密级状态：绝密(　　)　　秘密(　　)　　内部(　)　　公开(√)

RKNN Toolkit 问题排查手册

（技术部，图形计算平台中心）

文件状态： [　] 正在修改 [√] 正式发布	当前版本：	1.7.5
	作　　者：	HPC
	完成日期：	2023-07-31
	审　　核：	熊伟
	完成日期：	2023-07-31

瑞芯微电子股份有限公司

Rockchip Electronics Co., Ltd

更新记录

版本	修改人	修改日期	修改说明	核定人
0.9	HPC	2019-04-01	初稿	熊伟
1.0	HPC	2019-07-18	增加一些常见问题	熊伟
1.1	HPC	2019-08-22	增加深度神经网络模型设计建议	熊伟
1.2	HPC	2019-10-11	增加一些常见问题	熊伟
1.3	HPC	2019-12-25	增加一些常见问题，重新整理目录结构	熊伟
1.3.2	HPC	2020-04-13	增加一些常见问题	熊伟
1.4.0	HPC	2020-08-13	增加一些常见问题，完善问题排查步骤	熊伟
1.6.0	HPC	2020-12-31	增加一些常见问题	熊伟
1.6.1	HPC	2021-05-21	增加一些常见问题	熊伟
1.7.0	HPC	2021-08-08	增加一些常见问题	熊伟
1.7.1	HPC	2021-11-17	1.增加一些常见问题 2.文档结构调整	熊伟
1.7.3	HPC	2022-08-08	1.增加一些常见问题 2.文档结构调整	熊伟
1.7.5	HPC	2023-07-31	增加一些常见问题	熊伟

目 录

1 概述.....3

2 RKNN TOOLKIT 和 RKNPU 的版本关系.....4

3 RKNN TOOLKIT 安装问题.....5

4 模型转换常用参数说明.....8

5 各框架模型加载问题.....16

6 模型量化问题.....27

7 模拟器推理及连板推理、板端推理的说明.....32

8 模型评估常见问题.....34

9 深度神经网络模型设计建议.....52

10 附录.....54

1 概述

以下文档如无特殊说明，则仅适用于 RKNN Toolkit。由此工具转换出的模型适用于 RK3399Pro、RK1806、RK1808、RV1126、RV1109 平台。

参考本文档时，请确认环境中的 RKNN Toolkit 版本，否则部分内容可能不适用。

2 RKNN Toolkit 和 RKNPU 的版本关系

RKNN Toollkit 是基于 Python 语言实现的模型转换工具，可将其他训练框架导出的模型转为 RKNN，并提供较为有限的推理接口，协助用户测试模型转换效果。工程链接为：https://github.com/rockchip-linux/rknn-toolkit

RKNPU 是板端的组件，提供 NPU 驱动，并基于 C 语言，提供模型加载、模型推理等功能。相比 RKNN Toolkit 工具，RKNPU 的 C API 推理接口用法更灵活、性能更优。工程链接为：https://github.com/rockchip-linux/rknpu

RKNN Toolkit，RKNPU 具有一致的版本号。不同版本之间存在或多或少的不兼容问题，为了避免不必要的麻烦，推荐用户使用同一版本的 RKNN Toolkit、RKNPU；版本的更新通常包含 bug 修复与性能优化，建议用户使用最新版本。

3 RKNN Toolkit 安装问题

本章主要覆盖以下常见的工具安装问题：

1）安装 RKNN Toolkit 时出现未定义符号 PyFPE_jbuf 错误

2）RKNN Toolkit 依赖的环境限制太严格，导致无法成功安装

3）Tensorflow 依赖说明

4）PyTorch 依赖说明

5）ONNX 的依赖说明

6）RK3399Pro 非 Debian 的 Linux 系统是否可以使用 RKNN Toolkit 或 RKNN Toolkit Lite

7）RKNN Toolkit 安装包命名规则

8）模型训练环境和 RKNN Toolkit 使用环境

3.1 安装 RKNN Toolkit 时出现未定义符号 PyFPE_jbuf 错误

错误日志如下：

```
Undefined symbol: PyFPE_jbuf
```

出现该错误的原因是 Python 环境不干净，比如在两个不同的路径里都安装了 numpy。建议重新创建一个干净的 Python 环境后再试。

3.2 RKNN Toolkit 依赖的环境限制太严格，导致无法成功安装

在所有依赖库都已安装、但部分库的版本和要求不匹配时，可以尝试在安装指令后面加上 --no-deps 参数，取消安装 python 库时的环境检查。如 pip install rknn-toolkit --no-deps。

3.3 Tensorflow 依赖说明

RKNN Toolkit 的模型量化功能依赖于 TensorFlow 库。RKNN Toolkit 推荐使用的 TensorFlow 版本是 1.14.0 或 2.2.0。由于 TensorFlow 各版本之间的兼容性较差，其他版本可能会造成 RKNN Toolkit 工作异常。另外在加载 TensorFlow 模型时，建议导出模型使用的 TensorFlow 版本，与 RKNN Toolkit 依赖的 TensorFlow 版本一致。

对于 Tensorflow 版本引发的问题，通常会体现在 load_tensorflow、build 阶段，且出错信息会指向依赖的 TensorFlow 路径。

3.4 PyTorch 依赖说明

RKNN Toolkit 的 PyTorch 模型加载功能，依赖于 PyTorch。将 PyTorch 的模型分为浮点模型和已量化模型（包含 QAT 及 PTQ 量化模型）。对于浮点模型，PyTorch 1.5.1 与 PyTorch 1.6.0 导出的模型存在较大差异，两者混用时容易出现兼容性问题，比如使用 PyTorch 1.6.0 加载 1.5.1 版本导出的模型，可能会有部分参数加载异常。对于已量化模型(QAT、PTQ)，我们推荐使用 PyTorch 1.10 导出模型，并将 RKNN Toolkit 依赖的 PyTorch 版本升级至 1.10。另外在加载 PyTorch 模型时，建议导出原模型的 PyTorch 版本，要与 RKNN Toolkit 依赖的 PyTorch 版本一致。

推荐使用的 PyTorch 版本为 1.6.0、1.9.0 或 1.10 版本。

3.5 ONNX 的依赖说明

RKNN Toolkit 的 ONNX 模型加载功能，依赖 ONNX。由于 ONNX 各版本之间的兼容性较好，目前没发现版本导致的问题。

3.6 RK3399Pro 非 debian 的 Linux 系统是否可以使用 RKNN Toolkit 或 RKNN Toolkit Lite

如果目标系统的 Python 版本和 RKNN Toolkit 或 Lite 所要求的 Python 版本一致，则一般是可以使用的。

因为模型转换阶段需要较多的 CPU 和内存资源，不建议在开发板上安装 RKNN Toolkit 进行模型转换。建议在 PC 端进行模型转换和评估，部署阶段用 RKNN Toolkit Lite 提供的 Python 接口或者 RK3399Pro_npu(https://github.com/airockchip/RK3399Pro_npu)工程提供的 C/C++接口。

3.7 RKNN Toolkit 安装包命名规则

以 1.7.3 版本的发布件为例，RKNN Toolkit wheel 包命令规则如下：

- rknn_toolkit：工具名称
- 1.7.3：版本号
- cp<xx>-cp<xx>m：适用的 Python 版本，例如 cp36-cp36m 表示适用的 Python 版本是 3.6。
- linux_x86_64：系统架构，对于适用 x86 架构 cpu 的 pc 而言，这个字段有以下几种情况，linux_x86_64，Linux 系统；macosx_10_15_x86_64，MacOS 系统；win_amd64，Windows 操作系统。对于 ARM CPU 而言（Rockchip 的开发板都使用 ARM 架构），linux_aarch64 表示的是 ARM64 位架构的 Linux 系统。

在安装工具时，请确认自己所用的操作系统、CPU 架构和 Python 版本，再选择对应的工具包进行安装。否则安装会失败。

3.8 模型训练环境和 RKNN Toolkit 使用环境

建议训练环境中深度学习框架（如 PyTorch、TensorFlow 等）的版本和 RKNN Toolkit 中使用的版本一致。比如 PyTorch 1.9.0 训练的模型，在 RKNN Toolkit 环境中也安装 1.9.0 版本的 torch。

4 模型转换常用参数说明

本章节主要覆盖模型转换阶段常用参数的使用说明：

1） 根据模型确定参数

2） 各平台对应的模型是否兼容

3） 量化数据的格式及要求

4） 多输入模型 dataset 文件的填写方式

5） 确认 reorder 参数

6） mean/std、reorder 的计算顺序

7） mean/std 数值范围

8） 模型是非 3 通道输入或多输入时，mean/std 的设置问题

9） 量化类型的选取

10） 量化参数矫正算法和量化图片数量的选取

11） 量化模型与非量化模型，推理时输入输出的差异

12） 离线预编译和在线预编译

13） 离线预编译模型推理结果异常且耗时超过 20 秒

14） 更新 rknpu 驱动后使用在线预编译卡住，或者预编译得到的模型模型只有 2KB 大小，日志提示出错

15） RKNN Toolkit2 构建的 RKNN 模型可以在 RK3399Pro 平台上使用吗

16） 参数 force_builtin_perm 的作用

17） 混合量化模型能不能导出预编译模型

4.1 根据模型确定参数

模型转换时，config 和 build 接口会影响模型转换结果。load_onnx，load_tensorflow 指定输入输出节点，会影响模型转换结果。load_pytorch, load_tensorflow, load_mxnet 指定输入的尺寸大小会影响模型转换结果。

可以参考以下基本思路进行模型转换：

1. 准备量化数据，提供 dataset.txt 文件

2. 确定模型要使用的平台，如 RK1808、RV1109，并填写 config 中的 target_platform 参

数。

3. 当输入是 3 通道的图像，且量化数据采用的是图片格式(如 jpg, png 格式)时，需要确认模型的输入是 RGB 还是 BGR，以决定 config 接口中 reorder 参数的值。

4. 确认模型训练使用的归一化参数，以决定 config 接口中的 mean/std 参数的值。

5. 确认模型输入的尺寸信息，填入 load 接口相应参数中，如 load_pytorch 接口中的 input_size_list 参数。

6. 确认模型要量化比特数，以决定 config 接口中的 quantized_dtype 参数的值。**不对模型进行量化或加载的是已量化模型时可以忽略此步骤**。

7. 确认模型量化时使用的量化参数优化算法，以决定 config 接口中 quantized_algorithm 参数的值。**不对模型进行量化或加载已量化模型时可以忽略此步骤**。

8. 确认是否对模型进行量化，以决定 build 接口中 do_quantization 参数的值。选择对模型进行量化时，需要额外填写 build 接口中的 dataset 参数，指定量化矫正数据。

9. 模型使用离线预编译时，设置 build 接口中的 pre_compile 参数。**注：该参数在 1.7.5 版本已废弃，请使用在线预编译功能。**

4.2 各平台对应的模型是否兼容

对于 target_platform 设置的平台参数，兼容性关系如下：

- RK3399Pro、RK1808 平台使用的模型是相互兼容的；
- RV1126、RV1109 平台使用的模型是相互兼容的。

4.3 量化数据的格式及要求

量化数据的格式有两种选择，一种是图片格式（jpg，png），RKNN Toolkit 会调用 TensorFlow 接口进行读取；另一种是 npy 格式，RKNN Toolkit 会调用 numpy 接口进行读取。

当使用 npy 文件作为量化数据、且模型输入为 4 维时，需要将数据转为 NHWC 排布。例如输入为 1x3x112x224，使用 np.transpose，将数据转为 1x112x224x3 后保存为 npy 文件。如果是非 4 维输入，则跳过此步骤。

