密级状态：绝密(　　)　　秘密(　　)　　内部(　　)　　公开(√)

RKNN-Toolkit2 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：	V0.7.0
	作　　者：	HPC
	完成日期：	2021-3-30
	审　　核：	熊伟
	完成日期：	2021-3-30

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
V0.5.0	HPC	2020-12-18	初始版本	熊伟
V0.6.0	HPC	2021-2-24	1. 更新混合量化章节 2. 更新 RKNN 对象初始化参数 3. 更新 Caffe 加载 API	熊伟
V0.7.0	HPC	2021-3-30	1. 更新 mmse 量化支持 2. 更新部分接口说明 3. 更新量化算法的说明与建议	熊伟

目 录

1 概述

1.1 主要功能说明

RKNN-Toolkit2 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 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。
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-8、asymmetric_quantized-16），并支持混合量化功能。asymmetric_quantized-16 和混合量化目前暂不支持。
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：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。 目前暂不支持。
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。目前暂不支持。
6) 量化精度分析：该功能将给出模型量化前后每一层推理结果的余弦距离，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。

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 18.04	Windows 7/10	Debian 9/10 (aarch64)	MacOS Mojave / Catalina
模型转换	支持			

量化/混合量化	支持			
模型推理	支持			
性能评估				
内存评估				
多输入	支持			
批量推理				
设备查询				
SDK 版本查询				
量化精度分析	支持			
可视化功能				
模型优化开关	支持			

1.2 适用芯片

RKNN-Toolkit2 支持芯片的型号如下：

- RK3566
- RK3568

1.3 适用系统

RKNN-Toolkit2 是一个跨平台的开发套件，已支持的操作系统如下：

- Ubuntu: 18.04（x64）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	Ubuntu18.04（x64）及以上
Python 版本	3.6
Python 库依赖	numpy==1.16.6 onnx==1.7.0 onnxoptimizer==0.1.0 onnxruntime==1.7.0 tensorflow==1.14.0 tensorboard==1.14.0 protobuf==3.12.0 torch==1.6.0 torchvision==0.7.0 mxnet==1.7.0 psutil==5.6.2 ruamel.yaml==0.15.81 scipy==1.2.1 tqdm==4.27.0 requests==2.21.0 tflite==2.3.0 opencv-python==4.4.0.46 PuLP==2.4

注：

1. 本文档主要以 Ubuntu 18.04 / Python3.6 为例进行说明。其他操作系统请参考《Rockchip_Quick_Start_RKNN_Toolkit2_CN.pdf》。

3 使用说明

3.1 安装

目前提供两种方式安装 RKNN-Toolkit2：一是通过 Python 包安装与管理工具 pip 进行安装；二是运行带完整 RKNN-Toolkit2 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。

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 python3 python3-dev python3-pip
sudo apt-get install libxslt1-dev zlib1g zlib1g-dev libglib2.0-0 \
libsm6 libgl1-mesa-glx libprotobuf-dev gcc

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

2. 安装依赖库：

```
pip3 install -r doc/requirements.txt
```

注：RKNN-Toolkit2 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-Toolkit2

```
pip install package/rknn_toolkit2*.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）：

- **Python3.6 for x86_64**：rknn_toolkit2-0.7.0-cp36-cp36m-linux_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-Toolkit2，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-toolkit2-0.7.0-docker.tar.gz
```

加载成功后，执行“docker images”命令能够看到 rknn-toolkit2 的镜像，如下所示：

```
REPOSITORY          TAG         IMAGE ID            CREATED             SIZE
rknn-toolkit2       0.7.0       4f6bae6686d8        1 hours ago         4.13GB
```

