密级状态：绝密(　　)　　秘密(　　)　　内部(　)　　公开(√)

RKNN-Toolkit 用户使用指南

（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：	V1.6.0
	作　　者：	饶洪
	完成日期：	2020-12-31
	审　　核：	卓鸿添
	完成日期：	2020-12-31

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
V0.1	杨华聪	2018-08-25	初始版本	卓鸿添
V0.9.1	饶洪	2018-09-29	增加 RKNN-Toolkit 工具使用说明，包括主要功能、系统依赖、安装方式、使用场景及各 API 接口的详细说明。	卓鸿添
V0.9.2	卓鸿添	2018-10-12	优化性能预估方式	卓鸿添
V0.9.3	杨华聪	2018-10-24	添加连接开发板硬件说明	卓鸿添
V0.9.4	杨华聪	2018-11-03	添加 docker 镜像使用说明	卓鸿添
V0.9.5	饶洪	2018-11-19	1. 添加 npy 文件作为量化校正数据的使用说明； 2. build 接口 pre_compile 参数说明； 3. 完善 config 接口 reorder_channel 参数的使用说明	卓鸿添
V0.9.6	饶洪	2018-11-24	1. 新增接口 get_perf_detail_on_hardware 和 get_run_duration 的使用说明； 2. 更新 RKNN 初始化接口使用说明。	卓鸿添
V0.9.7	饶洪	2018-12-29	1. 接口优化：删除 get_run_duration、get_perf_detail_on_hardware 使用说明，重写 eval_perf 接口使用说明； 2. 重写 RKNN()接口使用说明； 3. 新增接口 init_runtime 的使用说明。	卓鸿添
V0.9.7.1	饶洪	2019-01-11	1. 解决多次调用 inference 后程序可能挂起的 BUG； 2. 接口调整：init_runtime 时不需要再指定 host，工具会自动判断。	卓鸿添
V0.9.8	饶洪	2019-01-30	1. 新增 verbose 选项，开启后可以打印模型加载、构建等阶段的日志信息，并写到指定文件中。	卓鸿添

版本	修改人	修改日期	修改说明	核定人
V0.9.9	饶洪	2019-03-06	1. 新增 eval_memory 接口，查看模型运行时的内存占用情况。 2. 优化 inference 接口；优化错误信息提示。 3. 新增 get_sdk_version 接口使用说明	卓鸿添
V1.0.0	饶洪	2019-05-08	1. 初始化运行时环境接口新增异步模式。 2. 推理接口新增输入透传模式。 3. 新功能：混合量化。 4. 优化 pre-compile 模型的加载时间。新版本工具生成的预编译模型无法在 NPU 驱动版本号小于 0.9.6 的设备上运行；旧版本生成的预编译模型也无法在新版本驱动上运行。 5. 调整模型推理结果的排列顺序：在 1.0.0 以前，如果原始模型输出的结果是按"NHWC"排列（如 TensorFlow），则工具会把结果转成"NCHW"；从 1.0.0 版本开始，将不做这个转换，而是保持跟原始模型的排列一致。	卓鸿添
V1.1.0	饶洪	2019-06-28	1. 新增对 TB-RK1808 AI 计算棒的支持。 2. 新增接口 list_devices，用来查询已连接设备信息。 3. 支持使用 Python 3.5 的 ARM64 Linux 平台。 4. 支持 Windows / Mac OS X 操作系统。	卓鸿添
V1.2.0	饶洪	2019-08-21	1. 新增对多 input 模型的支持。 2. 新增批量推理功能。 3. 新增模型分段功能，实现多模型同时运行。 4. 新增自定义算子功能。	卓鸿添

版本	修改人	修改日期	修改说明	核定人
V1.2.1	饶洪	2019-09-26	1. 解决使用 Python logging 模块报错的问题。 2. 解决多线程推理时在获取输出阶段发生段错误的问题。 3. 修复 dataset.txt 里所有文件路径都非法时构建量化模型会卡死的问题。 4. 调整 config 接口 batch_size 和 epochs 参数的默认值，修复 dataset.txt 中的数据未被充分利用的问题。 5. 新功能，load_rknn 接口支持直接加载 NPU 中的 rknn 模型。	卓鸿添
V1.3.0	饶洪	2019-12-23	1. 解决创建 RKNN 对象时间过长的问题。 2. 新增加载 Pytorch 模型功能。 3. 新增加载 MXNet 模型功能。 4. 新增对 4 通道输入的支持。 5. 新增误差分析功能。 6. 新增可视化功能。 7. 新增模型优化等级功能。 8. 优化混合量化功能。	卓鸿添

版本	修改人	修改日期	修改说明	核定人
V1.3.2	饶洪	2020-04-03	1. 增加对 RV1109、RV1126 的支持。 2. 完善 eval_perf 功能，不再需要填输入参数。 3. TensorFlow：增加对 reducemax 的支持；完善对 dilated convolution 的支持。 TFLite：增加对 dilated convolution 的支持。 Caffe：增加对 CRNN 的支持。 ONNX：增加对 Gather 和 Cast 的支持；完善对 avg_pool 的支持。 Pytorch：增加对 upsample_nearest2d, contiguous, softmax, permute, leaky_relu, prelue, log, deconv 和 sub 的支持；完善对 Reshape, Constant 的支持。 MXNet：增加对 Crop, UpSampling, SoftmaxActivation, _minus_scalar, log 的支持。 RKNN：完善对 reshape, concat, split 的支持。 4. 修复已知 bug。	卓鸿添

版本	修改人	修改日期	修改说明	核定人
V1.4.0	饶洪	2020-08-13	1. 新功能：增加逐层量化分析子功能；输入预处理支持多个 std_value；支持从开发板导出预编译模型。 2. 功能优化：优化 channel_mean_value 参数，改成 mean_values/std_values；移除 load_tensorflow 接口中的 mean_values 和 std_values；可视化完善对多输入的支持，增加对 RK1806/RV1109/RV1126 的支持；精度分析功能增加非归一化的余弦距离和欧式距离。 3. TensorFlow：增加对 dense 子图的支持。 TFLite：增加对 split_v 的支持；完善对 pad 的支持。 ONNX：完善对 prelu / deconvolution / avg_pool / clip 的支持。 Pytorch：增加对 pixel_shuffle, unsqueeze, sum,select, hardtanh, elu, slice, squeeze, exp,relu6, threshold_, matmul, exp, pad 的支持；完善对 adaptive_avg_pool2d, upsample_bilinear, relu6 的支持。 MXNet：完善对 fc 的支持。 Darknet: 增加对 mish 的支持；完善对 route 的支持。 4. 修复已知 bug。	卓鸿添

版本	修改人	修改日期	修改说明	核定人
V1.6.0	饶洪 洪启飞	2020-12-31	1. 新功能：支持 Keras 框架，并且支持 TF 2.0 导出的 h5 模型；支持 Pytorch 1.6.0；支持 ONNX 1.6.0；增加模型加密功能；支持通过命令行输出已连接设备；离线预编译支持多输入模型。 2. 功能优化：优化精度分析功能，完整模型比对时，遇到 Conv+Relu 等情况时，不再需要跳层比较；优化 RKNN 模型预处理，提高模型推理性能；优化性能分析功能，打印详情时，精简每层 OP；模型分段功能限制在 RK1806/RK1808/RV1109/RV1126 芯片范围；docker 镜像系统升级为 Ubuntu 18.04，Python 升级到 3.6。 3. 完善对各框架 OP 的支持。 4. 修复已知问题。	卓鸿添

目 录

1 概述

1.1 主要功能说明

RKNN-Toolkit 是为用户提供在 PC、 Rockchip NPU 平台上进行模型转换、推理和性能评估的开发套件，用户通过该工具提供的 Python 接口可以便捷地完成以下功能：

1) 模型转换：支持 Caffe、TensorFlow、TensorFlow Lite、ONNX、Darknet、Pytorch、MXNet 和 Keras 模型转成 RKNN 模型，支持 RKNN 模型导入导出，后续能够在 Rockchip NPU 平台上加载使用。从 1.2.0 版本开始支持多输入模型。从 1.3.0 版本开始支持 Pytorch 和 MXNet。从 1.6.0 版本开始支持 Keras 框架模型，并支持 TensorFlow 2.0 导出的 H5 模型。
2) 量化功能：支持将浮点模型转成量化模型，目前支持的量化方法有非对称量化（asymmetric_quantized-u8），动态定点量化（dynamic_fixed_point-8 和 dynamic_fixed_point-16）。从 1.0.0 版本开始，RKNN-Toolkit 开始支持混合量化功能，该功能的详细说明请参考第 3.3 章节。
3) 模型推理：能够在 PC 上模拟 Rockchip NPU 运行 RKNN 模型并获取推理结果；也可以将 RKNN 模型分发到指定的 NPU 设备上进行推理。
4) 性能评估：能够在 PC 上模拟 Rockchip NPU 运行 RKNN 模型，并评估模型性能（包括总耗时和每一层的耗时）；也可以将 RKNN 模型分发到指定 NPU 设备上运行，以评估模型在实际设备上运行时的性能。
5) 内存评估：评估模型运行时对系统和 NPU 内存的消耗情况。使用该功能时，必须将 RKNN 模型分发到 NPU 设备中运行，并调用相关接口获取内存使用信息。从 0.9.9 版本开始支持该功能。
6) 模型预编译：通过预编译技术生成的 RKNN 模型可以减少在硬件平台上的加载时间。对于部分模型，还可以减少模型尺寸。但是预编译后的 RKNN 模型只能在 NPU 设备上运行。目前只有 x86_64 Ubuntu 平台支持直接从原始模型生成预编译 RKNN 模型。RKNN-Toolkit 从 0.9.5 版本开始支持模型预编译功能，并在 1.0.0 版本中对预编译方法进行了升级，升级

后的预编译模型无法与旧驱动兼容。从 1.4.0 版本开始，也可以通过 NPU 设备将普通 RKNN 模型转成预编译 RKNN 模型，详情请参考接口 export_rknn_precompile_model 的使用说明。

7) 模型分段：该功能用于多模型同时运行的场景下，可以将单个模型分成多段在 NPU 上执行，借此来调节多个模型占用 NPU 的执行时间，避免因为一个模型占用太多执行时间，而使其他模型得不到及时执行。RKNN-Toolkit 从 1.2.0 版本开始支持该功能。目前，只有 RK1806/RK1808/RV1109/RV1126 芯片支持该功能，且 NPU 驱动版本要大于 0.9.8。

8) 自定义算子功能：如果模型含有 RKNN-Toolkit 不支持的算子（operator），那么在模型转换阶段就会失败。这时候可以使用自定义算子功能来添加不支持的算子，从而使模型能正常转换和运行。RKNN-Toolkit 从 1.2.0 版本开始支持该功能。自定义算子的使用和开发请参考《Rockchip_Developer_Guide_RKNN_Toolkit_Custom_OP_CN》文档。自定义算子目前只支持 TensorFlow 框架。

9) 量化精度分析功能：该功能将给出模型量化前后每一层推理结果的欧氏距离或余弦距离，以分析量化误差是如何出现的，为提高量化模型的精度提供思路。该功能从 1.3.0 版本开始支持。1.4.0 版本增加逐层量化精度分析子功能，将每一层运行时的输入指定为正确的浮点值，以排除逐层误差积累，能够更准确的反映每一层自身受量化的影响。