对于非 RGB/BGR 图片输入的模型，建议使用 numpy 的 npy 格式提供量化数据。

4.4 多输入模型 dataset.txt 文件的填写方式

模型量化需要用 dataset.txt 文件指定量化数据的路径。规则为一行作为一组输入，模型存在多输入时，多个输入写在同一行，并用空格隔开。

如单输入模型，使用两组量化数据：

```
sampleA.npy
sampleB.npy
```

如三个输入的模型，两组量化数据按如下方式填写：

```
sampleA_in0.npy sampleA_in1.npy sampleA_in2.npy
sampleB_in0.npy sampleB_in1.npy sampleB_in2.npy
```

4.5 确认 reorder 参数

当采用图片作为量化数据时，需要考虑设置 reorder 参数。

模型采用 RGB 图片进行训练时，则 reorder 参数设为‘0 1 2’。且在使用 Python inference 接口或 RKNPU C API 进行推理时，输入 RGB 图片。

模型采用 BGR 图片进行训练时，则 reorder 参数设为‘2 1 0’。且在使用 Python inference 接口或 RKNPU C API 进行推理时，输入 RGB 图片。

若量化数据采用 numpy 的 npy 格式，则建议不要使用 reorder 参数，避免产生使用混乱的问题。

4.6 mean/std、reorder 的计算顺序

对于 RKNN Toolkit 及 RKNPU C API，计算顺序如下：先对输入数据进行 reorder 操作，再进行减均值(mean)、除标准差(std)的操作。

如果原始框架的作用顺序和 RKNN Toolkit 一致，则 mean、std 以同样的顺序填写。

如果原始框架的作用顺序是先减均值除标准差、再进行通道转换(如 RGB 转 BGR 操作)，则需要对 mean, std 的数值顺序做个转换。

4.7 mean/std 数值范围

使用图片量化时，RKNN Toolkit 使用 TensorFlow 接口加载数据，加载后数值范围在 0～255 之间。如果原始框架加载数据的数值范围是 0～1 之间(常见于 PyTorch)，则需要对 mean、std 的数值乘以 255。

4.8 模型是非 3 通道输入或多输入时，mean/std 的设置问题

RKNN Toolkit 中 mean 和 std 的设置格式是一致的。这里以 mean 为例子。

假设输入有 N 个通道，则 channel_mean_value 的值为 [[channel_1, channel_2, channel_3, channel_4, ..., channel_n]]。

存在多输入时，则 channel_mean_value 的值为 [[channel_1, channel_2, channel_3, channel_4, ..., channel_n], [channel_1, channel_2, channel_3, channel_4, ..., channel_n]]

4.9 量化类型的选取

RKNN Toolkit 中量化类型 quantized_dtype 参数的可选值包括‘asymmetric_affine-u8’, ‘dynamic_fixed_point-i16’和‘dynamic_fixed_point-i8’，默认使用 u8 量化。其中 i8/u8 量化的推理速度更快一些，大部分时候 u8 量化的精度会比 i8 更高一点；而 i16 量化的精度损失较少，大部分时候没有损失（有损失的话可以反馈给 Rockchip NPU 相关部门）。

推荐先使用 u8 进行量化，如果精度不满足使用要求，再使用 i16 量化。

4.10 量化参数矫正算法和量化图片数量的选取

RKNN Toolkit 中量化矫正算法 quantized_algorithm 参数提供两种算法进行参数矫正，分别为‘normal’和‘mmse’，默认使用 normal。Normal 为常规的量化参数矫正算法；而 MMSE 会迭代中间层的计算结果，对权重数值进行一定范围的裁剪，以获得更高的推理精度。使用 MMSE 不一定能提升量化精度，但相比 normal 方式，量化时会占用更多的内存、耗费更长的模型转换时间。

建议先使用‘normal’算法，如果量化效果不佳，可尝试使用‘mmse’算法。

使用 normal 算法时，推荐给出 200-500 组数据进行量化。使用 MMSE 量化时，推荐使

用 20 组数据进行量化。

注：量化图片不需要标签或标注信息。RKNN Toolkit 使用量化矫正数据，主要是为了计算推理过程中，每一层计算结果的最大值、最小值，并以此计算、调整量化参数。在这个过程中，并不需要标签或者标注信息等。

4.11 量化模型与非量化模型，推理时输入输出的差异

调用常规 RKNPU C API 时（指不使用 pass_through、zero_copy 的方式调用 C API），输入数据的数据类型（如 uint8 数据，float 数据）与模型的量化与否没有关系。输出数据的数据类型可以选择自动处理成 float32 格式，也可以选择直接输出模型推理结果，此时数据类型与输出节点的数据类型一致。使用 Python 推理接口会有点差异，具体关系如下表：

表 4-1 量化与非量化模型输入、输出关系表

模型量化后	Python 推理(rknn.inference)	C API 推理(rknn.run) (非 pass_through、zero_copy)
输入类型是否有限制	无限制。 rknn.inference 有 data_type 参数，可以根据实际给的输入，指定 int8, uint8, int16, float16, float32 的数据类型，默认为 uint8。指定后，会将输入自动转成 RKNN 模型需要的数据格式。	无限制。 rknn inputs 的 rknn_tensor_type 参数可以根据实际输入，指定 RKNN_TENSOR_FLOAT32、RKNN_TENSOR_FLOAT16、RKNN_TENSOR_INT8、RKNN_TENSOR_UIN8、RKNN_TENSOR_INT16。指定后，会将输入自动转成 RKNN 模型需要的数据格式。
输出类型是否变化	无变化。 无论模型量化与否，Python 的 inference 接口总是返回 float 类型输出。无法选择其他数据类型。	有变化。 RKNPU C API 的 rknn outputs attr，可以设置 want_float=1，得到 float 类型的输出。而量化后，可以设置 want_float=0，此时可以输出最后一个节点的原始输出数据，如 u8 量化时，输出 uint8 数据。
输入 format 是否有变化（NCHW，NHWC）	无变化。 无论模型量化与否，rknn.inference 接口的 data_format 参数，可以根据需要可设置为 NCHW 或 NHWC	无。 无论模型量化与否，rknn inputs 结构体的 rknn_tensor_format 参数，可以根据需要设置为 NCHW 或 NHWC

4.12 离线预编译和在线预编译

预编译功能会提前将模型转成相应硬件上的运行指令，减少模型在板端的初始化（init_runtime 或 rknn_init）时间。预编译后，模型将无法使用模拟器进行推理。预编译模

型在不同的 NPU 驱动版本上，可能存在不兼容的情况。

离线预编译：使用 rknn.build 接口构建 RKNN 模型时，设置 pre_compile 参数为 True，此时使用模拟器对 RKNN 模型进行预编译。

在线预编译：使用 rknn.build 接口构建普通模型后，调用 export_rknn_precompile_model 接口，将模型推送至开发板，利用板端的驱动对 RKNN 模型进行预编译。具体可以参考 examples/common_function_demos/export_rknn_precompile_model 的例子。

由于模拟器可能与实际驱动有差异，推荐使用在线预编译。

注：从 1.7.5 版本开始，离线预编译功能已经被禁用，无法使用。

4.13 离线预编译模型推理结果异常且耗时超过 20 秒

出现这种现象一般是模拟器驱动 bug 导致。请尝试使用在线预编译。在线预编译的方法参考问题 4.12。

4.14 更新 rknpu 驱动后使用在线预编译接口卡住，或者预编译得到的模型只有 2KB 大小，日志提示出错

出现这种现象是因为使用某些版本的 adb 推文件时，没有保留二进制文件的可执行权限导致的。请在开发板上检查以下两个文件有可执行权限：

```
[root@RV1126_RV1109:/]# ls -lh /usr/bin/rknn_server
-rwxrwxrwx 1 root root 283K Aug 13    2022 /usr/bin/rknn_server

[root@RV1126_RV1109:/]# ls -lh /usr/lib/npu/rknn/memory_profile
-rwxrwxrwx 1 root root 9.6K Aug    1    2022 /usr/lib/npu/rknn/memory_profile
```

4.15 RKNN Toolkit2 构建的 RKNN 模型可以在 RK3399Pro 平台上使用吗？

不可以。

RKNN Toolkit2 构建的 RKNN 模型适用于 RK3566 / RK3568 / RK3588 / RK3588S / RV1103 / RV1106 等平台。RKNN Toolkit 构建的 RKNN 模型适用于 RK1806 / RK1808 /

RK3399Pro / RV1109 / RV1126 等平台。在 RK3399Pro 平台请使用 RKNN Toolkit 构建 RKNN 模型。

RKNN Toolkit 工具的使用说明请参考以下工程：

https://github.com/rockchip-linux/rknn-toolkit

RKNN Toolkit2 工具的使用说明请参考以下工程：

https://github.com/rockchip-linux/rknn-toolkit2

4.16 参数 force_builtin_perm 的作用

该参数的作用在模型转 RKNN 的过程中，为输入插一层 NHWC 转 NCHW 的 Transpose 算子，使得 RKNN 模型可以接受 NHWC 格式的数据作为输入。

对于 Caffe / Darknet / MXNet / ONNX / Pytorch 等框架的模型，在转成 RKNN 后，默认情况下，4 维输入数据的排列顺序是 NCHW。在推理时，如果应用获取到的数据是 NHWC 排列的数据（例如 OpenCV / CImg / STBI 等库读取图片都是按这个格式排列的），则需要手动做一次数据格式的重排，或者设置 data_fmt 为 NHWC，让 librknn_api 或 librknn_runtime 做这个工作，本质上都是在 CPU 上做。如果输入比较大，这一步的耗时就会比较多，加重 CPU 负担。通过 force_builin_perm 参数，可以把这个操作移到模型中去做，让 NPU 做加速，同时还可以减少 CPU 负担。

这个参数对 Keras, TensorFlow 或 TFLite 的模型无效，因为默认情况下，这几个框架的输入数据排列格式就是 NHWC 的，不需要转换。

4.17 混合量化模型能不能导出预编译模型

可以。

有两种预编译方法。一是在 hybrid_quantization_step2 中设置 pre_compile 参数为 True，做离线预编译（注：从 1.7.5 版本开始，无法使用这种方法）。二是先导出普通 RKNN 模型，再用 export_rknn_precompile_model 接口去做在线预编译。

推荐使用第二种方法，先导出普通 RKNN 模型，再做在线预编译。

在线预编译可以参考以下示例:

https://github.com/rockchip-linux/rknn-toolkit/tree/master/examples/common_function_demos/export_rknn_precompile_model

5 各框架模型加载问题

本章节主要覆盖以下模型加载问题：

1） RKNN Toolkit 支持的深度学习框架和对应版本

2） 各框架的 OP 支持列表

3） Caffe 模型转换常见问题

4） Darknet 模型转换常见问题

5） Keras 模型转换常见问题

6） ONNX 模型转换常见问题

7） Pytorch 模型转换常见问题

8） Tensorflow 模型转换常见问题

9） 加载模型出错时的排查步骤

5.1 RKNN Toolkit 支持的深度学习框架和对应版本

RKNN Toolkit 对各深度学习框架的支持情况如下表 5-1 和表 5-2 所示：

表 5-1 RKNN Toolkit 深度学习框架支持情况

RKNN Toolkit	Caffe	Darknet	Keras	MXNet
1.6.0	1.0	Commit ID：810d7f7	>=2.1.6-tf	>=1.4.0, <=1.5.1
1.6.1	1.0	Commit ID：810d7f7	>=2.1.6-tf	>=1.4.0, <=1.5.1
1.7.0	1.0	Commit ID：810d7f7	>=2.1.6-tf	>=1.4.0, <=1.5.1
1.7.1	1.0	Commit ID：810d7f7	>=2.1.6-tf	>=1.4.0, <=1.5.1
1.7.3	1.0	Commit ID：810d7f7	>=2.1.6-tf	>=1.4.0, <=1.5.1
1.7.5	1.0	Commit ID：810d7f7	>=2.1.6-tf	>=1.4.0, <=1.5.1

