Rockchip User Guide RKNN-Toolkit CN

V1.2.1

目 录

1 主要功能说明

RKNN-Toolkit 是为用户提供在 PC、RK3399Pro、RK1808、TB-RK1808 AI 计算棒或 RK3399Pro Linux 开发板上进行模型转换、推理和性能评估的开发套件，用户通过提供的 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 模型转成 RKNN 模型，支持 RKNN 模型导入导出，后续能够在硬件平台上加载使用。从 V1.2.0 版本开始支持多输入模型。

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 V1.0.0 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 上模拟运行模型并获取推理结果；也可以在指定硬件平台 RK3399Pro（或 RK3399Pro Linux 开发板）、RK1808、TB-RK1808 AI 计算棒上运行模型并获取推理结果。

4） 性能评估：能够在 PC 上模拟运行并获取模型总耗时及每一层的耗时信息；也可以通过联机调试的方式在指定硬件平台 RK3399Pro、RK1808、TB-RK1808 AI 计算棒上运行模型，或者直接在 RK3399Pro Linux 开发板上运行，以获取模型在硬件上完整运行一次所需的总时间和每一层的耗时情况。

5） 内存评估：评估模型运行时对系统和 NPU 内存的消耗情况。通过联机调试的方式获取模型在指定硬件平台 RK3399Pro、RK1808、TB-RK1808 AI 计算棒或 RK3399Pro Linux 开发板上运行时的内存使用情况。从 V0.9.9 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 模型只能在带有 NPU 的硬件平台上运行。目前只有 x86_64 Ubuntu 平台支持根据原始模型直接生成 RKNN 模型。RKNN-Toolkit 从 V0.9.5 版本开始支持模型预编译功能，并在 V1.0.0 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。

7） 模型分段：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执行时间，而使其他模型得不到及时执行。RKNN-Toolkit 从 V1.2.0 版本开始支持该功能。该功能必须在带有 NPU 的硬件上使用，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 V1.2.0 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。

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 9.8 (ARM 64)	MacOS Mojave
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	不支持	不支持	不支持
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

2 系统依赖说明

本开发套件支持运行于 Ubuntu、Windows、MacOS、Debian 等操作系统。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（x64）及以上
Python 版本	3.5/3.6
Python 库依赖	'numpy >= 1.16.1' 'scipy >= 1.1.0' 'Pillow >= 3.1.2' 'h5py >= 2.7.1' 'lmdb >= 0.92' 'networkx == 1.11' 'flatbuffers == 1.9', 'protobuf >= 3.5.2' 'onnx >= 1.4.0' 'onnx_tf == 1.2.0' 'flask >= 1.0.2' 'tensorflow >= 1.11.0' 'dill==0.2.8.2' 'opencv-python>=3.4.3.18' 'ruamel.yaml==0.15.82' 'psutils>=5.6.2'

注：

1. Windows 及 MacOS 只提供 Python3.6 的安装包。
2. 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_V1.2.1_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 pip install 命令安装；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：RKNN-Toolkit 在 RK3399Pro Linux 开发板上的安装流程可以参考以下链接：

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，请执行以下命令安装 TensorFlow 依赖
pip install tensorflow-gpu
# 如果要使用 TensorFlow CPU 版本，请执行以下
pip install tensorflow
# 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-1.2.1-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-1.2.1-cp35-cp35m-linux_x86_64.whl
- **Python3.5 for arm_x64**：rknn_toolkit-1.2.1-cp35-cp35m-linux_aarch64.whl
- **Python3.6 for x86_64**：rknn_toolkit-1.2.1-cp36-cp36m-linux_x86_64.whl
- **Python3.6 for arm_x64**: rknn_toolkit-1.2.1-cp36-cp36m-linux_aarch64.whl
- **Python3.6 for Windows x86_64:** rknn_toolkit-1.2.1-cp36-cp36m-win_amd64.whl
- **Python3.6 for Mac OS X:** rknn_toolkit-1.2.1-cp36-cp36m-macosx_10_9_x86_64.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-1.2.1-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    1.2.1    afa50891bb31    1 hours ago   2.18GB
```

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:1.2.1 /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:1.2.1 /bin/bash
```