10) 可视化功能：该功能以图形界面的形式呈现 RKNN-Toolkit 的各项功能，简化用户操作步骤。用户可以通过填写表单、点击功能按钮的形式完成模型的转换和推理等功能，而不需要再去手动编写脚本。有关可视化功能的具体使用方法请参考《Rockchip_User_Guide_RKNN_Toolkit_Visualization_CN》文档。1.3.0 版本开始支持该功能。1.4.0 版本完善了对多输入模型的支持，并且支持 RK1806, RV1109, RV1126 等新的 RK NPU 设备。1.6.0 版本增加对 Keras 框架的支持。

11) 模型优化等级功能：RKNN-Toolkit 在模型转换过程中会对模型进行优化，默认的优化选项可能会对模型精度产生一些影响。通过设置优化等级，可以关闭部分或全部优化选项。有关优化等级的具体使用方法请参考 config 接口中 optimization_level 参数的说明。该功能从 1.3.0 版本开始支持。

12) 模型加密功能：RKNN-Toolkit 从 1.6.0 版本开始支持模型加密功能。

注：部分功能受限于对操作系统或芯片平台的依赖，在某些操作系统或平台上无法使用。各操作系统（平台）的功能支持列表如下：

	Ubuntu 16.04/18.04	Windows 7/10	Debian 9/10 (aarch64)	MacOS Mojave / Catalina
模型转换	支持	支持	支持	支持
量化/混合量化	支持	支持	支持	支持
模型推理	支持	支持	支持	支持
性能评估	支持	支持	支持	支持
内存评估	支持	支持	支持	支持
模型预编译	支持	部分支持（只支持在线预编译）	部分支持（只支持在线预编译）	部分支持（只支持在线预编译）
模型分段	支持	支持	支持	支持
自定义算子	支持	不支持	不支持	不支持
多输入	支持	支持	支持	支持
批量推理	支持	支持	支持	支持
设备查询	支持	支持	支持	支持
SDK 版本查询	支持	支持	支持	支持
量化精度分析	支持	支持	支持	支持
可视化功能	支持	支持	不支持	支持
模型优化开关	支持	支持	支持	支持
模型加密	支持	支持	支持	支持

1.2 适用芯片

RKNN-Toolkit 支持瑞芯微目前已发布的以下带 NPU 的芯片，这些芯片的型号如下：

- RK1806

- RK1808
- RK3399Pro(D/X)
- RV1109
- RV1126

注：RK3566 和 RK3568 暂时还不支持，需要使用另外一套工具。

1.3 适用系统

RKNN-Toolkit 是一个跨平台的开发套件，已支持的操作系统如下：

- Ubuntu: 16.04（x64）及以上
- Windows：7（x64）及以上
- MacOS: 10.13.5（x64）及以上
- Debian: 9.8（aarch64）及以上

2 系统依赖说明

使用本开发套件时需要满足以下运行环境要求：

表 1 运行环境

操作系统版本	Ubuntu16.04（x64）及以上 Windows 7（x64）及以上 Mac OS X 10.13.5（x64）及以上 Debian 9.8（aarch64）及以上
Python 版本	3.5/3.6/3.7
Python 库依赖	'numpy == 1.16.3' 'scipy == 1.3.0' 'Pillow == 5.3.0' 'h5py == 2.8.0' 'lmdb == 0.93' 'networkx == 1.11' 'flatbuffers == 1.10', 'protobuf == 3.11.2' 'onnx == 1.6.0' 'onnx-tf == 1.2.1' 'flask == 1.0.2' 'tensorflow == 1.11.0' or 'tensorflow-gpu' 'dill==0.2.8.2' 'ruamel.yaml == 0.15.81' 'psutils == 5.6.2' 'ply == 3.11' 'requests == 2.22.0' 'torch == 1.2.0' or 'torch == 1.5.1' or 'torch==1.6.0' 'mxnet == 1.5.0'

注：

1. Windows 只提供 Python3.6 的安装包。
2. MacOS 提供 python3.6 和 python3.7 的安装包。
3. ARM64 平台（安装 Debian 9 或 10 操作系统）提供 Python3.5（Debain 9）和 Python3.7（Debian10）的安装包。

4. 因为 Pytorch / TensorFlow 等逐渐停止对 Python3.5 的支持，RKNN Toolkit 下一个大版本将移除 Linux x86 平台上 Python3.5 的安装包，转而提供 Python3.6 和 Python3.7 的安装包。
5. 除 MacOS 平台外，其他平台的 scipy 依赖为>=1.1.0。
6. 本文档主要以 Ubuntu 16.04 / Python3.5 为例进行说明。其他操作系统请参考《Rockchip_Quick_Start_RKNN_Toolkit_V1.6.0_CN.pdf》。

3 使用说明

3.1 安装

目前提供两种方式安装 RKNN-Toolkit：一是通过 Python 包安装与管理工具 pip 进行安装；二是运行带完整 RKNN-Toolkit 工具包的 docker 镜像。下面分别介绍这两种安装方式的具体步骤。

注：Toybrick 设备上的安装流程请参考以下链接：

http://t.rock-chips.com/wiki.php?mod=view&id=36

3.1.1 通过 pip install 命令安装

1. 创建 virtualenv 环境（如果系统中同时有多个版本的 Python 环境，建议使用 virtualenv 管理 Python 环境）

```
sudo apt install virtualenv
sudo apt-get install libpython3.5-dev
sudo apt install python3-tk

virtualenv -p /usr/bin/python3 venv
source venv/bin/activate
```

2. 安装 TensorFlow、python-opencv 等依赖库：

```
# 如果要使用 TensorFlow GPU 版本，请执行以下命令
pip install tensorflow-gpu==1.11.0
# 如果要使用 TensorFlow CPU 版本，请执行以下命令
pip install tensorflow==1.11.0
# 执行以下命令安装 pytorch(ubuntu16.04 python3.5 官方只提供到 1.5.1
# 的安装包)和 torchvision
pip3 install torch==1.5.1
pip3 install torchvision==0.4.0
# 执行以下命令安装 mxnet
pip3 install mxnet==1.5.0
# 执行以下命令安装 opencv-python
pip install opencv-python
```

注：RKNN-Toolkit 本身并不依赖 opencv-python，但是在 example 中的示例都会用到这个库来

读取图片，所以这里将该库也一并安装了。

3. 安装 RKNN-Toolkit

```
pip install package/rknn_toolkit-1.6.0-cp35-cp35m-linux_x86_64.whl
```

请根据不同的 python 版本及处理器架构，选择不同的安装包文件（位于 package/目录）：

- **Python3.5 for x86_64**：rknn_toolkit-1.6.0-cp35-cp35m-linux_x86_64.whl
- **Python3.5 for arm_x64**：rknn_toolkit-1.6.0-cp35-cp35m-linux_aarch64.whl
- **Python3.6 for x86_64**：rknn_toolkit-1.6.0-cp36-cp36m-linux_x86_64.whl
- **Python3.6 for Windows x86_64:** rknn_toolkit-1.6.0-cp36-cp36m-win_amd64.whl
- **Python3.6 for Mac OS X:** rknn_toolkit-1.6.0-cp36-cp36m-macosx_10_15_x86_64.whl
- **Python3.7 for Mac OS X:** rknn_toolkit-1.6.0-cp37-cp37m-macosx_10_15_x86_64.whl
- **Python3.7 for arm_x64:** rknn_toolkit-1.6.0-cp37-cp37m-linux_aarch64.whl

3.1.2 通过 DOCKER 镜像安装

在 docker 文件夹下提供了一个已打包所有开发环境的 Docker 镜像，用户只需要加载该镜像即可直接上手使用 RKNN-Toolkit，使用方法如下：

1、 安装 Docker

请根据官方手册安装 Docker（https://docs.docker.com/install/linux/docker-ce/ubuntu/）。

2、 加载镜像

执行以下命令加载镜像：

```
docker load --input rknn-toolkit-1.6.0-docker.tar.gz
```

加载成功后，执行“docker images”命令能够看到 rknn-toolkit 的镜像，如下所示：

```
REPOSITORY      TAG       IMAGE ID       CREATED        SIZE
rknn-toolkit    1.6.0     d2efba8caa7c   1 hours ago    4.69GB
```

3、 运行镜像

执行以下命令运行 docker 镜像，运行后将进入镜像的 bash 环境。

```
docker run -t -i --privileged -v /dev/bus/usb:/dev/bus/usb rknn-toolkit:1.6.0 /bin/bash
```

如果想将自己代码映射进去可以加上“-v <host src folder>:<image dst folder>”参数，例如:

```
docker run -t -i --privileged -v /dev/bus/usb:/dev/bus/usb -v /home/rk/test:/test rknn-toolkit:1.6.0 /bin/bash
```

4、 运行 demo

```
cd /example/tflite/mobilenet_v1
python test.py
```

注：

1. **从 RKNN Toolkit V1.6.0 版本开始，提供的 docker 镜像将基于 Ubuntu 18.04 和 Python3.6。**
2. **该 docker 镜像基于 x86 Ubuntu 制作，只能在 Linux x86 平台上使用。**

3.2 RKNN-Toolkit 的使用

目前 RKNN Toolkit 可以运行在 PC（Linux/Windows/MacOS x64）上，也可以运行在 RK3399Pro 开发板（Debian9 或 Debian10）或 RK1808 AI 计算棒（Debian10）上。

接下来将详细给出各使用场景下 RKNN Toolkit 的使用流程。

注：使用流程里涉及的所有接口，其详细说明参考第 3.7 章节 API 详细说明。

3.2.1 场景一：模型运行在模拟器上

这种场景下，RKNN Toolkit 运行在 PC 上，通过模拟的 RK1808/RV1109 运行模型，以实现推理/性能评估等功能。

根据模型类型的不同，这个场景又可以区分为两个子场景：一是模型为非 RKNN 模型，即 Caffe、TensorFlow、TensorFlow Lite、ONNX、Darknet、Pytorch、MXNet、Keras 等模型；二是 RKNN 模型（Rockchip 的专有模型），文件后缀为“rknn”。

注：模拟器只有 x86_64 Linux 平台支持。

3.2.1.1 运行非 RKNN 模型

运行非 RKNN 模型与 RKNN 模型的最大区别在于，进行模型推理或模型性能/内存评估前，需要先将非 RKNN 模型转成 RKNN 模型。该场景下 RKNN Toolkit 的完整使用流程如下图所示：