表 5-2 RKNN Toolkit 深度学习框架支持情况续

RKNN Toolkit	ONNX	Pytorch	TensorFlow	TF Lite
1.6.0	1.6.0	>=1.0.0, <=1.6.0	>=1.10.0, <=2.0.0	Schema version = 3
1.6.1	1.6.0	>=1.0.0, <=1.6.0	>=1.10.0, <=2.0.0	Schema version = 3
1.7.0	1.6.0	>=1.0.0, <=1.6.0	>=1.10.0, <=2.0.0	Schema version = 3
1.7.1	1.6.0	>=1.5.1, <=1.9.0	>=1.10.0, <=2.0.0	Schema version = 3
1.7.3	1.6.0	>=1.5.1, <=1.10.0	>=1.10.0, <=2.2.0	Schema version = 3
1.7.5	1.10.0	>=1.5.1, <=1.10.0	>=1.10.0, <=2.2.0	Schema version = 3

注：

1. 依照语义版本，用某一版本 TensorFlow 写出的任何图或检查点，都可以通过相同主要版本中更高（次要或补丁）版本的 TensorFlow 来进行加载和评估，所以理论上，1.14.0 之前版本的 TensorFlow 生成的 pb 文件，RKNN Toolkit 1.0.0 及之后的版本都是支持的。关于 TensorFlow 版本兼容性的更多信息，可以参考官方资料：https://www.tensorflow.org/guide/versions
2. 因为 tflite 不同版本的 schema 之间是互不兼容的，所以构建 tflite 模型时使用与 RKNN Toolkit 不同版本的 schema 可能导致加载失败。目前 RKNN Toolkit 使用的 tflite schema 是基于 TensorFlow Lite 官方 GitHub master 分支上的如下提交: 0c4f5dfea4ceb3d7c0b46fc04828420a344f7598 。具体的 schema 链接如下：https://github.com/tensorflow/tensorflow/commits/master/tensorflow/lite/schema/schema.fbs
3. RKNN Toolkit 所使用的 caffe protocol 有两种，一种是基于 berkeley 官方修改的 protocol，一种是包含 LSTM 层的 protocol。其中基于 berkeley 官方修改的 protocol 来自：https://github.com/BVLC/caffe/tree/master/src/caffe/proto ，commit 值为 21d0608，RKNN Toolkit 在这个基础上新增了一些 OP。而包含 LSTM 层的 protocol 参考以下链接：https://github.com/xmfbit/warpctc-caffe/tree/master/src/caffe/proto，commit 值为 bd6181b。这两种 protocol 通过 load_caffe 接口中的 proto 参数指定。
4. ONNX release version 和 opset version、IR version 之间的关系参考官网说明：https://github.com/microsoft/onnxruntime/blob/v1.10.0/docs/Versioning.md

表 5-3 ONNX release、opset 和 IR 版本间的对应关系

ONNX release version	ONNX opset version	Supported ONNX IR version
1.6.0	11	6
1.7.0	12	7

5. Darknet 官方 Github 链接：https://github.com/pjreddie/darknet，RKNN Toolkit 现在的转换规则是基于 master 分支的最新提交（commit 值：810d7f7）制定的。
6. RKNN Toolkit 目前主要支持以 TensorFlow 为 backend 的 Keras 版本，所测 Keras 版本均为 TensorFlow 自带的 Keras。
7. 使用 RKNN Toolkit 时，Torch 所用版本建议和导出 PyTorch 模型时所用的版本一致。
8. RKNN Toolkit 早期版本对各深度学习框架的支持情况请参考《Rockchip_User_Guide_RKNN_Toolkit_CN》文档。

5.2 各框架的 OP 支持列表

RKNN Toolkit 对不同框架的支持程度有差异，详细信息可以参考以下目录中的 RKNN_OP_Support 文档: https://github.com/rockchip-linux/rknn-toolkit/blob/master/doc/

5.3 Caffe 模型转换常见问题

5.3.1 转换模型时，提示过时的 Caffe 输入用法

错误日志信息如下：

```
Deprecated caffe input usage
```

出现该错误的原因是模型使用了旧版的 Caffe 输入层写法，需要将输入层修改成如下格式。

```
layer {
  name: "data"
  type: "Input"
  top: "data"
  input_param {
    shape {
      dim: 1
      dim: 3
      dim: 224
      dim: 224
    }
  }}
```

5.3.2 转换模型时，出现 caffe.PoolingParameter 没有 round_mode 字段的错误

错误日志如下：

```
Message type "caffe.PoolingParameter" has no field named "round_mode"
```

Pool 层的 round_mode 字段不能识别，可以改成 ceil_model，比如原来是 round_mode: CEIL，那么可以删掉（默认 ceil_mode 为 True）或者改成 ceil_mode:True。

5.3.3 在转换 Caffe 或者其他框架模型时，出现非法层名的错误

详细的错误日志如下：

```
T       raise ValueError("'%s' is not a valid scope name" % name)
T   ValueError: '_plus0_17' is not a valid scope name
```

出现这种情况是因为层名'_plusxxx' 用下划线开头不合法，要遵循 TensorFlow 的命名规则：

```
[A-Za-z0-9.][A-Za-z0-9_.\\-/]* (for scopes at the root)
[A-Za-z0-9_.\\-/]* (for other scopes)
```

5.3.4 Caffe 版本的 SSD 转换失败，出现非法 tensor id 的错误

具体的错误日志如下：

> “Invalid tensor id(1), tensor(@mbox_conf_flatten_188:out0)”

出现这个错误是因为 RKNN Toolkit 不支持 DetectionOutput 层。这是后处理中用到的算子，可以删掉，改成在 CPU 做。

5.3.5 Caffe 版本的 SSD 模型去掉 DetectionOutput 后应该有 3 个输出，但 RKNN 推理时实际只返回两个输出

这个缺失的输出是先验框，先验框信息不受训练、推理影响，数值始终不变。为了提高性能，RKNN Toolkit 在模型中将相关的层去掉。而要得到该先验框数据，可以在训练阶段将先验框的数据保存下来，或者用 Caffe 框架推理得到先验框数据。

5.3.6 转换 py-faster-rcnn 模型时出现非法 tensor 的错误

具体的错误日志如下：

> “ValueError: Invalid tensor id(1), tensor(@rpn_bbox_pred_18:out0)”

与官方相比需要修改 prototxt 中的'proposal'层，修改后（左边）与修改前的对比：

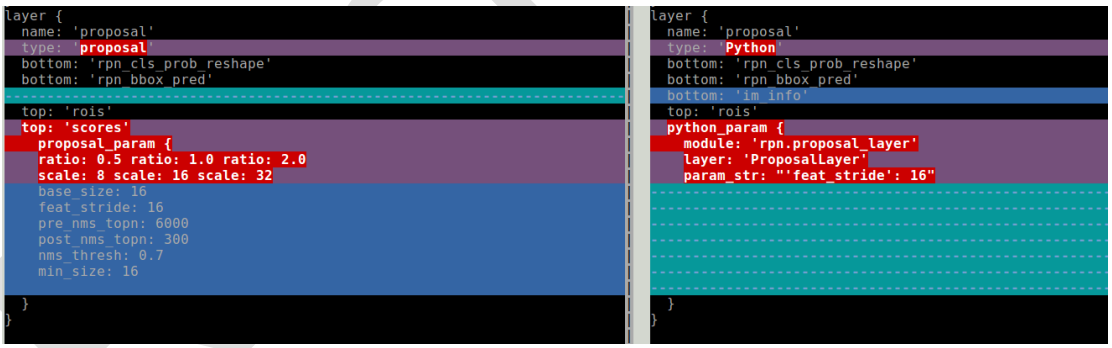

图 5-1 py-faster-rcnn proposal 层修改前后对比

修改后的 proposal 层定义如下：

```
layer {
  name: 'proposal'
  type: 'proposal'
  bottom: 'rpn_cls_prob_reshape'
  bottom: 'rpn_bbox_pred'
  top: 'rois'
  top: 'scores'
    proposal_param {
    ratio: 0.5 ratio: 1.0 ratio: 2.0
    scale: 8 scale: 16 scale: 32
    base_size: 16
    feat_stride: 16
    pre_nms_topn: 6000
    post_nms_topn: 300
    nms_thresh: 0.7
    min_size: 16
  }
}
```

5.3.7 Caffe 是否支持 Upsample 层

RKNN Toolkit 1.4.0 版本之前 Upsample 需要用 Resize 来代替，1.4.0 及之后的版本直接支持 Upsample 层。

5.3.8 Caffe 模型转换时出现不支持的模型版本错误

详细错误如下：

```
E Not supported caffenet model version(v0 layer or v1 layer)
E Catch exception when loading caffe model: ../model/vgg16.prototxt!
```

出现这个错误的原因是 Caffe 模型的版本太旧，需要更新模型。

模型更新方法如下（以 VGG16 为例）：

1) 从 https://github.com/BVLC/caffe.git 下载 Caffe 源码

2) 编译 Caffe

3) 使用如下命令将模型转为新的格式

```
/build_release/tools/upgrade_net_proto_text    vgg16_old/vgg16.prototxt \
vgg16_new/vgg16.prototxt

./build_release/tools/upgrade_net_proto_binary    vgg16_old/vgg16.caffemodel \
vgg16_new/vgg16.caffemodel
```

5.4 Darknet 模型转换常见问题

5.4.1 运行 examples/darknet/yolov3 示例时出错，错误信息提示 MultiDiGraph 对象没有 node 属性

具体的错误日志如下：

```
MultiDiGraph object has no attribute node
```

出现这个错误的原因是 networkx 版本太高。解决方法是将 networkx 版本降级到 1.11。

5.5 Keras 模型转换常见问题

5.5.1 调用 keras load_model_from_hdf5 接口报错导致 load_keras 出错

在加载 h5 模型时，出现如下错误：

```
E Catch exception when loading keras model: ./xception.h5!
E Traceback (most recent call last):
E   File "rknn/base/RKNNlib/converter/convert_keras.py", line 24, in
rknn.base.RKNNlib.converter.convert_keras.convert_keras.__init__
E   File
"/home/jxx/anaconda3/envs/rknn-test/lib/python3.6/site-packages/tensorflow/python/keras/saving/save.py", line 146, in load_model
E     return hdf5_format.load_model_from_hdf5(filepath, custom_objects, compile)
E   File "/home/rk/
envs/rknn-test/lib/python3.6/site-packages/tensorflow/python/keras/saving/hdf5_format.py",
line 210, in load_model_from_hdf5
E     model_config = json.loads(model_config.decode('utf-8'))
E AttributeError: 'str' object has no attribute 'decode'
```

出现该错误是因为当前使用的 TensorFlow 版本还不支持 h5py 3.0.0 及以上的版本。解决方法是降级 h5py，例如使用 2.6.0 版本。

5.6 ONNX 模型转换常见问题

5.6.1 转换时遇到模型 IR 版本过高的错误

具体的错误日志如下：

```
Your model ir_version is higher than the checker`s
```

RKNN Toolkit 1.1.0 及之前的版本只支持 1.3.2 及以下版本 ONNX 导出的模型。RKNN Toolkit 1.4.0 及之前的版本支持 1.4.1 及以下版本 ONNX 导出的模型；RKNN Toolkit 1.6.0 及之后的版本支持 1.6.0 版本 ONNX 导出的模型。