4、 运行 demo

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

3.2 RKNN-Toolkit 的使用

根据模型和设备的种类， RKNN-Toolkit 的用法可以分以下三种场景，每种场景的使用流程在接下来的章节里详细说明。

注：使用流程里涉及的所有接口，其详细说明参考第 3.4 章节。

3.2.1 场景一：模型运行在 PC 上

这种场景下，RKNN-Toolkit 运行在 PC 上，通过模拟的 RK1808 运行用户提供的模型，以实现推理或性能评估功能。

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 等模型；二是 RKNN 模型，Rockchip 的专有模型，文件后缀为“rknn”。

注：这种场景只有 x86_64 Linux 平台支持。

3.2.1.1 运行非 RKNN 模型

运行非 RKNN 模型时，RKNN-Toolkit 使用流程如下图所示：

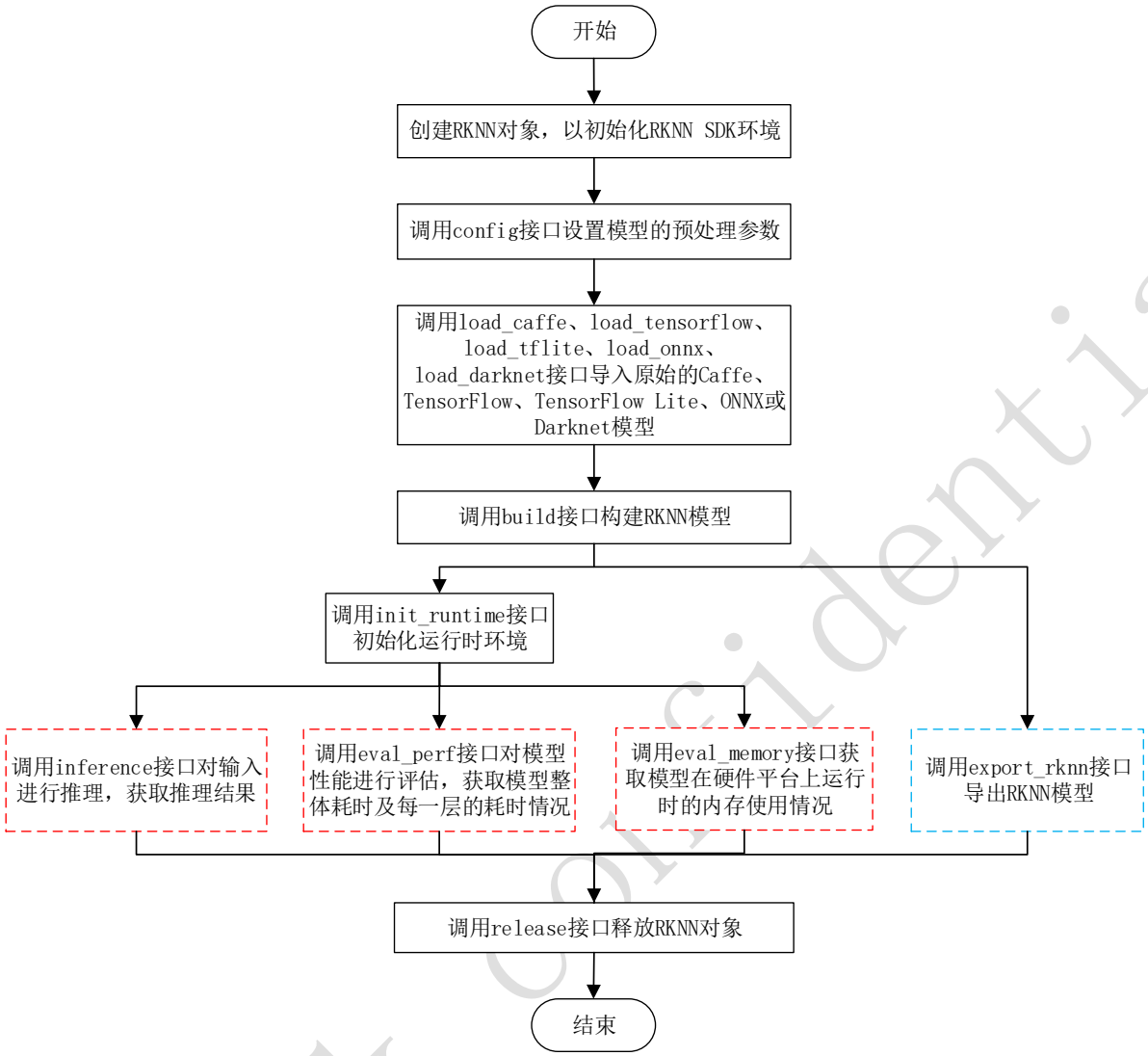

图 3-2-1-1-1 PC 上运行非 RKNN 模型时工具的使用流程

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、 只有当目标硬件平台是 RK3399Pro、RK1808、TB-RK1808 AI 计算棒或 RK3399Pro Linux 开发板时，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：

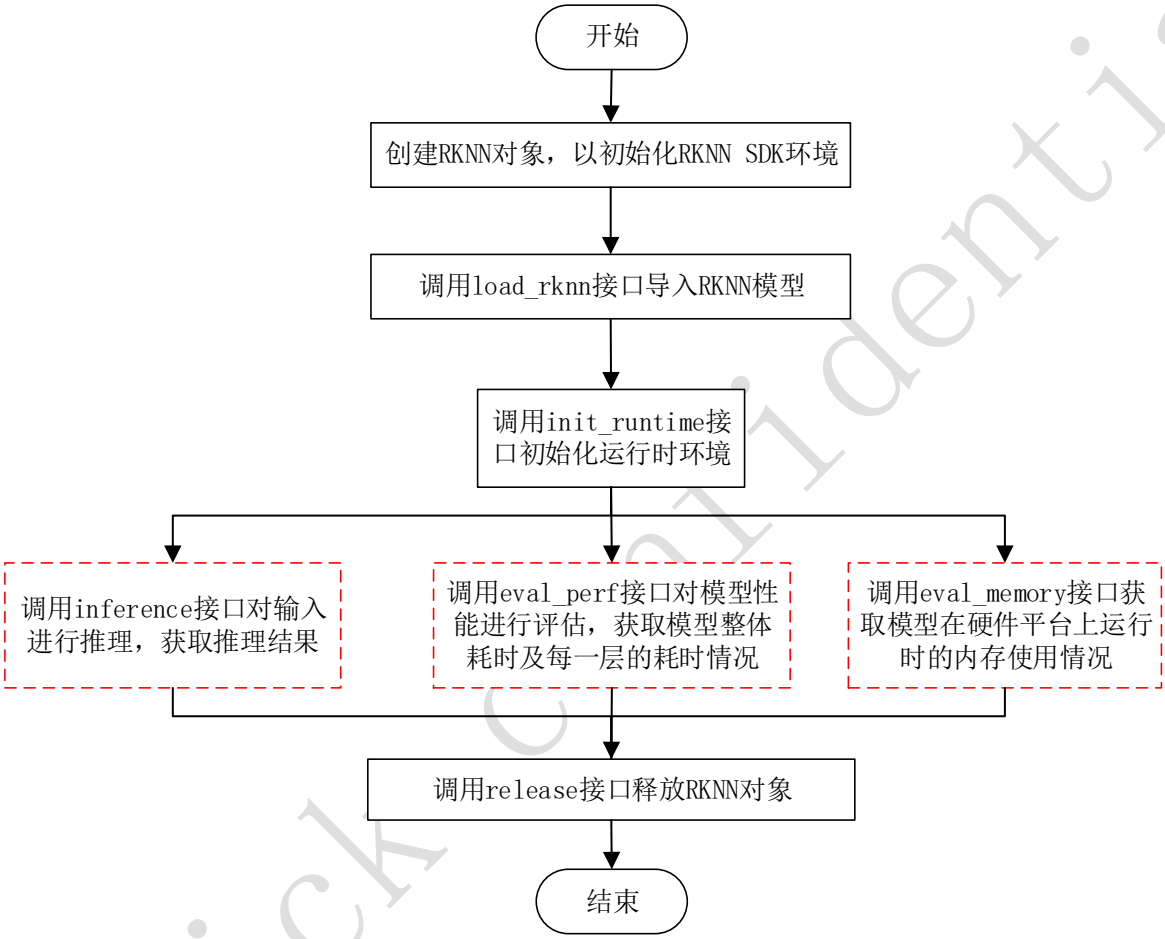

图 3-2-1-2-1 PC 上运行 RKNN 模型时工具的使用流程

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 RK3399Pro、RK1808 或 TB-RK1808 AI 计算棒上