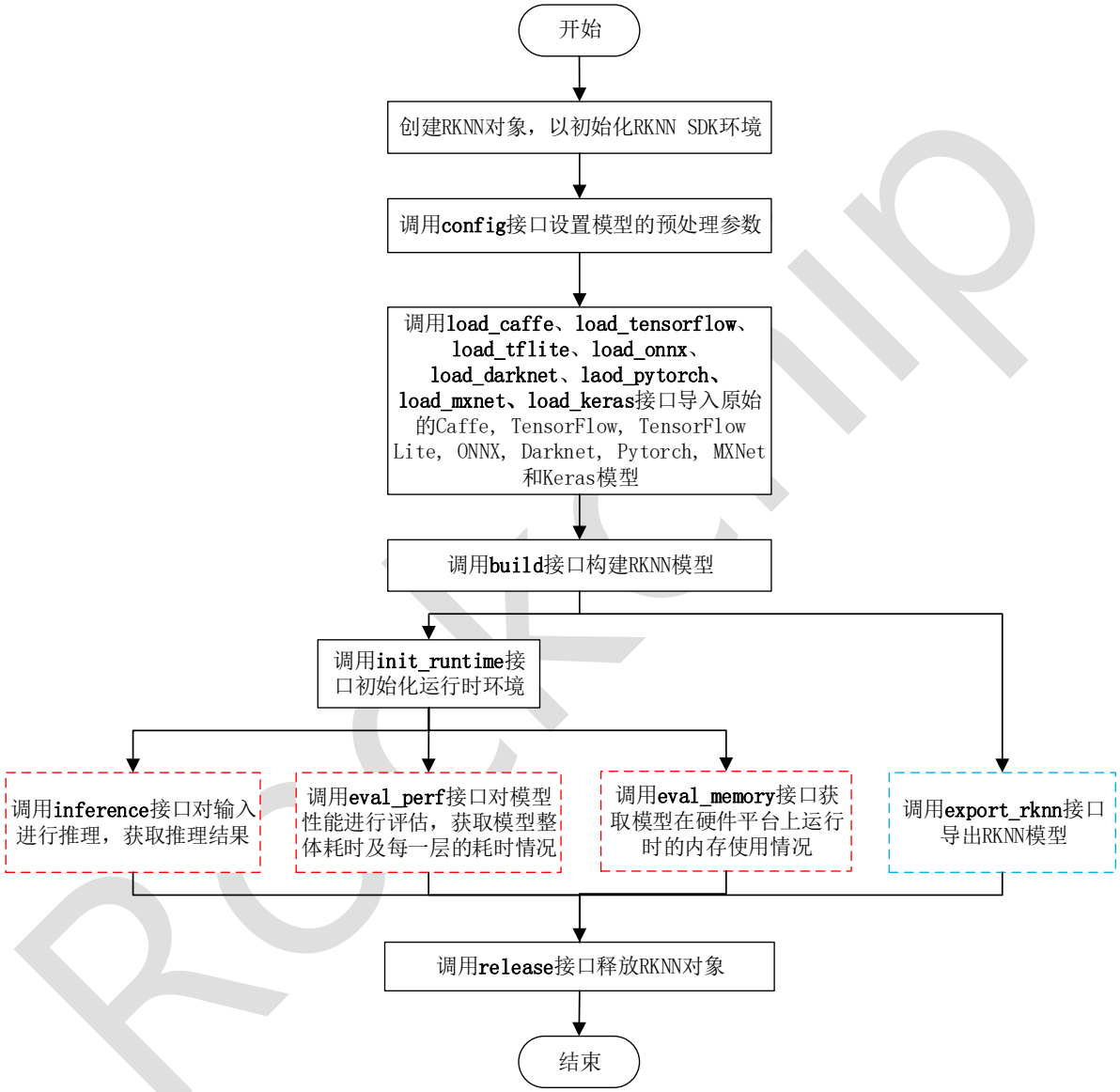

图 3-2-1-1-1 PC 上运行非 RKNN 模型时工具的使用流程

注：

1、 以上步骤请按顺序执行。

2、 蓝色框标注的步骤导出的 RKNN 模型可以通过 load_rknn 接口导入并使用。

3、 红色框标注的模型推理、性能评估和内存评估的步骤先后顺序不固定，根据实际使用情况决定。

4、 只有当目标平台是 Rockchip NPU 时，我们才可以调用 eval_memory 接口获取内存使用情况。

3.2.1.2 运行 RKNN 模型

运行 RKNN 模型时，用户不需要设置模型预处理参数，也不需要构建 RKNN 模型，其使用流程如下图所示：

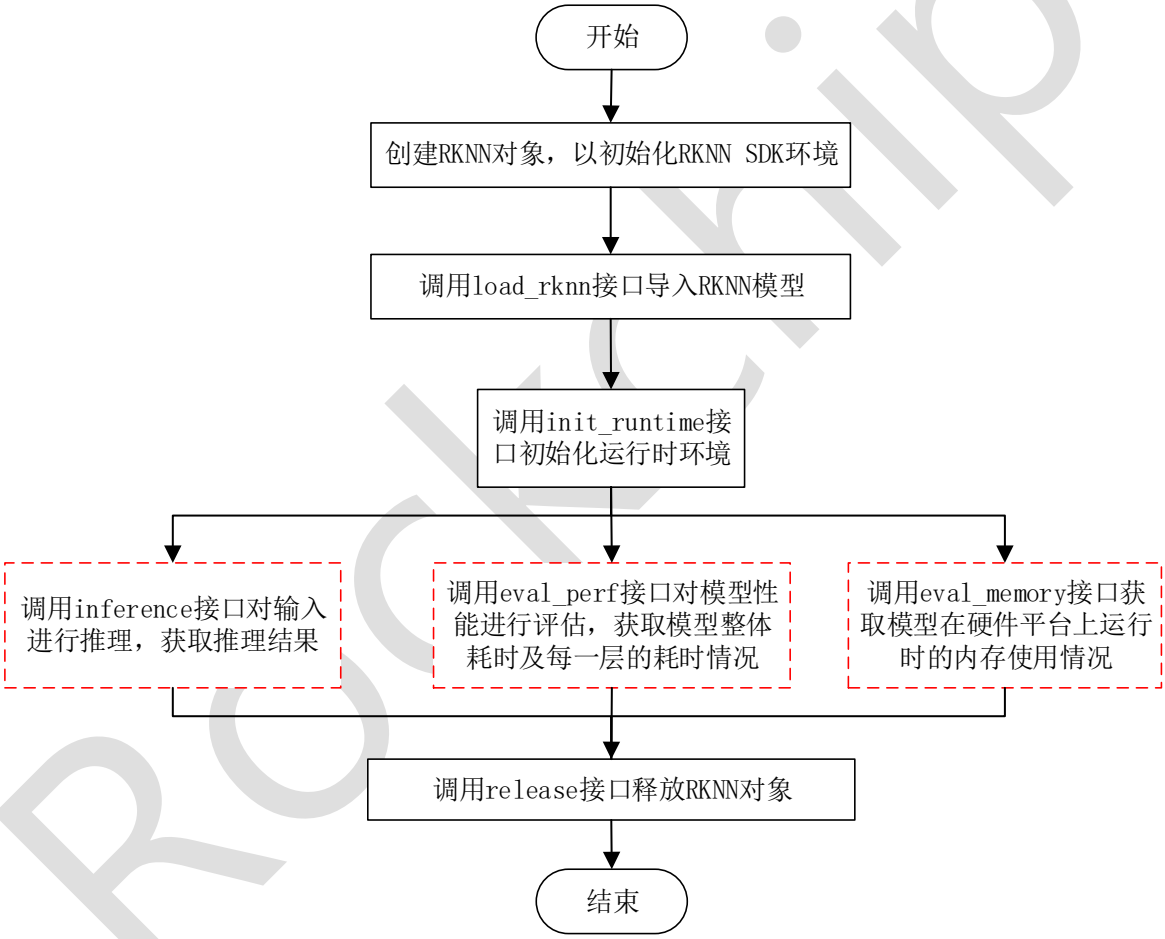

图 3-2-1-2-1 PC 上运行 RKNN 模型时工具的使用流程

注：

1、 以上步骤请按顺序执行。

2、 红色框标注的模型推理、性能评估和内存评估的步骤先后顺序不固定，根据实际使用情况决定。

3、 调用 eval_memory 接口获取内存使用情况时，模型必须运行在硬件平台上。

3.2.2 场景二：模型运行在与 PC 相连的 Rockchip NPU 平台上

RKNN Toolkit 目前支持的 Rockchip NPU 平台包括 RK1806, RK1808(或 TB-RK1808), RK3399Pro(D/X), RV1109 和 RV1126。

这种场景下，RKNN Toolkit 运行在 PC 上，通过 PC 的 USB 连接 NPU 设备。RKNN Toolkit 将 RKNN 模型传到 NPU 设备上运行，再从 NPU 设备上获取推理结果、性能信息等。

当模型为非 RKNN 模型（Caffe、TensorFlow、TensorFlow Lite、ONNX、Darknet、Pytorch、MXNet 等模型）时，RKNN-Toolkit 工具的使用流程及注意事项同场景一里的子场景一（见 3.2.1.1 章节）。

当模型为 RKNN 模型（后缀为“rknn”）时，RKNN-Toolkit 工具的使用流程及注意事项同场景一里的子场景二（见 3.2.1.2 章节）。

除此之外，在这个场景里，我们还需要完成以下两个步骤：

1、 确保开发板的 USB OTG 连接到 PC，并且正确识别到设备，即在 PC 上调用 python -m rknn.bin.list_devices 命令可以查询到相关设备。

2、 调用 init_runtime 接口初始化运行环境时需要指定 target 参数和 device_id 参数。其中 target 参数表明硬件类型，可选值为“rk1806”、“rk1808”、“rk3399pro”、“rv1109”或“rv1126”。当 PC 连接多个设备时，还需要指定 device_id 参数，即设备编号，设备编号可以通过 list_devices 接口或 python3 -m rknn.bin.list_devices 命令查询，示例如下：

```
all device(s) with adb mode:
[]
all device(s) with ntb mode:
['TB-RK1808S0', '515e9b401c060c0b']
```

初始化运行时环境**代码示例**如下：

```
# RK3399Pro
ret = init_runtime(target='rk3399pro', device_id='VGEJY9PW7T')

......

# RK1808
ret = init_runtime(target='rk1808', device_id='515e9b401c060c0b')
```

```
# TB-RK1808 AI 计算棒
ret = init_runtime(target='rk1808', device_id='TB-RK1808S0')

# RV1109
ret = init_runtime(target='rv1109', device_id='60a32d0000bb0709')

# RV1126
ret = init_runtime(target='rv1126', device_id='c3d9b8674f4b94f6')
```

注：

1. **目前 RK1806/RK1808/RV1109/RV1126 等设备支持 NTB 或 ADB 两种连接模式，RK3399Pro 开发板只支持 ADB 模式。使用多设备时，需要确保这些设备使用相同的连接模式，即 list_devices 查询出来的设备 ID 列表都是 ADB，或都是 NTB。**
2. **在 Linux 上第一次使用 NTB 设备时，非 root 用户需要获取该 USB 设备的读写权限，这可以通过执行 SDK/platform-tools/update_rk_usb_rule/linux/update_rk1808_usb_rule.sh 脚本完成，详细说明请参考同级目录下的 README.txt。**

3.2.3 场景三：模型运行在 RK3399Pro Linux 开发板上

这种场景下，RKNN-Toolkit 直接安装在 RK3399Pro Linux 系统里。构建或导入的 RKNN 模型直接在 RK3399Pro 上运行，以获取模型实际的推理结果或性能信息。

对于 RK3399Pro Linux 开发板，RKNN-Toolkit 工具的使用流程取决于模型种类，如果模型类型是非 RKNN 模型，则使用流程同场景一中的子场景一（见 3.2.1.1 章节）；否则使用流程同子场景二（见 3.2.1.2 章节）。

3.3 混合量化

RKNN-Toolkit 从 1.0.0 版本开始提供混合量化功能。

RKNN-Toolkit 提供的量化功能可以在提高模型性能的基础上尽量少地降低模型精度，但是不排除某些特殊模型在量化后出现精度下降较多的情况。为了让用户能够在性能和精度之间做更好的平衡，RKNN-Toolkit 从 1.0.0 版本开始引入混合量化功能，用户可以自己决定哪些层做量化还是不做量化，量化时候的参数也可以根据用户自己的经验进行修改。

注：

1. examples/common_function_demos 目录下提供了一个混合量化的例子 hybrid_quantization，可以参考该例子进行混合量化的实践。

3.3.1 混合量化功能用法

目前混合量化功能支持如下三种用法：

1. 将指定的量化层改成非量化层（用 float 进行计算），这种方式可能可以提高精度，但会损失一定的性能。
2. 将指定的非量化层改成量化层（量化方式与其他量化方式一样），这种方式可能会降低精度，但性能会提升。
3. 修改指定量化层的量化参数，这种方式对性能几乎不会产生影响，但精度可能更好、也可能更差。

