-in/op\_impl/ai\_core/tbe/config/<soc\_version>/aic-<soc\_version>-ops-info.json内置优化策略文件中查看“precision\_reduce”参数的取值：
  - 若取值为true（白名单），则表示允许将当前float32类型的算子，降低精度到float16。
  - 若取值为false（黑名单），则不允许将当前float32类型的算子降低精度到float16，相应算子仍旧使用float32精度。
  - 若网络模型中算子没有配置该参数（灰名单），当前算子的混合精度处理机制和前一个算子保持一致，即如果前一个算子支持降精度处理，当前算子也支持降精度；如果前一个算子不允许降精度，当前算子也不支持降精度。

参数默认值：fp16

## 推荐配置及收益

所配置的精度模式不同，网络模型精度以及性能有所不同，具体为：

精度高低排序：origin>mixed\_float16>fp16；性能优劣排序：  
fp16>=mixed\_float16>origin

## 示例

```
--precision_mode_v2=fp16
```

## 支持的型号

Atlas 200/300/500 推理产品

Atlas 推理系列产品

## 依赖约束

无。

### 7.3.3.3 --op\_precision\_mode

#### 功能说明

设置指定算子内部处理时的精度模式，支持指定一个算子或多个算子。

#### 关联参数

- 该参数不能与[7.3.3.5 --op\\_select\\_implmode](#)、[7.3.3.6 --optypelist\\_for\\_implmode](#)参数同时使用，若三个参数同时配置，则只有[7.3.3.3 --op\\_precision\\_mode](#)参数指定的模式生效。

#### 参数取值

**参数值：**设置算子精度模式的配置文件（.ini格式）路径以及文件名，配置文件中支持设置如下精度模式：

- high\_precision：表示高精度。
- high\_performance：表示高性能。
- support\_out\_of\_bound\_index：表示对gather、scatter和segment类算子的indices输入进行越界校验，校验会降低算子的执行性能。

具体某个算子支持配置的精度/性能模式取值，可以通过CANN软件安装后文件存储路径的opp/built-in/op\_impl/ai\_core/tbe/impl\_mode/all\_ops\_impl\_mode.ini文件查看。

**参数值格式：**路径和文件名：支持大小写字母（a-z，A-Z）、数字（0-9）、下划线（\_）、短横线（-）、句点（.）、中文汉字。

**参数值约束：**

- 当前仅支持通过.ini配置文件方式设置算子精度，配置文件中的内容以key\_value（算子类型=精度模式）形式呈现，每一行设置一个算子的精度模式。
- 算子类型必须为基于Ascend IR定义的算子的OpType，算子类型查看方法请参见[10.3 如何确定原始框架网络模型中的算子与昇腾AI处理器支持的算子的对应关系](#)。

#### 推荐配置及收益

- 该参数不建议配置，若使用高性能或者高精度模式，网络性能或者精度不是最优，则可以使用该参数，通过配置ini文件调整具体某个算子的精度模式。
- 通过该参数加载的ini配置文件，建议使用[7.3.3.5 --op\\_select\\_implmode](#)参数用户另存后的ini配置文件，详情请参见[推荐配置及收益](#)。

#### 示例

构造算子精度模式配置文件`op_precision.ini`，并在该文件中按照算子类型、节点名称设置精度模式，每一行设置一个算子类型或节点名称的精度模式，按节点名称设置精度模式的优先级高于按算子类型。

配置样例如下：

```
[ByOpType]
optype1=high_precision
optype2=high_performance
optype4=support_out_of_bound_index
```

```
[ByNodeName]
nodename1=high_precision
nodename2=high_performance
nodename4=support_out_of_bound_index
```

将配置好的 *op\_precision.ini* 文件上传到 ATC 工具所在服务器任意目录，例如上传到 *\$HOME/conf*，使用示例如下：

```
--op_precision_mode=$HOME/conf/op_precision.ini
```

## 支持的型号

Atlas 200/300/500 推理产品

Atlas 推理系列产品

## 依赖约束

无。

### 7.3.3.4 --modify\_mixlist

## 功能说明

混合精度场景下，修改算子使用混合精度黑白灰名单。

- 白名单：表示混合精度模式下，允许将当前 float32 类型的算子，降低精度到 float16。
- 黑名单：表示混合精度模式下，不允许将当前 float32 类型的算子降低精度到 float16，相应算子仍旧使用 float32 精度。
- 灰名单：表示混合精度模式下，当前算子的混合精度处理机制和前一个算子保持一致，即如果前一个算子支持降精度处理，当前算子也支持降精度；如果前一个算子不允许降精度，当前算子也不支持降精度。

## 关联参数

开启混合精度方式：

- [7.3.3.1 --precision\\_mode](#) 参数设置为 allow\_mix\_precision、allow\_mix\_precision\_fp16。
- [7.3.3.2 --precision\\_mode\\_v2](#) 参数设置为 mixed\_float16，与 [7.3.3.1 --precision\\_mode](#) 参数不能同时配置，建议使用 [7.3.3.2 --precision\\_mode\\_v2](#)。

混合精度场景下网络模型中 float32 数据类型的算子，按照内置的优化策略，自动将部分 float32 的算子降低精度到 float16；而使用 [7.3.3.4 --modify\\_mixlist](#) 参数后，用户可以在内置优化策略基础上进行调整，自行指定哪些算子允许降精度，哪些算子不允许降精度。

## 参数取值

**参数值：**混合精度名单路径以及文件名。

**参数值格式：**路径和文件名：支持大小写字母（a-z, A-Z）、数字（0-9）、下划线（\_）、短横线（-）、句点（.）、中文汉字。

**参数值约束：**

- 名单格式为\*.json格式，文件中的算子列表由用户指定，多个算子使用英文逗号分隔。
- 配置的算子类型必须为基于Ascend IR定义的算子的OpType，算子类型查看方法请参见[10.3 如何确定原始框架网络模型中的算子与昇腾AI处理器支持的算子的对应关系](#)。

**推荐配置及收益**

无。

**示例**

系统内置算子是否在黑名单、白名单或灰名单中，可从“OPP安装目录/opp/built-in/op\_impl/ai\_core/tbe/config/<soc\_version>/aic-<soc\_version>-ops-info.json”文件中查询每个算子的precision\_reduce下的flag参数值，flag参数值为true表示白名单，为false表示黑名单，不配置flag参数表示灰名单，查询示例如下：

```
"Conv2D":{
  "precision_reduce":{
    "flag":"true"
  }
}
```

混合精度名单样例如下，*ops\_info.json*为文件名示例，*OpTypeA*、*OpTypeB*、*OpTypeC*、*OpTypeD*为算子示例。

```
{
  "black-list": {           // 黑名单
    "to-remove": [         // 黑名单算子转换为灰名单算子，配置该参数时，请确保被转换的算子已经存在于
      "OpTypeA"            黑名单中
    ],
    "to-add": [             // 白名单或灰名单算子转换为黑名单算子
      "OpTypeB"
    ]
  },
  "white-list": {           // 白名单
    "to-remove": [         // 白名单算子转换为灰名单算子，配置该参数时，请确保被转换的算子已经存在于
      "OpTypeC"            白名单中
    ],
    "to-add": [             // 黑名单或灰名单算子转换为白名单算子
      "OpTypeD"
    ]
  }
}
```

- 假设算子A默认在白名单中，如果您希望将该算子配置为黑名单算子，则配置示例和系统处理逻辑为：

- a. 将该算子添加到黑名单中：

```
{
  "black-list": {
    "to-add": [A]
  }
}
```

则系统会将该算子从白名单中删除，并添加到黑名单中，最终该算子在黑名单中。

- b. 将该算子从白名单中删除，同时添加到黑名单中：

```
{
  "black-list": {
```

```
"to-add": [A]
}
"white-list": {
  "to-remove": [A]
}
}
```

则系统会将该算子从白名单中删除，并添加到黑名单中，最终该算子在黑名单中。

- 对于只从黑/白名单中删除，而不添加到白/黑名单的场景，系统会将该算子添加到灰名单中，配置示例如下（例如，从白名单删除某个算子）：

```
{
  "white-list": {
    "to-remove": [A]
  }
}
```

则系统会将该算子从白名单中删除，然后添加到灰名单中，最终该算子在灰名单中。

将配置好的 *ops\_info.json* 文件上传到 ATC 工具所在服务器任意目录，例如上传到 *\$HOME/module*，使用示例如下：

```
--precision_mode=allow_mix_precision --modify_mixlist=$HOME/module/ops_info.json
```

## 支持的型号

Atlas 200/300/500 推理产品

Atlas 推理系列产品

## 依赖约束

无。

### 7.3.3.5 --op\_select\_implmode

## 功能说明

昇腾 AI 处理器部分内置算子有高精度和高性能实现方式，用户可以通过该参数配置模型编译时算子选择哪种实现方式。

高精度是指在 float16 输入场景，通过泰勒展开/牛顿迭代等手段进一步提升算子的精度；高性能是指在 float16 输入的情况下，不影响网络精度前提的最优性能实现。

## 关联参数

无。

## 参数取值

参数值：

- high\_precision**：表示算子采用高精度实现模式。  
该参数采用系统内置的配置文件设置算子实现模式，内置配置文件路径为 *{INSTALL\_DIR}/opp/built-in/op\_impl/ai\_core/tbe/impl\_mode/high\_precision.ini*。

为保持兼容，该参数仅对high\_precision.ini文件中算子列表生效，通过该列表可以控制算子生效的范围并保证之前版本的网络模型不受影响。

- **high\_performance**：表示算子采用高性能实现模式。

该参数采用系统内置的配置文件设置算子实现模式，内置配置文件路径为\${INSTALL\_DIR}/opp/built-in/op\_impl/ai\_core/tbe/impl\_mode/high\_performance.ini。

为保持兼容，该参数仅对high\_performance.ini文件中算子列表生效，通过该列表可以控制算子生效的范围并保证之前版本的网络模型不受影响。

- **high\_precision\_for\_all**：表示算子采用高精度实现模式。

该参数采用系统内置的配置文件设置算子实现模式，内置配置文件路径为\${INSTALL\_DIR}/opp/built-in/op\_impl/ai\_core/tbe/impl\_mode/high\_precision\_for\_all.ini，该文件中列表后续可能会跟随版本更新。

**该实现模式不保证兼容**，如果后续新的软件包中有算子新增了实现模式（即配置文件中新增了某个算子的实现模式），之前版本使用high\_precision\_for\_all的网络模型，在新版本上性能可能会下降。

- **high\_performance\_for\_all**：表示算子采用高性能实现模式。

该参数采用系统内置的配置文件设置算子实现模式，内置配置文件路径为\${INSTALL\_DIR}/opp/built-in/op\_impl/ai\_core/tbe/impl\_mode/high\_performance\_for\_all.ini，该文件中列表后续可能会跟随版本更新。

**该实现模式不保证兼容**，如果后续新的软件包中有算子新增了实现模式（即配置文件中新增了某个算子的实现模式），之前版本使用high\_performance\_for\_all的网络模型，在新版本上精度可能会下降。

上述实现模式，根据算子的dtype进行区分。\${INSTALL\_DIR}请替换为CANN软件安装后文件存储路径。例如，若安装的Ascend-cann-toolkit软件包，则安装后文件存储路径为：\$HOME/Ascend/ascend-toolkit/latest。

**参数默认值：**high\_performance

## 推荐配置及收益

不建议用户使用**7.3.3.5 --op\_select\_implmode**参数设置算子的实现模式，该参数仅作为调测使用，推荐通过**7.3.3.3 --op\_precision\_mode**参数加载ini配置文件方式设置算子精度模式：

- 如果用户对性能有更高要求，则建议优先使用**high\_performance\_for\_all**参数，若经过验证性能满足要求，则建议用户复制一份high\_performance\_for\_all.ini文件，并且重命名为“*网络模型.ini*”文件，跟随网络使用，不同网络模型使用不同的ini文件，后续模型转换时，可以直接使用**7.3.3.3 --op\_precision\_mode**参数加载保存的“*网络模型.ini*”配置文件。
- 如果用户对精度有更高要求，则建议优先使用**high\_precision\_for\_all**参数，若经过验证精度满足要求，则建议用户复制一份high\_precision\_for\_all.ini文件，并且重命名为“*网络模型.ini*”文件，跟随网络使用，不同网络模型使用不同的ini文件，后续模型转换时，可以直接使用**7.3.3.3 --op\_precision\_mode**参数加载保存的“*网络模型.ini*”配置文件。
- 如果用户在使用**high\_performance\_for\_all**时，虽然性能得到很大的提升，但是发现精度不满足要求，发现是由于xxx算子使用了高性能模式引起的，则需要复制一份high\_performance\_for\_all.ini文件，重命名为“*网络模型.ini*”文件，并将文件中该xxx算子的实现模式调整为高精度模式，后续模型转换时，直接使用**7.3.3.3 --op\_precision\_mode**参数加载“*网络模型.ini*”配置文件。



- 如果用户在使用**high\_precision\_for\_all**时，虽然精度得到很大的提升，但是发现性能下降较厉害，发现是由于xxx算子使用了高精度模式引起的，则需要复制一份**high\_precision\_for\_all.ini**文件，重命名为“网络模型.ini”文件，并将文件中该xxx算子的实现模式调整为高性能模式，后续模型转换时，直接使用**7.3.3.3 --op\_precision\_mode**参数加载“网络模型.ini”配置文件。

**high\_\*.ini**文件中算子的实现模式以**all\_ops\_impl\_mode.ini**文件（路径为\${INSTALL\_DIR}/opp/built-in/op\_impl/ai\_core/tbe/impl\_mode）所列出的为准，不在该文件中的实现模式不支持配置。

上述路径中的\${INSTALL\_DIR}请替换为CANN软件安装后文件存储路径。例如，若安装的Ascend-cann-toolkit软件包，则安装后文件存储路径为：\$HOME/Ascend/ascend-toolkit/latest。

## 示例

```
--op_select_implmode=high_precision
```

## 支持的型号

Atlas 200/300/500 推理产品

Atlas 推理系列产品

## 使用约束

- 如果有新支持精度模式的算子也选择高性能或者高精度模式，又不想破坏已有网络的精度或性能，则可以通过如下两种方式进行配置：
  - 通过**7.3.3.6 --optypelist\_for\_implmode**参数指定新增的具体算子  
`--op_select_implmode=high_precision --optypelist_for_implmode=算子optype`
  - 通过**7.3.3.3 --op\_precision\_mode**参数设置算子的精度模式  
构造算子精度模式配置文件**op\_precision.ini**，并在该文件中设置算子的精度模式，每一行设置一个算子的精度模式，样例如下：  

```
optype1=high_precision  
optype2=high_performance
```

  
将配置好的**op\_precision.ini**文件上传到ATC工具所在服务器任意目录，例如上传到\$HOME/conf，使用示例如下：  
`--op_precision_mode=$HOME/conf/op_precision.ini`
- 7.3.3.5 --op\_select\_implmode**参数表示设置网络模型中所有算子的高精度或高性能模式，如果算子实现了高精度和高性能，则运行时选择**7.3.3.5 --op\_select\_implmode**参数指定的模式；如果算子只实现了一种，则按照算子实现的方式运行，例如：  
某个算子当前只支持高精度，而**7.3.3.5 --op\_select\_implmode**设置为高性能，则**7.3.3.5 --op\_select\_implmode**参数对于该算子不生效，使用该算子当前实现的高精度方式运行。

### 7.3.3.6 --optypelist\_for\_implmode

#### 功能说明

设置optype列表中算子的实现模式，算子实现模式包括high\_precision、high\_performance两种。



## 关联参数

- 该参数需要与[7.3.3.5 --op\\_select\\_implmode](#)参数配合使用，通过[7.3.3.6 --optypelist\\_for\\_implmode](#)参数设置的算子，统一使用[7.3.3.5 --op\\_select\\_implmode](#)参数指定的实现模式，不能为列表中的每个算子设置不同的实现模式。
- 该参数配合[7.3.3.5 --op\\_select\\_implmode](#)参数使用时，不能与[7.3.3.3 --op\\_precision\\_mode](#)参数同时使用，若同时配置，则只有[7.3.3.3 --op\\_precision\\_mode](#)参数指定的模式生效。上述三个算子运行流程请参见[7.3.3.1 --precision\\_mode](#)中的图7-3。

## 参数取值

**参数值：**算子列表。

**参数值约束：**

- 该列表中的算子OpType必须为基于Ascend IR定义的算子的OpType，算子类型查看方法请参见[10.3 如何确定原始框架网络模型中的算子与昇腾AI处理器支持的算子的对应关系](#)。
- 该列表中的算子使用[7.3.3.5 --op\\_select\\_implmode](#)参数指定的实现模式，且仅支持指定为high\_precision、high\_performance两种模式，多个算子使用英文逗号进行分隔。
- 该参数仅对指定的算子生效，不指定的算子按照默认实现方式选择。

## 推荐配置及收益

无。

## 示例

```
--op_select_implmode=high_precision --optypelist_for_implmode=Pooling,SoftmaxV2
```

上述配置示例表示对Pooling、SoftmaxV2算子使用统一的高精度模式，未指定算子使用算子的默认实现方式。

## 支持的型号

Atlas 200/300/500 推理产品

Atlas 推理系列产品

## 依赖约束

无。

### 7.3.3.7 --keep\_dtype

#### 功能说明

保持原始网络模型编译时个别算子的计算精度不变。

推理场景下，使用[7.3.3.1 --precision\\_mode](#)参数设置整个网络模型的精度模式，可能会有个别算子存在性能或精度问题，为此引入[7.3.3.7 --keep\\_dtype](#)参数，保持原始网

络模型编译时个别算子的计算精度不变，若原始网络模型中算子的计算精度，在昇腾AI处理器上不支持，则系统内部会自动采用算子支持的高精度来计算。

## 关联参数

- 该参数需要与[7.3.3.1 --precision\\_mode](#)参数配合使用，但当[7.3.3.1 --precision\\_mode](#)参数取值为must\_keep\_origin\_dtype时，该参数不生效。
- [7.3.3.8 --customize\\_dtypes](#)参数与[7.3.3.7 --keep\\_dtype](#)参数都用于设置算子的计算精度，若涉及需提升模型推理精度的场景，建议先使用[7.3.3.7 --keep\\_dtype](#)参数保持原图精度，若精度依然得不到提升，可以尝试使用[7.3.3.8 --customize\\_dtypes](#)参数自定义某个或某些算子的计算精度。  
但需注意，使用[7.3.3.8 --customize\\_dtypes](#)参数且通过配置算子名称的方式，可能会由于内部模型优化过程中的融合、拆分等操作导致算子名称发生变化，进而导致配置不生效，未达到提升精度的目的，可进一步获取日志定位问题，关于日志的详细说明请参见《[日志参考（开放态）](#)》。
- 若同时使用了[7.3.3.8 --customize\\_dtypes](#)参数与[7.3.3.7 --keep\\_dtype](#)参数，则以[7.3.3.8 --customize\\_dtypes](#)参数设置的精度为准。

## 参数取值

**参数值：**算子配置文件路径以及文件名，配置文件中列举需保持计算精度的算子名称或算子类型，每个算子单独一行。

**参数值约束：**若为算子类型，则以OpType::typeName格式进行配置，每个OpType单独一行，且算子OpType必须为基于Ascend IR定义的算子的OpType，算子类型查看方法请参见[10.3 如何确定原始框架网络模型中的算子与昇腾AI处理器支持的算子的对应关系](#)。

**参数值格式：**路径和文件名：支持大小写字母（a-z，A-Z）、数字（0-9）、下划线（\_）、短横线（-）、句点（.）、中文汉字。

## 推荐配置及收益

无。

## 示例

- 若配置文件中为算子名称，则配置样例如下（文件名举例为 *execeptionlist.cfg*）：

```
Opname1
Opname2
...
```

- 若配置文件中为算子类型，则配置样例为（文件名举例为 *execeptionlist.cfg*）：

```
OpType::TypeName1
OpType::TypeName2
...
```

以TensorFlow ResNet50网络模型中的Relu算子为例，其对应的Ascend IR定义的算子类型为Relu，配置样例如下：

```
#算子名称配置样例：
fp32_vars/Relu
#算子类型配置样例：
OpType::Relu
```

将配置好的 *execeptionlist.cfg* 文件上传到ATC工具所在服务器任意目录，例如上传到\$HOME，使用示例如下：

```
--keep_dtype=$HOME/exceptionlist.cfg --precision_mode=force_fp16
```

模型编译时，*exceptionlist.cfg*文件中的算子，保持原始网络模型精度，即精度不会改变，其余网络模型中的算子以**7.3.3.1 --precision\_mode**参数指定的精度模式进行编译。

## 支持的型号

Atlas 200/300/500 推理产品

Atlas 推理系列产品

## 依赖约束

无。

### 7.3.3.8 --customize\_dtypes

## 功能说明

模型编译时自定义某个或某些算子的计算精度。

## 关联参数

- 若本参数与**7.3.3.1 --precision\_mode**配合使用时，除本参数指定的算子，模型中其它算子按**7.3.3.1 --precision\_mode**参数配置的精度模式来编译。
- 7.3.3.8 --customize\_dtypes**参数与**7.3.3.7 --keep\_dtype**参数都用于设置算子的计算精度，若涉及需提升模型推理精度的场景，建议先使用**7.3.3.7 --keep\_dtype**参数保持原图精度，若精度依然得不到提升，可以尝试使用**7.3.3.8 --customize\_dtypes**参数自定义某个或某些算子的计算精度。

但需注意，使用**7.3.3.8 --customize\_dtypes**参数且通过配置算子名称的方式，可能会由于内部模型优化过程中的融合、拆分等操作导致算子名称发生变化，进而导致配置不生效，未达到提升精度的目的，可进一步获取日志定位问题，关于日志的详细说明请参见《[日志参考（开放态）](#)》。

- 若同时使用了**7.3.3.8 --customize\_dtypes**参数与**7.3.3.7 --keep\_dtype**参数，则以**7.3.3.8 --customize\_dtypes**参数设置的精度为准。

## 参数取值

**参数值：**算子配置文件路径以及文件名，配置文件中列举需要自定义计算精度的算子名称或算子类型，每个算子单独一行。

**参数值约束：**

- 若为算子名称，以**Opname::InputDtype:dtype1,...,OutputDtype:dtype1,...**格式进行配置，每个Opname单独一行，dtype1, dtype2..需要与可设置计算精度的算子输入，算子输出的个数一一对应。
- 若为算子类型，以**OpType::TypeName:InputDtype:dtype1,...,OutputDtype:dtype1,...**格式进行配置，每个OpType单独一行，dtype1, dtype2..需要与可设置计算精度的算子输入，算子输出的个数一一对应，且算子OpType必须为基于Ascend IR定义的算子的OpType，算子类型查看方法请参见**10.3 如何确定原始框架网络模型中的算子与昇腾AI处理器支持的算子的对应关系**。

- 对于同一个算子，如果同时配置了**Opname**和**OpType**的配置项，编译时以**Opname**的配置项为准。
- 使用该参数指定某个算子的计算精度时，如果模型转换过程中该算子被融合掉，则该算子指定的计算精度不生效。

**参数值格式：**路径和文件名：支持大小写字母（a-z，A-Z）、数字（0-9）、下划线（\_）、短横线（-）、句点（.）、英文冒号(:)、中文汉字。

## 推荐配置及收益

无。

## 示例

- 若配置文件中为算子名称，则配置样例为（文件名举例为 *customize\_dtypes.cfg*）：  
*Opname1::InputDtype: dtype1, dtype2, ..., OutputDtype: dtype1, ...*  
*Opname2::InputDtype: dtype1, dtype2, ..., OutputDtype: dtype1, ...*
- 若配置文件中为算子类型，则配置样例为（文件名举例为 *customize\_dtypes.cfg*）：  
*OpType:: TypeName1: InputDtype: dtype1, dtype2, ..., OutputDtype: dtype1, ...*  
*OpType:: TypeName2: InputDtype: dtype1, dtype2, ..., OutputDtype: dtype1, ...*

算子具体支持的计算精度可以从《[算子清单（开放态）](#)》中查看。

以TensorFlow ResNet50网络模型中的Relu算子为例，其对应的Ascend IR定义的算子类型为Relu，该算子输入和输出只有一个，该配置样例如下：

- 算子名称配置样例：  
*fp32\_vars/Relu::InputDtype: float16, OutputDtype: int8*
- 算子类型配置样例：  
*OpType:: Relu: InputDtype: float16, OutputDtype: int8*

将配置好的*customize\_dtypes.cfg*文件上传到ATC工具所在服务器任意目录，例如上传到\$HOME，使用示例如下：

```
--customize_dtypes=$HOME/customize_dtypes.cfg --precision_mode=force_fp16
```

模型编译时，*customize\_dtypes.cfg*文件中的算子，使用指定的计算精度，其余网络模型中的算子以[7.3.3.1 --precision\\_mode](#)参数指定的精度模式进行编译。