这种场景下，RKNN-Toolkit 通过 PC 的 USB 连接到开发板硬件，将构建或导入的 RKNN 模型传到 RK3399Pro、RK1808 或 TB-RK1808 AI 计算棒上运行，并从 RK3399Pro、RK1808 或 TB-RK1808 AI 计算棒上获取推理结果、性能信息。

当模型为非 RKNN 模型（Caffe、TensorFlow、TensorFlow Lite、ONNX、Darknet 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 RKNN-Toolkit 的 list_devices 接口可查询到相应的设备，关于该接口的使用方法，参见 3.5.13 章节。

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“rk1808”或“rk3399pro”，当 PC 连接多个设备时，还需要指定 device_id 参数，即设备编号，设备编号也可通过 list_devices 接口查询，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')
```

注：目前 RK1808 等设备支持 NTB 或 ADB 两种连接模式，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。

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 V1.0.0 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. example 目录下提供了一个混合量化的例子 ssd_mobilenet_v2，可以参考该例子进行混合量化的实践。
2. 混合量化目前暂不支持多输入模型。

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。

注：**每次混合量化，只能使用其中的一种用法**。

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
---
# hybrid_quantization_action can be delete, add or modify, only one of these can be set at a hybrid quantization
hybrid_quantization_action: delete
'@attach_concat_1/out0_0:out0':
    dtype: asymmetric_quantized
    method: layer
    max_value:
    -   10.568130493164062
    min_value:
    -   -53.3099365234375
    zero_point:
    -   213
    scale:
    -   0.25050222873687744
    qtype: u8

......

'@FeatureExtractor/MobilenetV2/Conv/Conv2D_230:bias':
    dtype: None
```

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混合量化时的操作。用户在使用混合量化功能时，需要指明是以哪一种用法来使用混合量化，即前一小节提到的三种用法，对应的 action 分别是：“delete”、“add”和

“modify”，默认值是“delete”。

接着是一个模型层列表，每一层都是一个字典。每个字典的 key 由@{layer_name}_{layer_id}:[weight/bias/out{port}]组成，其中 layer_name 是层名，layer_id 是层 id；量化时我们通常是对 weight/bias/out 进行量化，多个 output 时用 out0、out1 等进行区分。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：