3.3.2 混合量化配置文件

在使用混合量化功能时，第一步是生成一个混合量化配置文件，本节对该配置文件进行简要介绍。

当我们调用混合量化接口 hybrid_quantization_step1 后，会在当前目录下生成一个 {model_name}.quantization.cfg 的 yaml 配置文件。配置文件格式如下：

```
%YAML 1.2
# add layer name and corresponding quantized_dtype to customized_quantize_layers, e.g conv2_3: float32
customized_quantize_layers: {}
quantize_parameters:
    '@attach_concat_1/out0_0:out0':
        dtype: asymmetric_affine
        method: layer
        max_value:
        -   10.097497940063477
        min_value:
        -   -52.340476989746094
        zero_point:
        -   214
        scale:
```

```
        -    0.24485479295253754
        qtype: u8

    ......

        '@FeatureExtractor/MobilenetV2/Conv/Conv2D_230:bias':
            dtype: asymmetric_affine
            method: layer
            max_value:
            min_value:
            zero_point: 0
            scale:
            -    0.00026041566161438823
            qtype: i32
```

第一行是 yaml 的版本，第二行是一个分隔符，第三行是注释。后面是配置文件的主要内容。

配置文件正文第一行是一个自定义量化层的字典，将层名和相应的量化类型（可选值为 **asymmetric_affine-u8, dynamic_fixed_point-i8, dynamic_fixed_point-i16, float32**）添加到这儿，作为自定义的量化层。从 1.6.0 版本开始，第一步会生成一些默认的层，这些层仅做参考。

接着是模型每层的量化参数，每一层都是一个字典。每个字典的 key 由@{layer_name}_{layer_id}组成，其中 layer_name 是层名，layer_id 是层 id。字典的 value 即量化参数，如果没有经过量化，则 Value 里只有 dtype 项，且值为 None。

3.3.3 混合量化使用流程

使用混合量化功能时，可以分四步进行。

第一步，加载原始模型，生成量化配置文件和模型结构文件和模型配置文件。具体的接口调用流程如下：

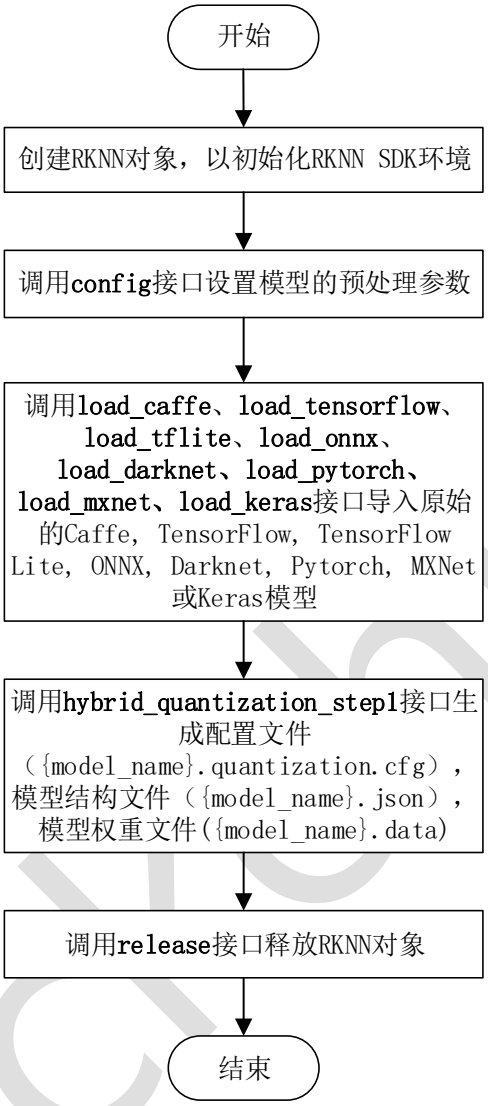

图 3-3-3-1 混合量化第一步接口调用流程

第二步，修改第一步生成的量化配置文件。

- 如果是将某些量化层改成非量化层，则找到不要量化的层，将这些层名加到 customized_quantize_layers 字典中，值为 float32，例如”<layer_name>: float32”。也可以使用其他量化方式，例如原来是用 asymmetric_affine-u8 的，这里也可以改成 dynamic_fixed_point-i8 或 dynamic_fixed_point-i16。但一个模型最多同时只能存在两种量化方式。层名最好用双引号括起来，避免因为特殊字符导致解析失败。
- 如果是将某些层从非量化改成量化，同样找到这些层，将它的层名加到 customized_quantize_layers 字典中，例如“<layername>: asymmetric_affine-u8”。
- 如果是要修改量化参数，直接修改指定层的量化参数即可，customized_quantize_layers 置

成空。

注：从 RKNN Toolkit 1.6.0 版本开始，第一步生成的量化配置文件会给出一些混合量化建议，仅供参考。

第三步，生成 RKNN 模型。具体的接口调用流程如下：

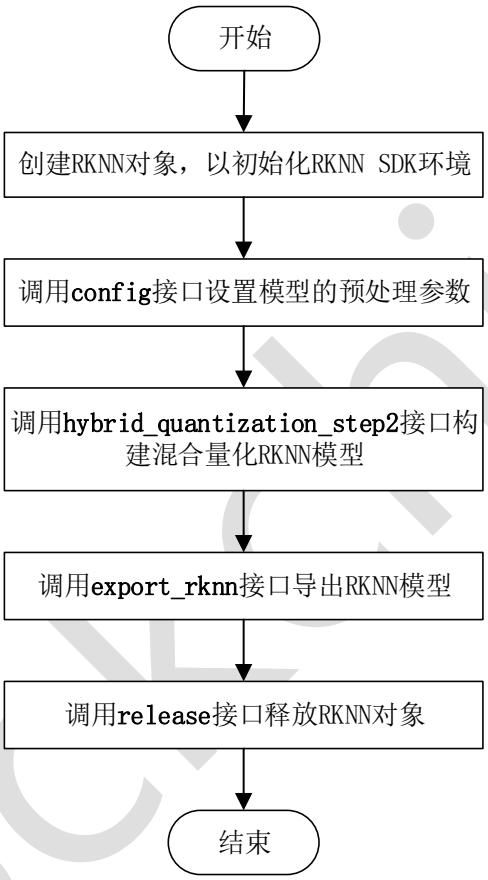

图 3-3-3-2 混合量化第三步接口调用流程

第四步，使用上一步生成的 RKNN 模型进行推理。

3.4 模型分段

RKNN-Toolkit 从 1.2.0 版本开始引入模型分段功能。该功能用于多模型同时跑的场景下，可以将单个模型分成多段在 NPU 上执行，借此来调节多个模型占用 NPU 的执行时间，避免因为一个模型占用太多的执行时间，而其他的模型得不到及时的执行。

每个分段（未启用模型分段功能的模型默认就是一个分段）抢占 NPU 的机会是均等的，在一个分段执行完成后，会主动让出 NPU（如果该模型还有下一分段，则会将该分段再次加入到命令

队列尾部），此时如果有其他的模型的分段在等待执行，则会按命令队列先后顺序执行其他模型的分段。

通过调用 export_rknn_sync_model 接口，可以将普通的 RKNN 模型分成多段，该接口的详细用法请参考 3.7.15 章节。

如果在单模型跑的场景下，则需要关闭该功能（不使用分段模型即可）。因为开启模型分段功能会降低单模型执行的效率，但多模型同时跑的场景下不会降低模型执行效率，因此只推荐在多模型同时跑的场景下才使用该功能。

3.5 精度分析

3.5.1 功能介绍

RKNN Toolkit 从 1.3.0 版本开始引入量化精度分析功能，用来分析量化模型和浮点模型每一层结果的差距，为提高模型精度提供改进思路。

1.3.0 版本的量化精度分析功能，其工作思路是在工具端用浮点模型完整推理一次，然后量化模型再完整推理一次。推理过程中，将每一层的结果保存到指定目录中，然后再计算每一层的欧式距离（归一化）和余弦距离（归一化），以判断每一层的误差。

在 1.4.0 版本的 RKNN Toolkit 中，该功能做了两个主要更新：一是在工具端引入逐层的误差分析；二是引入设备端的逐层误差分析。其中工具端的逐层误差分析是在原有的全模型误差分析基础上，将整个模型拆成一层一层，然后每一层以浮点模型前一层的结果作为输入，这样可以避免模型后面的层因为输入本身已经偏差较大，导致该层的结果也相差很大从而无法判断是否是量化引起的这种情况。前面完整模型的比较和逐层模型的比较，都是在工具端进行的，而第二个改进则是将 NPU 的结果也用来进行比较，这样可以更直观的反应模型在 NPU 运行时的误差情况。

在 1.3.0 到 1.4.0 版本之间，完整模型的误差分析，并没有考虑到 Conv/Relu 等层在量化后是合并在一起的，造成比较结果时，进行了错误的比较。1.6.0 版本对此类比较做了优化。

3.5.2 使用流程

量化精度分析功能使用流程如下：

图 3-5-2-1 量化精度分析使用流程

注：

1. 待分析的量化方法需要在 config 中指定；
2. 在 build 或 hybrid_quantization_step1 中的 dataset.txt 可以包含多张图片；但 accuracy_analysis 中指定的 inputs 只能包含一张图片；
3. 上述接口的调用顺序不能调换。

3.5.3 输出说明

如果没有设置 target，输出的目录结构如下：

```
├── entire_qnt
├── entire_qnt_error_analysis.txt
├── fp32
```

```
├─── individual_qnt
├─── individual_qnt_error_analysis.txt
```

各文件/目录含义如下：

- entire_qnt 目录：保存整个量化模型完整运行时每一层的结果（已转成 float32）；
- entire_qnt_error_analysis.txt：记录量化模型完整运算时每一层结果与浮点模型的余弦距离/欧式距离，归一化后的余弦距离/欧式距离。余弦距离越小或欧式距离越大，表明量化后的精度下降得越厉害。
- fp32 目录：保存整个浮点模型完整跑下来时每一层的结果，可以根据该目录中 order.txt 记录的顺序与原始模型进行对应。如果浮点模型本身结果不对，可以将该目录中每一层的结果与原始框架推理时每一层的结果进行对比，从而确定出问题的是哪一层，然后反馈给瑞芯微 NPU 团队。
- individual_qnt 目录：将量化模型拆成一层一层，逐层运行。每一层在推理时的输入为上一层用浮点模型推理的结果。该目录保存一层一层单独运行时的结果（已转成 float32）。这样做可以避免累积误差。
- individual_qnt_error_analysis.txt 文件：记录量化模型逐层运行时每一层的结果与浮点模型的余弦距离/欧氏距离，归一化后的余弦距离/欧式距离。余弦距离越小或欧式距离越大，表明量化后的精度下降得越厉害。

  如果设置了 target，该目录下会多出以下内容：

```
├─── individual_qnt_error_analysis_on_npu.txt
└─── qnt_npu_dump
```

- individual_qnt_error_analysis_on_npu.txt 文件：记录量化模型逐层在硬件设备上运行时每一层结果与浮点模型的余弦距离/欧氏距离，归一化后的余弦距离/欧式距离。余弦距离越小或欧式距离越大，表明量化后的精度下降得越厉害。
- qnt_npu_dump 目录：将量化模型拆成一层一层后逐个放到 NPU 设备上运行，所用的输入为浮点模型上一层的结果，该目录保存量化模型逐层在 NPU 上实际运行时的结果（dump 结果时会自动转成 float32，方便比较）。