## 支持的型号

Atlas 200/300/500 推理产品

Atlas 推理系列产品

## 使用约束

1. 使用该参数指定算子的计算精度，由于其优先级高于[7.3.3.1 --precision\\_mode](#)和[7.3.3.7 --keep\\_dtype](#)参数，可能会导致后续推理精度或者性能的下降。
2. 使用该参数指定算子的计算精度，如果指定的精度算子本身不支持，则会导致模型编译失败。

### 7.3.3.9 --op\_bank\_path

#### 功能说明

加载算子调优后自定义知识库的路径。  
算子调优详情请参见《[AOE工具使用指南](#)》。

#### 参数取值

**参数值：**算子调优后自定义知识库路径。  
**参数值格式：**支持大小写字母（a-z，A-Z）、数字（0-9）、下划线（\_）、中划线（-）、句点（.）。  
**参数默认值：**默认自定义知识库路径\$HOME/Ascend/latest/data/aoe/custom/op

#### 推荐配置及收益

无。

#### 示例

例如算子调优后自定义知识库的路径为\$HOME/custom\_tune\_bank，则使用示例为：

```
--op_bank_path=$HOME/custom_tune_bank
```

#### 使用约束

加载算子调优后自定义知识库路径优先级：**TUNE\_BANK\_PATH**环境变量设置路径>**7.3.3.9 --op\_bank\_path**参数加载路径>默认算子调优后自定义知识库路径。

1. 如果模型转换前，通过**TUNE\_BANK\_PATH**环境变量指定了算子调优自定义知识库路径，模型转换时又通过**7.3.3.9 --op\_bank\_path**参数加载了自定义知识库路径，该场景下以**TUNE\_BANK\_PATH**环境变量设置的路径为准，**7.3.3.9 --op\_bank\_path**参数加载的路径不生效。
2. **7.3.3.9 --op\_bank\_path**参数和环境变量指定路径都不生效前提下，使用默认自定义知识库路径。
3. 如果上述路径下都无可用的自定义知识库，则ATC工具会查找算子调优内置知识库。

#### 支持的型号

Atlas 200/300/500 推理产品  
Atlas 推理系列产品

### 7.3.3.10 --op\_debug\_level

#### 功能说明

TBE算子编译debug功能开关。

## 关联参数

如果要自行指定算子编译的过程文件存放路径，则需要通过[7.3.4.3 --debug\\_dir](#)参数指定，[7.3.3.10 --op\\_debug\\_level](#)取值为0时，使用[7.3.4.3 --debug\\_dir](#)参数不生效。

## 参数取值

### 参数值：

- 0：不开启算子debug功能，在当前执行路径**不生成**算子编译目录kernel\_meta。  
默认为0。
- 1：开启算子debug功能，在当前执行路径生成kernel\_meta文件夹，并在该文件夹下**生成**.o（算子二进制文件）、.json文件（算子描述文件）以及TBE指令映射文件（算子cce文件\*.cce和python-cce映射文件\*\_loc.json），用于后续分析AICore Error问题。
- 2：开启算子debug功能，在当前执行路径生成kernel\_meta文件夹，并在该文件夹下**生成**.o（算子二进制文件）、.json文件（算子描述文件）以及TBE指令映射文件（算子cce文件\*.cce和python-cce映射文件\*\_loc.json），用于后续分析AICore Error问题，同时设置为2，还会关闭编译优化开关、开启ccec调试功能（ccec编译器选项设置为-O0-g）。
- 3：不开启算子debug功能，在当前执行路径生成kernel\_meta文件夹，并在该文件夹中**生成**.o（算子二进制文件）和.json文件（算子描述文件），分析算子问题时可参考。
- 4：不开启算子debug功能，在当前执行路径生成kernel\_meta文件夹，并在该文件夹下**生成**.o（算子二进制文件）和.json文件（算子描述文件）以及TBE指令映射文件（算子cce文件\*.cce）和UB融合计算描述文件（{\$kernel\_name}\_compute.json），可在分析算子问题时进行问题复现、精度比对时使用。

### 参数值约束：

- 进行模型转换时，建议配置为0、3或4。如果需要定位AICore Error问题，则需要将参数值设置为1或2。设置为1或2后，由于加入了调试功能，会导致网络性能下降。
- 若[7.3.3.10 --op\\_debug\\_level](#)配置为0，同时又配置了[7.3.4.9 --op\\_debug\\_config](#)参数，该场景下在执行atc命令当前路径**仍旧会生成**算子编译目录kernel\_meta。

### 📖 说明

配置为2（即开启ccec编译选项）时，会导致算子Kernel（\*.o文件）大小增大。动态Shape场景下，由于算子编译时会遍历可能的Shape场景，因此可能会导致算子Kernel文件过大而无法进行编译，此种场景下，建议不要配置ccec编译选项。

由于算子Kernel文件过大而无法编译的报错日志示例如下：

```
message:link error ld.lld: error: InputSection too large for range extension thunk ./kernel_meta_XXXX.o
```

## 推荐配置及收益

无。

## 示例

```
--op_debug_level=1
```

## 支持的型号

Atlas 200/300/500 推理产品

Atlas 推理系列产品

## 使用约束

该参数优先级高于算子编译接口（TBE DSL的build接口或者TBE TIK的BuildCCE接口）中的**tbe\_debug\_level**的值。

## 7.3.4 调试选项

### 7.3.4.1 --dump\_mode

#### 功能说明

是否生成带shape信息的json文件。

#### 关联参数

该参数需要与[7.2.3.4 --json](#)、[7.2.1.2 --mode=1](#)、[7.2.2.4 --framework](#)、[7.2.2.3 --om](#)参数（需要为原始模型文件，如果为Caffe框架模型文件，还需要增加[7.2.2.2 --weight](#)参数）配合使用。

#### 参数取值

参数值：

- 0：不使能。
- 1：使能。

参数默认值：0

#### 推荐配置及收益

无。

#### 示例

```
atc --mode=1 --om=$HOME/module/resnet50_tensorflow*.pb --json=$HOME/module/out/tf_resnet50.json  
--framework=3 --dump_mode=1
```

## 支持的型号

Atlas 200/300/500 推理产品

Atlas 推理系列产品

## 依赖约束

无。



### 7.3.4.2 --log

#### 功能说明

设置ATC模型转换过程中日志的级别。

#### 关联参数

无。

#### 参数取值

- debug：输出debug/info/warning/error/event级别的运行信息。
- info：输出info/warning/error/event级别的运行信息。
- warning：输出warning/error/event级别的运行信息。
- error：输出/error/event级别的运行信息。
- null：不输出日志。**默认为null。**

#### 推荐配置及收益

无。

#### 示例

```
--log=debug
```

如果模型转换失败，则可以通过分析日志定位问题。日志格式如下，更多日志信息请参见《[日志参考（开放态）](#)》。

```
[Level] ModuleName(PID,PName):DateTimeMS [FileName:LineNumber]LogContent
```

各字段解释如下：

表 7-3 日志字段说明

字段	说明
Level	日志级别。运行日志存在5种日志级别：ERROR、WARNING、INFO、DEBUG、EVENT。
ModuleName	产生日志的模块的名称。
PID	进程ID。
PName	进程名称。
DateTimeMS	日志打印时间，格式为：yyyy-mm-dd-hh:mm:ss.SSS。
FileName:LineNumber	调用日志打印接口的文件及对应的行号。
LogContent	各模块具体的日志内容。

样例如下：



```
[INFO] FE(30741,atc.bin):2021-12-09-16:10:22.539.141 [fe_type_utils.cc:52]30741 GetRealPath:"path /usr/local/Ascend/opp/built-in/op_impl/ai_core/tbe/config/ascendxxx is not exist."  
[WARNING] FE(30741,atc.bin):2021-12-09-16:10:22.539.146 [sub_op_info_store.cc:52]30741 Initialize:"The config file[/usr/local/Ascend/opp/built-in/op_impl/ai_core/tbe/config/ascendxxx] of op information library[tbe-builtin] is not existed. "  
[ERROR] GE(30741,atc.bin):2021-12-09-16:10:22.539.201 [error_manager.cc:263]30741 ReportErrMsgage:[INIT][OPS_KER][Report][Error]error_code: W21000, arg path is not existed in map
```

问题定位思路:

表 7-4 问题定位思路

字段	说明	解决思路
GE	GE图编译或校验问题。	校验类报错，通常会给出明确的错误原因，此时需要针对性的修改模型转换使用的参数，以满足相关要求。
FE	算子融合问题。	无。
TEFUSION	<ul style="list-style-type: none"><li>算子预编译/编译问题。</li><li>融合算子编译问题。</li></ul>	常见错误信息以及解决思路： 1. ModuleNotFoundError: No module named 'decorator' 解决思路：根据提示信息安装pip包。 2. ModuleNotFoundError: No module named 'te' 解决思路：安装ATC工具所在软件包时，安装命令没有使用--pylocal，建议使用该参数重新安装相应软件包。
TBE	算子编译问题。	无。

支持的型号

Atlas 200/300/500 推理产品  
Atlas 推理系列产品

依赖约束

- 日志落盘：  
atc命令执行过程中，日志默认落盘到如下路径，由于7.3.4.2 --log默认值为null，即不输出日志，若下述路径存在日志信息，则为atc进程之外的其他日志信息，比如依赖Python相关信息；若想要日志体现atc进程相关信息，则模型转换时，需要设置7.3.4.2 --log参数（不能设置为null）。
  - \$HOME/ascend/log/debug/plog/plog-pid\*.log：调试日志。
  - \$HOME/ascend/log/run/plog/plog-pid\*.log：运行日志。
  - \$HOME/ascend/log/security/plog/plog-pid\*.log：安全日志。pid代表进程ID，“\*”表示该日志文件创建时的时间戳。
- 日志打屏：  
atc命令执行过程中，日志默认不打屏，如需打屏显示，则请在执行atc命令的当前窗口设置如下环境变量，然后再执行atc命令：

```
export ASCEND_SLOG_PRINT_TO_STDOUT=1
```

关于日志的更多信息请参见《[日志参考（开放态）](#)》。若设置上述环境变量后，仍旧未打屏有效信息，则请在atc命令设置[7.3.4.2 --log](#)参数（不能设置为null）显示相应的日志级别。

- 日志重定向：

如果不想日志落盘，而是重定向到文件，则模型转换前需要设置上述的日志打屏环境变量，并且atc命令需要设置[7.3.4.2 --log](#)参数（不能设置为null），样例如下：

```
atc xxx --log=debug >log.txt
```

### 7.3.4.3 --debug\_dir

#### 功能说明

用于配置模型转换过程中算子编译生成的调试相关过程文件的路径。

过程文件包括但不限于算子.o（算子二进制文件）、.json（算子描述文件）、.cce等文件，具体生成的文件以[7.3.3.10 --op\\_debug\\_level](#)参数设置的取值为准。

#### 关联参数

如果要自行指定算子编译的过程文件存放路径，需--debug\_dir参数与[7.3.3.10 --op\\_debug\\_level](#)参数配合使用，且当[7.3.3.10 --op\\_debug\\_level](#)取值为0时，使用[7.3.4.3 --debug\\_dir](#)参数不生效。

#### 参数取值

**参数值：**算子编译生成的调试相关文件的路径。

**参数值格式：**路径支持大小写字母（a-z, A-Z）、数字（0-9）、下划线（\_）、短横线（-）、句点（.）、中文汉字。

**参数值约束：**如果使用该参数，则在执行atc命令之前，请先创建该参数要指定的目录。

**参数默认值：**在执行atc命令的当前路径./kernel\_meta文件夹中生成算子编译的过程文件。

#### 推荐配置及收益

无。

#### 示例

例如创建的目录名为debug\_info，则执行命令为：

```
--debug_dir=$HOME/module/out/debug_info --op_debug_level=1
```

#### 支持的型号

Atlas 200/300/500 推理产品

Atlas 推理系列产品

## 使用约束

算子编译生成的调试文件存储路径，除[7.3.4.3 --debug\\_dir](#)参数设置的方式外，还可以配置环境变量ASCEND\_WORK\_PATH，几种方式优先级为：

配置参数“[7.3.4.3 --debug\\_dir](#)” > 环境变量ASCEND\_WORK\_PATH > 默认存储路径。

关于环境变量ASCEND\_WORK\_PATH的详细说明请参见《[环境变量参考](#)》。

### 7.3.4.4 --op\_compiler\_cache\_mode

## 功能说明

用于配置算子编译磁盘缓存模式。

## 关联参数

如果要自行指定算子编译磁盘缓存的路径，则需要通过[7.3.4.5 --op\\_compiler\\_cache\\_dir](#)参数指定。

## 参数取值

参数值：

- enable：表示启用算子编译缓存。启用后可以避免针对相同编译参数及算子参数的算子重复编译，从而提升编译速度。
- force：启用算子编译缓存功能，区别于enable模式，force模式下会强制刷新缓存，即先删除已有缓存，再重新编译并加入缓存。比如当用户的python变更、依赖库变更、算子调优后知识库变更等，需要先指定为force用于先清理已有的缓存，后续再修改为enable模式，以避免每次编译时都强制刷新缓存。
- disable：表示禁用算子编译缓存。

参数默认值：disable

参数值约束：

1. 由于force选项会先删除已有缓存，所以不建议在程序并行编译时设置，否则可能会导致其他模型使用的缓存内容被清除而导致失败。
2. 建议模型最终发布时设置编译缓存选项为disable或者force。
3. 如果算子调优后知识库变更，则需要通过设置为force来刷新缓存，否则无法应用新的调优知识库，从而导致调优应用执行失败。
4. 调试开关打开的场景下，即[7.3.3.10 --op\\_debug\\_level](#)配置非0值或者[7.3.4.9 --op\\_debug\\_config](#)配置非空时，会忽略[7.3.4.4 --op\\_compiler\\_cache\\_mode](#)参数的配置，不启用算子编译缓存。主要基于以下两点考虑：
  - 启用算子编译缓存功能（enable或force模式）后，相同编译参数的算子无需重复编译，编译过程日志无法完整记录。
  - 受限于缓存空间大小，对调试场景的编译结果不做缓存。

## 推荐配置及收益

推荐配置为enable：启用后可以避免针对相同编译参数及算子参数的算子重复编译，从而提升编译速度。

## 示例

```
--op_compiler_cache_mode=enable --op_compiler_cache_dir=$HOME/atc_data/kernel_cache --  
op_debug_level=3
```

## 支持的型号

Atlas 200/300/500 推理产品

Atlas 推理系列产品

## 依赖约束

启用算子编译缓存功能时，可以通过如下两种方式来设置缓存文件夹的磁盘空间大小：

### 1. 通过配置文件op\_cache.ini设置

ATC工具运行后，会在7.3.4.5 --op\_compiler\_cache\_dir参数指定路径下自动生成op\_cache.ini文件，用户可以通过该配置文件进行缓存磁盘空间大小的配置；若op\_cache.ini文件不存在，则需要手动创建。

打开该文件，增加如下信息：

```
#配置文件格式，必须包含，自动生成的文件中默认包括如下信息，手动创建时，需要输入  
[op_compiler_cache]  
#限制某个芯片下缓存文件夹的磁盘空间的大小，单位为MB  
ascend_max_op_cache_size=500  
#设置需要保留缓存的空间大小比例，取值范围：[1,100]，单位为百分比；例如80表示缓存空间不足时，  
删除缓存，保留80%  
ascend_remain_cache_size_ratio=80
```

- 上述文件中的ascend\_max\_op\_cache\_size和ascend\_remain\_cache\_size\_ratio参数取值都有效时，op\_cache.ini文件才会生效。
- 若多个使用者使用相同的用户登录ATC工具所在服务器，缓存路径相同，建议使用配置文件的方式进行设置，该场景下op\_cache.ini文件会影响所有使用者。

### 2. 通过环境变量设置

该场景下，可以通过环境变量ASCEND\_MAX\_OP\_CACHE\_SIZE来限制某个芯片下缓存文件夹的磁盘空间的大小，当编译缓存空间大小达到ASCEND\_MAX\_OP\_CACHE\_SIZE设置的取值，且需要删除旧的kernel文件时，可以通过环境变量ASCEND\_REMAIN\_CACHE\_SIZE\_RATIO设置需要保留缓存的空间大小比例。配置示例如下：

```
# ASCEND_MAX_OP_CACHE_SIZE环境变量默认值为500，单位为MB  
export ASCEND_MAX_OP_CACHE_SIZE=500  
  
# ASCEND_REMAIN_CACHE_SIZE_RATIO环境变量取值范围：[1,100]，默认值为50，单位为百分比；例如  
80表示缓存空间不足时，删除缓存，保留80%  
export ASCEND_REMAIN_CACHE_SIZE_RATIO=50
```

通过环境变量配置，只对当前用户生效。

若同时配置了op\_cache.ini文件和环境变量，则优先读取op\_cache.ini文件中的配置项，若op\_cache.ini文件和环境变量都未设置，则读取系统默认值：默认磁盘空间大小500M，默认保留缓存的空间50%。

### 7.3.4.5 --op\_compiler\_cache\_dir

#### 功能说明

用于配置算子编译磁盘缓存的路径。

#### 关联参数

如果要自行指定算子编译磁盘缓存的路径，需--op\_compiler\_cache\_dir与[7.3.4.4 --op\\_compiler\\_cache\\_mode](#)参数配合使用。

#### 参数取值

**参数值：**存放算子编译磁盘缓存的路径。

**参数值格式：**路径支持大小写字母（a-z，A-Z）、数字（0-9）、下划线（\_）、短横线（-）、句点（.）、中文汉字。

**参数值约束：**

- 如果[7.3.4.5 --op\\_compiler\\_cache\\_dir](#)参数指定的路径存在且有效，则在指定的路径下自动创建子目录kernel\_cache；如果指定的路径不存在但路径有效，则先自动创建目录，然后在该路径下自动创建子目录kernel\_cache。
- 用户请不要在**默认缓存目录**下存放其他自有内容，自有内容在软件包安装或升级时会同默认缓存目录一并被删除。
- 通过该参数指定的**非默认缓存目录**无法删除（软件包安装或升级时不会被删除）。

**参数默认值：**\$HOME/atc\_data

#### 推荐配置及收益

无。

#### 示例

```
--op_compiler_cache_dir=$HOME/atc_data --op_compiler_cache_mode=enable
```

#### 支持的型号

Atlas 200/300/500 推理产品

Atlas 推理系列产品

#### 使用约束

算子编译磁盘缓存路径，除[7.3.4.5 --op\\_compiler\\_cache\\_dir](#)参数设置的方式外，还可以配置环境变量ASCEND\_CACHE\_PATH，几种方式优先级为

配置参数“[7.3.4.5 --op\\_compiler\\_cache\\_dir](#)”>环境变量ASCEND\_CACHE\_PATH>默认存储路径。

关于环境变量ASCEND\_CACHE\_PATH的详细说明请参见《[环境变量参考](#)》。

### 7.3.4.6 --display\_model\_info

#### 功能说明

编译原始框架网络模型时，查询模型占用的关键资源信息、编译与运行环境等信息，查询出的信息直接在屏幕打印显示。

#### 关联参数

无。

#### 参数取值

参数值：

- 0：关闭查询功能。
- 1：打开查询功能。

**参数值约束：**该参数不支持单算子描述文件转离线模型时信息的查看，即不支持与 [7.2.2.11 --singleop](#) 参数同时使用。

**参数默认值：**0

#### 推荐配置及收益

无。

#### 示例

下面示例以TensorFlow框架网络模型为例进行说明：

```
atc --model=$HOME/module/resnet50_tensorflow*.pb --framework=3 --output=$HOME/module/out/tf_resnet50 --soc_version=<soc_version> --display_model_info=1
```

命令执行完毕，屏幕会打印类似如下信息：

```
===== Display Model Info start =====
# 模型转换使用的atc命令
Original Atc command line: ${INSTALL_DIR}/bin/atc.bin --model=$HOME/module/resnet50_tensorflow*.pb
--framework=3 --output=$HOME/module/out/tf_resnet50 --soc_version=<soc_version> --
display_model_info=1
# ATC软件版本信息、soc_version版本信息、原始框架信息
system info: atc_version[xxx], soc_version[xxx], framework_type[xxx].
# 运行时的占用内存、权重大小、逻辑stream数目、event数目
resource info: memory_size[xxx B], weight_size[xxx B], stream_num[xxx], event_num[xxx].
# 离线模型文件中各分区大小、包括ModelDef、权重、tbekernels、taskinfo占用的大小等
om info: modeldef_size[xxx B], weight_data_size[xxx B], tbe_kernels_size[xxx B],
cust_aicpu_kernel_store_size[xxx B], task_info_size[xxx B].
===== Display Model Info end =====
```

#### 支持的型号

Atlas 200/300/500 推理产品

Atlas 推理系列产品

#### 依赖约束

无。

### 7.3.4.7 --shape\_generalized\_build\_mode

#### 功能说明

图编译时Shape的编译方式。

#### 须知

该参数后续版本会废弃，请勿使用。

#### 关联参数

该参数不能与[7.2.2.7 --input\\_shape\\_range](#)、[7.2.2.8 --dynamic\\_batch\\_size](#)、[7.2.2.9 --dynamic\\_image\\_size](#)、[7.2.2.10 --dynamic\\_dims](#)同时使用。

#### 参数取值

##### 参数值：

- shape\_generalized：模糊编译：在编译时系统内部对可变维度做了泛化后再进行编译。如果算子Shape是固定，则可变维度会修改为-1（维度不变，例如原来Shape为4维，模糊编译后仍为4维）进行编译。

该参数使用场景为：用户想编译一次达到多次执行推理的目的时，可以使用模糊编译特性。

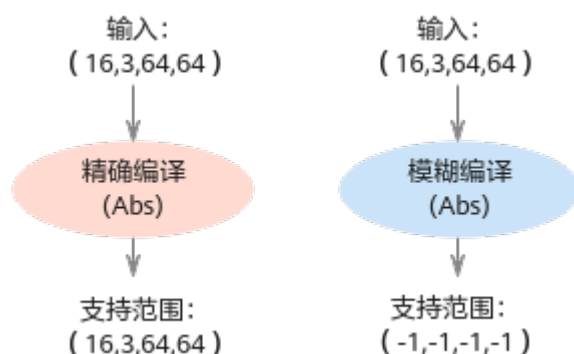
- shape\_precise：精确编译：是指按照用户指定的维度信息、在编译时系统内部不做任何转义直接编译。

**参数值约束：**如果算子本身不支持动态Shape、只支持固定Shape（无可变维度），此时按照固定Shape编译算子，不按模糊编译做泛化。

**参数默认值：**shape\_precise

[图7-4](#)为编译的两种方式。

图 7-4 编译模式



#### 推荐配置及收益

无。



## 示例

```
--shape_generalized_build_mode=shape_generalized
```

## 支持的型号

Atlas 200/300/500 推理产品

Atlas 推理系列产品

## 使用约束

如果模型转换时通过该参数设置了模糊编译，则使用应用工程进行模型推理时，需要在接口之前，增加接口，用于设置真实的shape取值

关于接口的具体使用方法，请参见《[CANN AscendCL 应用软件开发指南（C&C++）（开放态）](#)》手册“AscendCL API参考”。

### 7.3.4.8 --status\_check

## 功能说明

控制编译算子时是否添加溢出检测逻辑。

当模型计算精度有问题，并且怀疑是模型中算子有计算溢出时，模型编译时可以通过使能该参数，添加编译算子时的溢出检测逻辑，然后重新编译模型。

## 关联参数

使用该参数时，建议与[7.3.3.10 --op\\_debug\\_level](#)参数配合使用，这样在生成的算子\*.cce文件中，可以查看是否加入了溢出检测逻辑，加入了溢出检测逻辑的代码样例如下：

```
if (status_overflow[0]) {  
    xxxxxx  
}
```

## 参数取值

**参数值：**

- 0：不使能，算子编译时不添加溢出检测逻辑。
- 1：使能，算子编译时添加溢出检测逻辑。

**参数默认值：**0

## 示例

```
--status_check=1
```

## 支持的产品形态

Atlas 推理系列产品



## 使用约束

使用7.3.4.8 `--status_check`参数只是在模型编译后生成的算子\*.cce文件中加入了溢出检测逻辑, 如果想查看具体哪些算子有溢出, 则需要配合模型推理过程中的AscendCL `aclInit`接口使用, 在该接口入参的json配置文件中打开“dump\_debug”开关:

- 采集溢出算子信息

AscendCL `aclInit`接口json配置文件中的示例内容如下, 示例中的dump\_path以相对路径为例: (关于AscendCL `aclInit`接口的详细介绍请参见《[CANN AscendCL 应用软件开发指南 \(C&C++\) \(开放态\)](#)》)

```
{
  "dump":{
    "dump_path":"output",
    "dump_debug":"on"
  }
}
```

当dump\_path配置为相对路径时, 您可以在“应用可执行文件的目录/{dump\_path}”下查看导出的数据文件, 针对每个溢出算子, 会导出两个数据文件:

- 溢出算子的dump文件: 命名规则如{op\_type}.{op\_name}.{taskid}.{stream\_id}.{timestamp}, 如果op\_type、op\_name出现了“.”、“/”、“\”、空格时, 会转换为下划线表示。

用户可通过该信息知道具体出现溢出错误的算子, 并通过[解析溢出算子的dump文件](#)获取该算子的输入和输出信息。

- 算子溢出数据文件: 命名规则如OpDebug.Node\_Opdebug.{taskid}.{stream\_id}.{timestamp}, 其中taskid不是溢出算子的taskid, 用户不需要关注taskid的实际含义。

用户可通过[解析算子溢出数据文件](#)获取溢出相关信息, 包括溢出算子所在的模型、AICore的status寄存器状态等。

- 解析溢出算子的dump文件

- a. 请根据实际情况, 将{op\_type}.{op\_name}.{taskid}.{stream\_id}.{timestamp}上传到安装有Ascend-cann-toolkit开发套件包的环境。
- b. 进入解析脚本所在目录, 例如Ascend-cann-toolkit开发套件包安装目录为: /home/HwHiAiUser/Ascend/ascend-toolkit/latest。  

```
cd /home/HwHiAiUser/Ascend/ascend-toolkit/latest/tools/operator_cmp/compare
```
- c. 执行msaccucmp.py脚本, 转换dump文件为numpy文件。举例:  

```
python3 msaccucmp.py convert -d /home/HwHiAiUser/dump -out /home/HwHiAiUser/dumptonumpy -v 2
```

### 说明

-d参数支持传入单个文件, 对单个dump文件进行转换, 也支持传入目录, 对整个path下所有的dump文件进行转换。

- d. 调用Python, 转换numpy文件为txt文件。举例:

```
$ python3
>>> import numpy as np
>>> a = np.load("/home/HwHiAiUser/dumptonumpy/Pooling.pool1.1147.1589195081588018.output.0.npy")
>>> b = a.flatten()
>>> np.savetxt("/home/HwHiAiUser/dumptonumpy/Pooling.pool1.1147.1589195081588018.output.0.txt", b)
```

转换为.txt格式文件后, 维度信息、Dtype均不存在。详细的使用方法请参考numpy官网介绍。

- 解析算子溢出数据文件

由于生成的溢出数据是二进制格式，可读性较差，需要通过工具将bin文件解析为用户可读性好的json文件。

- a. 请根据实际情况，将溢出数据文件OpDebug.Node\_Opdebug.{taskid}.  
{timestamp}上传到安装有Toolkit软件包的环境。

- b. 进入解析脚本所在路径，例如Toolkit软件包安装目录为：/home/  
HwHiAiUser/Ascend/ascend-toolkit/latest。

```
cd /home/HwHiAiUser/Ascend/ascend-toolkit/latest/toolkit/tools/operator_cmp/compare
```

- c. 执行解析命令，例如：

```
python3 msaccucmp.py convert -d /home/HwHiAiUser/opdebug/  
Opdebug.Node_OpDebug.59.1597922031178434 -out /home/HwHiAiUser/result
```

关键参数：

- -d：溢出数据文件所在目录，包括文件名。
- -out：解析结果待存储目录，如果不指定，默认生成在当前目录下。

- d. 解析结果文件内容如下所示。

```
{  
  "DHA Atomic Add": {  
    "model_id": 0,  
    "stream_id": 0,  
    "task_id": 0,  
    "task_type": 0,  
    "pc_start": "0x0",  
    "para_base": "0x0",  
    "status": 0  
  },  
  "L2 Atomic Add": {  
    "model_id": 0,  
    "stream_id": 0,  
    "task_id": 0,  
    "task_type": 0,  
    "pc_start": "0x0",  
    "para_base": "0x0",  
    "status": 0  
  },  
  "AI Core": {  
    "model_id": 514,  
    "stream_id": 563,  
    "task_id": 57,  
    "task_type": 0,  
    "pc_start": "0x1008005b0000",  
    "para_base": "0x100800297000",  
    "kernel_code": "0x1008005ae000",  
    "block_idx": 1,  
    "status": 32  
  }  
}
```

参数解释：

- model\_id：标识溢出算子所在的模型id。
- stream\_id：标识溢出算子所在的streamid。
- task\_id：标识溢出算子的taskid。
- task\_type：标识溢出算子的task类型。
- pc\_start：标识溢出算子的代码程序的内存起始地址。
- para\_base：标识溢出算子的参数的内存起始地址。

- **kernel\_code**: 标识溢出算子的代码程序的内存起始地址, 和pc\_start相同。
- **block\_idx**: 标识溢出算子的blockid参数。
- **status**: AICore的status寄存器状态, 用户可以从status值分析得到具体溢出错误。status为10进制表示, 需要转换成16进制, 然后定位到具体错误。  
例如: status为272, 转换成16进制为0x00000110, 则可以判定出可能原因为0x00000010+0x00000100。
  - 0x00000008: 符号整数最小负数NEG符号位取反溢出
  - 0x00000010: 整数加法、减法、乘法或乘加操作计算有溢出
  - 0x00000020: 浮点计算有溢出
  - 0x00000080: 浮点数转无符号数的输入是负数
  - 0x00000100: FP32转FP16或32位符号整数转FP16中出现溢出
  - 0x00000400: CUBE累加出现溢出

### 7.3.4.9 --op\_debug\_config

#### 功能说明

使能Global Memory (DDR) 内存检测功能的配置文件路径及文件名。

#### 关联参数

无。

#### 参数取值

**参数值**: 配置文件路径及文件名。

**参数值格式**: 路径和文件名: 支持大小写字母 (a-z, A-Z)、数字 (0-9)、下划线 ( \_ )、短横线 ( - )、句点 ( . )、中文汉字。

**参数值约束**:

配置文件中支持配置如下选项, 多个选项使用英文逗号分隔。

- **oom**: 算子执行过程中, 检测Global Memory是否内存越界。
  - 配置该选项, 算子编译时, 在当前执行路径算子编译生成的kernel\_meta文件夹中保留.o (算子二进制文件) 和.json文件 (算子描述文件)。
  - 使用该选项后, 在算子编译过程中会加入如下的检测逻辑, 用户可以通过再使用**dump\_cce**参数, 在生成的.cce文件中查看如下的代码。

```
inline __aicore__ void CheckInvalidAccessOfDDR(XXX) {  
    if (access_offset < 0 || access_offset + access_extent > ddr_size) {  
        if (read_or_write == 1) {  
            trap(0X5A5A0001);  
        } else {  
            trap(0X5A5A0002);  
        }  
    }  
}
```

实际执行推理过程中，如果确实内存越界，会抛出“EZ9999”错误码。

- **dump\_bin**: 算子编译时，在当前执行路径算子编译生成的kernel\_meta文件夹中保留.o（算子二进制文件）和.json文件（算子描述文件）。
- **dump\_cce**: 算子编译时，在当前执行路径算子编译生成的kernel\_meta文件夹中保留算子cce文件\*.cce，以及.o（算子二进制文件）和.json文件（算子描述文件）。
- **dump\_loc**: 算子编译时，在当前执行路径算子编译生成的kernel\_meta文件夹中保留python-cce映射文件\*\_loc.json。
- **ccec\_O0**: 算子编译时，开启ccec编译器选项-O0，配置该选项不会对调试信息执行优化操作，用于后续分析AICore Error问题。
- **ccec\_g**: 算子编译时，开启ccec编译器选项-g，配置该选项会对调试信息执行优化操作，用于后续分析AICore Error问题。
- **check\_flag**: 算子执行时，检测算子内部流水线同步信号是否匹配。
  - 配置该选项，算子编译时，在当前执行路径算子编译生成的kernel\_meta文件夹中保留.o（算子二进制文件）和.json文件（算子描述文件）。
  - 使用该选项后，在算子编译过程中会加入如下的检测逻辑，用户可以通过再使用**dump\_cce**参数，在生成的.cce文件中查看如下的代码。

```
set_flag(PIPE_MTE3, PIPE_MTE2, EVENT_ID0);
set_flag(PIPE_MTE3, PIPE_MTE2, EVENT_ID1);
set_flag(PIPE_MTE3, PIPE_MTE2, EVENT_ID2);
set_flag(PIPE_MTE3, PIPE_MTE2, EVENT_ID3);
....
pipe_barrier(PIPE_MTE3);
pipe_barrier(PIPE_MTE2);
pipe_barrier(PIPE_M);
pipe_barrier(PIPE_V);
pipe_barrier(PIPE_MTE1);
pipe_barrier(PIPE_ALL);
wait_flag(PIPE_MTE3, PIPE_MTE2, EVENT_ID0);
wait_flag(PIPE_MTE3, PIPE_MTE2, EVENT_ID1);
wait_flag(PIPE_MTE3, PIPE_MTE2, EVENT_ID2);
wait_flag(PIPE_MTE3, PIPE_MTE2, EVENT_ID3);
...
```

实际执行推理过程中，如果确实存在算子内部流水线同步信号不匹配，则最终会在有问题的算子处超时报错，并终止程序，报错信息示例为：

```
Aicore kernel execute failed, ..., fault kernel_name=算子名,...
rtStreamSynchronizeWithTimeout execute failed....
```

## 说明

配置ccec编译选项（即ccec\_O0、ccec\_g选项）时，会导致算子Kernel（\*.o文件）大小增大。动态Shape场景下，由于算子编译时会遍历可能的Shape场景，因此可能会导致算子Kernel文件过大而无法进行编译，此种场景下，建议不要配置ccec编译选项。

由于算子Kernel文件过大而无法编译的报错日志示例如下：

```
message:link error ld.lld: error: InputSection too large for range extension thunk ./
kernel_meta_xxxx.o:
```

## 推荐配置及收益

无。

## 示例

假设使能Global Memory内存检测功能的配置文件名称为gm\_debug.cfg，文件内容配置示例如下：

```
op_debug_config=ccec_g.oom
```

将该文件上传到ATC工具所在服务器，例如上传到 `$HOME/module`，使用示例如下：

```
--op_debug_config=$HOME/module/gm_debug.cfg
```

## 支持的型号

Atlas 200/300/500 推理产品

Atlas 推理系列产品

## 使用约束

无。

### 7.3.4.10 --atomic\_clean\_policy

## 功能说明

是否集中清理网络中所有atomic算子（含有atomic属性的算子都是atomic算子）占用的内存。

## 关联参数

无。

## 参数取值

- 0：集中清理，默认为0。
- 1：单独清理，对网络中每一个atomic算子进行单独清理。当网络中atomic算子内存过大时建议使用此种清理方式，对降低使用内存有明显效果，但可能会导致一定的性能损耗。

## 推荐配置及收益

无。

## 示例

设置atomic算子清理策略为单独清理，使用示例如下：

```
--atomic_clean_policy=1
```

模型转换完毕，可以在日志中（[7.3.4.2 --log=debug](#)，并设置打屏环境变量或重定向到文件查看）中查看该参数的取值，若取值与命令设置的一致，则说明参数生效。例如上述命令生效的打屏日志为：

```
debug.log:3020:[DEBUG] GE(1989798,atc.bin):2022-10-24-14:27:29.100.362 [option_utils]
debug.log:35190:[INFO] GE(1989798,atc.bin):2022-10-24-14:27:34.930.191 [ops_kernel]
debug.log:243278:[INFO] FE(1989798,atc.bin):2022-10-24-14:27:55.293.930 [fe_graph_opt]
tition0_rank1_new_sub_graph1], [get_atomic_clean_policy_str[1]]."
```

## 支持的型号

Atlas 200/300/500 推理产品

Atlas 推理系列产品

## 使用约束

无。

### 7.3.4.11 --deterministic

## 功能说明

是否开启确定性计算。

默认情况下，不开启确定性计算，算子在相同的硬件和输入下，多次执行的结果可能不同。这个差异的来源，一般是因为在算子实现中，存在异步的多线程执行，会导致浮点数累加的顺序变化。当开启确定性计算功能时，算子在相同的硬件和输入下，多次执行将产生相同的输出。但启用确定性计算往往导致算子执行变慢。

## 关联参数

无。

## 参数取值

- 0：默认值，不开启确定性计算。
- 1：开启确定性计算。

## 推荐配置及收益

通常建议不开启确定性计算，因为确定性计算往往会导致算子执行变慢，进而影响性能。当发现模型多次执行结果不同，或者是进行精度调优时，可开启确定性计算，辅助模型调试、调优。

## 示例

```
--deterministic=1
```

## 支持的型号

Atlas 推理系列产品

## 依赖约束

无

# 8 专题

[定制网络修改 \(Caffe\)](#)

[定制网络修改 \(TensorFlow\)](#)

## 8.1 定制网络修改 (Caffe)

### 8.1.1 简介

本章节修改只适用于Caffe网络模型。

网络的算子可以分为如下几类：

- 标准算子：昇腾AI处理器支持的Caffe标准算子，比如Convolution等。
- 扩展算子：昇腾AI处理器支持的公开但非Caffe标准算子，分为 2 种：
  - 一种是基于Caffe框架进行自定义扩展的算子，比如Faster RCNN中的ROI Pooling、SSD中的归一化算子Normalize等。
  - 另外一种是来源于其他深度学习框架的自定义算子，比如YOLOv2中Passthrough等。

Faster RCNN、SSD等网络模型都包含了一些原始Caffe框架中没有定义的算子结构，如ROI Pooling、Normalize、PSROI Pooling和Upsample等。为了使昇腾AI处理器能支持这些网络，需要对原始的Caffe框架网络模型进行扩展，降低开发者开发自定义算子/开发后处理代码的工作量。若开发者的Caffe框架网络模型中使用了这些扩展算子，在进行模型转换时，需要先在prototxt中修改/添加扩展层的定义，才能成功进行模型转换。

本章节提供了昇腾AI处理器的扩展算子列表，并给出了如何根据扩展算子修改prototxt文件方法。

8.1.2 扩展算子列表

表 8-1 扩展算子列表

分类	算子类型	说明
过程算子	Proposal	用于Faster R-CNN，根据rpn_cls_prob的foreground，rpn_bbox_pred中的bounding box regression修正anchors获得精确的proposals。
	ROI Pooling	用于Faster R-CNN中，对Roi（Region of interest）进行池化操作，主要用于目标检测任务。
	PSROI Pooling	用于R-FCN中，进行位置敏感的候选区域池化操作，主要用于目标检测任务。
过程算子	Reverse	将输入tensor指定的轴进行反转。
	Upsample	使用pooling mask的上采样。用于yolo网络。
	Normallize	将SSD网络中Channel维度中的元素在L2进行归一化。
	Reorg	Reorg layer in Darknet，将通道数据转移到平面上，或反过来操作。 对应算子规格中的PassThrough算子。
	ROI Align	从feature map中获取ROI（range of interest）的特征矩阵。
	ShuffleChannel	把channel维度分为[group, channel/Group]，再在channel维度中进行数据转置[channel/Group,group]。
	Yolo（Yolo/Detection/Region）	Yolo/Detection/Region算子，都需要替换为Yolo算子，对卷积网络输出的feature map生成检测框的坐标信息，置信度信息及类别概率。
	PriorBox	用于SSD网络，根据输入参数，生成prior box。
	SpatialTransformer	用于仿射变换。



分类	算子类型	说明
后处理算子	YoloV3DetectionOutput	此算子用于YOLOv3的后处理过程，对卷积网络输出的feature map生成检测框的坐标信息，置信度信息及类别概率。
	YoloV2DetectionOutput	此算子用于YOLOv2的后处理过程，对卷积网络输出的feature map生成检测框的坐标信息，置信度信息及类别概率。
	SSDDetectionOutput	此算子用于SSD网络的后处理过程，用于整合预选框、预选框偏移以及得分三项结果，最终输出满足条件的目标检测框、目标的label和得分。
	FSRDetectionOutput	此算子用于Faster R-CNN网络的后处理过程，对结果进行分类，并对每个类输出最终的bbox数量、坐标、类别概率及类别索引。

### 8.1.3 扩展算子规则

扩展算子的实现可以在Caffe、TensorFlow、MindSpore深度学习框架中实现，扩展算子中的第一种是基于Caffe框架进行扩展实现的，如ROI Pooling、PSROI Pooling、Normalize和Upsample层等，因此有公开的prototxt标准定义。而扩展算子中的第二种并不是在Caffe框架下实现的，对于这类网络中的自定义算子，也需要给出prototxt的标准定义，用于在prototxt中定义网络中相应的算子。

#### Reverse

在指定的轴上反序，如输入为[1,2,3]，其反序为[3,2,1]。

算子定义如下：

1. 在LayerParameter中添加ReverseParameter

```
message LayerParameter {  
  ...  
  optional ReverseParameter reverse_param = 157;  
  ...  
}
```

2. 定义ReverseParameter类型以及属性参数

```
message ReverseParameter{  
  repeated int32 axis = 1;  
}
```

## ROI Pooling

在目标检测算法中，region proposal产生的ROI大小不一，而分类网络的FC层要固定的输入，所以ROI Pooling起到一个连接作用。

ROI Pooling层为Faster RCNN网络中用于将不同图像经过卷积层后得到的feature map进行维度统一的操作，输入为一个feature map以及需要从中提取的ROI框坐标值，输出为一个维度归一化的feature map。

ROI Pooling层需要对caffe.proto文件进行扩展，定义ROI PoolingParameter，扩展方式如下所示，定义的参数包括：

- spatial\_scale，即输入feature map与原始图片的尺寸比例，用于转换ROI的坐标，因为ROI坐标是在原图尺寸上的。
- pooled\_h、pooled\_w，输出feature map在spatial维度上h和w的大小。

1. 在LayerParameter中添加ROI PoolingParameter

```
message LayerParameter {  
    ...  
    optional ROI PoolingParameter roi_pooling_param = 161;  
    ...  
}
```

2. 定义ROI PoolingParameter类型以及属性参数

```
message ROI PoolingParameter {  
    required int32 pooled_h = 1;  
    required int32 pooled_w = 2;  
    optional float spatial_scale = 3 [default=0.0625];  
    optional float spatial_scale_h = 4;  
    optional float spatial_scale_w = 5;  
}
```

基于上述原则，ROI Pooling层在prototxt中定义的代码样例如下：

```
layer {  
    name: "roi_pooling"  
    type: "ROI Pooling"  
    bottom: "res4f"  
    bottom: "rois"  
    bottom: "actual_rois_num"  
    top: "roi_pool"  
    roi_pooling_param {  
        pooled_h: 14  
        pooled_w: 14  
        spatial_scale: 0.0625  
        spatial_scale_h: 0.0625  
        spatial_scale_w: 0.0625  
    }  
}
```

## PSROI Pooling

PSROI Pooling层的操作与ROI Pooling层类似，不同之处在于不同空间维度输出的图片特征来自不同的feature map channels，且对每个小区域进行的是Average Pooling，不同于ROI Pooling的Max Pooling。

对于一个输出  $k \times k$  的结果，不同空间维度的特征取自输入feature map中不同的组，即将输入的feature map均匀分为 $k \times k$ 组，每组的channel数与输出的channel一致，得到上述输出。

PSROI Pooling层需要对caffe.proto文件进行扩展，定义PSROI PoolingParameter，扩展方式如下所示，定义的参数包括：

- spatial\_scale, 即输入feature map与原始图片的尺寸比例。
- output\_dim, 输出feature map的channel。
- group\_size, 输出的spatial维度, 即上述的k。

1. 在LayerParameter中添加PSROIPoolingParameter

```
message LayerParameter {  
  ...  
  optional PSROIPoolingParameter psroi_pooling_param = 207;  
  ...  
}
```

2. 定义PSROIPoolingParameter类型以及属性参数

```
message PSROIPoolingParameter {  
  required float spatial_scale = 1;  
  required int32 output_dim = 2; // output channel number  
  required int32 group_size = 3; // number of groups to encode position-sensitive score maps  
}
```

基于上述原则, PSROIPooling层在prototxt中定义的代码样例如下:

```
layer {  
  name: "psroipooling"  
  type: "PSROIPooling"  
  bottom: "some_input"  
  bottom: "some_input"  
  top: "some_output"  
  psroi_pooling_param {  
    spatial_scale: 0.0625  
    output_dim: 21  
    group_size: 7  
  }  
}
```

## Upsample

Upsample层为Pooling层的逆操作, 其中每个Upsample层均与网络之前一个对应大小输入、输出Pooling层一一对应, 完成feature map在spatial维度上的扩充。

Upsample层需要对caffe.proto文件进行扩展, 定义UpsampleParameter, 扩展方式示例如下所示。定义的参数包括stride, 即输出与输入的尺寸比例, 如2。

1. 在LayerParameter中添加UpsampleParameter

```
message LayerParameter {  
  ...  
  optional UpsampleParameter upsample_param = 160;  
  ...  
}
```

2. 定义UpsampleParameter类型以及属性参数

```
message UpsampleParameter {  
  optional float scale = 1 [default = 1];  
  optional int32 stride = 2 [default = 2];  
  optional int32 stride_h = 3 [default = 2];  
  optional int32 stride_w = 4 [default = 2];  
}
```

基于上述原则, Upsample在prototxt中定义的代码样例如下:

```
layer {  
  name: "layer86-upsample"  
  type: "Upsample"  
  bottom: "some_input"  
  top: "some_output"  
  upsample_param {  
    scale: 1  
    stride: 2  
  }  
}
```

```
}  
}
```

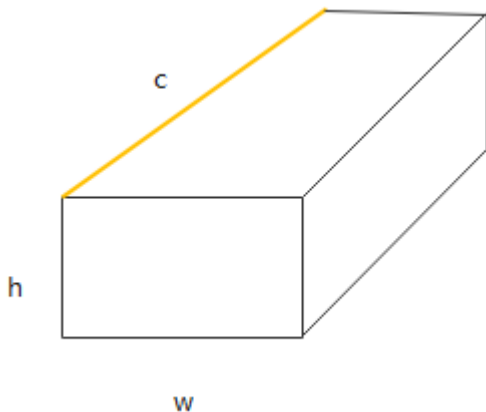
## NormalIize

Normalize层为SSD网络中的一个归一化层，主要作用是将空间或者通道内的元素归一化到0到1之间，其进行的操作为对于一个c\*h\*w的三维tensor，输出是同样大小的tensor，其中间计算为每个元素以channel方向的平方和的平方根求 normalize，其具体计算公式为：

$$x'_i = \frac{x_i}{\sqrt{\sum_{j=1}^c x_j^2}}$$

其中分母位置的平方和的累加向量为同一h与w位置的所有c方向的向量，如[图8-1](#)中的橙色区域。

图 8-1 Normalize 层原理图



经过上述计算归一化后，再对每个feature map做scale，每个channel对应一个scale值。

Normalize层需要对caffe.proto文件进行扩展，定义NormalizeParameter，扩展方式如下所示。定义的参数包括：

- across\_spatial参数表示是否对整个图片进行归一化，归一化的维度为：1 x c x h x w，否则对每个像素点进行归一化：1 x c x 1 x 1。
- channels\_shared表示scale是否相同，如果为true，则scale都是一样的，否则对于channel一样，对不同channel像素点是不一样的，默认为True。
- eps防止normalize过程中除以0的一个很小数值，默认为1e-10，可配置。

normalize的计算公式转换为：

$$x_i = \frac{x_i}{\left(\sum_1^n x_i^2 + \text{eps}\right)^{\frac{1}{2}}} \times \text{scale}_i$$

算子定义如下：

## 1. 在LayerParameter中添加NormalizeParameter

```
message LayerParameter {  
    ...  
    optional NormalizeParameter norm_param = 206;  
    ...  
}
```

## 2. 定义NormalizeParameter类型以及属性参数

```
message NormalizeParameter {  
    optional bool across_spatial = 1 [default = true];  
    // Initial value of scale. Default is 1.0 for all  
    optional FillerParameter scale_filler = 2;  
    // Whether or not scale parameters are shared across channels.  
    optional bool channel_shared = 3 [default = true];  
    // Epsilon for not dividing by zero while normalizing variance  
    optional float eps = 4 [default = 1e-10];  
}
```

基于上述原则，Normalize在prototxt中定义的代码样例如下：

```
layer {  
    name: "normalize_layer"  
    type: "Normalize"  
    bottom: ""some_input"  
    top: "some_output"  
    norm_param {  
        across_spatial: false  
        scale_filler {  
            type: "constant"  
            value: 20  
        }  
        channel_shared: false  
    }  
}
```

## Reorg

Reorg算子在昇腾AI处理器内部以PassThrough算子呈现，将通道数据转移到平面上，或反过来操作。

PassThrough层为Yolo v2中的一个自定义层，由于Yolo v2并不是使用Caffe框架实现，因此对于该层没有标准的定义。该层实现的功能为将feature map在spatial维度上的数据展开到channel维度上，原始在channel维度上连续的元素在展开后的feature map中依然是连续的。