5.6.2 Resize 问题

对于 RKNN Toolkit 1.7.0 及之前的版本，如果 Resize(upsample) OP 在解析的过程中出错，请更新到 RKNN Toolkit 1.7.3 及之后的版本，并在导出 ONNX 的时候选择设置 opset_version=11 或 12。

5.6.3 Slice 问题

对于 RKNN Toolkit 1.7.0 及之前的版本，如果 Slice OP 在解析的过程中出错，请更新到 RKNN Toolkit 1.7.3 及之后的版本。

5.6.4 动态输入问题

调用 load_onnx 接口时出错，提示“Loading dynamic inputs model is not supported, please fix it first”。

出现这个错误是因为 RKNN Toolkit 当前不支持动态输入，在导出 ONNX 模型时要固定输入的 shape。

出现这个问题的另外一种情况是旧版本的 PyTorch 或者 Paddle 等框架在导出 ONNX 时，没有严格遵守 ONNX 的规范，将一些常量也填在了 inputs 表中，导致 RKNN Toolkit 检查输入节点时误将这些常量当成了输入。这个问题可以通过升级 PyTorch 版本，重新导出 ONNX 模型或者更新 RKNN Toolkit 到 1.7.3 及之后的版本来解决。

5.7 Pytorch 模型转换常见问题

在 1.3.0 版本之前，RKNN Toolkit 通过 ONNX 间接支持 Pytorch，因此需要将 Pytorch 先转成 ONNX。在 1.3.0 及之后的版本中，RKNN Toolkit 支持直接加载 Pytorch 模型。如果转换过程遇到问题，请先将 RKNN Toolkit 升级到最新版本。

5.7.1 加载 Pytorch 模型时出现 torch._C 没有_jit_pass_inline 属性的错误

错误日志如下：

```
'torch._C' has no attribute '_jit_pass_inline'
```

请将 PyTorch 升级到 1.6.0 或之后的版本。

5.7.2 Pytorch 模型的保存格式

目前只支持 torch.jit.trace 导出的模型。torch.save 接口仅保存权重参数字典，缺乏网络结构信息，无法被正常导入并转成 RKNN 模型。

5.7.3 转换时遇到类似“assert(tsr.op_type == 'Constant')”的错误

这是 PyTorch 0.4.5 以后的版本引入的问题，在你的模型中，有类似“x = x.view(x.size(0), -1)”这样的语句，需要改成“x = x.view(int(x.size(0)), -1)”。

5.7.4 转换时遇到 PytorchStreamReader 失败的错误

详细错误如下：

```
E Catch exception when loading pytorch model: ./mobilenet0.25_Final.pth!
E Traceback (most recent call last):
……
E       cpp_module = torch._C.import_ir_module(cu, f, map_location, extra_files)
E RuntimeError: [enforce fail at inline container.cc:137]. PytorchStreamReader failed reading zip archive: faild finding central directory frame #0 ……

# 或者：
PytorchStreamReader failed loacting file constants.pkl: file not found
```

出错原因是输入的 PyTorch 模型没有网络结构信息。

通常 pth 只有权重信息，并没有网络结构信息。对于已保存的模型权重文件，可以通过初始化对应的网络结构，再使用 net.load_state_dict 加载 pth 权重文件。最后通过 torch.jit.trace 接口将网络结构和权重参数固化成一个 pt 文件。得到 torch.jit.trace 处理过以后的 pt 文件，就可以用 rknn.load_pytorch 接口将其转为 RKNN 模型。

5.7.5 转换时遇到 KeyError 的错误

错误日志如下：

```
E Catch exception when loading pytorch model: ./mobilenet0.25_Final.pth!
E Traceback (most recent call last):
……
E KeyError: 'aten::softmax'
```

出现形如 KeyError: 'aten::xxx'的错误信息时，表示该算子当前版本还不支持。RKNN Toolkit 在每次版本升级时都会修复此类 bug，请使用最新版本的 RKNN Toolkit。

5.7.6 转换时遇到"Syntax error in input! LexToken(xxx)"的错误

错误日志如下：

```
WARNING: Token 'COMMENT' defined, but not used
WARNING: There is 1 unused token
!!!!! Illegal character '"'
Syntax error in input! LexToken(NAMED_IDENTIFIER, 'fc', 1, 27)
!!!!! Illegal character '"'
```

这个错误的原因有很多种，请按照以下顺序排查：

1. 未继承 torch.nn.module 创建网络。请继承 torch.nn.module 这个基类来创建网络，然后再用 torch.jit.trace 生成 pt 文件。

2. 如果 RKNN Toolkit 版本是 1.6.0 或 1.6.1，建议 Pytorch 使用 1.5.0 到 1.6.0 之间的版本。

3. RKNN Toolkit 1.7.0 或之后的版本，torch 建议使用 1.6.0 或 1.9.0，1.10.0 版本。

5.8 Tensorflow 模型转换常见问题

5.8.1 转换谷歌官方的 ssd_mobilenet_v2 模型出现 AttributeError 错误

错误日志如下：

```
AttributeError: 'NoneType' object has no attribute op
```

出现该错误可能的原因是 input 节点没选对，可以尝试使用如下配置进行模型转换：

```
rknn.load_tensorflow(
tf_pb='./ssd_mobilenet_v2_coco_2018_03_29/frozen_inference_graph.pb',
inputs=['FeatureExtractor/MobilenetV2/MobilenetV2/input'],
outputs=['concat', 'concat_1'],
input_size_list=[[INPUT_SIZE, INPUT_SIZE, 3]])
```

5.8.2 转换 SSD_Resnet50_v1_FPN_640x640 模型出现无法转换数据类型的错误

错误日志如下：

```
Cannot convert value dtype (['resource', 'u1']) to a Tensorflow Dtype
```

需更新 RKNN Toolkit 到 0.9.8 及以后版本。

5.8.3 RKNN Toolkit 1.0.0 版本下，TensorFlow 模型的输出结果 shape 发生了变化

1.0.0 以前的版本如果模型输出的数据是按"NHWC"排列的，将转成"NCHW"。从 1.0.0 版本开始，输出的 shape 将与原始模型保持一致，不再进行"NHWC"到"NCHW"的转换。进行后处理时请注意 channel 所在的位置。

5.8.4 转换模型时，出现 TensorFlow 算子属性不存在的错误

错误日志如下：

```
E ValueError: NodeDef mentions attr 'explicit_paddings' not in Op<name=Conv2D;
```

这是由于该模型训练时用的 TensorFlow 的版本大于等于 1.14.0 版本，而转换 RKNN 模型时，采用的 TensorFlow 版本小于等于 1.13.2 版本导致的。解决方法是保持训练模型和转换 RKNN 模型时使用相同的 TensorFlow 版本。比如要么都用大于等于 1.14.0 版本，要么都用小于等于 1.13.2 版本。

5.8.5 加载模型 TFLite 模型时提示没有 TransposeOptions 属性

错误日志如下：

```
module “RKNNlib.convertert.lite.tflite” has no attribute ‘TransposeOptions’
```

1.3.2 及之前版本的 RKNN Toolkit 不支持 TFLite 的 Transpose 操作。需要更新到 1.4.0 或之后的版本。

5.9 加载模型出错时的排查步骤

首先确认原始深度学习框架是否可以加载该模型并进行正确的推理。

其次请将 RKNN Toolkit 升级到最新版本。如果模型有 RKNN Toolkit 不支持的层（或 OP），通过打开调试日志开关，在日志中可以看到哪一个算子是 RKNN Toolkit 不支持的，这类错误日志通常包含“Try match xxx failed”或“Not match xxx”等。如果模型在转换的时候出现 shape 处理不对的问题，可以在日志中找到如 concat/add/matmul 维度不匹配，或者计算 output tensor 出错等提示。

如果第一步不通过，请先检查原始模型是否有问题；如果升级到最新版本的工具后仍无法转换，或提示有算子不支持，请将所使用的工具的版本和出现转换问题时的详细日志反馈给瑞芯微 NPU 开发团队。

6 模型量化问题

本章节主要覆盖模型量化相关的问题：

1） 量化简介

2） QAT 与 PTQ 量化方式

3） 量化对模型体积的影响

4） 不量化时模型转换成功，开启量化后反而无法转换

5） 量化数据的作用

6） 模型量化时，图片是否需要和模型输入的尺寸一致

7） 模型量化时，程序运行一段时间后被 kill 掉或程序卡住

6.1 量化简介

模型量化后将使用更低的精度（如 int8/uint8/int16）保存模型的权重信息，在部署减少模型的内存空间占用，加快模型推理速度。

RKNN Toolkit 目前对量化模型的支持主要有以下两种形式：

- RKNN Toolkit 根据用户提供的量化数据集，对加载的浮点模型进行量化，生成量化的 RKNN 模型。
  - 支持的量化精度类型：int16，int8，uint8
  - 量化方式：训练后静态量化
- 由深度学习框架导出量化模型，RKNN Toolkit 加载并利用已有的量化信息,生成量化 RKNN 模型。
  - 支持的深度学习框架（括号内为主要支持版本，请尽量使用对应版本生成的量化模型）：PyTorch(v1.9.0 或 v1.10.0)、ONNX(Onnxruntime v1.5.1)、TensorFlow、TFLite
  - 支持的量化精度类型：int8, uint8
  - 量化方式：训练后量化, 量化感知训练(QAT)

6.2 QAT 与 PTQ 量化方式

- 训练后量化（PTQ 量化，Post train quantization）

RKNN Toolkit 加载用户训练好的浮点模型，然后根据 config 接口指定的量化方法和用户提供的校准数据集（训练数据或验证数据的一个小子集，大约 200~500 张）计算模型中网络层需要的量化参数。RKNN Toolkit 目前支持 3 种量化方法：

- asymmetric_quantized-u8(默认量化方法)

这是 TensorFlow 支持的训练后量化算法，也是 Google 推荐的。根据论文“Quantizing deep convolutional networks for efficient inference: A whitepaper”的描述，这种量化方式对精度的损失最小。

其计算公式如下：

$$quant = round\left(\frac{float\_num}{scale}\right) + zero\_point$$
$$quant = cast\_to\_bw$$

其中 quant 代表量化后的数，float_num 代表浮点数，scale 表示缩放系数（float32 类型），zero-points 代表实数为 0 时对应的量化值（int32 类型），最后把 quant 饱和到[range_min, range_max]，目前只支持 uint8 类型，所以 range_max 等于 255，range_min 等于 0

对应的反量化公式如下：

$$\text{float\_num} = scale(quant - zero\_point)$$

- dynamic_fixed_point-i8

少部分情况下，dynamic_fixed_point-i8 量化的精度比 asymmetric_quantized-u8 高。

其计算公式如下：

$$quant = round(float\_num * 2^{fl})$$
$$quant = cast\_to\_bw$$

其中 quant 代表量化后的数，float_num 代表浮点数，fl 是左移的位数，最后把量化后的数饱和到[range_min, range_max]。如果位宽 bw 等于 8，则范围是[-127, 127]。

- dynamic_fixed_point-i16

dynamic_fixed_point-i16 的量化公式与 dynamic_fixed_point-i8 一样，只不过它的位宽 bw 是 16。RK3399Pro 或 RK1808 的 NPU 自带 300Gops int16 计算单元，对于某些量化到 8 位后精度损失较大的模型，可以考虑使用此量化方式。

- 量化感知训练（QAT，quantization aware training）

通过量化感知训练可以得到一个带量化权重的模型。RKNN Toolkit 目前支持 TensorFlow 和 PyTorch 这两种框架量化感知训练得到的模型。量化感知训练技术细节请参考如下链接：