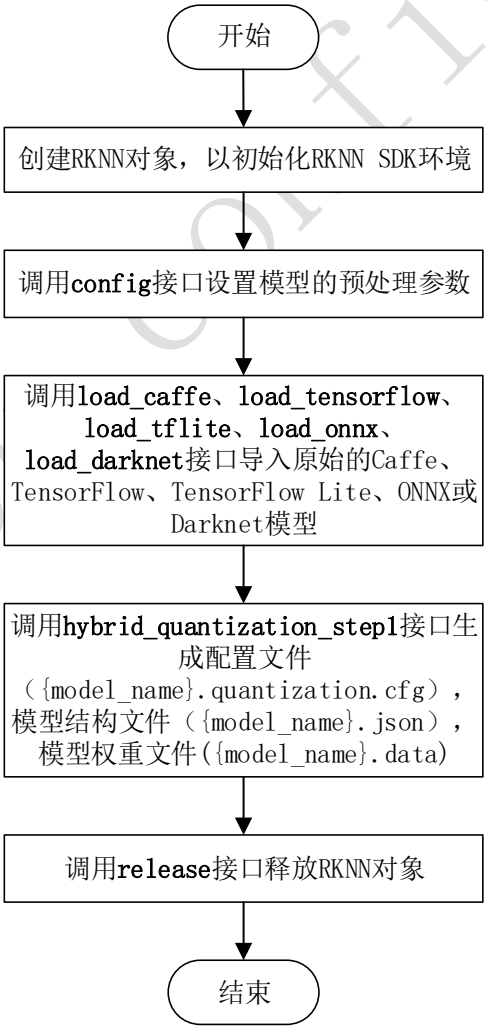

图 3-3-3-1 混合量化第一步接口调用流程

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，将它的 input 节点的 out 项、本层的 weight/bias 项从量化配置文件中删除。
- 如果是将某些层从非量化改成量化，则将量化配置文件中 hybrid_quantization_action 项的值改成”add”，然后在量化配置文件中找到该层，将它的 dtype 从 None 改成 asymmetric_quantized 或 dynamic_fixed_point 即可。注：dtype 需要和其他量化层保持一致。
- 如果是要修改量化参数，则将量化配置文件中 hybrid_quantization_action 项的值改成”modify”，然后直接修改指定层的量化参数即可。

第三步，生成 RKNN 模型。具体的接口调用流程如下：

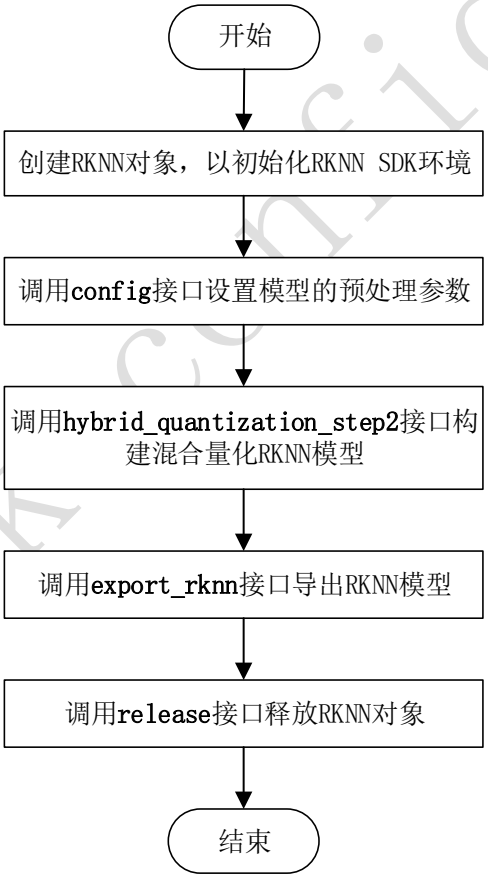

图 3-3-3-2 混合量化第三步接口调用流程

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 V1.2.0 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 3.7.13 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 示例

以下是加载 TensorFlow Lite 模型的示例代码（详细参见 example/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/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]: 0.8837890625
[155]: 0.0677490234375
[188 205]: 0.00867462158203125
[188 205]: 0.00867462158203125
[263]: 0.0057525634765625
```

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          152
62          convolution.relu.pooling.layer2_2          120
48          convolution.relu.pooling.layer2_2          116
63          convolution.relu.pooling.layer2_2          101
49          convolution.relu.pooling.layer2_2          185
64          convolution.relu.pooling.layer2_2          101
50          convolution.relu.pooling.layer2_2          111
65          convolution.relu.pooling.layer2_2          109
51          convolution.relu.pooling.layer2_2          213
66          convolution.relu.pooling.layer2_2          109
52          convolution.relu.pooling.layer2_2          213
67          convolution.relu.pooling.layer2_2          109
53          convolution.relu.pooling.layer2_2          213
68          convolution.relu.pooling.layer2_2          109
54          convolution.relu.pooling.layer2_2          213
69          convolution.relu.pooling.layer2_2          109
55          convolution.relu.pooling.layer2_2          213
70          convolution.relu.pooling.layer2_2          109
56          convolution.relu.pooling.layer2_2          174
71          convolution.relu.pooling.layer2_2          219
57          convolution.relu.pooling.layer2_2          353
58          fullyconnected.relu.layer_3                110
Total Time(us): 4772
FPS(800MHz): 209.56
============================================================
```

3.6 API 详细说明

3.6.1 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()
```

3.6.2 模型加载

RKNN-Toolkit 目前支持 Caffe、TensorFlow、TensorFlow Lite、ONNX、Darknet 五种非 RKNN 模型，它们在加载时调用的接口不同，下面详细说明这五种模型的加载接口。

3.6.2.1 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。
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')
```