算子定义如下：

## 1. 在LayerParameter中添加ReorgParameter

```
message LayerParameter {  
    ...  
    optional ReorgParameter reorg_param = 155;  
    ...  
}
```

## 2. 定义ReorgParameter类型以及属性参数

```
message ReorgParameter {  
    optional uint32 stride = 2 [default = 2];  
    optional bool reverse = 1 [default = false];  
}
```

基于上述原则，Reorg在prototxt中定义的代码样例如下：

```
layer {  
    bottom: "some_input"  
    top: "some_output"  
    name: "reorg"
```

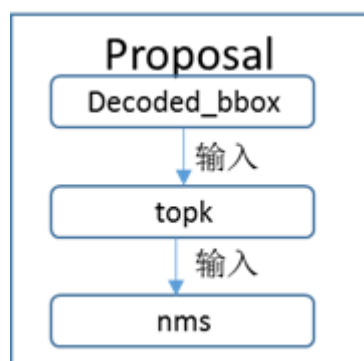
```
type: "Reorg"
reorg_param {
  stride: 2
}
```

## Proposal

Proposal算子根据rpn\_cls\_prob的foreground，rpn\_bbox\_pred中的bounding box regression修正anchors获得精确的proposals。

具体可以分为3个算子decoded\_bbox、topk和nms，实现如图8-2所示。

图 8-2 Proposal 算子实现



算子定义如下：

1. 在LayerParameter中添加ProposalParameter

```
message LayerParameter {
  ...
  optional ProposalParameter proposal_param = 201;
  ...
}
```

2. 定义ProposalParameter类型以及属性参数

```
message ProposalParameter {
  optional float feat_stride = 1 [default = 16];
  optional float base_size = 2 [default = 16];
  optional float min_size = 3 [default = 16];
  repeated float ratio = 4;
  repeated float scale = 5;
  optional int32 pre_nms_topn = 6 [default = 3000];
  optional int32 post_nms_topn = 7 [default = 304];
  optional float iou_threshold = 8 [default = 0.7];
  optional bool output_actual_rois_num = 9 [default = false];
}
```

基于上述原则，Proposal在prototxt中定义的代码样例如下：

```
layer {
  name: "faster_rcnn_proposal"
  type: "Proposal"           // 算子Type

  bottom: "rpn_cls_prob_reshape"
  bottom: "rpn_bbox_pred"
  bottom: "im_info"
  top: "rois"
  top: "actual_rois_num"     // 增加的算子输出
  proposal_param {
```

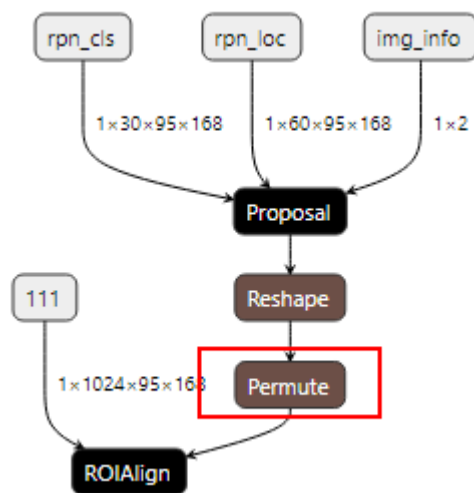
```
feat_stride: 16
base_size: 16
min_size: 16
pre_nms_topn: 3000
post_nms_topn: 304
iou_threshold: 0.7
output_actual_rois_num: true
}
```

### 说明

如果用户的网络模型中存在“Proposal+ROIAlign+业务算子”相关的网络结构，由于Proposal算子的输出为3维，而ROIAlign算子的rois是2维，因此需要在Proposal算子后面增加Reshape算子，用来改变Tensor shape，然后作为ROIAlign的输入。但该场景下输入给ROIAlign算子的rois数据，坐标排序混乱，不符合ROIAlign算子的要求。

基于上述问题，需要在Reshape算子后再增加Permute算子，进行转置之后，再次输出的数据符合ROIAlign算子的要求。

修改后的网络结构样例如下：



对应修改后的prototxt中代码样例如下：

```
layer {
  name: 'proposal'
  type: 'Proposal'
  bottom: 'rpn_cls'
  bottom: 'rpn_loc'
  bottom: 'img_info'
  top: 'roi_proposal'
  proposal_param {
    feat_stride: 16
    pre_nms_topn: 1000
    post_nms_topn: 16
    nms_thresh: 0.7
    base_size: 16
    min_size: 8
    ratio: [0.5, 1.0, 2.0]
    scale: [32, 64, 128, 256, 512]
  }
}
layer {
  name: "Reshape1"
  type: "Reshape"
  bottom: "roi_proposal"
  top: "roi_proposal_reshape"
  reshape_param {
    shape {
```

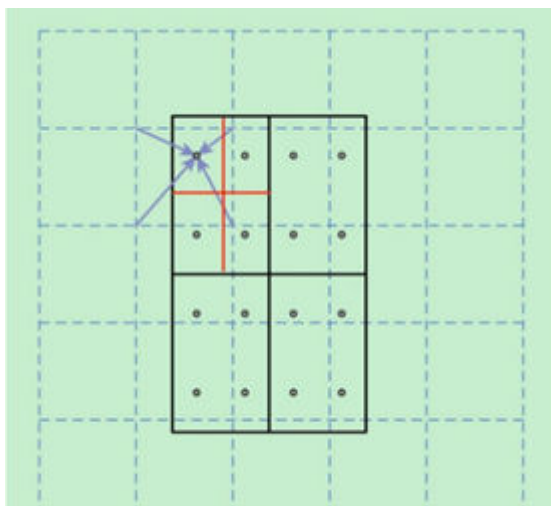
```
    dim: 5
    dim: 16
  }
}
layer {
  name: "Permute1"
  type: "Permute"
  bottom: "roi_proposal_reshape"
  top: "roi_proposal_permute"
  permute_param {
    order: 1
    order: 0
  }
}
layer {
  name: "align"
  type: "ROIAlign"
  bottom: "111"
  bottom: "roi_proposal_permute"
  top: "align"
  roi_align_param {
    pooled_w: 14
    pooled_h: 14
    spatial_scale: 0.0625
  }
}
```

## ROIAlign

ROIAlign是在Mask-RCNN论文里提出的一种区域特征聚集方式, 与ROI Pooling算法进行改进: 用双线性插值替换ROI Pooling操作中两次量化, 以解决ROI Pooling造成的区域不匹配的问题, 提高检测准确性。

ROIAlign算子是从feature map中获取ROI ( range of interest ), 分为pooled\_w x pooled\_h 个单元格, 每个单元格均分为sampling\_ratio\*sampling\_ratio个小方格, 每个小方格的中心点就是采样点。如图8-3所示, 虚线部分表示feature map, 实线表示ROI, 这里将ROI切分成2x2的单元格。如果采样点数是4, 则首先将每个单元格子均分成四个小方格 ( 如红色线所示 ), 每个小方格中心就是采样点。由于采样点的坐标通常是浮点数, 所以需要对采样点像素进行双线性插值 ( 如图8-3中的四个箭头所示 ), 就可以得到该像素点的值了。然后对每个单元格内的四个采样点取均值, 就可以得到最终的ROIAlign的结果。

图 8-3 ROIAlign 原理图





算子定义如下：

1. 在LayerParameter中添加ROIAlignParameter

```
message LayerParameter {  
    ...  
    optional ROIAlignParameter roi_align_param = 154;  
    ...  
}
```

2. 定义ROIAlignParameter类型以及属性参数

```
message ROIAlignParameter {  
    // Pad, kernel size, and stride are all given as a single value for equal  
    // dimensions in height and width or as Y, X pairs.  
    optional uint32 pooled_h = 1 [default = 0]; // The pooled output height  
    optional uint32 pooled_w = 2 [default = 0]; // The pooled output width  
    // Multiplicative spatial scale factor to translate ROI coords from their  
    // input scale to the scale used when pooling  
    optional float spatial_scale = 3 [default = 1];  
    optional int32 sampling_ratio = 4 [default = -1];  
    optional int32 roi_end_mode = 5 [default = 0];  
}
```

根据上述类型以及属性用户可以自定义prototxt。

## ShuffleChannel

ShuffleChannel是把channel维度分为[group, channel/Group]，然后再在channel维度中进行数据转置[channel/Group,group]。

例如对于channel=4，group=2，则执行此算子后，就是把channel[1]、channel[2]的数据交换位置。

算子定义如下：

1. 在LayerParameter中添加ShuffleChannelParameter

```
message LayerParameter {  
    ...  
    optional ShuffleChannelParameter shuffle_channel_param = 159;  
    ...  
}
```

2. 定义ShuffleChannelParameter类型以及属性参数

```
message ShuffleChannelParameter {  
    optional uint32 group = 1 [default = 1]; // The number of group  
}
```

基于上述原则，ShuffleChannel在prototxt中定义的代码样例如下：

```
layer {  
    name: "layer_shuffle"  
    type: "ShuffleChannel"  
    bottom: "some_input"  
    top: "some_output"  
    shuffle_channel_param {  
        group: 3  
    }  
}
```

## Yolo

YOLO算子出现在YOLO V2网络，且目前仅在YOLO V2、V3网络中使用，对数据做sigmoid和softmax操作。

- 在YOLO V2中，根据background和softmax的参数，有4种场景：

- a. background=false, softmax=true,  
对(x,y,h,w)中的(x,y)做sigmoid, 对b做sigmoid, 对classes做softmax。
  - b. background=false, softmax=false,  
对(x,y,h,w)中的(x,y)做sigmoid, 对b做sigmoid, 对classes做sigmoid。
  - c. background=true, softmax=false,  
对(x,y,h,w)中的(x,y)做sigmoid, 对b不做计算, 对classes做sigmoid。
  - d. background=true, softmax= true,  
对(x,y,h,w)中的(x,y)做sigmoid, 对b和classes放在一起做softmax。
- 在YOLO V3中, 只有一种场景: 对(x,y,h,w)中的(x,y)做sigmoid, 对b做sigmoid, 对classes做sigmoid。

输入数据格式为Tensor(n, coords+backgroup+classes,l,h,l.w), 其中n是anchor box的数量, coords表示x,y,w,h。

算子定义如下:

1. 在LayerParameter中添加YoloParameter

```
message LayerParameter {  
    ...  
    optional YoloParameter yolo_param = 199;  
    ...  
}
```

2. 定义YoloParameter类型以及属性参数

```
message YoloParameter {  
    optional int32 boxes = 1 [default = 3];  
    optional int32 coords = 2 [default = 4];  
    optional int32 classes = 3 [default = 80];  
    optional string yolo_version = 4 [default = "V3"];  
    optional bool softmax = 5 [default = false];  
    optional bool background = 6 [default = false];  
    optional bool softmaxtree = 7 [default = false];  
}
```

基于上述原则, Yolo在prototxt中定义的代码样例如下:

```
layer {  
    bottom: "layer82-conv"  
    top: "yolo1_coords"  
    top: "yolo1_obj"  
    top: "yolo1_classes"  
    name: "yolo1"  
    type: "Yolo"  
    yolo_param {  
        boxes: 3  
        coords: 4  
        classes: 80  
        yolo_version: "V3"  
        softmax: true  
        background: false  
    }  
}
```

## PriorBox

根据输入的参数, 生成prior box。

下面以conv7\_2\_mbox\_priorbox为例, 根据对应的参数生成prior box的数量。定义如下:

```
layer{  
    name:"conv7_2_mbox_priorbox"
```

```
type:"PriorBox"
bottom:"conv7_2"
bottom:"data"
top:"conv7_2_mbox_priorbox"
prior_box_param{
  min_size:162.0
  max_size:213.0
  aspect_ratio:2
  aspect_ratio:3
  flip:true
  clip:false
  variance:0.1
  variance:0.1
  variance:0.2
  variance:0.2
  img_size:300
  step:64
  offset:0.5
}
```

1. 宽高都为minsize生成prior box。
2. 如果存在max\_size, 则用 $\sqrt{\min\_size \times \max\_size}$ 确定宽高生成一个框(约束,  $\max\_size > \min\_size$ )。
3. 根据aspect\_ratio(如定义所示aspect\_ratio为2, 3, flip是true, 自动添加 aspect\_ratio=1/2、1/3), 生成对应的prior box。

因此, num\_priors(prior box的数量)= min\_size的数量+aspect\_ratio的数量 ( 这里为4 ) \*min\_size的数量 ( 这里为1 ) +max\_size的数量 ( max\_size的数量和min\_size 的数量一一对应 )。

算子定义如下:

1. 在LayerParameter中添加PriorBoxParameter

```
message LayerParameter {
  ...
  optional PriorBoxParameter prior_box_param = 203;
  ...
}
```

2. 定义PriorBoxParameter类型以及属性参数

```
message PriorBoxParameter {
  // Encode/decode type.
  enum CodeType {
    CORNER = 1;
    CENTER_SIZE = 2;
    CORNER_SIZE = 3;
  }
  // Minimum box size (in pixels). Required!
  repeated float min_size = 1;
  // Maximum box size (in pixels). Required!
  repeated float max_size = 2;
  // Various of aspect ratios. Duplicate ratios will be ignored.
  // If none is provided, we use default ratio 1.
  repeated float aspect_ratio = 3;
  // If true, will flip each aspect ratio.
  // For example, if there is aspect ratio "r",
  // we will generate aspect ratio "1.0/r" as well.
  optional bool flip = 4 [default = true];
  // If true, will clip the prior so that it is within [0, 1]
  optional bool clip = 5 [default = false];
  // Variance for adjusting the prior bboxes.
  repeated float variance = 6;
  // By default, we calculate img_height, img_width, step_x, step_y based on
  // bottom[0] (feat) and bottom[1] (img). Unless these values are explicitly
  // provided.
  // Explicitly provide the img_size.
```

```
optional uint32 img_size = 7;
// Either img_size or img_h/img_w should be specified; not both.
optional uint32 img_h = 8;
optional uint32 img_w = 9;

// Explicitly provide the step size.
optional float step = 10;
// Either step or step_h/step_w should be specified; not both.
optional float step_h = 11;
optional float step_w = 12;

// Offset to the top left corner of each cell.
optional float offset = 13 [default = 0.5];
}
```

基于上述原则，PriorBox在prototxt中定义的代码样例如下：

```
layer {
  name: "layer_priorbox"
  type: "PriorBox"
  bottom: "some_input"
  bottom: "some_input"
  top: "some_output"
  prior_box_param {
    min_size: 30.0
    max_size: 60.0
    aspect_ratio: 2
    flip: true
    clip: false
    variance: 0.1
    variance: 0.1
    variance: 0.2
    variance: 0.2
    step: 8
    offset: 0.5
  }
}
```

## SpatialTransformer

该算子计算过程其实做了一个仿射变换：仿射变换的参数，可以是在prototxt固定的，多个batch使用一份；也可以是层的第二个输入，每个batch使用不一样的参数。

计算步骤：

1. 对于输出坐标，通过如下公式将其变换成[-1,1]区间中的值。

$$x' = x \times \frac{1.0}{out_h} \times 2 - 1$$

$$y' = y \times \frac{1.0}{out_w} \times 2 - 1$$

计算代码如下：

```
Dtype* data = output_grid.mutable_cpu_data();
for(int i=0; i< output_H_ * output_W_; ++i) {
  data[3 * i] = (i / output_W_) * 1.0 / output_H_ * 2 - 1;
  data[3 * i + 1] = (i % output_W_) * 1.0 / output_W_ * 2 - 1;
  data[3 * i + 2] = 1;
}
```

2. 通过仿射变换，转换为输入的坐标，其中s为输入坐标，t为输出坐标，计算公式如下：

$$\begin{pmatrix} x_i^s \\ y_i^s \end{pmatrix} = \begin{bmatrix} \theta_{11} & \theta_{12} & \theta_{13} \\ \theta_{21} & \theta_{22} & \theta_{23} \end{bmatrix} \begin{pmatrix} x_i^t \\ y_i^t \\ 1 \end{pmatrix}$$

计算代码如下：

```
caffe_cpu_gemm<Dtype>(CblasNoTrans, CblasTrans, output_H_ * output_W_, 2, 3, (Dtype)1.,  
output_grid_data, full_theta_data + 6 * i, (Dtype)0., coordinates);
```

3. 通过输入坐标取对应位置的值，赋值给输出的对应位置。

因为在第一步对输出坐标做了一次转换，所以对应的输入坐标，使用同样的方式再缩放回去，代码样例如下：

```
Dtype x = (px + 1) / 2 * H;  
Dtype y = (py + 1) / 2 * W;  
if(debug) std::cout<<prefix<<"(x, y) = ("<<x<< ", "<<y<<")"<<std::endl;  
  
for(int m = floor(x); m <= ceil(x); ++m)  
for(int n = floor(y); n <= ceil(y); ++n) {  
    if(debug) std::cout<<prefix<<"(m, n) = ("<<m<< ", "<<n<<")"<<std::endl;  
    if(m >= 0 && m < H && n >= 0 && n < W) {  
        res += (1 - abs(x - m)) * (1 - abs(y - n)) * pic[m * W + n];  
        if(debug) std::cout<<prefix<<" pic["<<m * W + n<<"] = "<<std::endl;  
    }  
}
```

算子定义如下：

1. 在LayerParameter中添加SpatialTransformParameter

```
message LayerParameter {  
    ...  
    optional SpatialTransformParameter spatial_transform_param = 153;  
    ...  
}
```

2. 定义SpatialTransformParameter类型以及属性参数

```
message SpatialTransformParameter {  
    optional uint32 output_h = 1 [default = 0];  
    optional uint32 output_w = 2 [default = 0];  
    optional float border_value = 3 [default = 0];  
    repeated float affine_transform = 4;  
    enum Engine {  
        DEFAULT = 0;  
        CAFFE = 1;  
        CUDNN = 2;  
    }  
    optional Engine engine = 15 [default = DEFAULT];  
}
```

基于上述原则，SpatialTransform层在prototxt中定义的代码样例如下：

```
layer {  
    name: "st_1"  
    type: "SpatialTransformer"  
    bottom: "data"  
    bottom: "theta"  
    top: "transformed"  
    st_param {  
        to_compute_dU: false  
        theta_1_1: -0.129  
        theta_1_2: 0.626  
        theta_2_1: 0.344  
        theta_2_2: 0.157  
    }  
}
```

## 8.1.4 样例参考

本章节介绍几种常用网络的修改方法。

### 8.1.4.1 FasterRCNN 网络模型 prototxt 修改

#### 说明

本章节所有的代码样例都不能直接复制到网络模型中使用，需要用户根据使用的网络模型，自行调整相应参数，比如bottom、top中的参数要和具体网络模型中的bottom、top——对应，并且bottom和top对应的参数顺序不能更改。

如下以FasterRCNN Resnet34网络模型为例进行说明。

1. proposal算子修改。

根据《[算子清单（开放态）](#)》>AI框架算子支持清单>支持Caffe算子清单，该算子有3个输入，2个输出；根据上述原则，对原始proposal算子进行修改，type修改为caffe.proto文件中的类型，增加actual\_rois\_num输出节点。参见caffe.proto文件中的属性定义增加相应属性信息。修改情况如[图8-4](#)所示，其中左边为原始算子prototxt，右边是适配昇腾AI处理器的prototxt。

图 8-4 原始算子与修改后的算子比对 1



代码样例如下：

```
layer {
  name: "faster_rcnn_proposal"
  type: "Proposal" // 算子Type

  bottom: "rpn_cls_prob_reshape"
  bottom: "rpn_bbox_pred"
  bottom: "im_info"
  top: "rois"
  top: "actual_rois_num" // 增加的算子输出
  proposal_param {

    feat_stride: 16
    base_size: 16
    min_size: 16
    pre_nms_topn: 3000
    post_nms_topn: 304
    iou_threshold: 0.7
    output_actual_rois_num: true
  }
}
```

参数解释请参见《[算子清单（开放态）](#)》>AI框架算子支持清单>支持Caffe算子清单。

2. 最后一层增加FSRDetectionOutput算子，用于输出最终的检测结果。

对于FasterRCNN网络，参考[8.1.2 扩展算子列表](#)在原始prototxt文件的最后增加后处理算子层FSRDetectionOutput，参见《[算子清单（开放态）](#)》>AI框架算子支持清单>支持Caffe算子清单，FSRDetectionOutput算子有五个输入，两个输出，此算子的Type及属性定义如下：

代码样例如下：

```
layer {
  name: "FSRDetectionOutput_1"
  type: "FSRDetectionOutput"

  bottom: "rois"
  bottom: "bbox_pred"
  bottom: "cls_prob"
  bottom: "im_info"
  bottom: "actual_rois_num"
  top: "actual_bbox_num1"
  top: "box1"

  fsrdetectionoutput_param {
    num_classes:3
    score_threshold:0.0
    iou_threshold:0.7
    batch_rois:1
  }
}
```

参数解释请参见《[算子清单（开放态）](#)》>AI框架算子支持清单>支持Caffe算子清单。

### 8.1.4.2 YOLOv3 网络模型 prototxt 修改

#### 说明

本章节所有的代码样例都不能直接复制到网络模型中使用，需要用户根据使用的网络模型，自行调整相应参数，比如bottom、top中的参数要和具体网络模型中的bottom、top——对应，并且bottom和top对应的参数顺序不能更改。

1. upsample算子upsample\_param属性参数修改。

参见《[算子清单（开放态）](#)》>AI框架算子支持清单>支持Caffe算子清单，需要将原始算子prototxt中的scale:2修改为scale:1 stride:2。

对比情况如[图8-5](#)所示，其中左边为原始算子prototxt，右边是适配昇腾AI处理器的prototxt。

图 8-5 原始算子与修改后的算子比对 4

```
layer {
  bottom: "layer85-conv"
  top: "layer86-upsample"
  name: "layer86-upsample"
  type: "Upsample"
  upsample_param {
    scale: 2
  }
}

layer {
  bottom: "layer85-conv"
  top: "layer86-upsample"
  name: "layer86-upsample"
  type: "Upsample"
  upsample_param {
    scale: 1
    stride: 2
  }
}
```

参数解释请参见《[算子清单（开放态）](#)》>AI框架算子支持清单>支持Caffe算子清单。

2. 新增三个Yolo算子。

由于Yolo算子+DetectionOutput算子才是特征检测网络完整的后处理逻辑，根据原始算子prototxt所示，在增加YoloV3DetectionOutput算子之前需要先增加三个Yolo算子。

根据《[算子清单（开放态）](#)》>AI框架算子支持清单>支持Caffe算子清单，该算子有1个输入，3个输出，根据上述原则，构造的Yolo算子代码样例如下：

- 算子1代码样例：

```
layer {
  bottom: "layer82-conv"
  top: "yolo1_coords"
  top: "yolo1_obj"
  top: "yolo1_classes"
  name: "yolo1"
  type: "Yolo"
  yolo_param {
    boxes: 3
    coords: 4
    classes: 80
    yolo_version: "V3"
    softmax: true
    background: false
  }
}
```

- 算子2代码样例：

```
layer {
  bottom: "layer94-conv"
  top: "yolo2_coords"
  top: "yolo2_obj"
  top: "yolo2_classes"
  name: "yolo2"
  type: "Yolo"
  yolo_param {
    boxes: 3
    coords: 4
    classes: 80
    yolo_version: "V3"
    softmax: true
    background: false
  }
}
```

- 算子3代码样例：

```
layer {
  bottom: "layer106-conv"
  top: "yolo3_coords"
  top: "yolo3_obj"
  top: "yolo3_classes"
  name: "yolo3"
  type: "Yolo"
  yolo_param {
    boxes: 3
    coords: 4
    classes: 80
    yolo_version: "V3"
    softmax: true
    background: false
  }
}
```

参数解释请参见《[算子清单（开放态）](#)》>AI框架算子支持清单>支持Caffe算子清单。

3. 最后一层增加YoloV3DetectionOutput算子。

对于YOLOv3网络，参考[8.1.2 扩展算子列表](#)在原始prototxt文件的最后增加后处理算子层YoloV3DetectionOutput，参见《[算子清单（开放态）](#)》>AI框架算子支持清单>支持Caffe算子清单，YoloV3DetectionOutput算子有十个输入，两个输出；根据该原则，构造的后处理算子代码样例如下：

```
layer {
  name: "detection_out3"
  type: "YoloV3DetectionOutput"
```



```
bottom: "yolo1_coords"
bottom: "yolo2_coords"
bottom: "yolo3_coords"
bottom: "yolo1_obj"
bottom: "yolo2_obj"
bottom: "yolo3_obj"
bottom: "yolo1_classes"
bottom: "yolo2_classes"
bottom: "yolo3_classes"
bottom: "img_info"
top: "box_out"
top: "box_out_num"
yolov3_detection_output_param {
    boxes: 3
    classes: 80
    relative: true
    obj_threshold: 0.5
    score_threshold: 0.5
    iou_threshold: 0.45
    pre_nms_topn: 512
    post_nms_topn: 1024
    biases_high: 10
    biases_high: 13
    biases_high: 16
    biases_high: 30
    biases_high: 33
    biases_high: 23
    biases_mid: 30
    biases_mid: 61
    biases_mid: 62
    biases_mid: 45
    biases_mid: 59
    biases_mid: 119
    biases_low: 116
    biases_low: 90
    biases_low: 156
    biases_low: 198
    biases_low: 373
    biases_low: 326
}
```