TensorFlow: https://www.tensorflow.org/model_optimization/guide/quantization/training

PyTorch: https://pytorch.org/blog/introduction-to-quantization-on-pytorch/

这种方法要求用户使用原始框架训练（或 fine tune）得到一个量化模型，接着使用 RKNN Toolkit 导入这个量化模型（模型转换时需要在 rknn.build 接口设置 do_quantization=False）。此时 RKNN Toolkit 将使用模型自身的量化参数，因此理论上几乎不会有精度损失。

注：RKNN Toolkit 也支持 ONNX，PyTorch，TensorFlow，TensorFlow Lite 的训练后量化模型，模型转换方法与量化感知训练模型一样，调用 build 接口时 do_quantization 要设置成 False。加载 ONNX 量化模型，请更新 RKNN Toolkit 到 1.7.0 或更新的版本；加载 PyTorch 量化模型，请更新 RKNN Toolkit 到 1.7.1 及以上版本。

6.3 量化对模型体积的影响

分两种情况，当导入的模型是量化的模型时，do_quantization=False 会使用该模型里面的量化参数。当导入的模型是非量化模型时，do_quantization=False 不会做量化的操作，但是会把权重从 float32 转成 float16，这块几乎不会有精度损失。这两种情况都减少了模型权重的体积，从而使得整个模型占用空间变小。

6.4 不量化时模型转换成功，开启量化后反而无法转换

错误信息如下：

```
T Caused by op 'fifo_queue_DequeueMany', defined at:
T       File "test.py", line 52, in <module>
T             ret = rknn.build(do_quantization=True, dataset='./dataset.txt')
……
T OutOfRangeError (see above for traceback): FIFOQueue '_0_fifo_queue' is closed and has insufficient elements (requested 1, current size 0)
```

出现这个错误可能的原因是 dataset.txt 没有数据、或是缺乏有效数据导致的。该文件的编写可以参考本文档第 4 章节的内容。

6.5 量化数据的作用

RKNN Toolkit 在量化过程中，会根据量化数据集，计算推理结果所需要的量化参数。基于这个原因，校准数据集里的数据最好是从训练集或验证集中取一个有代表性的子集，建议数量在 200~500 张之间。

6.6 模型量化时，图片是否需要和模型输入的尺寸一致

不需要。RKNN Toolkit 会自动对这些图片进行缩放处理。但是缩放操作也可能会使图片信息发生改变，对量化精度产生一定影响，所以最好使用尺寸相近的图片。

6.7 模型量化时，程序运行一段时间后被 kill 掉或程序卡住

rknn.config 的参数 batch_size(默认值为 100)越大，量化过程中 RKNN Toolkit 会申请越

大的系统内存，有可能造成程序被 kill 掉或卡住。

解决方法：增加 PC 内存；或者将 config 接口中的 batch_size 设小一点，例如 8 或者 16。

7 模拟器推理及连板推理、板端推理的说明

本章节主要覆盖以下内容：

1） 术语说明

2） 模拟器推理结果与板端不一致

3） 连板推理的工作原理

4） 连板推理与板端推理结果有差异

5） 板端推理的速度比连板推理更快

6） 涉及连板调试、连板推理功能时，获取详细的错误日志

7.1 术语说明

模拟器推理：RKNN Toolkit 在 Linux x86_64 平台提供模拟器功能，可以在没有开发版的情况下进行推理。(该功能在部分模型上可能存在精度问题，更推荐使用连板推理或板端推理)

连板推理：指在开发板已连接 PC 的情况下，调用 RKNN Toolkit 的 Python API 推理模型，获取推理结果。开发板的连接方法，请参考 Rockchip_User_Guide_RKNN_Toolkit_CN 文档的 2.4 小节)

板端推理：指在开发板上调用 RKNPU 的 C API 接口推理模型，获取推理结果。

7.2 模拟器推理结果与板端不一致

注：RKNN Toolkit 从 1.7.5 版本开始禁用模拟器相关功能。

发生此情况时请以板端的结果为准。

由于硬件和驱动的差异，模拟器不保证可以和板端获取一模一样的结果。在连板调试可用的情况下，不建议使用模拟器进行评估。模拟器仅在没有开发板的情况下作为一个可选的调试方式。推荐使用连板调试，避免模拟器出错时引发的各种问题。

7.3 连板推理的工作原理

使用连板推理时，RKNN Toolkit 会在后台启动一个 npu_transfer_proxy 的进程，此进程

会与板端的 rknn-server 进行通信，通信时会将模型、模型的输入由 PC 端传至板端，随后调用 RKNPU C API 进行模型推理，板端推理完成后将结果回传至 PC 端。

7.4 连板推理与板端推理结果有差异

连板推理是基于 RKNPU 的 C API 接口实现的，理论上连板推理结果会与 RKNPU C API 推理结果一致。当这两者出现较大差异时，请确认输入的预处理、数据类型、数据的排布方式(NCHW, NHWC)是否有差异。

需指出，如差异很小且发生在小数点后 3 位及之后的数值上，则属于正常现象。差异可能产生在使用不同的库读取图片、转换数据类型等步骤上。

7.5 板端推理的速度比连板推理更快

由于连板推理存在额外的数据拷贝、传输过程，会导致连板推理的性能不如板端的 RKNPU C API 推理性能。因此，NPU 实际推理性能以 RKNPU C API 的推理性能为准。

7.6 涉及连板调试、连板推理功能时，获取详细的错误日志

连板调试、连板推理时，模型的初始化、推理等操作主要在开发板上完成，此时日志信息主要产生在板端上。

为了获取具体的板端调试信息，可以通过串口进入开发板操作系统。然后执行以下两条命令设置获取日志的环境变量。保持串口窗口不要关闭，再进行连板调试，此时板端的错误信息就会显示在串口窗口上：

```
export RKNN_LOG_LEVEL=5
restart_rknn.sh
```

8 模型评估常见问题

本章内容：

1) 模型的输入输出数据格式
2) 量化模型精度不及预期
3) dump 网络每层输出
4) 支持哪些框架的已量化模型
5) 精度分析时保存的文件格式说明
6) 使用 adb 查询连接设备时提示没有权限
7) 连板调试时，rknn_ini 失败，返回-6 或模型非法的错误
8) 连板调试时，rknn_init 失败，出现设备不可用的错误
9) 在 Windows 平台初始化模型时提示 dlopen 失败
10) 在 Windows 平台安装 RV1109/RV1126 USB 驱动失败
11) Windows 平台无法检测到 RV1109/RV1126 设备
12) rknn.inference 耗时与 rknn.eval_perf 理论速度不一致
13) rknn.inference 对多输入和多 batch 的支持
14) 运行多个 RKNN 模型
15) 模型推理的耗时非常长，而且得到的结果全是 0
16) 非量化模型推理慢
17) 调用 rknn.build 时如果设置 pre_compile=True 报错，不设置可以转换成功。
18) 使用 RK3399Pro 时如何保存每一层的推理结果
19) 在 Windows 系统中如果调用 RKNN Toolkit 接口失败后直接退出
20) RK3399Pro 在推理时，是否使用到 CPU，是独立内存还是共享内存
21) 性能评估时，开启 perf_debug 与关闭该参数，性能数据存在差异
22) 模型转换时 rknn_batch_size 设成 8，推理时只传一张图片，但推理耗时却和 8 张的一样
23) 环境用的 docker，之前连板推理正常，重启 docker 后，推理卡在环境初始化阶段
24) Rockchip 开发板 NPU 定频方法
25) RKNN Toolkit Lite 的速度比 RKNN Toolkit 快吗
26) 模型量化时出现离群值警告

8.1 模型的输入输出数据格式

RKNN 模型的输入输出属性主要记录在 rknn_input_output_num 和 rknn_tensor_attr 两个结构体中，可以使用 C/C++ API 中的 rknn_query 接口查询。

其中 rknn_input_ouput_num 结构体记录了 RKNN 模型输入节点和输出节点的数量，分别用 n_input / n_ouput 属性表示。

而 rknn_tensor_attr 结构体则记录了具体某一个输入或输出节点的详细信息。其中和数据 buf 密切相关的属性如下：

- fmt：输入或输出节点数据的排列格式，目前支持两种格式：RKNN_TENSOR_NCHW 和 RKNN_TENSOR_NHWC。对于 TensorFlow / TFLite / Keras 这三类框架导出的 RKNN 模型，默认格式是 RKNN_TENSOR_NHWC。但如果在模型转换阶段，config 接口中设置了 remove_tensorflow_output_permute，则输入、输出节点的排列方式会变成 RKNN_TENSOR_NCHW。以 ssd_mobilenet_v1 模型为例，该模型的输入节点 shape 是{1, 300, 300, 3}，设置该参数后，查询 RKNN 输入节点，得到的 shape 是{1, 3, 300, 300}。对于其他框架的模型，导出 RKNN 模型时，默认的数据排列格式是 RKNN_TENSOR_NCHW。但如果在模型转换阶段，config 接口的 force_builtin_perm 参数被设成 True，则输入输出节点的排列格式会变成 RKNN_TENSOR_NHWC。例如 ONNX yolov5 模型的输入 shape 是{1, 3, 640, 640}，转成 RKNN 模型后会变成{1, 640, 640, 3}。如果模型的输入或输出节点不是 4 维的，RKNN 模型会保持跟原始框架一样的排列顺序。例如三维输入{1, 20, 256}，RKNN 模型该输入节点的 shape 就是{1, 20, 256}。
- n_dims: 节点的维度个数。对于图像处理类的模型的输入节点，这个值通常是 4。
- dims: 输入或输出节点的 shape 信息。注意：通过 C/C++ rknn_query 接口查询到的 dims 顺序和数据的实际排列顺序是反过来的。例如 RKNN 模型的输出 shape 实际是{1, 255, 20, 20}，但 dims 中的数值存储的是{20, 20, 255, 1}。
- type: 输入或输出节点的数据类型。如果该节点没有量化，则类型应该是 RKNN_TENSOR_FLOAT16；如果是用 dynamic_fixed_point-i16 量化，则类型应该是 RKNN_TENSOR_INT16；如果是用 dynamic_fixed_point-i8 量化，则类型应该是 RKNN_TENSOR_INT8；如果是用 asymmetric_affine-u8 量化，则类型应该是 RKNN_TENSOR_UINT8。注：虽然在模型转换阶段设置了量化，但因为输入或输

出节点遇到一些特殊情况，这些节点可能是没有量化的。节点是否量化以 qnt_type 属性的值为准。

- n_elems：输入或输出节点的元素个数。例如输入节点 shape 为{1, 3, 300, 300}时，有 1* 3 * 300 * 300，共 270000 个元素。如果我们要为这个输入分配内存时，需要分配 27000 * sizeof(type)字节的空间。
- size: 输入或输出节点需要占用的内存空间，单位为 Byte。同样以输入节点 shape 为{1, 3, 300, 300}为例，如果模型的输入节点没有量化，type 为 RKNN_TENSOR_FLOAT16，则需要的内存空间为 27000 * 16，一共 4320000 字节。

注：以上这些属性是模型转换完后就固定的，但应用拿到的输入数据并不一定是按照上述格式去存储的。这时候就需要对数据类型、数据排列方式等进行转换。这个转换可以应用程序外部去做，也可以交给 rknn_inputs_set 接口去完成。

8.2 量化模型精度不及预期

- **首先确保 float 类型的精度和原始平台测试结果相近：**

(1) 导出不量化的模型，rknn.build(do_quantization=False)；

(2) 参考第 4 章正确设置 **mean_values/std_values** 参数，确保其和训练模型时使用的参数相同;