3.6 示例

以下是加载 TensorFlow Lite 模型的示例代码（详细参见 example/tflite/mobilenet_v1 目录），如果在 PC 上执行这个例子，RKNN 模型将在模拟器上运行：

```
import numpy as np
import cv2
from rknn.api import RKNN

def show_outputs(outputs):
    output = outputs[0][0]
    output_sorted = sorted(output, reverse=True)
    top5_str = 'mobilenet_v1\n-----TOP 5-----\n'
    for I in range(5):
        value = output_sorted[i]
        index = np.where(output == value)
        for j in range(len(index)):
            if (I + j) >= 5:
                break
            if value > 0:
                topi = '{}: {}\n'.format(index[j], value)
            else:
                topi = '-1: 0.0\n'
            top5_str += topi
    print(top5_str)

def show_perfs(perfs):
    perfs = 'perfs: {}\n'.format(outputs)
    print(perfs)

if __name__ == '__main__':

    # Create RKNN object
    rknn = RKNN()

    # pre-process config
    print('→ config model')
    rknn.config(channel_mean_value='103.94 116.78 123.68 58.82',
reorder_channel='0 1 2')
    print('done')

    # Load tensorflow model
    print('→ Loading model')
    ret = rknn.load_tflite(model='./mobilenet_v1.tflite')
    if ret != 0:
        print('Load mobilenet_v1 failed!')
```

```
        exit(ret)
    print('done')

    # Build model
    print('--> Building model')
    ret = rknn.build(do_quantization=True, dataset='./dataset.txt')
    if ret != 0:
        print('Build mobilenet_v1 failed!')
        exit(ret)
    print('done')

    # Export rknn model
    print('--> Export RKNN model')
    ret = rknn.export_rknn('./mobilenet_v1.rknn')
    if ret != 0:
        print('Export mobilenet_v1.rknn failed!')
        exit(ret)
    print('done')

    # Set inputs
    img = cv2.imread('./dog_224x224.jpg')
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

    # init runtime environment
    print('--> Init runtime environment')
    ret = rknn.init_runtime()
    if ret != 0:
        print('Init runtime environment failed')
        exit(ret)
    print('done')

    # Inference
    print('--> Running model')
    outputs = rknn.inference(inputs=[img])
    show_outputs(outputs)
    print('done')

    # perf
    print('--> Begin evaluate model performance')
    perf_results = rknn.eval_perf(inputs=[img])
    print('done')

    rknn.release()
```

其中 dataset.txt 是一个包含测试图片路径的文本文件，例如我们在 example/tflite/mobilent_v1 目录下有一张 dog_224x224.jpg 的图片，那么对应的 dataset.txt 内容如下

```
dog_224x224.jpg
```

demo 运行模型预测时输出如下结果：

```
-----TOP 5-----
[156]: 0.85107421875
[155]: 0.09173583984375
[205]: 0.01358795166015625
[284]: 0.006465911865234375
[194]: 0.002239227294921875
```

评估模型性能时输出如下结果（在 PC 上执行这个例子，结果仅供参考）：

Performance

Layer ID	Name	Time(us)
0	tensor.transpose_3	72
44	convolution.relu.pooling.layer2_2	363
59	convolution.relu.pooling.layer2_2	201
45	convolution.relu.pooling.layer2_2	185
60	convolution.relu.pooling.layer2_2	243
46	convolution.relu.pooling.layer2_2	98
61	convolution.relu.pooling.layer2_2	149
47	convolution.relu.pooling.layer2_2	104
62	convolution.relu.pooling.layer2_2	120
48	convolution.relu.pooling.layer2_2	72
63	convolution.relu.pooling.layer2_2	101
49	convolution.relu.pooling.layer2_2	92
64	convolution.relu.pooling.layer2_2	99
50	convolution.relu.pooling.layer2_2	110
65	convolution.relu.pooling.layer2_2	107
51	convolution.relu.pooling.layer2_2	212
66	convolution.relu.pooling.layer2_2	107
52	convolution.relu.pooling.layer2_2	212
67	convolution.relu.pooling.layer2_2	107
53	convolution.relu.pooling.layer2_2	212
68	convolution.relu.pooling.layer2_2	107
54	convolution.relu.pooling.layer2_2	212
69	convolution.relu.pooling.layer2_2	107
55	convolution.relu.pooling.layer2_2	212
70	convolution.relu.pooling.layer2_2	107
56	convolution.relu.pooling.layer2_2	174
71	convolution.relu.pooling.layer2_2	220
57	convolution.relu.pooling.layer2_2	353
28	pooling.layer2_1	36
58	fullyconnected.relu.layer_3	110
30	softmaxlayer2.layer_1	90

Total Time(us): 4694

FPS(800MHz): 213.04

3.7 API 详细说明

3.7.1 RKNN 初始化及对象释放

在使用 RKNN Toolkit 的所有 API 接口时，都需要先调用 RKNN()方法初始化一个 RKNN 对象，并在用完后调用该对象的 release()方法将对象释放掉。

初始化 RKNN 对象时，可以设置 ***verbose*** 和 ***verbose_file*** 参数，以打印详细的日志信息。其中 verbose 参数指定是否要在屏幕上打印详细日志信息；如果设置了 verbose_file 参数，且 verbose 参数值为 True，日志信息还将写到这个参数指定的文件中。

举例如下：

```
# 将详细的日志信息输出到屏幕，并写到 mobilenet_build.log 文件中
rknn = RKNN(verbose=True, verbose_file='./mobilenet_build.log')
# 只在屏幕打印详细的日志信息
rknn = RKNN(verbose=True)
...
rknn.release()
```

3.7.2 RKNN 模型配置

在构建 RKNN 模型之前，需要先对模型进行通道均值、通道顺序、量化类型等的配置，这可以通过 config 接口完成。

API	**config**
描述	设置模型参数
参数	batch_size：批处理大小，默认值为 100。量化时将根据该参数决定每一批次喂的数据量，以校正量化结果。如果 dataset 中的数据量小于 100，则该参数值将自动调整为 dataset 中的数据量。
	mean_values：输入的均值。该参数与 channel_mean_value 参数不能同时设置。参数格式是一个列表，列表中包含一个或多个均值子列表，多输入模型对应多个子列表，每个子列表的长度与该输入的通道数一致，形如[[128,128,128]]，表示一个输入的三

	个通道的值减去 128。如果 reorder_channel 设置成'2 1 0'，则优先做通道调整，再做减均值。
	std_values：输入的归一化值。该参数与 channel_mean_value 参数不能同时设置。参数格式是一个列表，列表中包含一个或多个归一化值子列表，多输入模型对应多个子列表，每个子列表的长度与该输入的通道数一致，形如[[128,128,128]]，表示设置一个输入的三个通道的值减去均值后再除以 128。如果 reorder_channel 设置成'2 1 0'，则优先做通道调整，再减均值和除以归一化值。
	epochs：量化时的迭代次数，每迭代一次，就选择 batch_size 指定数量的图片进行量化校正。默认值为-1，此时 RKNN-Toolkit 会根据 dataset 中的图片数量自动计算迭代次数以最大化利用数据集中的数据。
	reorder_channel：表示是否需要对图像通道顺序进行调整。'0 1 2'表示按照输入的通道顺序来推理，比如图片输入时是 RGB，那推理的时候就根据 RGB 顺序传给输入层；'2 1 0'表示会对输入做通道转换，比如输入时通道顺序是 RGB，推理时会将其转成 BGR，再传给输入层，同样的，输入时通道的顺序为 BGR 的话，会被转成 RGB 后再传给输入层。如果有多个输入，对应的每个输入的参数以"#"进行分割，如 '0 1 2#0 1 2'。
	need_horizontal_merge：是否需要进行水平合并，默认值为 False。如果模型是 inception v1/v3/v4，建议开启该选项，可以提高推理时的性能。
	quantized_dtype：量化类型，目前支持的量化类型有 asymmetric_quantized-u8、dynamic_fixed_point-8、dynamic_fixed_point-16，默认值为 asymmetric_quantized-u8。
	optimization_level：模型优化等级。通过修改模型优化等级，可以关掉部分或全部模型转换过程中使用到的优化规则。该参数的默认值为 3，打开所有优化选项。值为 2 或 1 时关闭一部分可能会对部分模型精度产生影响的优化选项，值为 0 时关闭所有优化选项。
	target_platform: 指定 RKNN 模型是基于哪个目标芯片平台生成的。目前支持 RK1806、RK1808、RK3399Pro、RV1109 和 RV1126。其中基于 RK1806、RK1808 或 RK3399Pro 生成的 RKNN 模型可以在这两个平台上通用，基于 RV1109 或 RV1126 生成的 RKNN 模

	型可以在这两个平台通用。如果模型要在 RK1806、RK1808 或 RK3399Pro 上运行，该参数的值可以是 [“rk1806”], [“rk1808”], [“rk3399pro”] 或 [“rk1806”, “rk1808”, “rk3399pro”]等；如果模型要在 RV1109 或 RV1126 上运行，该参数的值可以是 [“rv1126”], [“rv1109”]或[“rv1109”, “rv1126”]等。这个参数的值不可以是类似[“rk1808”, “rv1126”]这样的组合，因为这两款芯片互相是不兼容的。如果不填该参数，则默认是 [“rk1808”]，生成的 RKNN 模型可以在 RK1806、RK1808 和 RK3399Pro 平台上运行。该参数的值大小写不敏感。
返回值	无

举例如下：

```
# model config
rknn.config(mean_values=[[103.94, 116.78, 123.68]],
            std_values=[[58.82, 58.82, 58.82]],
            reorder_channel='0 1 2',
            need_horizontal_merge=True,
            target_platform=['rk1808', 'rk3399pro'])
```

3.7.3 模型加载

RKNN-Toolkit 目前支持 Caffe、TensorFlow、TensorFlow Lite、ONNX、Darknet、Pytorch、MXNet 和 Keras 等非 RKNN 模型，它们在加载时调用的接口不同，下面详细说明这些框架对应的加载接口。

3.7.3.1 Caffe 模型加载接口

API	**load_caffe**
描述	加载 caffe 模型
参数	model：caffe 模型文件（.prototxt 后缀文件）所在路径。
	proto：caffe 模型的格式（可选值为'caffe'或'lstm_caffe'）。为了支持 RNN 模型，增加了相关网络层的支持，此时需要设置 caffe 格式为'lstm_caffe'。

	blobs：caffe 模型的二进制数据文件（.caffemodel 后缀文件）所在路径。该参数值可以为 None，RKNN Toolkit 将随机生成权重等参数。
返回值	0：导入成功
	-1：导入失败

举例如下：

```
# 从当前路径加载 mobilenet_v2 模型
ret = rknn.load_caffe(model='./mobilenet_v2.prototxt',
                      proto='caffe',
                      blobs='./mobilenet_v2.caffemodel')
```

3.7.3.2 TensorFlow 模型加载接口

API	**load_tensorflow**
描述	加载 TensorFlow 模型
参数	tf_pb：TensorFlow 模型文件（.pb 后缀）所在路径。
	inputs：模型输入节点，支持多个输入节点。所有输入节点名放在一个列表中。
	input_size_list：每个输入节点对应的图片的尺寸和通道数。如示例中的 mobilenet-v1 模型，其输入节点对应的输入尺寸是[224, 224, 3]。
	outputs：模型的输出节点，支持多个输出节点。所有输出节点名放在一个列表中。
	predef_file：为了支持一些控制逻辑，需要提供一个 npz 格式的预定义文件。可以通过以下方法生成预定义文件：np.savez('prd.npz', placeholder_name=prd_value)。如果“placeholder_name”中包含'/'，请用'#'替换。
返回值	0：导入成功
	-1：导入失败