3.6.2.2 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。
	mean_values：输入的均值。只有当导入的模型是已量化过的模型时才需要设置该参数，且模型输入的三个通道均值都相同。
	std_values：输入的 scale 值。只有当导入的模型是已量化过的模型时才需要设置该参数。
返回值	0：导入成功
	-1：导入失败

举例如下：

```
# 从当前目录加载 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]])
```

3.6.2.3 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')
```

3.6.2.4 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')
```

3.6.2.5 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')
```

3.6.3 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。

	channel_mean_value：包括四个值(M0 M1 M2 S0)，前三个值为均值参数，后面一个值为 Scale 参数。对于输入数据是三通道的(Cin0, Cin1, Cin2)数据来讲，经过预处理后，输出的数据为(Cout0,Cout1, Cout2)，计算过程如下:
	`Cout0 = (Cin0 - M0)/S0`
	`Cout1 = (Cin1 - M1)/S0`
	`Cout2 = (Cin2 - M2)/S0`
	例如，如果需要将输入数据归一化到[-1, 1]之间，则可以设置这个参数为(128 128 128 128)；如果需要将输入数据归一化到[0, 1]之间,则可以设置这个参数为 (0 0 0 255)。

	如果有多个输入，对应的每个输入的参数以“#”进行分割，如 ’128 128 128 128#128 128 128 128’。
	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。
返回值	无

举例如下：

```
# model config
rknn.config(channel_mean_value='103.94 116.78 123.68 58.82',
            reorder_channel='0 1 2',
            need_horizontal_merge=True)
```

3.6.4 构建 RKNN 模型

API	**build**
描述	根据导入的 Caffe、TensorFlow、TensorFlow Lite 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')
```

3.6.5 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')
.....
```

3.6.6 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')
```

3.6.7 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（RK3399Pro、RK3399Pro Linux、RK1808、TB-RK1808 AI 计算棒）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”、“rk1808”。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

	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)
```

3.6.8 使用模型对输入进行推理

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	使用模型对指定的输入进行推理，得到推理结果。 如果 RKNN-Toolkit 运行在 PC 上，且初始化运行环境时设置 target 为“rk3399pro”或“rk1808”，得到的是模型在硬件平台上的推理结果。其中“rk1808”包含了 TB-RK1808

	AI 计算棒。 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。 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。
	outputs： 指定输出数据的格式，格式是 ndarray list，用于指定输出的 shape 和 dtype。默认值为 None，此时返回的数据 dtype 为 float32。
	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。 **注：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，output 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。**

举例如下：

对于分类模型，如 mobilenet_v1，代码如下（完整代码参考 example/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]: 0.8837890625
[155]: 0.0677490234375
[188 205]: 0.00867462158203125
[188 205]: 0.00867462158203125
[263]: 0.0057525634765625
```