(3) 务必确保测试时输入图像通道顺序为 R,G,B。（不论训练时使用的图像通道顺序如何，使用 RKNN 做测试时都按 R,G,B 输入）；

(4) 在 **rknn.config** 函数里面设置 **reorder_channel** 参数，“0 1 2”代表 RGB，“2 1 0”代表 BGR，务必和训练时候图像通道顺序一致。

- **量化后的精度测试建议**

(1) 使用多张图进行量化校准，确保量化精度稳定。

在 rknn.config 中设置 batch_size 参数 (建议设置 batch_size = 200) 并且在 dataset.txt 中给出大于 200 张图像路径用于量化。

如果内存不够，可以设置 batch_size =10, epochs=20 代替 batch_size = 200 进行量化。

(2) 精度对比，尽量用较大数据集进行测试。分类网络比较 top-1,top-5 精度，检测网络比较数据集的 mAP, Recall 等。

(3) 如果是人脸识别，不能使用 float 模型的结果和量化模型的结果进行特征比较。比如

A1、A2 两张图片，用 float 模型跑出来的结果 A1fp、A2fp，用量化模型跑的结果 A1u8、A2u8，这时可以算 A1fp 和 A2fp 的欧式距离来计算两个图片相似度，也可以算 A1u8 和 A2u8 的欧式距离来计算两个图片相似度，但是不能算 A1fp 和 A1u8 的欧式距离来计算两个图片相似度。

- **浮点模型结果不对**

1) 目前 PC 模拟器可以支持 dump 出每一层网络的数据，在执行 inference 的脚本前需要设置一个环境变量，命令如下：export NN_LAYER_DUMP=1。
2) 执行完之后，会在当前目录生成每层网络的 tensor 数据文件，这样可以和原始框架的数据进行逐层比对。注意：有些层会被合并，比如 conv+bn+scale 会合并成一个 conv，这时候就需要和原来模型的 scale 层的输出进行对比。
3) 如果对比数据的时候发现第一层结果差距就很大，则通常是因为输入预处理没正确设置导致的。请首先检查 config 中的 mean_values / std_values / reorder_channel 是否设置正确；推理时图像是否做了其他处理。如果是最后一层结果全为 0，则留意串口日志，是否出现 GPU Hang，并将该模型反馈给瑞芯微 NPU 团队分析。如果是中间层出错，可以将该层的参数，输入输出等信息反馈给瑞芯微 NPU 团队进行进一步分析。
4) 在 RK1808/RV1109/RV1126 等开发板上也可以保存模型推理时每一层的中间结果。具体方法如下：通过 DEBUG 口连接到开发板上，在/userdata 目录下（这个目录空间一般比较大）创建一个目录用来保存中间结果，例如 dump_data；接着进入该目录，设置环境变量（export NN_LAYER_DUMP=1）；如果是通过 PC 连接板子进行调试的，需要重启 rknn_server，方法为执行 restart_rknn.sh 命令。进行以上设置以后再运行模型推理的程序就可以 dump 每一层中间结果了。对于 RK3399Pro 板子，需要连接板子上的 NPU 的串口，具体位置可以参考下图用红色框标注的位置。

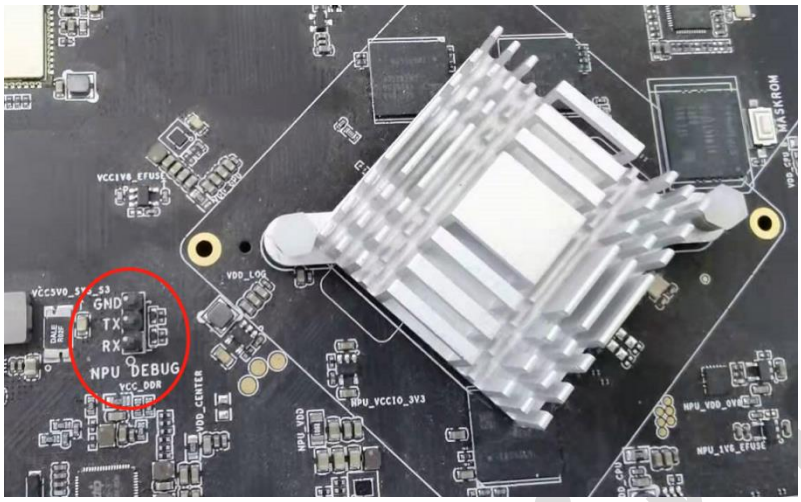

图 8-1 RK3399Pro NPU 串口位置

- **量化模型结果不对**

1） 检查浮点模型是否正确，不正确，请先按照前一小节进行排查；正确的话进行下一步。

2） 使用 accuracy_analysis 接口进行精度分析，找出造成精度下降的层。也可以手动 dump 每一层的结果,并进行比较后找出精度下降的层。

3） 这些精度下降的层，存在三种情况，一是驱动实现有问题（此时量化结果通常会和浮点结果差距巨大）；二是该层本身对量化就不友好；也可能是某些模型优化导致精度下降,例如用 conv 替换 add 或者 average pool 等。对于第一种场景，可以尝试用混合量化的方式进行规避。对于第二种场景，可以先尝试用 MMSE 或 KL 散度算法优化量化参数，还无法提高精度的情况下，考虑使用混合量化，用更高精度的方法（浮点或动态定点 i16 量化）去计算对量化不友好的层；也可以考虑使用 TensorFlow 或 PyTorch 进行量化感知训练，然后直接导入量化模型。对于第三种场景，可以尝试将优化等级下调(下调的方法是在 config 接口中将 optimization_level 设成 2 或 1)。如果精度还是无法满足要求，请将相关模型、精度分析结果和精度测试方法等反馈给瑞芯微 NPU 团队。

注：精度分析、MMSE/KL 散度算法和混合量化相关接口的用法请参考 RKNN Toolkit 使用说明文档《Rockchip_User_Guide_RKNN_Toolkit_CN.pdf》。

8.3 dump 网络每层输出

参考前一问题。

8.4 支持哪些框架的已量化模型

RKNN Toolkit 1.6.1 及之前的版本支持 TensorFlow，TensorFlow Lite 这两种框架的已量化模型。从 1.7.0 开始，支持 ONNX 框架的已量化模型。从 1.7.1 开始，支持 PyTorch 框架的已量化模型。

8.5 精度分析时保存的文件格式说明

是一维 numpy 数组，可以通过 numpy 的 loadtxt 接口加载。

数据按照 NHWC 的格式保存。目前不可以修改数据排列顺序。

8.6 使用 adb 查询连接设备时提示没有权限

错误信息如下：

```
test@test:~/ $ adb devices
List of devices attached
* daemon not running; starting now at tcp:5037
* daemon started successfully
1109        no permissions (user hpcci is not in the plugdev group); see
[http://developer.android.com/tools/device.html]
1126        no permissions (user hpcci is not in the plugdev group); see
[http://developer.android.com/tools/device.html]
```

解决方法：

找到脚本：<sdk>/platform-tools/update_rk_usb_rule/linux/update_rk1808_usb_rule.sh。执行该脚本。然后再尝试查询设备的命令。如果还提示没有权限，请重新插拔设备，或者重启机器。权限正常时，输出如下：

```
test@test:~/ $ adb devices
List of devices attached
1109        device
1126        device
```

8.7 连板调试时，rknn_ini 失败，返回-6 或模型非法的错误

错误信息如下：

```
E RKNNAPI: rknn_init,    msg_load_ack fail, ack = 1, expect 0!
E Catch exception when init runtime!
T Traceback (most recent call last):
T      File "rknn/api/rknn_base.py", line 646, in
rknn.api.rknn_base.RKNNBase.init_runtime
T      File "rknn/api/rknn_runtime.py", line 378, in
rknn.api.rknn_runtime.RKNNRuntime.build_graph
T Exception: RKNN init failed. error code: RKNN_ERR_MODEL_INVALID
```

出现该错误一般有以下几种情况：

1）在生成 rknn 模型时，使用了 pre_compile=True 的选项（离线预编译方式在 1.7.5 版本将废弃）。此时请检查 RKNN Toolkit 和 NPU 驱动版本是否匹配。RKNN Toolkit 和 NPU 驱动的匹配关系可以参考表 8-1。建议将 RKNN Toolkit 和板子的 NPU 驱动都升级到最新的版本，NPU 驱动的更新方法请参考 rknpu 工程：https://github.com/rockchip-linux/rknpu.git

2）在生成 rknn 模型时，没有使用 pre_compile=True 的选项，这时一般是系统固件太老了，建议将板子的固件升级到最新的版本。NPU 驱动更新方法请参考 rknpu 工程相关说明。

更新驱动时请确保所有文件都推送成功，建议在升级后检查下板端相关文件的路径和 MD5 值是否与 rknpu 工程中的一致。

3）没有正确设置 target_platform。例如不设置 config 接口中的 target_platform 时，生成的 RKNN 模型只能在 RK1806/RK1808/RK3399Pro 上运行，而不能在 RV1109/RV1126 上运行。如果要在 RV1109/RV1126 上运行，需要在调用 config 接口时设置 target_platform=["rv1109", "rv1126"]。

4）如果是在 Docker 容器中推理时出现该问题，有可能是因为宿主机上的 npu_transfer_proxy 进程没有结束，导致通信异常。可以先退出 Docker 容器，将宿主机上的 npu_transfer_proxy 进程结束掉，然后再进入容器执行推理脚本。

5）也可能是 RKNN Toolkit 转出来的模型本身有问题。这时可以获取如下信息，反馈给瑞芯微 NPU 团队：如果是用模拟器，初始化 RKNN 对象时设置 verbose=True，打印详细日志，并记录下来，在模型初始化的地方会有更详细的日志说明导致模型检查失败的原因；如果是 PC 连开发板调试，或者在开发板上运行模型，可以串口连到开发板，然后设置环境变量 RKNN_LOG_LEVEL=5，之后执行 restart_rknn.sh，然后再重跑程序，将开发板上的详细日志记录下来。

8.8 连板调试时，rknn_init 失败，返回设备不可用的错误

错误信息如下：

```
E RKNNAPI: rknn_init, driver open fail! ret = -9!
E Catch exception when init runtime!
T Traceback (most recent call last):
T File "rknn/api/rknn_base.py", line 617, in
rknn.api.rknn_base.RKNNBase.init_runtime
T File "rknn/api/rknn_runtime.py", line 378, in rknn.api.rknn_runtime.RKNNRuntime.
T build_graph
T Exception: RKNN init failed. error code: RKNN_ERR_DEVICE_UNAVAILABLE
```

这个问题的原因比较复杂，请按以下方式排查：

1） 确保RKNN Toolkit及板子的系统固件都已经升级到最新版本。各版本RKNN Toolkit和系统固件各组件之间的对应关系如下：

表 8-1 RKNN Toolkit 和 RKNN Server 等组件的版本对应关系

RKNN Toolkit	rknn_server	NPU 驱动	librknn_runtime
1.0.0	0.9.6/0.9.7	6.3.3.3718	0.9.8/0.9.9
1.1.0	0.9.8	6.3.3.03718	1.0.0
1.2.0	0.9.9	6.4.0.213404	1.1.0
1.2.1	1.2.0	6.4.0.213404	1.2.0
1.3.0	1.3.0	6.4.0.227915	1.3.0
1.3.2	1.3.2	6.4.0.7915	1.3.2
1.4.0	1.4.0	6.4.0.27915	1.4.0
1.6.0	1.6.0	6.4.3.5.293908	1.6.0
1.6.1	1.6.1	6.4.3.5.293908	1.6.1
1.7.0	1.7.0	6.4.6.5.351518	1.7.0
1.7.1	1.7.1	6.4.6.5.351518	1.7.1
1.7.3	1.7.3	6.4.6.5.351518	1.7.3
1.7.5	1.7.5	6.4.6.5.351518	1.7.5