举例如下：

```
# 从当前目录加载 ssd_mobilenet_v1_coco_2017_11_17 模型
ret = rknn.load_tensorflow(
      tf_pb='./ssd_mobilenet_v1_coco_2017_11_17.pb',
      inputs=[`FeatureExtractor/MobilenetV1/MobilenetV1/Conv2d_0
```

```
            /BatchNorm/batchnorm/mul_1'],
        outputs=['concat', 'concat_1'],
        input_size_list=[[300, 300, 3]])
```

3.7.3.3 TensorFlow Lite 模型加载接口

API	**load_tflite**
描述	加载 TensorFlow Lite 模型。 注： 因为 tflite 不同版本的 schema 之间是互不兼容的，所以构建的 tflite 模型使用与 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 是基于官网 master 分支上的提交: 0c4f5dfea4ceb3d7c0b46fc04828420a344f7598。官网地址如下： https://github.com/tensorflow/tensorflow/commits/master/tensorflow/lite/schema/schema.fbs
参数	model：TensorFlow Lite 模型文件（.tflite 后缀）所在路径
返回值	0：导入成功
	-1：导入失败

举例如下：

```
# 从当前目录加载 mobilenet_v1 模型
ret = rknn.load_tflite(model = './mobilenet_v1.tflite')
```

3.7.3.4 ONNX 模型加载

API	**load_onnx**
描述	加载 ONNX 模型
参数	model：ONNX 模型文件（.onnx 后缀）所在路径。
返回值	0：导入成功

	-1：导入失败

举例如下：

```
# 从当前目录加载 arcface 模型
ret = rknn.load_onnx(model = `./arcface.onnx')
```

3.7.3.5 Darknet 模型加载接口

API	**load_darknet**
描述	加载 Darknet 模型
参数	model：Darknet 模型文件（.cfg 后缀）所在路径。
	weight：权重文件（.weights 后缀）所在路径
返回值	0：导入成功
	-1：导入失败

举例如下：

```
# 从当前目录加载 yolov3-tiny 模型
ret = rknn.load_darknet(model = `./yolov3-tiny.cfg',
                        weight='./yolov3.weights')
```

3.7.3.6 Pytorch 模型加载接口

API	**load_pytorch**
描述	加载 Pytorch 模型
参数	model：Pytorch 模型文件（.pt 后缀）所在路径，而且需要是 torchscript 格式的模型。必填参数。
	input_size_list：每个输入节点对应的图片的尺寸和通道数。例如[[1,224,224],[3,224,224]]表示有两个输入，其中一个输入的 shape 是[1,224,224]，另外一个输入的 shape 是[3,224,224]。必填参数。

	convert_engine：RKNN Toolkit 1.6.0 版本引入该参数，用以指定 pytorch 模型的转换引擎。RKNN Toolkit 目前提供 torch1.2, torc 这两种转换引擎。其中“torch1.2”沿袭旧版转换引擎，只能转换 torch1.1 到 1.2 版本之间的 pytorch 模型。而“torch”则可以支持到 Pytorch 1.6.0，要求 torch 版本高于 1.5.0，这也是该版本默认使用的转换引擎。这是一个选填参数，默认值为“torch”。
返回值	0：导入成功
	-1：导入失败

举例如下：

```
# 从当前目录加载 resnet18 模型
ret = rknn. Load_pytorch(model = ‘./resnet18.pt’,
                         input_size_list=[[3,224,224]])
```

3.7.3.7 MXNet 模型加载接口

API	**load_mxnet**
描述	加载 MXNet 模型
参数	symbol：MXNet 模型的网络结构文件，后缀是 json。必填参数。
	params：MXnet 模型的参数文件，后缀是 params。必填参数。
	input_size_list：每个输入节点对应的图片的尺寸和通道数。例如[[1,224,224],[3,224,224]]表示有两个输入，其中一个输入的 shape 是[1,224,224]，另外一个输入的 shape 是[3,224,224]。必填参数。
返回值	0：导入成功
	-1：导入失败

举例如下：

```
# 从当前目录加载 resnext50 模型
ret = rknn.load_mxnet(symbol=’resnext50_32x4d-symbol.json’,
                      params=’resnext50_32x4d-4ecf62e2.params’,
```

```
                                input_size_list=[[3,224,224]] )
```

3.7.3.8 Keras 模型加载接口

API	**load_keras**
描述	加载 Keras 模型
参数	model：Keras 模型文件（后缀为.h5）。必填参数。
	convert_engine：转换引擎，可以是'Keras'或'tflite'。默认转换引擎为 Keras。可选参数。
返回值	0：导入成功
	-1：导入失败

举例如下：

```
# 从当前目录加载 xception 模型
ret = rknn.load_keras(model='./xception_v3.h5')
```

3.7.4 构建 RKNN 模型

API	**build**
描述	根据导入的 Caffe、TensorFlow、TensorFlow Lite 等模型，构建对应的 RKNN 模型。
参数	do_quantization：是否对模型进行量化，值为 True 或 False。
	dataset：量化校正数据的数据集。目前支持文本文件格式，用户可以把用于校正的图片（jpg 或 png 格式）或 npy 文件路径放到一个.txt 文件中。文本文件里每一行一条路径信息。如： a.jpg b.jpg 或 a.npy b.npy

	如有多个输入，则每个输入对应的文件用空格隔开，如： a.jpg a2.jpg b.jpg b2.jpg 或 a.npy a2.npy b.npy b2.npy
	pre_compile：预编译开关，如果设置成 True，可以减小模型大小，及模型在硬件设备上的首次启动速度。但是打开这个开关后，构建出来的模型就只能在硬件平台上运行，无法通过模拟器进行推理或性能评估。如果硬件有更新，则对应的模型要重新构建。 **注：** **1. 该选项不能在 RK3399Pro Linux 开发板 / Windows PC / Mac OS X PC 上使用。** **2. RKNN-Toolkit-V1.0.0 及以上版本生成的预编译模型不能在 NPU 驱动版本小于 0.9.6 的设备上运行；RKNN-Toolkit-V1.0.0 以前版本生成的预编译模型不能在 NPU 驱动版本大于等于 0.9.6 的设备上运行。驱动版本号可以通过 get_sdk_version 接口查询。** **3. 如果多个输入，该选项需要设置成 False。**
	rknn_batch_size：模型的输入 Batch 参数调整，默认值为 1。如果大于 1，则可以在一次推理中同时推理多帧输入图像或输入数据，如 MobileNet 模型的原始 input 维度为[1, 224, 224, 3]，output 维度为[1, 1001]，当 rknn_batch_size 设为 4 时，input 的维度变为[4, 224, 224, 3]，output 维度变为[4, 1001]。 **注：** **1. rknn_batch_size 的调整并不会提高一般模型在 NPU 上的执行性能，但却会显著增加内存消耗以及增加单帧的延迟。** **2. rknn_batch_size 的调整可以降低超小模型在 CPU 上的消耗，提高超小模型的平均帧率。（适用于模型太小，CPU 的开销大于 NPU 的开销）。**

	3. rknn_batch_size 的值建议小于 32，避免内存占用太大而导致推理失败。 **4. rknn_batch_size 修改后，模型的 input/output 维度会被修改，使用 inference 推理模型时需要设置相应的 input 的大小，后处理时，也需要对返回的 outputs 进行处理。**
返回值	0：构建成功
	-1：构建失败

举例如下：

```
# 构建 RKNN 模型，并且做量化
ret = rknn.build(do_quantization=True, dataset='./dataset.txt')
```

3.7.5 导出 RKNN 模型

前一个接口构建的 RKNN 模型可以保存成一个文件，之后如果想要再使用该模型进行结果预测或性能分析，直接通过 load_rknn 接口加载模型即可，无需再用原始模型构建对应的 RKNN 模型。

API	**export_rknn**
描述	将 RKNN 模型保存到指定文件中（.rknn 后缀）。
参数	export_path：导出模型文件的路径。
返回值	0：导出成功
	-1：导出失败

举例如下：

```
……
# 将构建好的 RKNN 模型保存到当前路径的 mobilenet_v1.rknn 文件中
ret = rknn.export_rknn(export_path = './mobilenet_v1.rknn')
……
```

3.7.6 加载 RKNN 模型

API	**load_rknn**

描述	加载 RKNN 模型。
参数	path：RKNN 模型文件路径。
	load_model_in_npu: 是否直接加载 npu 中的 rknn 模型。其中 path 为 rknn 模型在 npu 中的路径。只有当 RKNN-Toolkit 运行在 RK3399Pro Linux 开发板或连有 NPU 设备的 PC 上时才可以设为 True。默认值为 False。
返回值	0：加载成功
	-1：加载失败

举例如下：

```
# 从当前路径加载 mobilenet_v1.rknn 模型
ret = rknn.load_rknn(path='./mobilenet_v1.rknn')
```

3.7.7 初始化运行时环境

在模型推理或性能评估之前，必须先初始化运行时环境，确定模型在哪一个硬件平台上运行或直接通过模拟器运行。

API	**init_runtime**
描述	初始化运行时环境。确定模型运行的设备信息（硬件平台信息、设备 ID）；性能评估时是否启用 debug 模式，以获取更详细的性能信息。
参数	target：目标硬件平台，目前支持“rk3399pro”、“rk1806”、“rk1808”、“rv1109”、“rv1126”。默认为 None，即在 PC 使用工具时，模型在模拟器上运行，在 RK3399Pro Linux 开发板运行时，模型在 RK3399Pro 自带 NPU 上运行，否则在设定的 target 上运行。其中“rk1808”包含了 TB-RK1808 AI 计算棒。
	device_id：设备编号，如果 PC 连接多台设备时，需要指定该参数，设备编号可以通过”***list_devices***”接口查看。默认值为 None。 ***注：MAC OS X 系统当前版本还不支持多个设备。***
	perf_debug：进行性能评估时是否开启 debug 模式。在 debug 模式下，可以获取到每一层的运行时间，否则只能获取模型运行的总时间。默认值为 False。

	eval_mem: 是否进入内存评估模式。进入内存评估模式后，可以调用 eval_memory 接口获取模型运行时的内存使用情况。默认值为 False。
	async_mode：是否使用异步模式。调用推理接口时，涉及设置输入图片、模型推理、获取推理结果三个阶段。如果开启了异步模式，设置当前帧的输入将与推理上一帧同时进行，所以除第一帧外，之后的每一帧都可以隐藏设置输入的时间，从而提升性能。在异步模式下，每次返回的推理结果都是上一帧的。该参数的默认值为 False。
返回值	0：初始化运行时环境成功。
	-1：初始化运行时环境失败。

举例如下：

```
# 初始化运行时环境
ret = rknn.init_runtime(target='rk1808', device_id='012345789AB')
if ret != 0:
    print('Init runtime environment failed')
    exit(ret)
```

3.7.8 模型推理

在使用模型进行推理前，必须先构建或加载一个 RKNN 模型。

API	**inference**
描述	使用模型进行推理，返回推理结果。 如果 RKNN-Toolkit 运行在 PC 上，且初始化运行环境时设置 target 为 Rockchip NPU 设备，得到的是模型在硬件平台上的推理结果。 如果 RKNN-Toolkit 运行在 PC 上，且初始化运行环境时没有设置 target，得到的是模型在模拟器上的推理结果。模拟器可以模拟 RK1808,也可以模拟 RV1126,具体模拟哪款芯片,取决于 RKNN 模型的 target_platform 参数值. 如果 RKNN-Toolkit 运行在 RK3399Pro Linux 开发板上，得到的是模型在实际硬件上的推理结果。
参数	inputs：待推理的输入，如经过 cv2 处理的图片。格式是 ndarray list。

	data_type：输入数据的类型，可填以下值：’float32’, ‘float16’, ‘int8’, ‘uint8’, ‘int16’。默认值为’uint8’。
	data_format：数据模式，可以填以下值: “nchw”, “nhwc”。默认值为’nhwc’。这两个的不同之处在于 channel 放置的位置。
	inputs_pass_through: 将输入透传给 NPU 驱动。非透传模式下，在将输入传给 NPU 驱动之前，工具会对输入进行减均值、除方差等操作；而透传模式下，不会做这些操作。这个参数的值是一个数组，比如要透传 input0，不透彻 input1，则这个参数的值为[1, 0]。默认值为 None，即对所有输入都不透传。
返回值	results：推理结果，类型是 ndarray list。