参数解释请参见《[算子清单（开放态）](#)》>AI框架算子支持清单>支持Caffe算子清单。

#### 4. 新增输入：

由于YoloV3DetectionOutput算子有img\_info输入，故模型输入时增加该输入。对比情况如图8-6所示，其中左边为原始算子prototxt，右边是适配昇腾AI处理器的prototxt。

图 8-6 原始算子与修改后的算子比对 3

```
layer {
  name: "Input_1"
  type: "Input"
  top: "data"
  input_param {
    shape {
      dim: 1
      dim: 3
      dim: 416
      dim: 416
    }
  }
}
```

```
input: "data"
input_shape {
  dim: 1
  dim: 3
  dim: 416
  dim: 416
}

input: "img_info"
input_shape {
  dim: 1
  dim: 4
}
```

代码样例如下，参数为[batch,4]，4表示netH、netW、scaleH、scaleW四个维度。其中netH,netW为网络模型输入的HW，scaleH,scaleW为原始图片的HW。

```
input: "img_info"
input_shape {
  dim: 1
```

```
dim: 4  
}
```

### 8.1.4.3 YOLOv2 网络模型 prototxt 修改

#### 说明

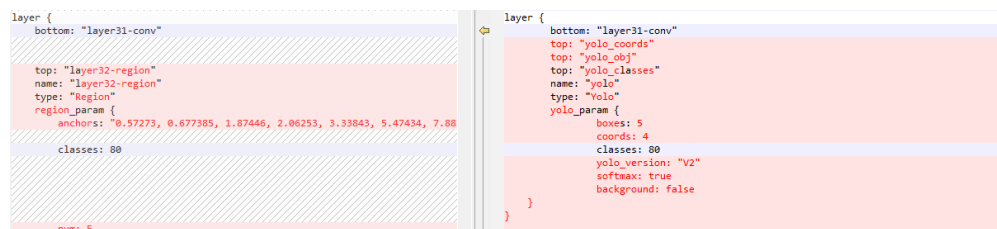
本章节所有的代码样例都不能直接复制到网络模型中使用，需要用户根据使用的网络模型，自行调整相应参数，比如bottom、top中的参数要和具体网络模型中的bottom、top——对应，并且bottom和top对应的参数顺序不能更改。

#### 1. 修改Region算子。

由于Yolo算子+DetectionOutput算子才是特征检测网络完整的后处理逻辑，在增加YoloV2DetectionOutput算子之前需要将Region过程算子，替换为Yolo算子。

参见《[算子清单（开放态）](#)》>AI框架算子支持清单>支持Caffe算子清单，Yolo算子有一个输入，三个输出；基于该原则，修改后的Yolo算子如下，对比情况如图8-7所示，其中左边为原始算子prototxt，右边是适配昇腾AI处理器的prototxt。

图 8-7 原始算子与修改后的算子比对 2



构造的代码样例如下：

```
layer {  
  bottom: "layer31-conv"  
  top: "yolo_coors"  
  top: "yolo_obj"  
  top: "yolo_classes"  
  name: "yolo"  
  type: "Yolo"  
  yolo_param {  
    boxes: 5  
    coords: 4  
    classes: 80  
    yolo_version: "V2"  
    softmax: true  
    background: false  
  }  
}
```

参数解释请参见《[算子清单（开放态）](#)》>AI框架算子支持清单>支持Caffe算子清单。

#### 2. 最后一层增加YoloV2DetectionOutput算子。

对于YoloV2网络，参考[8.1.2 扩展算子列表](#)在原始prototxt文件的最后增加后处理算子层YoloV2DetectionOutput，参见《[算子清单（开放态）](#)》>AI框架算子支持清单>支持Caffe算子清单，YoloV2DetectionOutput算子有四个输入，两个输出；根据上述原则，构造的后处理算子代码样例如下：

```
layer {  
  name: "detection_out2"  
  type: "YoloV2DetectionOutput"  
  bottom: "yolo_coors"  
  bottom: "yolo_obj"  
  bottom: "yolo_classes"
```

```
bottom: "img_info"
top: "box_out"
top: "box_out_num"
yolov2_detection_output_param {
  boxes: 5
  classes: 80
  relative: true
  obj_threshold: 0.5
  score_threshold: 0.5
  iou_threshold: 0.45
  pre_nms_topn: 512
  post_nms_topn: 1024
  biases: 0.572730
  biases: 0.677385
  biases: 1.874460
  biases: 2.062530
  biases: 3.338430
  biases: 5.474340
  biases: 7.882820
  biases: 3.527780
  biases: 9.770520
  biases: 9.168280
}
```

参数解释请参见《[算子清单（开放态）](#)》>AI框架算子支持清单>支持Caffe算子清单。

### 3. 新增输入：

由于YoloV2DetectionOutput算子有img\_info输入，故模型输入时增加该输入。对比情况如[图8-8](#)所示，其中左边为原始算子prototxt，右边是适配昇腾AI处理器的prototxt。

图 8-8 原始算子与修改后的算子比对 1

```
layer {
  name: "data"
  type: "Input"
  top: "data"
  input_param {
    shape {
      dim: 1
      dim: 3
      dim: 608
      dim: 608
    }
  }
}

input: "data"
input_shape {
  dim: 1
  dim: 3
  dim: 608
  dim: 608
}

input: "img_info"
input_shape {
  dim: 1
  dim: 4
}
```

代码样例如下，参数为[batch,4]，4表示netH、netW、scaleH、scaleW四个维度。其中netH,netW为网络模型输入的HW，scaleH,scaleW为原始图片的HW。

```
input: "img_info"
input_shape {
  dim: 1
  dim: 4
}
```

## 8.1.4.4 SSD 网络模型 prototxt 修改

### 说明

本章节所有的代码样例都不能直接复制到网络模型中使用，需要用户根据使用的网络模型，自行调整相应参数，比如bottom、top中的参数要和具体网络模型中的bottom、top一一对应，并且bottom和top对应的参数顺序不能更改。

对于SSD网络，参考[8.1.2 扩展算子列表](#)在原始prototxt文件的最后增加后处理算子层SSDDetectionOutput。

参见caffe.proto文件（该文件路径为\${INSTALL\_DIR}/include/proto），先在LayerParameter message中添加自定义层参数的声明（如下自定义层在caffe.proto中已经声明，用户无需再次添加）：

```
message LayerParameter {  
...  
  optional SSDDetectionOutputParameter ssddetectionoutput_param = 232;  
...  
}
```

参见caffe.proto文件，此算子的Type及属性定义如下：

```
message SSDDetectionOutputParameter {  
  optional int32 num_classes = 1 [default = 2];  
  optional bool share_location = 2 [default = true];  
  optional int32 background_label_id = 3 [default = 0];  
  optional float iou_threshold = 4 [default = 0.45];  
  optional int32 top_k = 5 [default = 400];  
  optional float eta = 6 [default = 1.0];  
  optional bool variance_encoded_in_target = 7 [default = false];  
  optional int32 code_type = 8 [default = 2];  
  optional int32 keep_top_k = 9 [default = 200];  
  optional float confidence_threshold = 10 [default = 0.01];  
}
```

参见《[算子清单（开放态）](#)》>AI框架算子支持清单>支持Caffe算子清单，SSDDetectionOutput算子有3个输入，2个输出，基于上述原则，构造的代码样例如下：

```
layer {  
  name: "detection_out"  
  type: "SSDDetectionOutput"  
  bottom: "bbox_delta"  
  bottom: "score"  
  bottom: "anchors"  
  top: "out_boxnum"  
  top: "y"  
  ssddetectionoutput_param {  
    num_classes: 2  
    share_location: true  
    background_label_id: 0  
    iou_threshold: 0.45  
    top_k: 400  
    eta: 1.0  
    variance_encoded_in_target: false  
    code_type: 2  
    keep_top_k: 200  
    confidence_threshold: 0.01  
  }  
}
```

- bottom输入中的bbox\_delta对应caffe原始网络中的mbox\_loc，score对应caffe原始网络中的mbox\_conf\_flatten，anchors对应caffe原始网络中的mbox\_priorbox；num\_classes取值需要与原始网络模型中的取值保持一致。
- top输出多batch场景下：
  - out\_boxnum输出shape是(batchnum,8)，每个batchnum的第一个值是实际框的个数。
  - y输出shape是(batchnum,len,8)，其中len是keep\_top\_k 128对齐后的取值（如batch为2，keep\_top\_k为200，则最后输出shape为(2,256,8)），前256\*8个数据为第一个batch的结果。

参数解释请参见《[算子清单（开放态）](#)》>AI框架算子支持清单>支持Caffe算子清单。

### 8.1.4.5 BatchedMatMul 算子 prototxt 修改

#### 说明

本章节所有的代码样例都不能直接复制到网络模型中使用，需要用户根据使用的网络模型，自行调整相应参数，比如bottom、top中的参数要和具体网络模型中的bottom、top一一对应，并且bottom和top对应的参数顺序不能更改。

该算子用户输出两个张量的乘积： $y=x1*x2$ （ $x1$ 和 $x2$ 的张量相乘， $x1$ 和 $x2$ 维数必须大于2而小于等于8）。如果网络模型使用该算子，则请参见该章节，修改相应prototxt后，再进行模型转换。

参见caffe.proto文件（该文件路径为 $\${INSTALL\_DIR}/include/proto$ ），先在LayerParameter message中添加自定义层参数的声明（如下自定义层在caffe.proto中已经声明，用户无需再次添加）：

```
message LayerParameter {  
...  
  optional BatchMatMulParameter batch_matmul_param = 235;  
...  
}
```

参见caffe.proto文件，此算子的Type及属性定义如下：

```
message BatchMatMulParameter{  
  optional bool adj_x1 = 1 [default = false];  
  optional bool adj_x2 = 2 [default = false];  
}
```

参见《[算子清单（开放态）](#)》>AI框架算子支持清单>支持Caffe算子清单，BatchedMatMul算子有2个输入，1个输出，基于上述原则，构造的代码样例如下：

```
layer {  
  name: "batchmatmul"  
  type: "BatchedMatMul"  
  bottom: "matmul_data_1"  
  bottom: "matmul_data_2"  
  top: "batchmatmul_1"  
  batch_matmul_param {  
    adj_x1:false  
    adj_x2:true  
  }  
}
```

参数解释请参见《[算子清单（开放态）](#)》>AI框架算子支持清单>支持Caffe算子清单。

### 8.1.4.6 SENet 网络模型 prototxt 修改

#### 说明

本章节所有的代码样例都不能直接复制到网络模型中使用，需要用户根据使用的网络模型，自行调整相应参数，比如bottom、top中的参数要和具体网络模型中的bottom、top一一对应，并且bottom和top对应的参数顺序不能更改。

该网络模型中的Axy算子，需要修改为Reshape、Scale、Eltwise三个算子，修改样例如下：



修改后的代码样例如下：

```
layer {
  name: "conv3_1_axpy_reshape"
  type: "Reshape"
  bottom: "conv3_1_1x1_up"
  top: "conv3_1_axpy_reshape"
  reshape_param {
    shape {
      dim: 0
      dim: -1
    }
  }
}
layer {
  name: "conv3_1_axpy_scale"
  type: "Scale"
  bottom: "conv3_1_1x1_increase"
  bottom: "conv3_1_axpy_reshape"
  top: "conv3_1_axpy_scale"
  scale_param {
    axis: 0
    bias_term: false
  }
}
layer {
  name: "conv3_1_axpy_eltwise"
  type: "Eltwise"
  bottom: "conv3_1_axpy_scale"
  bottom: "conv3_1_1x1_proj"
  top: "conv3_1"
}
```

Reshape、Scale、Eltwise算子相关参数解释请参见《[算子清单（开放态）](#)》>AI框架算子支持清单>支持Caffe算子清单。

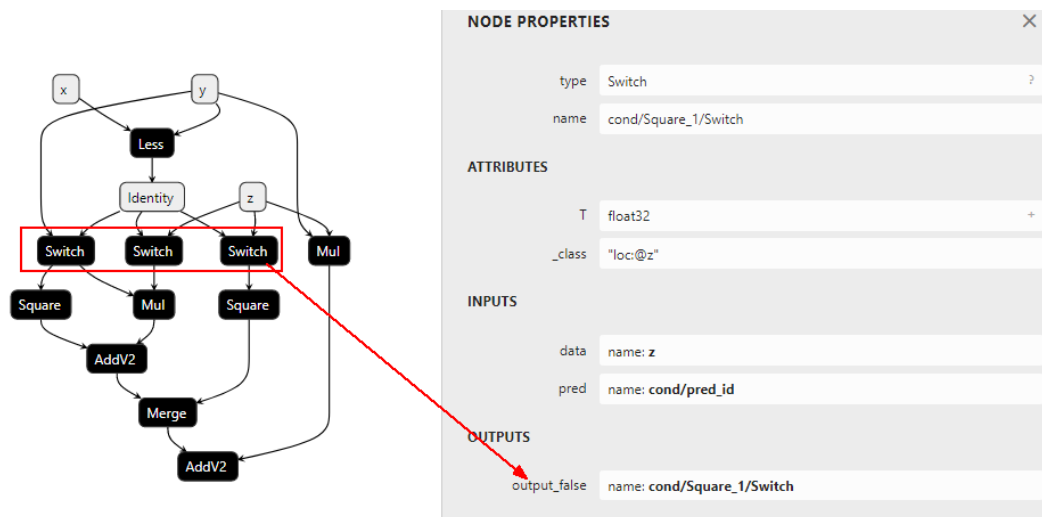
## 8.2 定制网络修改（TensorFlow）

### 8.2.1 概述

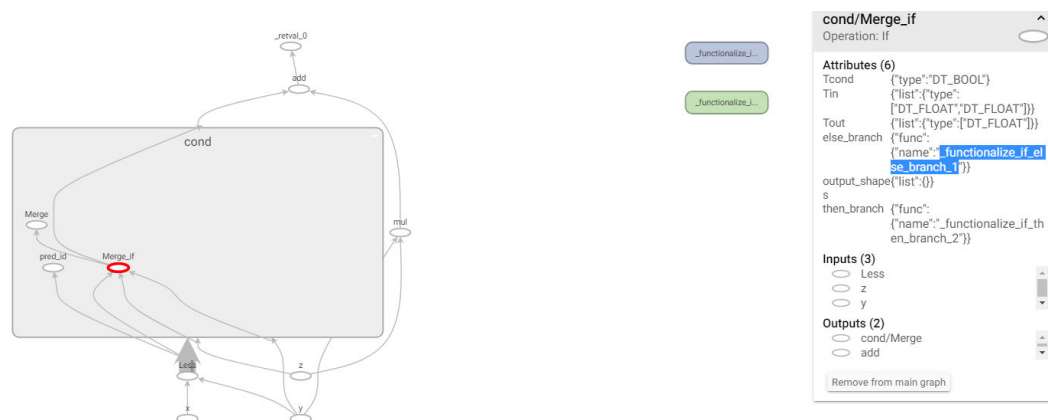
## 简介

本章节介绍如何将TensorFlow有控制流算子的网络模型（如图8-9所示）转成函数类算子的网络模型（如图8-10所示），然后利用ATC工具转换成适配昇腾AI处理器的离线模型。

图 8-9 有控制流算子的网络模型



**图 8-10 函数类算子的网络模型**



## 使用前提

- 使用该工具时，请确保工具所在服务器能连接网络。
  - 安装bazel编译工具。
- 请参见<https://docs.bazel.build/versions/master/install-ubuntu.html>官方地址安装bazel编译工具。
- 安装TensorFlow以及依赖future。
- ATC安装路径下的func2graph.py脚本依赖TensorFlow，使用**pip3.7.5 list**查看列表中是否有TensorFlow依赖，若有则不用安装，否则执行如下命令安装：
- ```
pip3.7.5 install tensorflow==1.15 --user
```



bazel编译工具依赖python, 请使用**pip3.7.5 list**命令查看是否安装future、patch, 若有则不用安装, 否则执行如下命令安装:

```
pip3.7.5 install future --user  
pip3.7.5 install patch --user  
pip3.7.5 install numpy --user
```

如果执行上述命令时无法连接网络, 且提示“Could not find a version that satisfies the requirement xxx”, 请参见[8.2.4.2 使用pip3.7.5 install软件时提示“Could not find a version that satisfies the requirement xxx”](#)解决。

## 8.2.2 编译可执行文件

请以ATC软件包安装用户进行如下操作:

**步骤1** 从<https://github.com/tensorflow/tensorflow/archive/v1.15.0.tar.gz>链接下载Tensorflow源码, 然后将下载的源码上传到ATC工具所在Linux服务器任意目录。

**步骤2** 登录Linux服务器, 切换到Tensorflow源码所在路径, 执行如下命令解压源码包:

```
tar -zxvf tensorflow-1.15.0.tar.gz
```

**步骤3** 进入解压后的tensorflow-1.15.0, 安装补丁。

参见[8.2.4.4 获取xlacompile.patch补丁文件](#)获取xlacompile.patch补丁, 然后上传到Linux服务器tensorflow-1.15.0路径下, 执行如下命令安装补丁:

```
patch -p1 < xlacompile.patch
```

**步骤4** 切换到tensorflow-1.15.0目录, 执行如下命令编译xlacompile工具:

```
bazel build --config=monolithic //tensorflow/compiler/aot:xlacompile
```

若出现类似如下信息, 则说明编译成功, 编译大约需要10分钟左右; 若编译失败, 请参见[8.2.4.1 使用bazel编译工具编译时提示“An error occurred during the fetch of repository 'io\\_bazel\\_rules\\_docker'”](#), 编译失败解决。

```
Target //tensorflow/compiler/aot:xlacompile up-to-date:
```

```
  bazel-bin/tensorflow/compiler/aot/xlacompile
```

```
INFO: Elapsed time: 214.550s, Critical Path: 73.38s
```

```
INFO: 1511 processes: 1511 local.
```

```
INFO: Build completed successfully, 1513 total actions
```

编译成功后, 会在\$HOME/.cache/bazel/\_bazel\_test/abd37aaac8a380ca5a3f13938322fcb2/external/org\_tensorflow/bazel-out/k8-opt/bin/tensorflow/compiler/aot路径生成xlacompile可执行文件(该路径只是样例, 请以用户实际编译后的为准)。

xlacompile可执行文件用于将控制流算子的网络模型转成函数类算子的网络模型。

**步骤5** 切换到tensorflow-1.15.0目录, 执行如下命令编译summarize\_graph工具:

```
bazel build --config=monolithic -c opt //tensorflow/tools/graph_transforms:summarize_graph
```

若出现类似如下信息, 则说明编译成功:

```
Target //tensorflow/tools/graph_transforms:summarize_graph up-to-date:
```

```
  bazel-bin/tensorflow/tools/graph_transforms/summarize_graph
```

```
INFO: Elapsed time: 70.474s, Critical Path: 53.16s
```

```
INFO: 1028 processes: 1028 local.
```

```
INFO: Build completed successfully, 1053 total actions
```

编译成功后, 会在\$HOME/.cache/bazel/\_bazel\_test/abd37aaac8a380ca5a3f13938322fcb2/execroot/org\_tensorflow/bazel-out/k8-opt/bin/tensorflow/tools/graph\_transforms路径生成summarize\_graph可执行文件(该路径只是样例, 请以用户实际编译后的为准)。



**summarize\_graph**可执行文件用来查看有控制流算子网络模型的输入输出节点，方便用户构造config.pbtxt输入输出配置文件。

----结束

## 8.2.3 转换模型

下面以Switch\_v1.pb网络模型为例进行说明，演示如何将控制流算子网络模型转换成函数类算子网络模型，然后通过ATC工具转换成适配昇腾AI处理器的离线模型。请先参见[8.2.4.3 获取Switch\\_v1.pb网络模型](#)获取Switch\_v1.pb网络模型。

**步骤1** 获取控制流算子网络模型的输出。

切换到**summarize\_graph**可执行文件所在路径，执行如下命令：

```
./summarize_graph --in_graph=/home/test/module/Switch_v1.pb
```

若返回如下信息，则说明执行成功：

```
Found 3 possible inputs: (name=x, type=float(1), shape=<unknown>) (name=y, type=float(1), shape=<unknown>) (name=z, type=float(1), shape=<unknown>)
No variables spotted.
Found 1 possible outputs: (name=add, op=AddV2)
Found 0 (0) const parameters, 0 (0) variable parameters, and 0 control_edges
Op types used: 3 Placeholder, 3 Switch, 2 AddV2, 2 Mul, 2 Square, 1 Identity, 1 Less, 1 Merge
To use with tensorflow/tools/benchmark:benchmark_model try these arguments:
bazel run tensorflow/tools/benchmark:benchmark_model -- --graph=/home/test/module/Switch_v1.pb --
show_flops --input_layer=x,y,z --input_layer_type=float,float,float --input_layer_shape=: --output_layer=add
```

**步骤2** 构造config.pbtxt输出配置文件。

在任意路径执行**vim config.pbtxt**命令创建config.pbtxt文件，本示例以在\$HOME/module路径创建为例进行说明。

根据**步骤1**所示，该网络模型有一个输出，构造的config.pbtxt配置文件样例如下（如下配置文件只是样例，需要用户根据实际情况修改输出算子的name，其中fetch表示输出）：

```
fetch {
  id { node_name: "add" }
}
```

配置完成后，执行**wq!**命令退出。

**步骤3** 生成函数类算子网络模型的配置文件。

在任意路径执行如下命令设置**xlacompile**命令执行过程中的打屏日志信息：

```
export TF_CPP_MIN_LOG_LEVEL=0
export TF_CPP_MIN_VLOG_LEVEL=1
```

切换到**xlacompile**可执行文件所在路径，执行如下命令：

```
./xlacompile --graph=/home/test/module/Switch_v1.pb --config=/home/test/module/config.pbtxt --output=/home/test/module/Switch_v1_v2
```

如果提示“Successfully convert ...”信息，则说明转换成功。切换到**--output**参数指定的路径，可以看到如下输出文件：

```
-rw-rw-r-- 1 test test 1236 Jun 20 17:13 Switch_v1_v2.pb
-rw-rw-r-- 1 test test 4803 Jun 20 17:13 Switch_v1_v2.pbtxt
```

**步骤4** 函数类算子网络模型生成graph子图。后续使用ATC工具进行模型转换时，需要使用该文件生成子图。

在任意路径执行如下命令，将函数类算子网络模型生成子图：

```
python3.7.5 ${INSTALL_DIR}/compiler/python/func2graph/func2graph.py -m /home/test/module/  
Switch_v1_v2.pb
```

若提示如下信息，则说明生成成功。

```
graph_def_file: /home/test/module/graph_def_library.pbtxt  
INFO: Convert to subgraphs successfully.
```

#### 步骤5 函数类算子网络模型转换成适配昇腾AI处理器的离线模型。

参见[设置环境变量](#)设置ATC工具执行时需设置的环境变量，然后执行如下命令进行模型转换：

```
atc --model=/home/test/module/Switch_v1_v2.pb --framework=3 --output=/home/test/module/out/  
Switch_v1_v2_to_om --soc_version=<soc_version>
```

若提示如下信息，则说明模型转换成功：

```
ATC run success
```

成功执行命令后，在--output参数指定的路径下，可查看模型文件。

----结束

## 8.2.4 FAQ

### 8.2.4.1 使用 bazel 编译工具编译时提示 “An error occurred during the fetch of repository 'io\_bazel\_rules\_docker'”，编译失败

#### 问题描述

使用**bazel build --config=monolithic //tensorflow/compiler/aot:xlacompile**命令编译过程中，提示 “ERROR: An error occurred during the fetch of repository 'io\_bazel\_rules\_docker'”，检查服务器，能够连接网络，但仍旧提示如下错误信息：

```
ERROR: An error occurred during the fetch of repository 'io_bazel_rules_docker':  
  java.io.IOException: Error downloading [https://github.com/bazelbuild/rules_docker/releases/download/  
v0.14.3/rules_docker-v0.14.3.tar.gz] to /home/test/.cache/bazel/_bazel_test/  
abd37aaac8a380ca5a3f13938322fcb2/external/io_bazel_rules_docker/rules_docker-v0.14.3.tar.gz: PKIX path  
building failed: sun.security.provider.certpath.SunCertPathBuilderException: unable to find valid certification  
path to requested target  
ERROR: no such package '@io_bazel_rules_docker//repositories': java.io.IOException: Error downloading  
[https://github.com/bazelbuild/rules_docker/releases/download/v0.14.3/rules_docker-v0.14.3.tar.gz]  
to /home/test/.cache/bazel/_bazel_test/abd37aaac8a380ca5a3f13938322fcb2/external/  
io_bazel_rules_docker/rules_docker-v0.14.3.tar.gz: PKIX path building failed:  
sun.security.provider.certpath.SunCertPathBuilderException: unable to find valid certification path to  
requested target  
INFO: Elapsed time: 24.586s  
INFO: 0 processes.  
FAILED: Build did NOT complete successfully (0 packages loaded)
```