对于目标检测的模型，如 mobilenet_ssd，代码如下(完整代码参考 example/mobilent-ssd)：

```
# 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）：

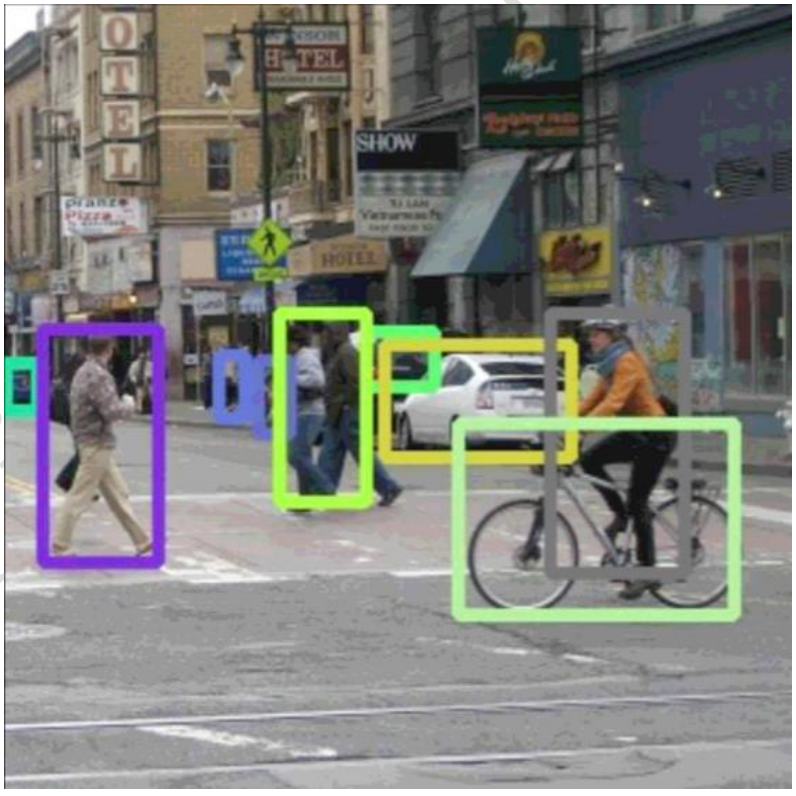

图 3-4-8-1 mobilenet-ssd inference 结果

3.6.9 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。 模型运行在与 PC 连接的 RK3399Pro 或 RK1808 或 TB-RK1808 AI 计算棒上，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
参数	inputs：输入数据，如经过 cv2 处理得图片。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'。
	is_print：是否以规范格式打印性能评估结果。默认值为 True。

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/mobilenet-ssd，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   325
105         convolution.relu.pooling.layer2_2   331
72          convolution.relu.pooling.layer2_2   437
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   201
109         convolution.relu.pooling.layer2_2   250
76          convolution.relu.pooling.layer2_2   320
110         convolution.relu.pooling.layer2_2   250
77          convolution.relu.pooling.layer2_2   165
111         convolution.relu.pooling.layer2_2   257
78          convolution.relu.pooling.layer2_2   319
112         convolution.relu.pooling.layer2_2   257
79          convolution.relu.pooling.layer2_2   319
113         convolution.relu.pooling.layer2_2   257
80          convolution.relu.pooling.layer2_2   319
```

```
114             convolution.relu.pooling.layer2_2       257
81              convolution.relu.pooling.layer2_2       319
115             convolution.relu.pooling.layer2_2       257
82              convolution.relu.pooling.layer2_2       319
83              convolution.relu.pooling.layer2_2       181
27              tensor.transpose_3                      48
84              convolution.relu.pooling.layer2_2       45
28              tensor.transpose_3                      6
116             convolution.relu.pooling.layer2_3       297
85              convolution.relu.pooling.layer2_2       233
117             convolution.relu.pooling.layer2_2       311
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       125
90              convolution.relu.pooling.layer2_3       588
91              convolution.relu.pooling.layer2_2       96
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       154
95              convolution.relu.pooling.layer2_2       50
47              tensor.transpose_3                      6
96              convD_2                                 6
48              tensor.transpose_3                      4
97              convolution.relu.pooling.layer2_2       17
98              convolution.relu.pooling.layer2_3       153
99              convolution.relu.pooling.layer2_2       49
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): 10462
FPS(800MHz): 95.58
================================================
```

3.6.10 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 RK3399Pro、RK1808、TB-RK1808 AI 计算棒上，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 中 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 : 41.53 MiB
    total allocation   : 43.86 MiB
NPU memory:
    maximum allocation : 34.53 MiB
    total allocation   : 34.54 MiB

Total memory:
    maximum allocation : 76.06 MiB
    total allocation   : 78.40 MiB

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

3.6.11 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。且该接口只能在硬件平台 RK3399Pro、RK1808、TB-RK1808 AI 计算棒上使用。**
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:
  API: 1.2.0 (1190a71 build: 2019-09-25 12:39:14)
  DRV: 1.2.0 (6897f97 build: 2019-09-25 10:17:41)
=============================================
```

3.6.12 混合量化

3.6.12.1 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')
.....
```

3.6.12.2 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，可以加快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')
……
```

3.6.13 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。获取时需要 PC 连接到 RK3399Pro/RK1808/RK1808 AI 计算棒上获取，或者在 RK3399Pro Linux 开发板上获取。** 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()
```

3.6.14 获取设备列表

API	**list_devices**
描述	列出已连接的 RK3399PRO/RK1808 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 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。

3.6.15 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(...)
```