以 RK1808 为例，这些组件的版本查询方法如下：

```
# execute these commands on RK1808
dmesg | grep -i galcore       # 查询 NPU 驱动版本
strings /usr/bin/rknn_server | grep build       # 查询 rknn_server 版本
strings /usr/lib/librknn_runtime.so | grep version       # 查询 librknn_runtime 版本
```

也可以通过 get_sdk_version 接口查询版本信息，其中的 DRV 版本对应的就是 rknn_server 的版本。

2） 确保 adb devices 能看到设备，并且 init_runtime()的 target 和 device_id 设置正确。

3） 如果使用 RKNN Toolkit 1.1.0 及以上版本，请确保 rknn.list_devices()能看到设备，并且 init_runtime()的 target 和 device_id 设置正确。

4） 如果使用的是计算棒或者 RK1808 EVB 版的 NTB 模式，请确保已经调用 update_rk1808_usb_rule.sh （在 RKNN Toolkit 发布包中）来获得 USB 设备的读写权限。（可以先使用 sudo 运行该 python 脚本看是否还出现问题）。

5） 如果既连接了 ADB 设备，又连接了 NTB 设备，请确保同时只有一类设备在使用。使用 ADB 设备时需要将 npu_transfer_proxy 进程结束掉；ADB 设备只能在 Linux x86_64 平台上使用。

6） 如果是 Mac OS 平台，请确保只有一个设备在使用，如果要使用另外一个设备，请先手动将 npu_transfer_proxy 进程结束掉。

7） 如果是 Windows 平台，使用前请先将 360 安全卫士/腾讯电脑管家等程序关掉，否则也可能出现该错误。第一次使用开发板需要安装相应驱动，驱动安装方法参考<Rockchip_Quick_Start_RKNN_SDK>文档 4.1 章节。

8） 如果是直接在 RK3399/RK3399Pro 上运行 AARCH64 版本的 RKNN Toolkit，请确保系统固件都已经升级到最新版本。调用 init_runtime 的时候 target/device_id 参数不要填。

9） 如果使用的是 RV1109/RV1126，请检查是否使用了 Mini Driver，Mini Driver 不支持联机调试。

10） 如果是 RK3399Pro，只有安装 Android 系统固件的开发板可以在 Linux_X86 PC 上通过联机进行调试，其他固件可以直接登录相应系统安装 AARCH64 版本 RKNN Toolkit 进行开发和测试。

8.9 在 Windows 平台初始化模型时提示 dlopen 失败

出现该错误时的错误日志如下：

```
--> Init runtime environment
E Catch exception when init runtime!
……
E     File “rknn\api\rknn_runtime.py”, line xxx, in
rknn.api.rknn_runtime.RKNNRuntime.__init__,
E     File “C:\Users\xxx\Python\Python36\lib\ctypes\__init__.py”,
E          self._handle = _dlopen(self._name, mode)
E OSError: [WinError 126]  找不到指定的模块。
E Current device id is: None
E Devices connected:
E [‘1808s1’]
Init runtime environment failed
```

如果出现这种错误，需要把 librknn_api.so 的路径加到系统路径 PATH 中，或者将 RKNN Toolkit 版本升级到最新版本。

首先，通过“pip show rknn-toolkit”找到 RKNN Toolkit 的安装路径<RKNN_INSTALL_PATH>，然后在系统 PATH 添加如下两条路径：<RKNN_INSTALL_PATH>/rknn/api/lib/hardware/LION/Windows_x64 和<RKNN_INSTALL_PATH>/rknn/api/lib/hardware/PUMA/Windows_x64。

8.10 在 Windows 平台安装 RV1109/RV1126 USB 驱动失败

现象：执行驱动安装程序时 Driver/USB ID 等全部为空，Installer Driver 为灰色，无法点击，如下图所示：

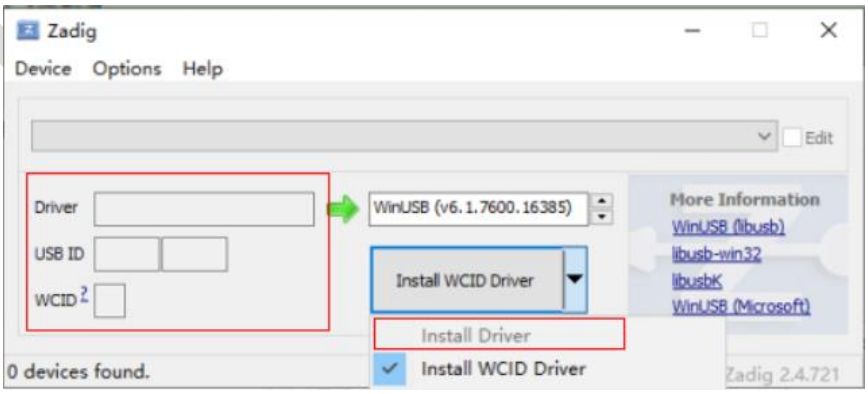

图 8-2 驱动安装界面

此时，通过 Windows 的设备管理器查看设备列表，识别到的是 Android Device，而不是通用串行总线设备。

原因：RV1109/RV1126 的固件默认没有开启 ntb 通信方式。

解决方法如下：

1. 通过串口或者 adb 进入开发板系统；
2. 找到/etc/init.d/.usb_config 这个文件，打开后添加一行：usb_ntb_en；
3. 保存并重启开发板；
4. 再次执行驱动安装程序。

8.11 Windows 平台无法检测到 RV1109/RV1126 设备

现象：在 Windows 上执行“python -m rknn.bin.list_devices”命令或者调用 list_devices 接口，无法列出 RV1109 或 RV1126 开发板。

原因：RV1109/RV1126 开发板的固件默认没有开启 NTB 通信方式。

解决方法：

1. 需要先开启 NTB，开启方法同 8.10。
2. 安装开发板驱动，安装方法请参考《Rockchip_Quick_Start_RKNN_SDK_CN》。

8.12 rknn.inference 耗时与 rknn.eval_perf 理论速度不一致

这个问题有两方面的现象：

1) 进行前向推理测试速度慢，经测试 mobilenet-ssd 有的图片耗时在 0.5 秒以上

2) 模型 rknn.inference 的时间和 rknn.eval_perf 时间相差较大，比如

理论计算时间(单图)	1.79ms	8.23ms	7.485ms	30.55ms
实际计算时间(单图)	21.37ms	39.82ms	33.12ms	76.13ms

实测帧率慢的问题，有两方面的原因：

1. 使用 PC + adb 的方式传图片比较慢，这种对高帧率的网络影响很大比如理论 1.79ms 的网络。

2. 在 0.9.8 及之前版本的实现中，推理包含了一些额外的时间，0.9.9 及之后的版本已经做了优化。

对于更真实的帧率，建议直接在开发板上使用 RKNPU C API 进行测试。

8.13 rknn.inference 对多输入和多 batch 的支持

RKNN Toolkit 需要升级到 1.2.0 或之后的版本，并且需要在构建 RKNN 模型时就指定输入图片的数量，详细用法参考《Rockchip_User_Guide_RKNN_Toolkit_CN》中关于 build 接口的说明。

另外，当 rknn_batch_size 大于 1（如等于 4 时），python 里推理的调用要由：

```
outputs = rknn.inference(inputs=[img])
```

修改为：

```
img = np.concatenate((img, img, img, img), axis=0)
outputs = rknn.inference(inputs=[img])
```

完整示例请参考：<sdk>/examples/common_function_demos/batch_size/。

8.14 运行多个 RKNN 模型

运行两个或多个模型时，需要创建多个 RKNN 对象。一个 RKNN 对象对应一个模型，类似一个上下文。每个模型在各自的上下文里初始化模型，推理，获取推理结果，互不干涉。这些模型在 NPU 上推理时是串行进行的。

8.15 模型推理的耗时非常长，而且得到的结果全是 0

如果推理耗时超过 20s，且结果全是 0，这通常是 NPU 出现了 GPU hang 的 BUG。如果遇到这个问题，可以尝试将 NPU 驱动更新到 1.5.0 或之后的版本。如果普通模型能正常推理，而离线预编译后出现 GPU Hang 问题，请使用在线预编译。

8.16 非量化模型推理慢

NPU 浮点算力比较弱，针对量化模型 NPU 有更好的优化，所以量化模型性能会比浮点模型好很多。在 NPU 上，建议用量化后的模型。

8.17 调用 rknn.build 时如果设置 pre_compile=True 报错，不设置可以转换成功

错误信息如下：

```
E Catch exception when building RKNN model!
T Traceback (most recent call last):
T   File "rknn/api/rknn_base.py", line 515, in rknn.api.rknn_base.RKNNBase.build
T   File "rknn/api/rknn_base.py", line 439, in rknn.api.rknn_base.RKNNBase._build
T   File "rknn/base/ovxconfiggenerator.py", line 187, in
rknn.base.ovxconfiggenerator.generate_vx_config_from_files
T   File "rknn/base/RKNNlib/app/exporter/ovxlib_case/casegenerator.py", line 380, in
rknn.base.RKNNlib.app.exporter.ovxlib_case.casegenerator.CaseGenerator.generate
T   File "rknn/base/RKNNlib/app/exporter/ovxlib_case/casegenerator.py", line 352, in
rknn.base.RKNNlib.app.exporter.ovxlib_case.casegenerator.CaseGenerator._gen_special_cas
e
T   File "rknn/base/RKNNlib/app/exporter/ovxlib_case/casegenerator.py", line 330, in
rknn.base.RKNNlib.app.exporter.ovxlib_case.casegenerator.CaseGenerator._gen_nb_file
T AttributeError: 'CaseGenerator' object has no attribute 'nbg_graph_file_path'
```

请确认：

1) 系统装有 gcc 编译工具链

2) 模型的名称只包含“字母”、“数字”、“_”，特别注意有些复制过来的模型名称会自动加上“（1）”，这时会失败。也可以升级到 1.3.0 或之后的版本，由 RKNN Toolkit 自动处理这些特殊字符。

如果不属于上述情况，可以尝试使用 export_rknn_precompile_model 接口从开发板上导出预编译模型。

注：RKNN Toolkit 从 1.7.5 版本开始禁用离线预编译功能。

8.18 使用 RK3399Pro 时如何保存每一层的推理结果？

需要通过串口程序进入 NPU 系统，在 NPU 系统中创建一个目录存放中间结果。例如在 /data/ 目录下创建一个 dumps 目录。然后进入 dumps 目录执行如下命令 export NN_LAYER_DUMP=1 && restart_rknn.sh。接着再调用模型推理脚本，推理时 RKNPU 会将每一层的结果保存在 dumps 目录中。

通过串口进入 NPU 系统的方法请参考 user guide 文档。

8.19 在 Windows 系统中如果调用 RKNN Toolkit 接口失败后直接退出，会报错，提示 PermissionError。

出现这个错误是因为出错时没有调用 rknn.release 接口释放资源，导致 Python 回收资源时出现冲突导致的。解决方式是在调用 RKNN Toolkit 接口的地方都加上返回值的判断，如果返回码等于-1 就调用 rknn.release 接口释放资源。

8.20 RK3399Pro 在模型推理阶段，使用的是 NPU 的独立性能？还是也使用了一部分 CPU 性能？内存是独立的吗？

RK3399Pro 在模型推理阶段可以认为使用的都是 NPU 上的计算资源。但是在设置输入和获取输出阶段，需要进行数据的处理和传输，这时候会使用一部分 RK3399Pro 自身的计算、内存资源。NPU 的内存是独立的。