#### 解决方案

如果连网情况下，仍旧提示上述无法下载压缩包的问题，则请参见如下方法解决：

#### 步骤1 在本地Windows PC将如下链接中的附件下载到本地，然后上传到linux服务器任意路径，例如上传到\$HOME/bazel\_tools路径。

```
https://github.com/bazelbuild/rules_docker/releases/download/v0.14.3/rules_docker-v0.14.3.tar.gz  
https://github.com/bazelbuild/bazel-skylib/releases/download/0.8.0/bazel-skylib.0.8.0.tar.gz  
https://github.com/bazelbuild/rules_swift/releases/download/0.11.1/rules_swift.0.11.1.tar.gz  
https://github.com/llvm-mirror/llvm/archive/7a7e03f906aada0cf4b749b51213fe5784eeff84.tar.gz
```

则上述文件在linux服务器绝对路径为:

```
/home/test/bazel_tools/rules_docker-v0.14.3.tar.gz  
/home/test/bazel_tools/bazel-skylib.0.8.0.tar.gz  
/home/test/bazel_tools/rules_swift.0.11.1.tar.gz  
/home/test/bazel_tools/llvm-7a7e03f906aada0cf4b749b51213fe5784eeff84.tar.gz
```

## 步骤2 修改bazel编译工具相关文件中的下载链接:

1. 切换到tensorflow-1.15.0目录, 使用**vi WORKSPACE**命令打开WORKSPACE, 修改该文件中的下载链接, 将下载链接修改为linux服务器绝对路径地址:

```
bazel_toolchains_repositories()  
http_archive(  
  name = "io_bazel_rules_docker",  
  sha256 = "6287241e033d247e9da5ff705dd6ef526bac39ae82f3d17de1b69f8cb313f9cd",  
  strip_prefix = "rules_docker-0.14.3",  
  urls = ["file:///home/test/bazel_tools/rules_docker-v0.14.3.tar.gz"],  
)  
  
load(  
  "@io_bazel_rules_docker//repositories:repositories.bzl",  
  container_repositories = "repositories",  
)  
container_repositories()  
  
load("//third_party/toolchains/preconfig/generate:workspace.bzl",  
  "remote_config_workspace")  
remote_config_workspace()  
  
# Apple and Swift rules.  
http_archive(  
  name = "build_bazel_rules_apple",  
  sha256 = "6efdde60c91724a2be7f89b0c0a64f01138a45e63ba5add2dca2645d981d23a1",  
  urls = ["https://github.com/bazelbuild/rules_apple/releases/download/0.17.2/rules_apple.0.17.2.tar.gz"],  
) # https://github.com/bazelbuild/rules_apple/releases  
http_archive(  
  name = "build_bazel_rules_swift",  
  sha256 = "96a86afcbdab215f8363e65a10cf023b752e90b23abf02272c4fc668fcb70311",  
  urls = ["file:///home/test/bazel_tools/rules_swift.0.11.1.tar.gz"],  
) # https://github.com/bazelbuild/rules_swift/releases
```

执行**wq!**保存退出。

2. 修改tensorflow-1.15.0/tensorflow路径下**workspace.bzl**文件中llvm对应的链接:

```
# TODO(phawkins): currently, this rule uses an unofficial LLVM mirror.  
# Switch to an official source of snapshots if/when possible.  
tf_http_archive(  
  name = "llvm",  
  build_file = clean_dep("//third_party/llvm:llvm.autogenerated.BUILD"),  
  sha256 = "599b89411df88b9e2be40b019e7ab0f7c9c10dd5ab1c948cd22e678cc8f8f352",  
  strip_prefix = "llvm-7a7e03f906aada0cf4b749b51213fe5784eeff84",  
  urls = [  
    "https://mirror.bazel.build/github.com/llvm-mirror/llvm/archive/  
7a7e03f906aada0cf4b749b51213fe5784eeff84.tar.gz",  
    "file:///home/test/bazel_tools/llvm-7a7e03f906aada0cf4b749b51213fe5784eeff84.tar.gz",  
  ],  
)
```

3. 修改完成后, 重新切换到tensorflow-1.15.0目录, 执行如下命令编译**xlacompile**工具:

```
bazel build --config=monolithic //tensorflow/compiler/aot:xlacompile
```

若提示 “ERROR: An error occurred during the fetch of repository 'bazel\_skylib':”, 则进入下一步继续修改bazel\_skylib相关的文件。

4. 修改**.cache**目录中的相关链接:

切换到\$HOME目录, 使用**grep -r bazel-skylib.0.8.0 .cache/**命令查看.cache目录哪个文件引用**bazel-skylib.0.8.0.tar.gz**的url, 根据返回信息, 进入引用该url文

件所在的目录，例如.cache/bazel/\_bazel\_test/  
abd37aac8a380ca5a3f13938322fcb2/external/io\_bazel\_rules\_closure/closure/  
repositories.bzl (该路径只是样例，请以用户实际编译后的为准)

打开该文件，将其中的url改为：

```
file:///home/test/bazel_tools/bazel-skylib.0.8.0.tar.gz
```

执行wq!保存退出。

----结束

## 8.2.4.2 使用 pip3.7.5 install 软件时提示" Could not find a version that satisfies the requirement xxx"

### 问题描述

安装依赖时，使用**pip3.7.5 install xxx**命令安装相关软件时提示无法连接网络，且提示“Could not find a version that satisfies the requirement xxx”，使用**apt-get update**命令检查源可用。提示信息如下图所示。

图 8-11 使用 pip3.7.5 安装软件提示信息

```
ascend@dggphicpr32833:~$ pip3 install numpy --user
Collecting numpy
  Retrying (Retry(total=4, connect=None, read=None, redirect=None, status=None)) after connection broken by 'NewConnectionError('<urllib3.connection.VerifiedHTTPSConnection object at 7f5f725b9320>: Failed to establish a new connection: [Errno 101] Network is unreachable')': /simple/numpy/
  Retrying (Retry(total=3, connect=None, read=None, redirect=None, status=None)) after connection broken by 'NewConnectionError('<urllib3.connection.VerifiedHTTPSConnection object at 7f5f725cd9e8>: Failed to establish a new connection: [Errno 101] Network is unreachable')': /simple/numpy/
  Retrying (Retry(total=2, connect=None, read=None, redirect=None, status=None)) after connection broken by 'NewConnectionError('<urllib3.connection.VerifiedHTTPSConnection object at 7f5f725cd9e8>: Failed to establish a new connection: [Errno 101] Network is unreachable')': /simple/numpy/
  Retrying (Retry(total=1, connect=None, read=None, redirect=None, status=None)) after connection broken by 'NewConnectionError('<urllib3.connection.VerifiedHTTPSConnection object at 7f5f725cd9e8>: Failed to establish a new connection: [Errno 101] Network is unreachable')': /simple/numpy/
  Retrying (Retry(total=0, connect=None, read=None, redirect=None, status=None)) after connection broken by 'NewConnectionError('<urllib3.connection.VerifiedHTTPSConnection object at 7f5f725cd9e8>: Failed to establish a new connection: [Errno 101] Network is unreachable')': /simple/numpy/
Could not find a version that satisfies the requirement numpy (from versions: )
```

### 可能原因

没有配置pip源。

### 解决方法

配置pip源，配置方法如下：

**步骤1** 使用ATC软件包安装用户，执行如下命令：

```
cd ~/.pip
```

如果提示目录不存在，则执行如下命令创建：

```
mkdir ~/.pip
cd ~/.pip
```

在.pip目录下创建pip.conf 文件，命令为：

```
touch pip.conf
```

**步骤2** 编辑pip.conf文件。

使用**vi pip.conf**命令打开pip.conf文件，写入如下内容：

```
[global]
#可用的源，请根据实际情况进行替换。
index-url=http://xxx
[install]
#可信主机，请根据实际情况进行替换。
trusted-host=xxx
```

**步骤3** 执行:wq!命令保存文件。

----结束

### 8.2.4.3 获取 Switch\_v1.pb 网络模型

**步骤1** 将如下文件中的脚本复制到.py文件中，例如复制到Switch\_v1.py文件中。

```
import os
import numpy as np
import tensorflow as tf

x = tf.compat.v1.placeholder(tf.float32, name='x')
y = tf.compat.v1.placeholder(tf.float32, name='y')
z = tf.compat.v1.placeholder(tf.float32, name='z')

def then_branch(x, y, z):
    m = tf.square(x)
    return m + tf.multiply(y, z)

def else_branch(x, y, z):
    m = tf.pow(x, y)
    return m - tf.div(y, z)

# 控制流算子使用入口，执行脚本之后，在图中生成对应的V1控制流算子
def testDefun(x, y, z):
    return tf.cond(pred=x < y, true_fn=lambda: then_branch(x, y, z), false_fn=lambda: else_branch(x, y, z)), y

def testCase(x, y, z):
    a, b = testDefun(x, y, z)
    return a + b * z

with tf.compat.v1.Session() as sess:
    result = sess.run(testCase(x, y, z), feed_dict={x: 1., y: .6, z: .2})

    with tf.io.gfile.GFile('./Switch_v1.pb', 'wb') as f:
        f.write(sess.graph_def.SerializeToString())
```

**步骤2** 切换到Switch\_v1.py脚本所在目录，执行如下命令生成Switch\_v1.pb网络模型：

```
python3.7.5 Switch_v1.py
```

命令执行完毕，在当前目录会生成Switch\_v1.pb网络模型。

----结束

### 8.2.4.4 获取 xlacompile.patch 补丁文件

用户安装完xlacompile.patch补丁，编译成xlacompile工具后，该工具可以将有控制流的V1网络模型转成函数类的V2网络模型。

将如下代码复制到文件中，并另存为xlacompile.patch，然后上传到Linux服务器tensorflow-1.15.0路径下：

```
---
WORKSPACE | 7 +
tensorflow/compiler/aot/BUILD | 27 +
tensorflow/compiler/aot/xlacompile_main.cc | 170 +
tensorflow/compiler/tf2xla/tf2xla.cc | 6 +
tensorflow/compiler/tf2xla/tf2xla.h | 4 +
5 files changed, 195 insertions(+)
create mode 100644 tensorflow/compiler/aot/xlacompile_main.cc

diff --git a/WORKSPACE b/WORKSPACE
index 74ea14d..d2265f9 100644
```

```

--- a/WORKSPACE
+++ b/WORKSPACE
@@ -34,6 +34,13 @@ load(

    bazel_toolchains_repositories()

+http_archive(
+  name = "io_bazel_rules_docker",
+  sha256 = "6287241e033d247e9da5ff705dd6ef526bac39ae82f3d17de1b69f8cb313f9cd",
+  strip_prefix = "rules_docker-0.14.3",
+  urls = ["https://github.com/bazelbuild/rules_docker/releases/download/v0.14.3/rules_docker-v0.14.3.tar.gz"],
+)
+
load(
  "@io_bazel_rules_docker//repositories:repositories.bzl",
  container_repositories = "repositories",
)

diff --git a/tensorflow/compiler/aot/BUILD b/tensorflow/compiler/aot/BUILD
index f871115..b2620db 100644
--- a/tensorflow/compiler/aot/BUILD
+++ b/tensorflow/compiler/aot/BUILD
@@ -106,6 +106,33 @@ cc_library(
     ],
 )

+tf_cc_binary(
+  name = "xlacompile",
+  visibility = ["/visibility:public"],
+  deps = [":xlacompile_main"],
+)
+
+cc_library(
+  name = "xlacompile_main",
+  srcs = ["xlacompile_main.cc"],
+  visibility = ["/visibility:public"],
+  deps = [
+      ":tfcompile_lib",
+      "//tensorflow/compiler/tf2xla:tf2xla_proto",
+      "//tensorflow/compiler/tf2xla:tf2xla_util",
+      "//tensorflow/compiler/xla:debug_options_flags",
+      "//tensorflow/compiler/xla/service:compiler",
+      "//tensorflow/core:core_cpu",
+      "//tensorflow/core:core_cpu_internal",
+      "//tensorflow/core:framework",
+      "//tensorflow/core:framework_internal",
+      "//tensorflow/core:graph",
+      "//tensorflow/core:lib",
+      "//tensorflow/core:protos_all_cc",
+      "@com_google_absl//absl/strings",
+  ],
+)
+
# NOTE: Most end-to-end tests are in the "tests" subdirectory, to ensure that
# tfcompile.bzl correctly handles usage from outside of the package that it is
# defined in.

diff --git a/tensorflow/compiler/aot/xlacompile_main.cc b/tensorflow/compiler/aot/xlacompile_main.cc
new file mode 100644
index 0000000..bc795ef
--- /dev/null
+++ b/tensorflow/compiler/aot/xlacompile_main.cc
@@ -0,0 +1,170 @@
+/* Copyright 2017 The TensorFlow Authors. All Rights Reserved.
+
+Licensed under the Apache License, Version 2.0 (the "License");
+you may not use this file except in compliance with the License.
+You may obtain a copy of the License at
+

```

```
+ http://www.apache.org/licenses/LICENSE-2.0
+
+Unless required by applicable law or agreed to in writing, software
+distributed under the License is distributed on an "AS IS" BASIS,
+WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
+See the License for the specific language governing permissions and
+limitations under the License.
+=====*/
+
+#include <memory>
+#include <string>
+#include <utility>
+#include <vector>
+#include <map>
+
+#include "tensorflow/compiler/aot/flags.h"
+#include "tensorflow/compiler/tf2xla/tf2xla.h"
+#include "tensorflow/compiler/tf2xla/tf2xla_util.h"
+#include "tensorflow/core/platform/init_main.h"
+
+namespace tensorflow {
+namespace xlacompile {
+
+const char kUsageHeader[] =
+  "xlacompile performs ahead-of-time compilation of a TensorFlow graph,\n"
+  "resulting in an object file compiled for your target architecture, and a\n"
+  "header file that gives access to the functionality in the object file.\n"
+  "A typical invocation looks like this:\n"
+  "\n"
+  " $ xlacompile --graph=mygraph.pb --config=config.pbtxt --output=output.pbtxt\n"
+  "\n";
+
+void AppendMainFlags(std::vector<Flag>* flag_list, tfcompile::MainFlags* flags) {
+  const std::vector<Flag> tmp = {
+    {"graph", &flags->graph,
+     "Input GraphDef file. If the file ends in '.pbtxt' it is expected to "
+     "be in the human-readable proto text format, otherwise it is expected "
+     "to be in the proto binary format."},
+    {"config", &flags->config,
+     "Input file containing Config proto. If the file ends in '.pbtxt' it "
+     "is expected to be in the human-readable proto text format, otherwise "
+     "it is expected to be in the proto binary format."},
+    {"output", &flags->out_session_module,
+     "Output session module proto. Will generate '.pb' and '.pbtxt' file."},
+  };
+  flag_list->insert(flag_list->end(), tmp.begin(), tmp.end());
+}
+
+Status ReadProtoFile(const string& fname, protobuf::Message* proto) {
+  if (absl::EndsWith(fname, ".pbtxt")) {
+    return ReadTextProto(Env::Default(), fname, proto);
+  } else {
+    return ReadBinaryProto(Env::Default(), fname, proto);
+  }
+}
+
+Status Main(tfcompile::MainFlags& flags) {
+  // Process config.
+  tf2xla::Config config;
+  if (flags.config.empty()) {
+    return errors::InvalidArgument("Must specify --config");
+  }
+  TF_RETURN_IF_ERROR(ReadProtoFile(flags.config, &config));
+  TF_RETURN_IF_ERROR(ValidateConfig(config));
+  if (flags.dump_fetch_nodes) {
+    std::set<string> nodes;
+    for (const tf2xla::Fetch& fetch : config.fetch()) {
+      nodes.insert(fetch.id().node_name());
+    }
+  }
+}
```

```
+ std::cout << absl::StrJoin(nodes, ",");
+ return Status::OK();
+ }
+
+ // Read and initialize the graph.
+ if (flags.graph.empty()) {
+   return errors::InvalidArgument("Must specify --graph");
+ }
+ if (flags.out_session_module.empty()) {
+   return errors::InvalidArgument("Must specify --output");
+ }
+
+ string output_pb_bin = flags.out_session_module + ".pb";
+ string output_pb_txt = flags.out_session_module + ".pbtxt";
+ if (output_pb_bin == flags.config || output_pb_bin == flags.graph ||
+     output_pb_txt == flags.config || output_pb_txt == flags.graph) {
+   return errors::InvalidArgument("Must different --config --graph --output");
+ }
+
+ GraphDef graph_def;
+ TF_RETURN_IF_ERROR(ReadProtoFile(flags.graph, &graph_def));
+ std::unique_ptr<Graph> graph;
+ TF_RETURN_IF_ERROR(ConvertGraphDefToXla(graph_def, config, graph));
+
+ std::map<string, string> arg_name_maps;
+ GraphDef new_graph_def;
+ graph->ToGraphDef(&new_graph_def);
+ // Delete _class attribute for: expects to be colocated with unknown node
+ for (int i = 0; i < new_graph_def.node_size(); ++i) {
+   NodeDef *node = new_graph_def.mutable_node(i);
+   node->mutable_attr()->erase("_class");
+   if (node->op() == "_Retval") {
+     node->set_name(absl::StrCat(node->attr().at("index").i(), "_Retval"));
+   }
+   if (node->op() == "_Arg") {
+     const string name = node->name();
+     node->set_name(absl::StrCat(node->attr().at("index").i(), "_Arg"));
+     arg_name_maps[name] = node->name();
+   }
+ }
+
+ for (int i = 0; i < new_graph_def.node_size() && !arg_name_maps.empty(); ++i) {
+   NodeDef *node = new_graph_def.mutable_node(i);
+   for (int j = 0; j < node->input_size(); ++j) {
+     auto it = arg_name_maps.find(node->input(j));
+     if (it != arg_name_maps.end()) {
+       *node->mutable_input(j) = it->second;
+     }
+   }
+ }
+
+ TF_RETURN_IF_ERROR(WriteBinaryProto(Env::Default(), output_pb_bin, new_graph_def));
+ std::cerr << "Successfully convert: " << output_pb_bin << "\n";
+ TF_RETURN_IF_ERROR(WriteTextProto(Env::Default(), output_pb_txt, new_graph_def));
+ std::cerr << "Successfully convert: " << output_pb_txt << "\n";
+ return Status::OK();
+}
+
+} // end namespace xlacompile
+} // end namespace tensorflow
+
+int main(int argc, char** argv) {
+  tensorflow::tfcompile::MainFlags flags;
+  flags.target_triple = "x86_64-pc-linux";
+  flags.out_function_object = "out_model.o";
+  flags.out_metadata_object = "out_helper.o";
+  flags.out_header = "out.h";
+  flags.entry_point = "entry";
+}
```



```
+ std::vector<tensorflow::Flag> flag_list;
+ tensorflow::xla::Compile::AppendMainFlags(&flag_list, &flags);
+
+ tensorflow::string usage = tensorflow::xla::Compile::kUsageHeader;
+ usage += tensorflow::Flags::Usage(argv[0], flag_list);
+ if (argc > 1 && absl::string_view(argv[1]) == "--help") {
+   std::cerr << usage << "\n";
+   return 0;
+ }
+ bool parsed_flags_ok = tensorflow::Flags::Parse(&argc, argv, flag_list);
+ QCHECK(parsed_flags_ok) << "\n" << usage;
+
+ tensorflow::port::InitMain(usage.c_str(), &argc, &argv);
+ QCHECK(argc == 1) << "\nERROR: This command does not take any arguments "
+   "other than flags\n\n";
+   << usage;
+ tensorflow::Status status = tensorflow::xla::Compile::Main(flags);
+ if (status.code() == tensorflow::error::INVALID_ARGUMENT) {
+   std::cerr << "INVALID ARGUMENTS: " << status.error_message() << "\n\n";
+   << usage;
+   return 1;
+ } else {
+   TF_QCHECK_OK(status);
+ }
+ return 0;
+}

diff --git a/tensorflow/compiler/tf2xla/tf2xla.cc b/tensorflow/compiler/tf2xla/tf2xla.cc
index 3c2b256..3872776 100644
--- a/tensorflow/compiler/tf2xla/tf2xla.cc
+++ b/tensorflow/compiler/tf2xla/tf2xla.cc
@@ -410,4 +410,10 @@ Status ConvertGraphDefToXla(const GraphDef& graph_def,
   return Status::OK();
 }

+Status ConvertGraphDefToXla(const GraphDef &graph_def,
+  const tf2xla::Config &config,
+  std::unique_ptr<Graph> &graph) {
+  return InitGraph(graph_def, config, &graph);
+}

} // namespace tensorflow

diff --git a/tensorflow/compiler/tf2xla/tf2xla.h b/tensorflow/compiler/tf2xla/tf2xla.h
index 432a12a..969500c 100644
--- a/tensorflow/compiler/tf2xla/tf2xla.h
+++ b/tensorflow/compiler/tf2xla/tf2xla.h
@@ -20,6 +20,7 @@ limitations under the License.
#include "tensorflow/compiler/xla/client/client.h"
#include "tensorflow/compiler/xla/client/xla_computation.h"
#include "tensorflow/core/framework/graph.pb.h"
#include "tensorflow/core/graph/graph.h"

namespace tensorflow {

@@ -34,6 +35,9 @@ Status ConvertGraphDefToXla(const GraphDef& graph_def,
  const tf2xla::Config& config, xla::Client* client,
  xla::XlaComputation* computation);

+Status ConvertGraphDefToXla(const GraphDef &graph_def,
+  const tf2xla::Config &config,
+  std::unique_ptr<Graph> &graph);
+} // namespace tensorflow

#endif // TENSORFLOW_COMPILER_TF2XLA_TF2XLA_H_

--
1.8.3.1
```

# 9 参考

[dump图详细信息](#)

[算子规格参考](#)

[开启AI CPU Cast算子自动插入特性](#)

[支持量化的层及约束](#)

本章节给出不同框架可量化的层以及相关约束。

[简易配置文件](#)

如果要自动控制量化过程，比如控制哪些层是否量化、控制使用什么量化算法，则可以通过本章节构造的cfg配置文件实现。

## 9.1 dump 图详细信息

模型转换前，通过设置如下两个环境变量：

```
export DUMP_GE_GRAPH=1    #控制dump图的内容多少
export DUMP_GRAPH_LEVEL=1 #控制dump图的个数
```

在执行atc命令的当前路径会生成如下文件，关于环境变量的详细介绍请参见[2](#)。

- `ge_onnx*.pbtxt`：基于ONNX的开源模型描述结构，可以使用Netron等可视化软件打开。
- `ge_proto*.txt`：protobuf格式存储的文本文件，该文件可以转成json格式文件方便用户定位问题。该文件与`ge_onnx*.pbtxt`成对出现，但是比`ge_onnx*.pbtxt`文件会多string类型的属性信息，用户选择其中一种文件打开即可。

上述每个文件对应模型编译过程中的一个步骤，比如以`ge_onnx_00000001_graph_0_PreRunBegin.pbtxt`开始，以`ge_onnx_00000078_graph_0_PreRunAfterBuild.pbtxt`结尾。每个文件中包括完成该步骤所涉及的所有算子，关于dump图每个阶段的子图详细作用请参见[9.1 dump图详细信息](#)（每个模型生成的dump子图可能不一致，但是主流程基本一致）。