举例如下：

对于分类模型，如 mobilenet_v1，代码如下（完整代码参考 example/tflite/mobilent_v1）：

```
# 使用模型对图片进行推理，得到 TOP5 结果
.....
outputs = rknn.inference(inputs=[img])
show_outputs(outputs)
.....
```

输出的 TOP5 结果如下：

```
-----TOP 5-----
[156]: 0.85107421875
[155]: 0.09173583984375
[205]: 0.01358795166015625
[284]: 0.006465911865234375
[194]: 0.002239227294921875
```

对于目标检测的模型，如 ssd_mobilenet_v1，代码如下(完整代码参考 example/tensorflow/ssd_mobilenet_v1)：

```
# 使用模型对图片进行推理，得到目标检测结果
.....
outputs = rknn.inference(inputs=[image])
.....
```

输出的结果经过后处理后输出如下图片（物体边框的颜色是随机生成的，所以每次运行这个

example 得到的边框颜色会有所不同）：

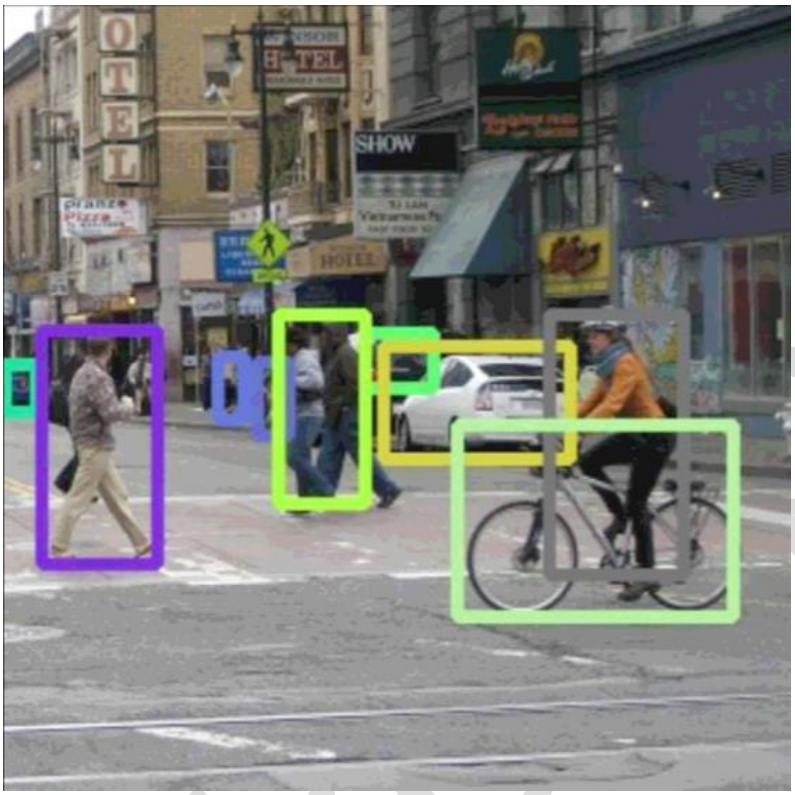

图 3-4-8-1 ssd_mobilenet_v1 inference 结果

3.7.9 评估模型性能

API	**eval_perf**
描述	评估模型性能。 模型运行在 PC 上，初始化运行环境时不指定 target，得到的是模型在模拟器上运行的性能数据，包含逐层的运行时间及模型完整运行一次需要的时间。模拟器可以模拟 RK1808,也可以模拟 RV1126,具体模拟哪款芯片,取决于 RKNN 模型的 target_platform 参数值。 模型运行在与 PC 连接的 Rockchip NPU 上，且初始化运行环境时设置 perf_debug 为 False，则获得的是模型在硬件上运行的总时间；如果设置 perf_debug 为 True，除了返回总时间外，还将返回每一层的耗时情况。 模型运行在 RK3399Pro Linux 开发板上时，如果初始化运行环境时设置 perf_debug 为 False，获得的也是模型在硬件上运行的总时间；如果设置 perf_debug 为 True，返回

	总时间及每一层的耗时情况

返回值	perf_result：性能信息。类型为字典。在硬件平台上运行，且初始运行环境时设置 perf_debug 为 False 时，得到的字典只有一个字段'total_time'，示例如下：
	`{ 'total_time': 1000 }`
	其他场景下，得到的性能信息字典多一个'layers'字段，这个字段的值也是一个字典，这个字典以每一层的 ID 作为 key，其值是一个包含'name'（层名）、'operation'（操作符，只有运行在硬件平台上时才有这个字段）、'time'（该层耗时）等信息的字典。举例如下：
	`{ 'total_time', 4568, 'layers', { '0': { 'name': 'convolution.relu.pooling.layer2_2', 'operation': 'CONVOLUTION', 'time', 362 } '1': { 'name': 'convolution.relu.pooling.layer2_2', 'operation': 'CONVOLUTION', 'time', 158 } } }`

举例如下：

```
# 对模型性能进行评估
……
rknn.eval_perf(inputs=[image], is_print=True)
……
```

如 example/tensorflow/ssd_mobilenet_v1，其性能评估结果打印如下（以下是在 PC 模拟器上得到的结果，连接硬件设备时得到的详情与该结果略有不同）：

```
============================================
                Performance
============================================
Layer ID     Name                                Time(us)
```

0	tensor.transpose_3	125
71	convolution.relu.pooling.layer2_3	324
105	convolution.relu.pooling.layer2_2	331
72	convolution.relu.pooling.layer2_2	438
106	convolution.relu.pooling.layer2_2	436
73	convolution.relu.pooling.layer2_2	223
107	convolution.relu.pooling.layer2_2	374
74	convolution.relu.pooling.layer2_2	327
108	convolution.relu.pooling.layer2_3	533
75	convolution.relu.pooling.layer2_2	167
109	convolution.relu.pooling.layer2_2	250
76	convolution.relu.pooling.layer2_2	293
110	convolution.relu.pooling.layer2_2	249
77	convolution.relu.pooling.layer2_2	164
111	convolution.relu.pooling.layer2_2	256
78	convolution.relu.pooling.layer2_2	319
112	convolution.relu.pooling.layer2_2	256
79	convolution.relu.pooling.layer2_2	319
113	convolution.relu.pooling.layer2_2	256
80	convolution.relu.pooling.layer2_2	319
114	convolution.relu.pooling.layer2_2	256
81	convolution.relu.pooling.layer2_2	319
115	convolution.relu.pooling.layer2_2	256
82	convolution.relu.pooling.layer2_2	319
83	convolution.relu.pooling.layer2_2	173
27	tensor.transpose_3	48
84	convolution.relu.pooling.layer2_2	45
28	tensor.transpose_3	6
116	convolution.relu.pooling.layer2_3	299
85	convolution.relu.pooling.layer2_2	233
117	convolution.relu.pooling.layer2_2	314
86	convolution.relu.pooling.layer2_2	479
87	convolution.relu.pooling.layer2_2	249
35	tensor.transpose_3	29
88	convolution.relu.pooling.layer2_2	30
36	tensor.transpose_3	5
89	convolution.relu.pooling.layer2_2	122
90	convolution.relu.pooling.layer2_3	715
91	convolution.relu.pooling.layer2_2	98
41	tensor.transpose_3	10
92	convolution.relu.pooling.layer2_2	11
42	tensor.transpose_3	5
93	convolution.relu.pooling.layer2_2	31
94	convolution.relu.pooling.layer2_3	205
95	convolution.relu.pooling.layer2_2	51
47	tensor.transpose_3	6
96	convolution.relu.pooling.layer2_2	6
48	tensor.transpose_3	4
97	convolution.relu.pooling.layer2_2	17
98	convolution.relu.pooling.layer2_3	204

```
99          convolution.relu.pooling.layer2_2          51
53          tensor.transpose_3                         5
100         convolution.relu.pooling.layer2_2          6
54          tensor.transpose_3                         4
101         convolution.relu.pooling.layer2_2          10
102         convolution.relu.pooling.layer2_2          21
103         fullyconnected.relu.layer_3                13
104         fullyconnected.relu.layer_3                8
Total Time(us): 10622
FPS(800MHz): 94.14
============================================
```

3.7.10 获取内存使用情况

API	**eval_memory**
描述	获取模型在硬件平台运行时的内存使用情况。 模型必须运行在与 PC 连接的 Rockchip NPU 设备上，或直接运行在 RK3399Pro Linux 开发板上。 **注：使用该功能时，对应的驱动版本必须要大于等于 0.9.4。驱动版本可以通过 get_sdk_version 接口查询。**
参数	is_print: 是否以规范格式打印内存使用情况。默认值为 True。
返回值	memory_detail：内存使用情况。类型为字典。 内存使用情况按照下面的格式封装在字典中： `{ 'system_memory', { 'maximum_allocation': 128000000, 'total_allocation': 152000000 }, 'npu_memory', { 'maximum_allocation': 30000000, 'total_allocation': 40000000 }, 'total_memory', { 'maximum_allocation': 158000000, 'total_allocation': 192000000 } }` ● ’system_memory’字段表示系统内存占用。

	● ’npu_memory’表示 NPU 内部内存的使用情况。 ● ’total_memory’是系统内存和 NPU 内部内存的和。 ● ’maximum_allocation’是内存使用的峰值，单位是 Byte。表示从模型运行开始到结束内存的最大分配值，这是一个峰值。 ● ’total_allcation’表示整个运行过程中分配的所有内存之和。

举例如下：

```
# 对模型内存使用情况进行评估
……
memory_detail = rknn.eval_memory()
……
```

如 example/tflite 中的 mobilenet_v1，它在 RK1808 上运行时内存占用情况如下：

```
==============================================
            Memory Profile Info Dump
==============================================
System memory:
    maximum allocation : 22.65 MiB
    total allocation   : 72.06 MiB
NPU memory:
    maximum allocation : 33.26 MiB
    total allocation   : 34.57 MiB

Total memory:
    maximum allocation : 55.92 MiB
    total allocation   : 106.63 MiB

INFO: When evaluating memory usage, we need consider
the size of model, current model size is: 4.10 MiB
==============================================
```

3.7.11 混合量化

3.7.11.1 hybrid_quantization_step1

使用混合量化功能时，第一阶段调用的主要接口是 hybrid_quantization_step1，用于生成模型结构文件（{model_name}.json）、权重文件（{model_name}.data）和量化配置文件

（{model_name}.quantization.cfg）。接口详情如下：

API	**hybrid_quantization_step1**
描述	根据加载的原始模型，生成对应的模型结构文件、权重文件和量化配置文件。
参数	dataset: 量化校正数据的数据集。目前支持文本文件格式，用户可以把用于校正的图片（jpg 或 png 格式）或 npy 文件路径放到一个.txt 文件中。文本文件里每一行一条路径信息。如： a.jpg b.jpg 或 a.npy b.npy
返回值	0：成功
	-1：失败

举例如下：

```
# Call hybrid_quantization_step1 to generate quantization config
……
ret = rknn.hybrid_quantization_step1(dataset='./dataset.txt')
……
```