8.21 性能评估时，开启 perf_debug 与关闭该参数，性能数据的差异

开启 perf_debug 时，为了收集每层的信息，会添加一些调试代码。而且在实际运行的时候，有些算子可能是并行算的，而 perf_debug=True 是单纯逐层相加，所以耗时比 perf_debug=False 时多。

开启 perf_debug 的主要作用是检查模型中哪些层耗时比较多，并以此为依据来设计优化方案。

8.22 模型转换时 rknn_batch_size 设成 8，推理时只传 1 张图片，但耗时却和推理 8 张图片一样长

设置 rknn_batch_size 后，推理都是按设置的 batch_size 去推理的。即使只填了一张图片的数据，但推理时，其他的图片会被全部填成 0，参与推理。

8.23 环境用的 docker，之前连板推理正常，重启 docker 后，推理时卡在初始化环境阶段？

这个是因为 docker 重启时 npu_transfer_proxy 类似于异常退出的状态，导致开发板上的 rknn_server 无法检测到上端连接已经断开导致的。这时需要重启下开发板，重置 rknn_server 的连接状态。

8.24 Rockchip 开发板 NPU 定频方法

在进行性能评估时，因为 NPU / CPU / DDR 的频率都会影响最终的性能数据，所以评估性能时，建议先对开发板的 NPU / CPU / DDR 进行定频。其中 DDR 出厂固件一般是定频的，测试时只需确认下所用频率即可。

对于 RK1808，RV1109，RV1126，CPU 定频方法如下：

```
# 通过 adb 或 ssh 等方式进入开发板系统，以 adb 为例
adb shell

# 查看 cpu 频率设置策略
cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_available_governors
# 结果说明
userspace: 自定义频率，定频时要把 scaling_governor 设成这个值。
ondemand: 动态调整，默认策略
performance：性能模式，功耗会比较大

# 查看可用的 cpu 频率
cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_available_frequencies

# 查看可设置的最大/最小 cpu 频率
cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_max_freq
cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_min_freq

# 以 1.2GHz 为例，设置 cpu 频率
echo userspace > /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor
echo 1200000 > /sys/devices/system/cpu/cpu0/cpufreq/scaling_setspeed
# 查看 cpu 频率
cat /sys/kernel/debug/clk/clk_summary | grep arm
```

NPU 定频方法如下：

```
# 通过 adb 或 ssh 等方式进入开发板系统，以 adb 为例
adb shell

# 查看 npu 频率设置策略
cat /sys/class/devfreq/ffbc0000.npu/available_governors
# 结果说明
userspace: 自定义频率，定频时要把 scaling_governor 设成这个值。
ondemand: 动态调整，默认策略
performance：性能模式，功耗会比较大

# 查看可设置 npu 频率
cat /sys/class/devfreq/ffbc0000.npu/available_frequencies

# 查看可设置的最大/最小 npu 频率
cat /sys/class/devfreq/ffbc0000.npu/max_freq
cat /sys/class/devfreq/ffbc0000.npu/min_freq

# 以 934MHz 为例，设置 NPU 频率
echo userspace > /sys/class/devfreq/ffbc0000.npu/governor
echo 934000000 > /sys/class/devfreq/ffbc0000.npu/userspace/set_freq

# 查看 NPU 频率，结果在 clk_npu_np5 行第 5 列，单位是 Hz
cat /sys/kernel/debug/clk/clk_summary | grep npu
```

DDR 频率查看方法：

```
# 通过 adb 或 ssh 等方式进入开发板系统，以 adb 为例
adb shell

# DDR 频率查询命令
cat   /sys/kernel/debug/clk/clk_summary | grep dpll

# 结果说明
     pll_dpll      1      1      0      462000000      0      0      50000
         dpll      1      1      0      462000000      0      0      50000
# 其中的 462000000 即读或写的频率，单位是 Hz
```

注：

1. RK3399Pro 中的 NPU 已经定频，不需要设置。
2. 以上方法基于 Rockchip 的开发板，如果是第三方厂商的开发板，请找对应厂商确认相应方法。

8.25 RKNN Toolkit Lite 的速度比 RKNN Toolkit 快吗

没有更快。

RKNN Toolkit Lite 是 RKNN Toolkit 的裁剪版，只保留了模型推理等少部分功能，在推理时并不会比 RKNN Toolkit 更快。

8.26 模型量化时出现离群值警告

量化时，如果显示存在离群值警告信息，可留意对应的网络层是否在精度分析中存在精度损失的情况。目前只对卷积的 weight 进行检测，离群值警告信息示例及解释如下：

W found outlier value, this may affect quantization accuracy

const name	abs_mean	abs_std	outlier value
convolution_at_input.12_1_1:weight	2.44	2.47	-17.494
convolution_at_input.9_130_130:weight	0.18	0.20	-11.216
convolution_at_input.17_143_143:weight	0.12	0.19	13.361, 13.317
convolution_at_input.1_156_156:weight	0.09	0.14	-10.215

- Const name 指明对应的卷积层名。
- Abs mean 指 weight 的绝对值取均值。
- Abs std 指 weight 的绝对值取标准差。
- Outlier value 指具体的离群数值，通常会是数倍于标准差、均值，导致整个权重的分布不均匀，最终导致量化精度不佳。请注意这里只打印最显著的离群点，不代表其余数值分布均匀。

9 深度神经网络模型设计建议

9.1 如何设计卷积神经网络，使其能在 RKNN 上实现最佳性能

有以下 5 点建议：

1） 卷积核设置

推荐在设计的时候尽量使用 3x3 的卷积核，这样可以实现最高的乘加运算单元（MAC）利用率，使得 NPU 的性能最佳。

NPU 也可以支持大尺寸的卷积核。支持的最小卷积核为[1]，最大值为[11 * stride - 1]。同时 NPU 也支持非对称卷积核，不过会增加一些额外的计算开销。

2） 融合结构设计

NPU 会对卷积后面的 ReLU 和 MAX Pooling 进行融合的优化操作，能在运行中减少计算和带宽开销。所以在搭建网络时，能针对这一特性，进行设计。

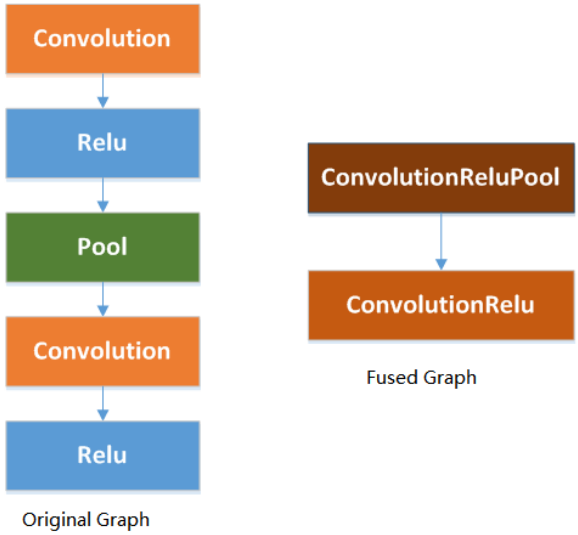

图 9-1 融合操作示例

模型量化后，卷积算子与下一层的 ReLU 算子可以被合并。另外为了确保 Max Pooling 算子也能被合并加速，请在设计网络时参考以下几点：

- pool size 必须是 2x2 或者 3x3，而步长 stride=2
- 2x2 池化的输入图片尺寸必须是偶数，而且不能有填充
- 3x3 池化的输入图片尺寸必须是非 1 的奇数，而且不能有填充
- 如果是 3x3 的池化，则水平输入大小必须小于 64（8-bit 模型）或 32（16-bit 模型）

3） 关于 2D 卷积和 Depthwise 卷积的使用

NPU 支持常规 2D 卷积和 Depthwise 卷积加速。由于 Depthiwise 卷积特定的结构，使得它对于量化(int8)模型不太友好，而 2D 卷积的优化效果更好。所以设计网络时建议尽量使用 2D 卷积。

如果必须使用 Depthwise 卷积，建议按照下面的规则进行修改，能提高量化后模型的精度:

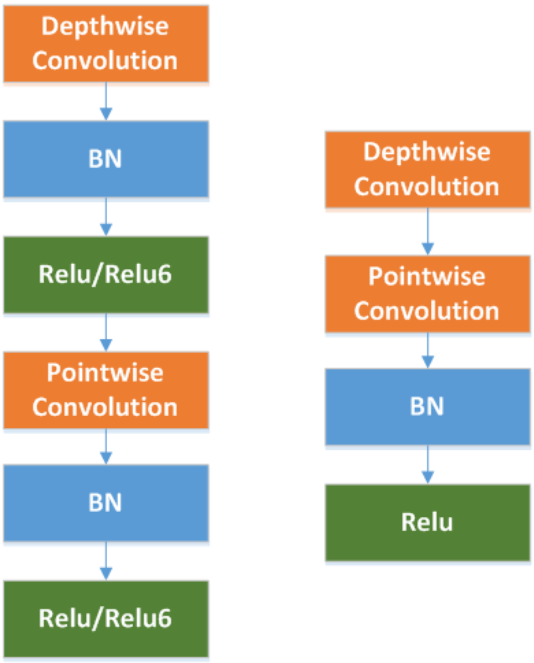

图 9-2 Depthwise 卷积改进思路

- 如果网络中的激活函数使用的是 ReLU6，建议将其都改为 ReLU。
- 在 Depthwise 卷积层的 BN 层和激活层，建议去除。
- 在训练时，针对 Depthwise 卷积层，对它的权重进行 L2 正则化。

4） 网络稀疏化

当前的神经网络存在过度参数化现象，并且在其设计时会存在很多冗余。NPU 针对稀疏矩阵，有进行跳零计算和内存提取方面的优化。所以建议在设计网络的时候，可以针对性的进行网络稀疏化设计，以利用该技术进一步提高网络性能。

5） 空洞卷积的使用

当使用 tensorflow 来创建带 dilations 参数的卷积时，请使用 tf.nn.atrous_conv2d 来创建卷积。目前 toolkit 不支持直接使用 tf.nn.conv2d 来创建带 dilations 参数的卷积，否则推理结果会出错。

另外，tensorflow 的版本需要高于 1.14.0，RKNN Toolkit 的版本需要高于 v1.4.0。

10 附录

10.1 参考文档

OP 支持列表：《RKNN_OP_Support.md》

快速上手指南：《Rockchip_Quick_Start_RKNN_SDK_CN.pdf》

RKNN Toolkit 使用指南：《Rockchip_User_Guide_RKNN_Toolkit_CN.pdf》

RKNN Toolkit Lite 使用指南：《Rockchip_User_Guide_RKNN_Toolkit_Lite_CN.pdf》

问题排查手册：《Rockchip_Trouble_Shooting_RKNN_Toolkit_CN.pdf》

自定义 OP 使用指南：《Rockchip_Developer_Guide_RKNN_Toolkit_Custom_OP_CN.pdf》

可视化功能使用指南：《Rockchip_User_Guide_RKNN_Toolkit_Visualization_CN.pdf》

以上文档均放在 SDK/doc 目录中，也可以访问以下链接查阅：

https://github.com/rockchip-linux/rknn-toolkit/tree/master/doc

10.2 问题反馈渠道

请通过 RKNN QQ 交流群，Github Issue 或瑞芯微 redmine 将问题反馈给 Rockchip NPU 团队。

RKNN QQ 交流群：1025468710

Github issue: https://github.com/rockchip-linux/rknn-toolkit/issues

Rockchip Redmine: https://redmine.rock-chips.com/

注：Redmine 账号需要通过销售或业务人员开通。如果是第三方开发板，请先找对应厂商反馈问题。