表 9-1 dump 图详细信息说明

| 子图名称                                                            | 涉及组件 | 含义                                                              |
|-----------------------------------------------------------------|------|-----------------------------------------------------------------|
| ge_proto_00000000_RunCustomPassBegin.txt                        | GE   | GE获取到的经过parse处理的整张下沉图，在此图前还会进行parse处理，TensorFlow场景还会经过scope融合处理 |
| ge_proto_00000001_PreRunBegin.txt                               | GE   | 用户自定义优化处理之后的图结构                                                 |
| ge_proto_00000002_PrepareAfterCheckAndUpdateInput.txt           | GE   | 校验并更新图输入数据处理之后的图结构                                              |
| ge_proto_00000003_PrepareAfterPropagateFormatIfNeed.txt         | GE   | 单算子模式下，对算子做format推导处理之后的图结构                                     |
| ge_proto_00000004_PreRunAfterInitPreparation.txt                | GE   | 经历了图准备阶段所有初始化处理之后的图结构                                           |
| ge_proto_00000005_OptimizeGraph_TagNoConstFoldingAfter.txt      | FE   | 量化场景使用，FE会给算子打上不做常量折叠标签，GE在执行常量折叠时会判断此标签，如果存在，则不执行常量折叠          |
| ge_proto_00000006_PreRunAfterOptimizeGraphPrepare.txt           | GE   | 经过各算子信息库原图准备处理（OptimizeGraphPrepare接口调用）之后的图结构                  |
| ge_proto_00000007_PreRunAfterHandleSummaryOp.txt                | GE   | 对Summary节点做处理之后的图结构                                             |
| ge_proto_00000008_PrepareAfterGraphEquivalentTransformation.txt | GE   | 将For循环图结构同等替换成While循环图结构处理之后的图结构                                |
| ge_proto_00000009_PrepareAfterProcessOutput.txt                 | GE   | 对图数据进行相关处理之后的图结构                                                |
| ge_proto_00000010_PrepareAfterProcessMultiBatch.txt             | GE   | 在动态档位开关下，对图结构做相关处理之后的图结构                                        |
| ge_proto_00000011_PrepareAfterInsertAipp.txt                    | GE   | 在配置了aipp参数下，对图进行aipp相关处理之后的图结构                                  |
| ge_proto_00000012_PrepareAfterProcessBeforeInfershape.txt       | GE   | 对条件算子进行死边消除处理之后的图结构                                             |
| ge_proto_00000013_after_first_inferformat.txt                   | GE   | 经过全图inferformat处理之后的图结构                                         |
| ge_proto_00000014_after_infershape.txt                          | GE   | 经过全图infershape处理之后的图结构，会伴随常量折叠                                  |

| 子图名称                                                               | 涉及组件 | 含义                                                          |
|--------------------------------------------------------------------|------|-------------------------------------------------------------|
| ge_proto_00000015_PrepareAfterInferFormatAndShape.txt              | GE   | 经历完所有inferformat与infershape处理之后的图结构，与上图间经历了第二次全图inferformat |
| ge_proto_00000016_PrepareAfterCtrlFlowPreProcess.txt               | GE   | 对条件算子做预处理之后的图结构                                             |
| ge_proto_00000017_PrepareAfterGetDynamicOutputShape.txt            | GE   | 动态档位下，对图输出做处理之后的图结构                                         |
| ge_proto_00000018_PrepareAfterProcessAippStage2.txt                | GE   | 在aipp模式下，对图输入节点做相关处理之后的图结构                                  |
| ge_proto_00000019_PrepareAfterPrepareOptimize.txt                  | GE   | 在图准备阶段，做相关优化处理之后的图结构                                        |
| ge_proto_00000020_PreRunAfterPrepare.txt                           | GE   | 目前和上张图相同，经历过所有图准备处理之后的图结构                                   |
| ge_proto_00000021_OptimizeQuantGraph_FeGraphFusionAfter.txt        | FE   | 图优化阶段的量化流程结束后的图结构                                           |
| ge_proto_00000022_OptimizeOriginalGraph_FeGraphFusionAfter.txt     | FE   | 图融合流程结束后的图结构                                                |
| ge_proto_00000023_OptimizeOriginalGraph_FeTopoSortingAfter.txt     | FE   | 图融合后进行拓扑排序，排查融合后是否成环的图结构                                    |
| ge_proto_00000024_PreRunAfterOptimizeOriginalGraph.txt             | GE   | 经过各算子信息库原图优化处理（OptimizeOriginalGraph接口调用）之后的图结构             |
| ge_proto_00000025_PrepareAfterUpdateInputOutputByUserOptions.txt   | GE   | 根据用户参数，对图输入输出做相关处理之后的图结构                                    |
| ge_proto_00000026_PrepareAfterUpdateVariableFormats.txt            | GE   | 对变量的Format进行相关处理之后的图结构                                      |
| ge_proto_00000027_PreRunAfterPrepareRunningFormatRefiner.txt       | GE   | 与上图相同                                                       |
| ge_proto_00000028_OptimizeOriginalGraph_FeOpJudgeAfter.txt         | FE   | opjudge流程后的图结构                                              |
| ge_proto_00000029_OptimizeOriginalGraph_FeDistHeavyFormatAfter.txt | FE   | 重型算子扩散后的图结构                                                 |

| 子图名称                                                                             | 涉及组件 | 含义                                                       |
|----------------------------------------------------------------------------------|------|----------------------------------------------------------|
| ge_proto_00000030_OptimizeOriginalGraph_FeInsertTransNodeAfter.txt               | FE   | 插入转换算子后的图结构                                              |
| ge_proto_00000031_PreRunAfterRefineRunningFormat.txt                             | GE   | 经过各算子信息库优化处理（OptimizeOriginalGraphJudgeInsert接口调用）之后的图结构 |
| ge_proto_00000032_PreRunAfterSubexpressionMigration.txt                          | GE   | 公共子表达式消除优化处理之后的图结构                                       |
| ge_proto_00000033_OptimizeStage1_1.txt                                           | GE   | 图优化1_1阶段处理之后的图结构                                         |
| ge_proto_00000034_OptimizeStage1_2.txt                                           | GE   | 图优化1_2阶段处理之后的图结构                                         |
| ge_proto_00000035_PreRunAfterOptimize1.txt                                       | GE   | 所有图优化1阶段处理之后的图结构                                         |
| ge_proto_00000036_PreRunAfterOptimizeAfterStage1.txt                             | GE   | 经过各算子信息库优化处理（OptimizeAfterStage1接口调用）之后的图结构              |
| ge_proto_00000037_PreRunAfterInferShape2.txt                                     | GE   | 第二次inershape处理之后的图结构                                     |
| ge_proto_00000038_AfterPipelinePartition.txt                                     | GE   | 为本地队列流水做图拆分之后的图结构，helper场景使用                             |
| ge_proto_00000039_AfterDynamicShapePartition.txt                                 | GE   | 动态shape图拆分之后的图结构                                         |
| ge_proto_00000040_MergedComputeGraphAfterCompositeEnginePartition.txt            | GE   | 经历对立子图拆分与子图优化处理之后的合并图结构                                  |
| ge_proto_00000041_partition0_rank0_inputNodeGraph_AtomicEnginePartitioning.txt   | GE   | 原子引擎规则图拆分后，输入节点子图的图结构                                    |
| ge_proto_00000042_partition0_rank1_new_sub_graph1_AtomicEnginePartitioning.txt   | GE   | 原子引擎规则图拆分后，子图1的图结构                                       |
| ge_proto_00000043_partition0_rank2_new_sub_graph110_AtomicEnginePartitioning.txt | GE   | 原子引擎规则图拆分后，子图110的图结构                                     |
| ge_proto_00000044_OptimizeSubGraphBefore.txt                                     | GE   | 子图优化操作前的子图结构，每张子图都有一份，同名不同序号，总个数根据子图个数确定                 |

| 子图名称                                                                     | 涉及组件 | 含义                                         |
|--------------------------------------------------------------------------|------|--------------------------------------------|
| ge_proto_00000045_OptimizeSubGraphBefore.txt                             | GE   | 子图优化操作前的子图结构，每张子图都有一份，同名不同序号，总个数根据子图个数确定   |
| ge_proto_00000046_OptimizeSubGraphAfter.txt                              | GE   | 子图优化操作后的子图结构，每张子图都有一份，同名不同序号，总个数根据子图个数确定   |
| ge_proto_00000047_partition0_rank1_new_sub_graph1_lxfusion_input.txt     | AOE  | ATC场景和AOE baseline场景的sgat输入图               |
| ge_proto_00000048_partition0_rank1_new_sub_graph1_after_rebuild.txt      | AOE  | AOE sgat内部流程UB融合图                          |
| ge_proto_00000049_OptimizeSubGraphAfter.txt                              | GE   | 子图优化操作后的子图结构，每张子图都有一份，同名不同序号，总个数根据子图个数确定   |
| ge_proto_00000050_mergedComputeGraph.txt                                 | GE   | 图合并之后的图结构，与上图相同                            |
| ge_proto_00000051_MergedComputeGraphAfterAtomicEnginePartition.txt       | GE   | 经历对立原子引擎拆分与子图优化处理之后的合并图结构                  |
| ge_proto_00000052_PreRunAfterOptimizeSubgraph.txt                        | GE   | 子图优化处理之后的图结构                               |
| ge_proto_00000053_OptimizeWholeGraphaicpu_tf_optimizer.txt               | GE   | 调用各引擎的原图优化接口后的图信息，OptimizeWholeGraph后为引擎名称 |
| ge_proto_00000054_OptimizeWholeGraphaicpu_ascend_optimizer.txt           | GE   | 调用各引擎的原图优化接口后的图信息，OptimizeWholeGraph后为引擎名称 |
| ge_proto_00000055_OptimizeWholeGraphAlcoreEngine.txt                     | GE   | 调用各引擎的原图优化接口后的图信息，OptimizeWholeGraph后为引擎名称 |
| ge_proto_00000056_OptimizeWholeGraphDNN_VM_RTS_GRAPH_OPTIMIZER_STORE.txt | GE   | 调用各引擎的原图优化接口后的图信息，OptimizeWholeGraph后为引擎名称 |
| ge_proto_00000057_OptimizeWholeGraphDNN_VM_HOST_CPU_OPTIMIZER.txt        | GE   | 调用各引擎的原图优化接口后的图信息，OptimizeWholeGraph后为引擎名称 |
| ge_proto_00000058_PreRunAfterOptimizeWholeGraph.txt                      | GE   | 经过各算子信息库优化处理（OptimizeWholeGraph接口调用）之后的图结构 |

| 子图名称                                                                       | 涉及组件 | 含义                                                   |
|----------------------------------------------------------------------------|------|------------------------------------------------------|
| ge_proto_00000059_BeforeHandleMemConflict.txt                              | GE   | -                                                    |
| ge_proto_00000060_PreRunAfterOptimize2.txt                                 | GE   | 所有图优化2阶段处理之后的图结构                                     |
| ge_proto_00000061_PreRunAfterOptimizeGraphBeforeBuild.txt                  | GE   | 经过各算子信息库优化处理 ( OptimizeGraphBeforeBuild接口调用 ) 之后的图结构 |
| ge_proto_00000062_partition0_rank0_inputNodeGraph_SecondPartitioning.txt   | GE   | 二拆操作后, 输入节点子图的图结构                                    |
| ge_proto_00000063_partition0_rank1_new_sub_graph1_SecondPartitioning.txt   | GE   | 二拆操作后, 子图1的图结构                                       |
| ge_proto_00000064_partition0_rank2_new_sub_graph110_SecondPartitioning.txt | GE   | 二拆操作后, 子图2的图结构                                       |
| ge_proto_00000065_BeforePreBuildModel.txt                                  | GE   | 经历过二次图拆分, 在图编译动作处理之前的图结构                             |
| ge_proto_00000066_AfterPreBuildModel.txt                                   | GE   | 经历图编译前预处理动作后的图结构                                     |
| ge_proto_00000067_AfterCalcOppParam.txt                                    | GE   | 对图中所有节点的tensor做size计算之后的图结构                          |
| ge_proto_00000068_BeforeAssignedLogicalStreams.txt                         | GE   | 在逻辑流分配处理之前的图结构                                       |
| ge_proto_00000069_AfterAssignedLogicalStreams.txt                          | GE   | 完成逻辑流分配处理之后的图结构                                      |
| ge_proto_00000070_BeforeRefreshRealStream.txt                              | GE   | 在流同步激活关系处理之前的图结构, 与上图间会经历内存分配的动作                     |
| ge_proto_00000071_AfterRefreshRealStream.txt                               | GE   | 经历流同步激活关系处理之后的图结构                                    |
| ge_proto_00000072_AfterBuildModel.txt                                      | GE   | 权重合并, 生成模型基础数据之后的图结构                                 |
| ge_proto_00000073_AfterOptimizeStreamedSubGraph.txt                        | GE   | 对流分配结果经过相关优化处理之后的图结构                                 |
| ge_proto_00000074_GenerateTaskBefore.txt                                   | GE   | 在节点生成task处理之前的图结构                                    |

| 子图名称                                    | 涉及组件 | 含义                                            |
|-----------------------------------------|------|-----------------------------------------------|
| ge_proto_00000075_GenerateTaskAfter.txt | GE   | 经历节点生成task处理之后的图结构，其中会调用各算子信息库的GenerateTask接口 |
| ge_proto_00000076_AfterGetTask.txt      | GE   | 在经历了所有task生成处理之后的图结构，与上图相同                    |
| ge_proto_00000077_Build.txt             | GE   | 完成图编译之后的图结构                                   |
| ge_proto_00000078_PreRunAfterBuild.txt  | GE   | 与上图相同                                         |

## 9.2 算子规格参考

### 9.2.1 Caffe 框架算子规格

该算子规格仅适用于Caffe框架原生IR定义的网络模型，算子详情请参见《[算子清单（开放态）](#)》>AI框架算子支持清单>支持Caffe算子清单。

如果要查看基于Ascend IR定义的单算子信息，请参见《[算子清单（开放态）](#)》>CANN算子清单。

当前支持的Caffe版本为caffe-master分支commit id为9b891540183ddc834a02b2bd81b31afae71b2153。

### 9.2.2 TensorFlow 框架算子规格

该算子规格仅适用于TensorFlow框架原生IR定义的网络模型。算子详情请参见《[算子清单（开放态）](#)》>AI框架算子支持清单>支持TensorFlow算子清单。

如果要查看基于Ascend IR定义的单算子信息，请参见《[算子清单（开放态）](#)》>CANN算子清单。

当前支持的TensorFlow版本为1.15.0。

### 9.2.3 ONNX 算子规格

该算子规格仅适用于ONNX原生IR定义的网络模型。算子详情请参见《[算子清单（开放态）](#)》>AI框架算子支持清单>支持ONNX算子清单。

如果要查看基于Ascend IR定义的单算子信息，请参见《[算子清单（开放态）](#)》>CANN算子清单。

当前支持的ONNX版本为1.12.0、Opset版本详见各算子支持的ONNX版本描述、ONNX Runtime版本为1.14.0。



## 9.3 开启 AI CPU Cast 算子自动插入特性

### 简介

模型编译时，若遇到AI CPU算子不支持某种数据类型导致编译失败的场景，可通过启用Cast算子自动插入特性快速将输入转换为算子支持的数据类型，从而实现网络的快速打通。

如图9-1，表示MatrixInverse算子的输入x不支持float16的数据类型。

图 9-1 报错示例

```
2021-05-13 04:19:51.515060: W tensorflow/core/framework/op_kernel.cc:1639] Unavailable: failed
2021-05-13 04:19:55.776673: F tf_adapter/kernels/geop_npu.cc:786] GeOp1_0GEOP::DoRunAsync Failed
Error Message is :
E13002: Op type[MatrixInverse] of ops kernel[aicpu_tf_kernel] is unsupported, data_type DT_FLOAT16 of input x is unsupported by current kernel info store. op
type[MatrixInverse].
Can't find any supported ops kernel and engine of [MatrixInverse], type is [MatrixInverse]
Aborted (core dumped)
root@ubuntu:/home/.../op_tf_testcaseSmoke_host# /usr/local/python3.7.5/lib/python3.7/multiprocessing/semaphore_tracker.py:144: UserWarning: semaphore
tracker: There appear to be 91 leaked semaphores to clean up at shutdown
  (on{cache})
```

此种场景下，即可开启Cast算子自动插入特性，详细操作方法见[操作步骤](#)。

### 操作步骤

#### 步骤1 打开AutoCast开关。

修改“Ascend-cann-toolkit安装目录/ascend-toolkit/latest”目录中“lib64/plugin/opskernel/config/init.conf”文件，将“AutoCastMode”参数的值修改为1，如下所示：

```
...
AutoCastMode = 1
```

#### 步骤2 修改对应的算子信息库（内置AI CPU算子信息库存储在opp安装目录下的“built-in/op\_impl/aicpu/aicpu\_kernel/config”目录下），在需要修改的算子中插入Cast转换规则。

如下所示，MatrixInverse算子的输入x不支持float16，算子信息库配置如下：

```
"MatrixInverse":{
  "input0":{
    "name":"x",
    "type":"DT_FLOAT,DT_DOUBLE,DT_COMPLEX128,DT_COMPLEX64"
  },
  "opInfo":{
    "computeCost":"100",
    "engine":"DNN_VM_AICPU",
    "flagAsync":"False",
    "flagPartial":"False",
    "formatAgnostic":"False",
    "opKernelLib":"TFKernel",
    "opsFlag":"OPS_FLAG_OPEN",
    "subTypeOfInferShape":"1"
  },
  "output0":{
    "name":"y",
    "type":"DT_FLOAT,DT_DOUBLE,DT_COMPLEX128,DT_COMPLEX64"
  }
},
```

为了让其支持float16，需要做如下修改：

## 1. 对输入信息进行修改，增加支持的数据类型，并增加数据类型转换规则。

例如，对MatrixInverse算子，输入增加对float16类型的支持，并增加cast规则，将float16转换为float32，代表在此输入前会插入一个float16到float32的cast算子。

```
"input0":{
  "name":"x",
  "type":["DT_FLOAT,DT_DOUBLE,DT_COMPLEX128,DT_COMPLEX64,DT_FLOAT16",
  "srcAutoCastType":"DT_FLOAT16",
  "dstAutoCastType":"DT_FLOAT"
},
```

- 支持的“type”中增加“DT\_FLOAT16”数据类型，支持的数据类型可参见对应的算子信息库中Cast算子的定义。
- 增加配置“srcAutoCastType”，代表输入数据的类型。
- 增加配置“dstAutoCastType”，代表需要转换成的目标数据类型。

## 2. 对输出信息进行修改，增加支持的数据类型，并增加数据类型转换规则。

例如，对MatrixInverse算子，输出增加对float16类型的支持，并增加cast规则，将float32转换为float16，代表在此输出后插入一个float32到float16的cast算子。

```
"output0":{
  "name":"y",
  "type":["DT_FLOAT,DT_DOUBLE,DT_COMPLEX128,DT_COMPLEX64,DT_FLOAT16",
  "srcAutoCastType":"DT_FLOAT",
  "dstAutoCastType":"DT_FLOAT16"
}
```

- 支持的“type”中增加“DT\_FLOAT16”数据类型，支持的数据类型可参见对应的算子信息库中Cast算子的定义。
- 增加配置“srcAutoCastType”，代表输入数据的类型。
- 增加配置“dstAutoCastType”，代表需要转换成的目标数据类型。

**须知**

- 若算子的多个输入、多个输出要求具有相同的数据类型，则每个输入、输出都需要按照上述规则进行修改。
- 由于插入Cast算子，精度会有一定程度的损失，具体损失大小与转换的数据类型有关。

----结束

## 9.4 支持量化的层及约束

本章节给出不同框架可量化的层以及相关约束。

**说明**

- 若网络模型输入数据类型或权重数据类型为Float16或混合精度类型（Float32/Float16共存），AMCT会关闭如下算子的量化功能：  
AvgPool、Pooling、AvgPoolV2、MaxPool、MaxPoolV3、Pooling、Add、Eltwise。

表 9-2 均匀量化支持的层及约束

| 框架         | 支持的层类型                              | 约束                                                                                                                                                                | 对应 Ascend IR 定义的层类型 |
|------------|-------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------|
| Caffe      | InnerProduct: 全连接层                  | transpose属性为false, axis为1                                                                                                                                         | FullyConnection     |
|            | Convolution: 卷积层                    | filter维度为4                                                                                                                                                        | Conv2D              |
|            | Deconvolution: 反卷积层                 | dilation为1、filter维度为4                                                                                                                                             | Deconvolution       |
|            | Pooling                             | <ul style="list-style-type: none"><li>mode为1, 全量化 ( weight +tensor ), global_pooling为false, 不支持移位N操作</li><li>mode为0, 只做tensor量化</li></ul>                         | Pooling             |
|            | Eltwise                             | 只做tensor量化且operation=1                                                                                                                                            | Eltwise             |
| TensorFlow | MatMul: 全连接层                        | <ul style="list-style-type: none"><li>transpose_a为False, transpose_b为False, adjoint_a为False, adjoint_b为False</li><li>weight的输入来源不含有placeholder等可动态变化的节点</li></ul> | MatMulV2            |
|            | Conv2D: 卷积层                         | weight的输入来源不含有placeholder等可动态变化的节点                                                                                                                                | Conv2D              |
|            | DepthwiseConv2dNative: Depthwise卷积层 | weight的输入来源不含有placeholder等可动态变化的节点                                                                                                                                | DepthwiseConv2D     |
|            | Conv2DBackpropInput                 | dilation为1, weight的输入来源不含有placeholder等可动态变化的节点                                                                                                                    | Conv2DBackpropInput |
|            | BatchMatMulV2                       | <ul style="list-style-type: none"><li>adj_x=False, 第二路输入要求为2维const</li><li>weight的输入来源不含有placeholder等可动态变化的节点</li></ul>                                           | BatchMatMulV2       |
|            | AvgPool                             | 不支持移位N操作                                                                                                                                                          | AvgPool             |
|            | Conv3D                              | dilation_d为1                                                                                                                                                      | Conv3D              |
|            | MaxPool                             | 只做tensor量化                                                                                                                                                        | MaxPool、MaxPoolV3   |
|            | Add                                 | 只做tensor量化                                                                                                                                                        | Add                 |

| 框架   | 支持的层类型              | 约束                                                                                                                       | 对应Ascend IR定义的层类型 |
|------|---------------------|--------------------------------------------------------------------------------------------------------------------------|-------------------|
| ONNX | Conv: 卷积层           | <ul style="list-style-type: none"><li>filter维度为5的情况下，要求dilation_d为1</li><li>weight的输入来源不含有placeholder等可动态变化的节点</li></ul> | Conv2D、Conv3D     |
|      | Gemm: 广义矩阵乘         | <ul style="list-style-type: none"><li>transpose_a=false</li><li>weight的输入来源不含有placeholder等可动态变化的节点</li></ul>             | MatMulV2          |
|      | ConvTranspose: 转置卷积 | <ul style="list-style-type: none"><li>dilation为1、filter维度为4</li><li>weight的输入来源不含有placeholder等可动态变化的节点</li></ul>         | Conv2DTranspose   |
|      | MatMul              | <ul style="list-style-type: none"><li>第二路输入要求为2维const</li><li>weight的输入来源不含有placeholder等可动态变化的节点</li></ul>               | BatchMatMulV2     |
|      | AveragePool         | global_pooling为false，不支持移位N操作                                                                                            | AvgPoolV2         |
|      | MaxPool             | 只做tensor量化                                                                                                               | MaxPool、MaxPoolV3 |
|      | Add                 | 只做tensor量化                                                                                                               | Add               |

表 9-3 非均匀量化支持的层及约束

| 框架         | 支持的层类型             | 约束                       | 对应Ascend IR定义的层类型 |
|------------|--------------------|--------------------------|-------------------|
| Caffe      | Convolution: 卷积层   | dilation为1、filter维度为4    | Conv2D            |
|            | InnerProduct: 全连接层 | transpose属性为false，axis为1 | FullyConnection   |
| TensorFlow | Conv2D: 卷积层        | dilation为1               | Conv2D            |
|            | MatMul: 全连接层       | transpose_a为False        | MatMulV2          |
| ONNX       | Conv: 卷积层          | -                        | Conv2D            |
|            | Gemm: 广义矩阵乘        | transpose_a=false        | MatMulV2          |

表 9-4 仅权重量化场景支持的层及约束

| Ascend<br>IR定义的层类型 | 仅权重量化<br>权重ARQ中<br>channel_wise=true | 仅权重量化<br>权重ARQ中<br>asymmetric=true | 权重和数据都量化<br>权重ARQ中<br>channel_wise=true | 权重和数据都量化<br>权重ARQ中<br>asymmetric=true | 约束                               |
|--------------------|--------------------------------------|------------------------------------|-----------------------------------------|---------------------------------------|----------------------------------|
| MatMulV2           | √                                    | √                                  | ×                                       | ×                                     | 第二路的输入来源不含有placeholder等可动态变化的节点。 |
| BatchMatMulV2      | √                                    | √                                  | ×                                       | ×                                     | 第二路的输入来源不含有placeholder等可动态变化的节点。 |
| Pooling            | 不支持                                  | 不支持                                | ×                                       | ×                                     | -                                |
| AvgPool            | 不支持                                  | 不支持                                | ×                                       | ×                                     | -                                |
| AvgPoolV2          | 不支持                                  | 不支持                                | ×                                       | ×                                     | -                                |

- 其中：
- √表示支持，×表示该场景量化会异常。
  - 权重ARQ中channel\_wise=true：表示每个channel独立量化，量化因子不同。
  - 权重ARQ中asymmetric=true：表示权重量化使用非对称量化。
  - 不支持：表示该算子不支持仅权重量化特性。