3.7.11.2 hybrid_quantization_step2

使用混合量化功能时，生成混合量化 RKNN 模型阶段调用的主要接口是 hybrid_quantization_step2。接口详情如下：

API	**hybrid_quantization_step2**
描述	接收模型结构文件、权重文件、量化配置文件、校正数据集作为输入，生成混合量化后的 RKNN 模型。
参数	model_input: 第一步生成的模型结构文件，形如“{model_name}.json”。数据类型为字符串。必填参数。

	data_input: 第一步生成的权重数据文件，形如“{model_name}.data”。数据类型为字符串。必填参数。
	model_quantization_cfg: 经过修改后的模型量化配置文件，形如“{model_name}.quantization.cfg”。数据类型为字符串。必填参数。
	dataset: 量化校正数据的数据集。目前支持文本文件格式，用户可以把用于校正的图片（jpg 或 png 格式）或 npy 文件路径放到一个.txt 文件中。文本文件里每一行一条路径信息。如： a.jpg b.jpg 或 a.npy b.npy
	pre_compile：预编译开关，如果设置成 True，可以减小模型大小，及模型在硬件设备上的首次启动速度。但是打开这个开关后，构建出来的模型就只能在硬件平台上运行，无法通过模拟器进行推理或性能评估。如果硬件有更新，则对应的模型要重新构建。 **注：** **1. 该选项不能在 RK3399Pro Linux 开发板 / Windows PC / Mac OS X PC 上使用。** **2. RKNN-Toolkit-V1.0.0 及以上版本生成的预编译模型不能在 NPU 驱动版本小于 0.9.6 的设备上运行；RKNN-Toolkit-V1.0.0 以前版本生成的预编译模型不能在 NPU 驱动版本大于等于 0.9.6 的设备上运行。驱动版本号可以通过 get_sdk_version 接口查询。** **3. 如果多个输入，该选项需要设置成 False。**
返回值	0：成功
	-1：失败

举例如下：

```
# Call hybrid_quantization_step2 to generate hybrid quantized RKNN model
......
ret = rknn.hybrid_quantization_step2(
        model_input='./ssd_mobilenet_v2.json',
        data_input='./ssd_mobilenet_v2.data',
        model_quantization_cfg='./ssd_mobilenet_v2.quantization.cfg',
        dataset='./dataset.txt')
......
```

3.7.12 量化精度分析

该接口的功能是进行浮点、量化推理并产生每层的数据，用于量化精度分析。

API	**accuracy_analysis**
描述	使用浮点和量化模型推理并将每一层的结果做快照。之后再根据快照里的数据，比较量化模型和浮点模型每一层的差距，用于评估量化所产生的误差或错误。 **注：** 1. **该接口只能在 build 或 hybrid_quantization_step1 之后调用，并且原始模型应该为非量化的模型，否则会调用失败。** 2. **该接口使用的量化方式与 config 中指定的一致。**
参数	inputs：包含输入图像或数据的数据集文本文件（与量化校正数据的数据集文件同格式，但只能包含一行字符串，详见“构建 RKNN 模型章节”）。
	output_dir：输出目录，所有快照都保存在该目录。该目录内容的详细说明见 3.5.3 章节。
	calc_qnt_error：是否计算量化误差（默认为 True）。
	target: 指定设备类型。如果指定 target，在逐层量化误差分析时，还将连接到 NPU 上获取每一层的真实结果，以跟 float 结果相比较。这样可以更准确的反映实际运行时的误差。
	device_id: 如果 PC 连接了多个 NPU 设备，需要指定具体设备的 ID。
返回值	0：成功
	-1：失败

举例如下：

```
……

print('--> config model')
rknn.config(channel_mean_value='0 0 0 1', reorder_channel='0 1 2')
print('done')

print('--> Loading model')
ret = rknn.load_onnx(model='./mobilenetv2-1.0.onnx')
if ret != 0:
    print('Load model failed! Ret = {}'.format(ret))
    exit(ret)
print('done')

# Build model
print('--> Building model')
ret = rknn.build(do_quantization=True, dataset='./dataset.txt')
if ret != 0:
    print('Build rknn failed!')
    exit(ret)
print('done')

print('--> Accuracy analysis')
rknn.accuracy_analysis(inputs='./dataset.txt', target='rk1808')
print('done')

……
```

3.7.13 注册自定义算子

API	**register_op**
描述	注册自定义算子。
参数	op_path：算子编译生成的 rknnop 文件的路径
返回值	无

参考代码如下所示。注意，该接口要在模型转换前调用。自定义算子的使用和开发请参考《Rockchip_Developer_Guide_RKNN_Toolkit_Custom_OP_CN》文档。

```
rknn.register_op('./resize_area/ResizeArea.rknnop')
```

```
rknn.load_tensorflow(...)
```

3.7.14 导出预编译模型（在线预编译）

构建 RKNN 模型时，可以指定预编译选项以导出预编译模型，这被称为离线编译。同样，RKNN Toolkit 也提供在线编译的接口：export_rknn_precompile_model。使用该接口，可以将普通 RKNN 模型转成预编译模型。

API	**export_rknn_precompile_model**
描述	经过在线编译后导出预编译模型。 **注：** **1. 使用该接口前必须先调用 load_rknn 接口加载普通 rknn 模型；** 2. **使用该接口前必须调用 init_runtime 接口初始化模型运行环境，target 必须是 RK NPU 设备，不能是模拟器；而且要设置 rknn2precompile 参数为 True。**
参数	export_path: 导出模型路径。必填参数。
返回值	0：成功
	-1：失败

举例如下：

```
from rknn.api import RKNN

if __name__ == '__main__':
    # Create RKNN object
    rknn = RKNN()

    # Load rknn model
    ret = rknn.load_rknn('./test.rknn')
    if ret != 0:
        print('Load RKNN model failed.')
        exit(ret)

    # init runtime
    ret = rknn.init_runtime(target='rk1808', rknn2precompile=True)
    if ret != 0:
        print('Init runtime failed.')
        exit(ret)
```

```
# Note: the rknn2precompile must be set True when call init_runtime
ret = rknn.export_rknn_precompile_model('./test_pre_compile.rknn')
if ret != 0:
    print('export pre-compile model failed.')
    exit(ret)

rknn.release()
```

3.7.15 导出分段模型

该接口的功能是将普通的 RKNN 模型转成分段模型，分段的位置由用户指定。

API	**export_rknn_sync_model**
描述	在用户指定的模型层后面插入 sync 层，用于将模型分段，并导出分段后的模型。
参数	input_model: 待分段的 rknn 模型路径。数据类型为字符串。必填参数。
	sync_uids: 待插入 sync 节点层的层 uid 列表，RKNN-Toolkit 将在这些层后面插入 sync 层。 **注：** **1. uid 只能通过 eval_perf 接口获取，且需要在 init_runtime 时设置 perf_debug 为 True。此时，PC 需要连接到 RK1806/RK1808/RK1808 AI 计算棒/RV1109/RV1126 上。** 2. **uid 的值不可以随意填写，一定需要是在 eval_perf 获取性能详情的 uid 列表中，否则可能出现不可预知的结果。**
	output_model: 导出模型的保存路径。数据类型：字符串。默认值为 None，如果不填该参数，导出的模型将保存在 input_model 指定的文件中。
返回值	0：成功
	-1：失败

举例如下：

```
from rknn.api import RKNN

if __name__ == '__main__':
```

```
rknn = RKNN()
ret = rknn.export_rknn_sync_model(
            input_model='./ssd_inception_v2.rknn',
            sync_uids=[206, 186, 152, 101, 96, 67, 18, 17],
            output_model='./ssd_inception_v2_sync.rknn')
if ret != 0:
    print('export sync model failed.')
    exit(ret)
rknn.release()
```

3.7.16 导出加密模型

该接口的功能是将普通的 RKNN 模型进行加密，得到加密后的模型。

API	**export_encrypted_rknn_model**
描述	根据用户指定的加密等级对普通的 RKNN 模型进行加密。
参数	input_model: 待加密的 RKNN 模型路径。数据类型为字符串。必填参数。
	output_model: 模型加密后的保存路径。数据类型为字符串。选填参数，如果不填该参数，则使用{original_model_name}.crypt.rknn 作为加密后的模型的名字。
	crypt_level: 加密等级。等级越高，安全性越高，解密越耗时；反之，安全性越低，解密越快。数据类型为整型，默认值为 1，目前安全等级有 3 个等级，1，2 和 3。
返回值	0：成功
	-1：失败

举例如下：

```
from rknn.api import RKNN

if __name__ == '__main__':
    rknn = RKNN()
    ret = rknn.export_encrypted_rknn_model('test.rknn')
    if ret != 0:
        print('Encrypt RKNN model failed.')
        exit(ret)
    rknn.release()
```

3.7.17 查询 SDK 版本

API	**get_sdk_version**
描述	获取 SDK API 和驱动的版本号。 **注：使用该接口前必须完成模型加载和初始化运行环境。且该接口只有在 target 是 Rockchip NPU 或 RKNN Toolkit 运行在 RK3399Pro Linux 开发板才可以使用。**
参数	无
返回值	sdk_version：API 和驱动版本信息。类型为字符串。

举例如下：

```
# 获取 SDK 版本信息
.....
sdk_version = rknn.get_sdk_version()
.....
```

返回的 SDK 信息如下：

```
=============================================
RKNN VERSION:
  RKNNAPI:    API: 1.4.0 (b4a8096 build: 2020-08-12 10:15:19)
  RKNNAPI:    DRV: 1.4.0 (b4a8096 build: 2020-08-13 08:27:47)
=============================================
```

3.7.18 获取设备列表

API	**list_devices**
描述	列出已连接的 RK3399PRO/RK1808/RV1109/RV1126 或 TB-RK1808 AI 计算棒。 注：目前设备连接模式有两种：ADB 和 NTB。其中 RK3399PRO 目前只支持 ADB 模式，TB-RK1808 AI 计算棒只支持 NTB 模式，RK1808/RV1109 支持 ADB/NTB 模式。多设备连接时请确保他们的模式都是一样的。
参数	无。
返回值	返回 adb_devices 列表和 ntb_devices 列表，如果设备为空，则返回空列表。 例如我们的环境里插了两个 TB-RK1808 AI 计算棒，得到的结果如下：

	adb_devices = [] ntb_devices = ['TB-RK1808S0', '515e9b401c060c0b']

举例如下：

```
from rknn.api import RKNN

if __name__ == '__main__':
    rknn = RKNN()
    rknn.list_devices()
    rknn.release()
```

返回的设备列表信息如下（这里有两个 RK1808 开发板，它们的连接模式都是 adb）：

```
**************************
all device(s) with adb mode:
['515e9b401c060c0b', 'XGOR2N4EZR']
**************************
```

注：使用多设备时，需要保证它们的连接模式都是一致的，否则会引起冲突，导致设备连接失败。

3.7.19 查询模型可运行平台

API	**list_support_target_platform**
描述	查询给定 RKNN 模型可运行的芯片平台。
参数	rknn_model：RKNN 模型路径。如果不指定模型路径，则按类别打印 RKNN Toolkit 当前支持的芯片平台。
返回值	support_target_platform: Returns runnable chip platforms, or None if the model path is empty.

参考代码如下所示：

```
rknn.list_support_target_platform(rknn_model='mobilenet_v1.rknn')
```

参考结果如下：

```
**************************************************
```

```
    Target platforms filled in RKNN model:              []
    Target platforms supported by this RKNN model: ['RK1806', 'RK1808',
'RK3399PRO']
    ************************************************
```