9.5 简易配置文件

如果要自动控制量化过程，比如控制哪些层是否量化、控制使用什么量化算法，则可以通过本章节构造的cfg配置文件实现。

表 9-5 calibration\_config.proto 参数说明

| 消息         | 是否必填     | 类型   | 字段                | 说明                                                                                                                          |
|------------|----------|------|-------------------|-----------------------------------------------------------------------------------------------------------------------------|
| AMCTConfig | -        | -    | -                 | AMCT训练后量化的简易配置。                                                                                                             |
|            | optional | bool | activation_offset | 数据量化是否带offset。全局配置参数。 <ul style="list-style-type: none"><li>• 带offset：数据量化使用非对称量化。</li><li>• 不带offset：数据量化使用对称量化。</li></ul> |

| 消息 | 是否必填     | 类型                | 字段                     | 说明                                                                                                                                                                |
|----|----------|-------------------|------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|    | optional | bool              | joint_quant            | 是否进行Eltwise联合量化，默认为false，表示关闭联合量化功能。<br>开启后对部分网络可能会存在性能提升但是精度下降的问题。                                                                                               |
|    | repeated | string            | skip_layers            | 不需要量化层的层名。                                                                                                                                                        |
|    | repeated | string            | skip_layer_types       | 不需要量化的层类型。                                                                                                                                                        |
|    | optional | int32             | version                | 简易配置文件的版本。                                                                                                                                                        |
|    | optional | CalibrationConfig | common_config          | 通用的量化配置，全局量化配置参数。若某层未被override_layer_types或者override_layer_configs重写，则使用该配置。<br>参数优先级：<br>override_layer_configs>override_layer_types>common_config               |
|    | repeated | OverrideLayerType | override_layer_types   | 重写某一类型层的量化配置，即对哪些层进行差异化量化。<br>例如全局量化配置参数配置的量化因子搜索步长为0.01，可以通过该参数对部分层进行差异化量化，可以配置搜索步长为0.02。<br>参数优先级：<br>override_layer_configs>override_layer_types>common_config |
|    | repeated | OverrideLayer     | override_layer_configs | 重写某一层的量化配置，即对哪些层进行差异化量化。<br>例如全局量化配置参数配置的量化因子搜索步长为0.01，可以通过该参数对部分层进行差异化量化，可以配置搜索步长为0.02。<br>参数优先级：<br>override_layer_configs>override_layer_types>common_config   |
|    | optional | bool              | do_fusion              | 是否开启BN融合功能，默认为true，表示开启该功能。                                                                                                                                       |

| 消息                | 是否必填     | 类型                | 字段                 | 说明                                                                                                                                                                                                                                          |
|-------------------|----------|-------------------|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                   | repeated | string            | skip_fusion_layers | 跳过bn融合的层，配置之后这些层不会进行bn融合。                                                                                                                                                                                                                   |
|                   | repeated | Tensor Quantize   | tensor_quantize    | 对网络模型中指定节点的输入Tensor进行训练后量化，来提高数据搬运时的推理性能。<br><b>当前仅支持对MaxPool/Add/eltwise算子做tensor量化。</b>                                                                                                                                                   |
|                   | optional | bool              | enable_auto_nuq    | 是否开启权重自动非均匀量化功能。默认为false，表示不开启该功能。<br>开启该功能，不影响用户已强制配置的量化层（通过简易配置文件中override_layer_configs配置的层），只会在剩余的均匀量化层中自动搜索因权重过大导致性能瓶颈的层，对其量化，提高权重的压缩率，从而达到降低带宽、提升性能的目的。<br>若某层配置了仅支持权重量化（通过weight_compress_only配置为true），在剩余的均匀量化层中搜索时，不会再搜索仅支持权重量化的层。 |
| OverrideLayerType | required | string            | layer_type         | 支持量化的层类型的名称。                                                                                                                                                                                                                                |
|                   | required | CalibrationConfig | calibration_config | 重置的量化配置。                                                                                                                                                                                                                                    |
| OverrideLayer     | -        | -                 | -                  | 重置某层量化配置。                                                                                                                                                                                                                                   |
|                   | required | string            | layer_name         | 被重置层的层名。                                                                                                                                                                                                                                    |
|                   | required | CalibrationConfig | calibration_config | 重置的量化配置。                                                                                                                                                                                                                                    |
| CalibrationConfig | -        | -                 | -                  | Calibration量化的配置。                                                                                                                                                                                                                           |
|                   | -        | ARQuantize        | arq_quantize       | 权重量化算法配置。<br>arq_quantize: ARQ量化算法配置。                                                                                                                                                                                                       |
|                   | -        | FMRQuantize       | ifmr_quantize      | 数据量化算法配置。<br>ifmr_quantize: IFMR量化算法配置。                                                                                                                                                                                                     |

| 消息           | 是否必填     | 类型         | 字段                   | 说明                                                                                                                                                                                                                                                                                                                                                                                               |
|--------------|----------|------------|----------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|              | -        | NUQuantize | nuq_quantize         | 权重量化算法配置。<br>nuq_quantize: 非均匀量化算法配置。                                                                                                                                                                                                                                                                                                                                                            |
|              | optional | bool       | weight_compress_only | 是否只进行权重量化。仅权重量化场景，支持的数据类型必须为Float32, Float16。 <ul style="list-style-type: none"><li>true: 只进行权重量化。</li><li>false: 权重和数据都量化。默认为false。</li></ul> <b>只进行权重量化场景下，不支持同时配置IFMR数据量化和NUQ非均匀量化。</b>                                                                                                                                                                                                       |
| ARQ quantize | -        | -          | -                    | ARQ权重量化算法配置。                                                                                                                                                                                                                                                                                                                                                                                     |
|              | optional | bool       | channel_wise         | 是否对每个channel采用不同的量化因子。 <ul style="list-style-type: none"><li>true: 每个channel独立量化，量化因子不同。</li><li>false: 所有channel同时量化，共享量化因子。</li></ul>                                                                                                                                                                                                                                                          |
|              | optional | bool       | asymmetric           | 是否对权重进行非对称量化。用于控制逐层量化算法的选择。<br><b>只在weight_compress_only为true时生效，若weight_compress_only设置为false，则asymmetric只能设置为false。</b> <ul style="list-style-type: none"><li>true: 权重量化使用非对称量化（offset不为0）。</li><li>false: 权重量化使用对称量化（offset为0），默认为false。</li></ul> 如果override_layer_configs、override_layer_types、common_config配置项都配置该参数，则生效优先级为：<br>override_layer_configs>override_layer_types>common_config |
| FMR Quantize | -        | -          | -                    | FMR数据量化算法配置。                                                                                                                                                                                                                                                                                                                                                                                     |
|              | optional | float      | search_range_start   | 量化因子搜索范围左边界。                                                                                                                                                                                                                                                                                                                                                                                     |
|              | optional | float      | search_range_end     | 量化因子搜索范围右边界。                                                                                                                                                                                                                                                                                                                                                                                     |
|              | optional | float      | search_step          | 量化因子搜索步长。                                                                                                                                                                                                                                                                                                                                                                                        |



| 消息             | 是否必填     | 类型                   | 字段               | 说明                                                                                                                                                                                                                                                                                                |
|----------------|----------|----------------------|------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                | optional | float                | max_percentile   | 最大值搜索位置。                                                                                                                                                                                                                                                                                          |
|                | optional | float                | min_percentile   | 最小值搜索位置。                                                                                                                                                                                                                                                                                          |
|                | optional | bool                 | asymmetric       | 是否对数据进行非对称量化。用于控制逐层量化算法的选择。 <ul style="list-style-type: none"><li>• true: 非对称量化</li><li>• false: 对称量化</li></ul> 如果override_layer_configs、override_layer_types、common_config配置项都配置该参数，或者配置了activation_offset参数，则生效优先级为：override_layer_configs>override_layer_types>common_config>activation_offset |
|                | optional | Calibration DataType | dst_type         | 量化位宽，数据量化是采用INT8量化还是INT16量化，默认为INT8量化。 <b>当前版本仅支持INT8量化。</b>                                                                                                                                                                                                                                      |
| TensorQuantize | -        | -                    | -                | 需要进行训练后量化的输入Tensor配置。                                                                                                                                                                                                                                                                             |
|                | required | string               | layer_name       | 需要对节点输入Tensor进行训练后量化的节点名称，当前仅支持对MaxPool算子的输入Tensor进行量化。                                                                                                                                                                                                                                           |
|                | required | uint32               | input_index      | 需要对节点输入Tensor进行训练后量化的节点的输入索引。                                                                                                                                                                                                                                                                     |
|                | -        | FMRQuantize          | ifmr_quantize    | 数据量化算法配置。<br>ifmr_quantize: IFMR量化算法配置。默认为IFMR量化算法。                                                                                                                                                                                                                                               |
| NUQuantize     | -        | -                    | -                | 非均匀权重量化算法配置。                                                                                                                                                                                                                                                                                      |
|                | optional | uint32               | num_steps        | 非均匀量化的台阶数。当前仅支持设置为16和32。                                                                                                                                                                                                                                                                          |
|                | optional | uint32               | num_of_iteration | 非均匀量化优化的迭代次数。当前仅支持设置为{0,1,2,3,4,5}，0表示没有迭代。                                                                                                                                                                                                                                                       |

- 基于该文件构造的**均匀量化简易配置文件***quant.cfg*样例如下所示：*Otype*需要配置为基于Ascend IR定义的算子类型，详细对应关系请参见[9.4 支持量化的层及约束](#)。

```
# global quantize parameter
activation_offset : true
joint_quant : false
enable_auto_nuq : false
version : 1
skip_layers : "Otype"
skip_layer_types:"Otype"
do_fusion: true
skip_fusion_layers : "Otype"
common_config : {
    arq_quantize : {
        channel_wise : true
    }
    ifmr_quantize : {
        search_range_start : 0.7
        search_range_end : 1.3
        search_step : 0.01
        max_percentile : 0.999999
        min_percentile : 0.999999
        asymmetric : true
    }
}

override_layer_types : {
    layer_type : "Otype"
    calibration_config : {
        arq_quantize : {
            channel_wise : false
        }
        ifmr_quantize : {
            search_range_start : 0.8
            search_range_end : 1.2
            search_step : 0.02
            max_percentile : 0.999999
            min_percentile : 0.999999
            asymmetric : false
        }
    }
}

override_layer_configs : {
    layer_name : "Opname"
    calibration_config : {
        arq_quantize : {
            channel_wise : true
        }
        ifmr_quantize : {
            search_range_start : 0.8
            search_range_end : 1.2
            search_step : 0.02
            max_percentile : 0.999999
            min_percentile : 0.999999
            asymmetric : false
        }
    }
}

tensor_quantize {
    layer_name: "Opname"
    input_index: 0
    ifmr_quantize: {
        search_range_start : 0.7
        search_range_end : 1.3
        search_step : 0.01
        min_percentile : 0.999999
        asymmetric : false
    }
}
```

```
}  
tensor_quantize {  
  layer_name: "Opname"  
  input_index: 0  
}
```

- 基于该文件构造的仅权重量化简易配置文件`quant.cfg`配置示例:

```
activation_offset : true  
joint_quant : false  
version : 1  
do_fusion: true  
common_config : {  
  weight_compress_only : true  
  arq_quantize : {  
    channel_wise : true  
    asymmetric : false  
  }  
}  
  
override_layer_types : {  
  layer_type : "Otype"  
  calibration_config : {  
    weight_compress_only : true  
    arq_quantize : {  
      channel_wise : true  
      asymmetric : true  
    }  
  }  
}  
  
override_layer_configs : {  
  layer_name : "Opname"  
  calibration_config : {  
    weight_compress_only : true  
    arq_quantize : {  
      channel_wise : true  
      asymmetric : true  
    }  
  }  
}
```

- 基于该文件构造的非均匀量化简易配置文件`quant.cfg`配置示例:

```
# global quantize parameter  
activation_offset : true  
joint_quant : false  
enable_auto_nuq : false  
  
common_config : {  
  arq_quantize : {  
    channel_wise : true  
  }  
  ifmr_quantize : {  
    search_range_start : 0.7  
    search_range_end : 1.3  
    search_step : 0.01  
    max_percentile : 0.999999  
    min_percentile : 0.999999  
    asymmetric : true  
  }  
}  
  
override_layer_types : {  
  layer_type : "Otype"  
  calibration_config : {  
    arq_quantize : {  
      channel_wise : false  
    }  
    ifmr_quantize : {  
      search_range_start : 0.7  
      search_range_end : 1.3  
    }  
  }  
}
```

```
        search_step : 0.01
        max_percentile : 0.999999
        min_percentile : 0.999999
        asymmetric : false
    }
}
}
override_layer_configs : {
    layer_name : "Opname"
    calibration_config : {
        nuq_quantize : {
            num_steps : 32
            num_of_iteration : 1
        }
        ifmr_quantize : {
            search_range_start : 0.8
            search_range_end : 1.2
            search_step : 0.02
            max_percentile : 0.999999
            min_percentile : 0.999999
            asymmetric : false
        }
    }
}
tensor_quantize {
    layer_name : "Opname"
    input_index : 0
    ifmr_quantize : {
        search_range_start : 0.7
        search_range_end : 1.3
        search_step : 0.01
        min_percentile : 0.999999
        asymmetric : false
    }
}
tensor_quantize {
    layer_name : "Opname"
    input_index : 0
}
```

# 10 FAQ

开发环境架构为Arm ( aarch64 ) 时模型转换耗时较长  
使用AIPP色域转换模型时如何判断视频流的格式标准  
如何确定原始框架网络模型中的算子与昇腾AI处理器支持的算子的对应关系  
模型中存在不支持量化的层，量化模型失败  
量化模型时模型输入大小过大，AI Core执行任务失败，量化模型失败  
量化模型时校准集数据大小与模型输入大小不匹配，量化模型失败

## 10.1 开发环境架构为 Arm ( aarch64 ) 时模型转换耗时较长

### 问题现象

若开发环境操作系统以及架构为Arm ( aarch64 )，发现模型转换的耗时较长。

### 解决办法

可以使用numactl工具指定CPU核后进行模型转换，步骤如下：

1. 以ATC安装用户登录开发环境，执行**su root**命令切换到root用户。
2. 确保开发环境已连接网络后，执行以下命令安装numactl工具。  
`yum -y install numactl`
3. 切换到ATC安装用户执行如下命令，通过**numactl -C**指定编号16到31的CPU核来处理模型转换操作。

此处建议指定编号16到31的CPU核，处理性能更好，用户也可以根据实际情况修改。

```
numactl -C 16-31 --localalloc <args>
```

<args>请替换为具体使用ATC模型转换命令。

## 10.2 使用 AIPP 色域转换模型时如何判断视频流的格式标准

### 问题现象

使用AIPP色域转换模型时无法判断视频流的格式标准。

### 解决办法

此处以第三方ffprobe工具为例，演示如何进行判断。

**步骤1** 从官方路径下载工具及相关文档：<https://www.ffmpeg.org/ffprobe-all.html#Description>。

**步骤2** 通过 `ffprobe -show_frames filename` 参数获取对应视频信息。

**参数功能：** Show information about each frame and subtitle contained in the input multimedia stream. The information for each single frame is printed within a dedicated section with name "FRAME" or "SUBTITLE".

**步骤3** 通过结果中如下信息确认是哪种视频标准。

"color\_range": "tv" 或者 "pc"

"color\_space": "bt709" 或者 "bt601"

其中tv代表：tv表示 limited，即narrow range。pc代表：pc表示full，即wide range。

例如查询结果为color\_range=tv，color\_space=bt709，则代表BT-709，NARROW。

#### 说明

如命令变化，请以工具官方说明为准。

----结束

## 10.3 如何确定原始框架网络模型中的算子与昇腾 AI 处理器支持的算子的对应关系

### 问题现象

用户使用精度比对工具或者性能比对工具进行算子精度或者性能分析时，若发现某些算子精度或者性能有问题，可能会考虑使用ATC工具中的某些参数调整算子的计算精度后，重新进行模型转换然后推理，比如通过7.3.3.4 `--modify_mixlist` 参数将有问题的算子配置为黑名单等，该场景下，ATC中的参数要求配置的必须为基于Ascend IR定义的算子的OpType。

那如何获取此类算子的OpType？或者如何通过原始框架网络模型中的算子，来获取我们昇腾AI处理器对应支持的算子的OpType呢？

## 解决办法

- 如果用户正在使用Profiling工具进行算子性能分析，该场景下直接获取昇腾AI处理器支持的算子类型即可，参见《[CANN 开发工具指南（开放态）](#)》中的“性能分析工具”章节手册：
  - 导出summary数据中的“AI Core和AI CPU算子数据”，文件名为“op\_summary\_\*.csv”格式。
  - 该文件中的“OP Type”列即为昇腾AI处理器支持的算子的OpType，从该列中找到有问题的算子即可。
- 如果用户正在使用精度比对工具进行算子精度分析：
  - 参见《[CANN 开发工具指南（开放态）](#)》中的“精度比对工具”章节手册获取精度比对结果文件result\_\*.csv。
  - 根据该文件中的“NPUDump”列找到有问题的算子名，然后到对应dump数据文件中检索对应的OpType。

dump数据的第一段即为昇腾AI处理器支持的算子的OpType，例如下图dump数据中标红部分的算子信息：

```
Pooling.MaxPool2D_1.5.3.1656495909948325
Pooling.MaxPool2D_2.14.3.1656495909974359
Pooling.MaxPool2D_3.23.3.1656495909993443
SoftmaxV2_softmax.42.3.1656495910027798
TransData.trans_TransData_1.3.3.1656495909934641
```

## 10.4 模型中存在不支持量化的层，量化模型失败

### 问题现象

执行ATC模型转换命令时，通过--compression\_optimize\_conf参数配置模型量化（将模型中的权重由浮点数float32量化到低比特整数int8）相关的选项，结果报错提示如下：

```
ATC start working now, please wait for a moment.
[ERROR][ProcessScale][52] Not support scale greater than 1 / FLT_EPSILON.
[ERROR][WtsArqCalibrationCpuKernel][188] ArqQuantCPU scale is illegal.
[ERROR][ArqQuant][301] WtsArqCalibrationCpuKernel of format CO_CI_KH_KW failed.
[ERROR] AMCT(14815,atc.bin):2023-04-14-12:23:19[weight_algorithm.cpp:137]Default/network-DeepLabV3/resnet-Resnet/layer4-SequentialCell/0-Bottleneck/downsample-SequentialCell/0-Conv2d/Conv2D-op311 arq weight fake quant failed!
[ERROR] AMCT(14815,atc.bin):2023-04-14-12:23:19[weight_calibration_pass.cpp:90]Fail to excute WeightFakeQuant without trans!
[ERROR] AMCT(14815,atc.bin):2023-04-14-12:23:19[weight_calibration_pass.cpp:185]layer Default/network-DeepLabV3/resnet-Resnet/layer4-SequentialCell/0-Bottleneck/downsample-SequentialCell/0-Conv2d/Conv2D-op311 run WeightFakeQuantArq failed
[ERROR] AMCT(14815,atc.bin):2023-04-14-12:23:19[graph_optimizer.cpp:43]pass run failed
[ERROR] AMCT(14815,atc.bin):2023-04-14-12:23:19[quantize_api.cpp:227]Do GenerateCalibrationGraph optimizer pass failed.
[ERROR] AMCT(14815,atc.bin):2023-04-14-12:23:19[quantize_api.cpp:363]Generate calibration Graph failed.
[ERROR] AMCT(14815,atc.bin):2023-04-14-12:23:22[inner_graph_calibration.cpp:78]Failed to excute InnerQuantizeGraph failed.
```

### 解决办法

通过报错提示layer xxxxxx run WeightFakeQuantArq failed可知，当前模型中有权重相关的层不支持量化，可以通过配置跳过不支持量化的层。

**步骤1** 增加配置，跳过不支持量化的层。

新增一个配置文件，文件名后缀为.cfg，例如`simple_config.cfg`，文件内容如下，加粗部分为报错提示中不支持量化的层：

```
skip_layers:"Default/network-DeepLabV3/resnet-Resnet/layer4-SequentialCell/0-Bottleneck/  
downsample-SequentialCell/0-Conv2d/Conv2D-op311"
```

同时，在`--compression_optimize_conf`参数指定的量化配置文件中，增加`config_file`参数：

```
calibration:  
{  
  input_data_dir: xxxxxx  
  config_file: simple_config.cfg  
  input_shape: xxxxxx  
  infer_soc: xxxxxx  
}
```

**步骤2** 重新执行模型转换。**步骤3** 重新执行推理。

如果跳过不支持量化的层影响模型推理的结果数据，则需要用户自行调整模型，再重新量化模型。

----结束

## 10.5 量化模型时模型输入大小过大，AI Core 执行任务失败，量化模型失败

### 问题现象

模型转换命令示例如下，模型输入大小与`input_shape`参数指定的shape有关：

```
atc --model=xxxxxx.pb --framework=3 --output=xxxxxx --soc_version=xxxxxx --  
input_shape="input:64,224,224,3" --input_format=NHWC --compression_optimize_conf=config/quant.cfg
```

模型转换时，报错示例如下：

```
[ERROR] AMCT(757013,atc.bin):2023-03-14-14:15:54[model_process.cpp:299]execute model failed, modelId  
is 1, errorCode is 507011  
[ERROR] AMCT(757013,atc.bin):2023-03-14-14:15:54[sample_process.cpp:320]execute inference failed  
[ERROR] AMCT(757013,atc.bin):2023-03-14-14:15:55[sample_process.cpp:275]ACL model infer failed.  
[ERROR] AMCT(757013,atc.bin):2023-03-14-14:15:55[quantize_api.cpp:242]sample process failed  
[ERROR] AMCT(757013,atc.bin):2023-03-14-14:15:55[quantize_api.cpp:378]Do Calibration failed.  
[ERROR] AMCT(757013,atc.bin):2023-03-14-14:15:56[inner_graph_calibration.cpp:77]Failed to excute  
InnerQuantizeGraph failed.  
ATC run failed, Please check the detail log, Try 'atc --help' for more information  
EZ9999: Inner Error!  
EZ9999 Aicore kernel execute failed, device_id=0, stream_id=11, report_stream_id=2, task_id=209,  
flip_num=0, fault kernel_name=17786444594805609729-1_0_1_vgg_16/conv1/conv1_2/Conv2D_histo,  
program id=206, hash=1846532111878224358.[FUNC:GetError][FILE:stream.cc][LINE:1131]  
  TraceBack (most recent call last):  
    Model synchronize execute failed, model_id=1![FUNC:GetStreamToSyncExecute][FILE:model.cc]  
[LINE:630]  
  rtModelExecute execute failed, reason=[the model stream execute failed][FUNC:FuncErrorReason]  
[FILE:error_message_manage.cc][LINE:49]  
[Exec][Model]Execute model failed, ge result[507011], modelId[1][FUNC:ReportCallError]  
[FILE:log_inner.cpp][LINE:161]  
[Exec][Model]modelId[1] execute failed, result[507011][FUNC:ReportInnerError][FILE:log_inner.cpp]  
[LINE:145]  
  An unknown error occurred. Please check the log.
```



## 解决办法

AI Core执行任务失败，猜测可能是因为input\_shape参数处的Batch size值过大，导致AI Core上的算子执行失败。

可将input\_shape参数处的Batch size值调小，例如：--  
input\_shape="input:8,224,224,3"，调整参数值之后，再重新转换模型。

## 10.6 量化模型时校准集数据大小与模型输入大小不匹配，量化模型失败

### 问题现象

执行ATC模型转换命令时，通过--compression\_optimize\_conf参数配置模型量化（将模型中的权重由浮点数float32量化到低比特整数int8）相关的选项，结果报错提示如下：

```
[ERROR] AMCT(21177,atc.bin):2023-04-14-14:43:17[utils_acl.cpp:133]input image size[1579014] is not equal to model input size[3158028]
[ERROR] AMCT(21177,atc.bin):2023-04-14-14:43:17[sample_process.cpp:234]memcpy device buffer failed
[ERROR] AMCT(21177,atc.bin):2023-04-14-14:43:17[sample_process.cpp:298]execute PreProcess failed
[ERROR] AMCT(21177,atc.bin):2023-04-14-14:43:17[sample_process.cpp:275]ACL model infer failed.
[ERROR] AMCT(21177,atc.bin):2023-04-14-14:43:17[quantize_api.cpp:240]sample process failed
[ERROR] AMCT(21177,atc.bin):2023-04-14-14:43:17[quantize_api.cpp:376]Do Calibration failed.
[ERROR] AMCT(21177,atc.bin):2023-04-14-14:43:20[inner_graph_calibration.cpp:78]Failed to excute InnerQuantizeGraph failed.
...
ATC run failed, Please check the detail log, Try 'atc --help' for more information
```

### 解决办法

通过报错提示input image size[xxxxxx] is not equal to model input size[xxxxxx]可知，量化模型时校准集数据大小与模型输入大小不匹配，不匹配可能是校准集数据的shape与模型输入shape不一致，也有可能是校准集数据的数据类型与模型输入数据类型不一致，因此需要逐一排查shape、数据类型。

量化时，模型输入shape通过量化配置文件中的input\_shape参数配置，模型输入数据类型需由用户从获取模型的网站获取或通过第三方软件打开模型文件查看。