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-toolkit2:0.7.0 /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-toolkit2:0.7.0 /bin/bash
```

4、 运行 demo

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

3.2 RKNN-Toolkit2 的使用

接下来将详细给出各使用场景下 RKNN Toolkit2 的使用流程。

3.2.1 场景一：模型运行在模拟器上

这种场景下，RKNN Toolkit2 运行在 PC 上，通过模拟器运行模型，以实现相应功能。

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

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 Toolkit2 的完整使用流程如下图所示：

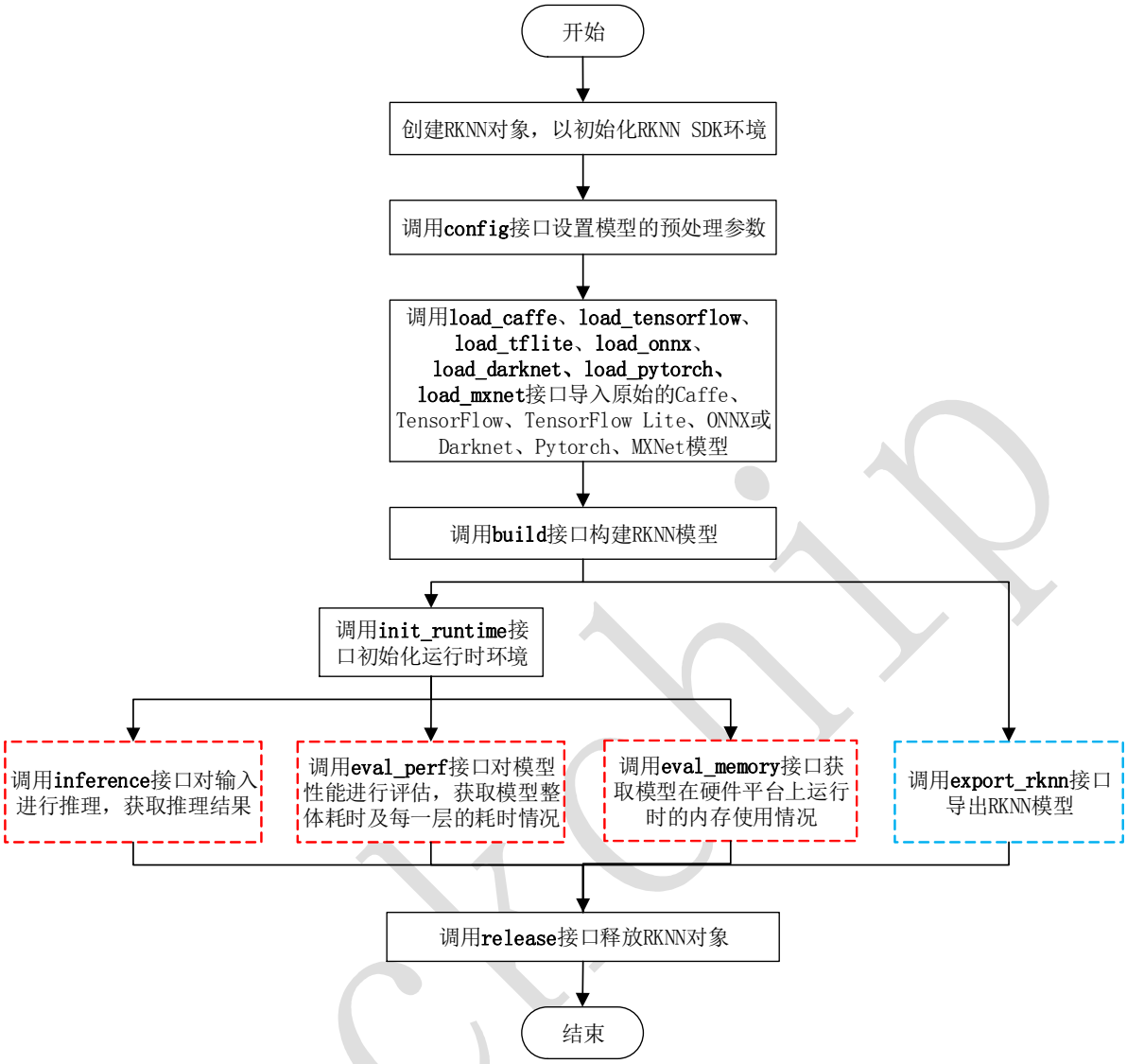

图 3-2-1-1-1 PC 上运行非 RKNN 模型时工具的使用流程

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。性能评估和内存评估目前暂不支持。

4、 只有当目标平台是 Rockchip NPU 时，我们才可以调用 inference / eval_perf / eval_memory 接口获取内存使用情况。目前暂不支持。

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

目前该功能暂不支持。

RKNN Toolkit2 目前支持的 Rockchip NPU 平台包括 RK3566, RK3568。

这种场景下，RKNN Toolkit2 运行在 PC 上，通过 PC 的 USB 连接 NPU 设备。RKNN Toolkit2 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。

首先，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-Toolkit2 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 3.5.15 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“rk3566”、“rk3568”。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：

```
# RK3566
ret = init_runtime(target='rk3566', device_id='VGEJY9PW7T')

# RK3568
ret = init_runtime(target='rk3568', device_id='515e9b401c060c0b')
```

3.2.2.1 运行非 RKNN 模型

当模型为非 RKNN 模型（Caffe、TensorFlow、TensorFlow Lite、ONNX、Darknet、Pytorch 等模型）时，RKNN-Toolkit2 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）。

3.2.2.2 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：

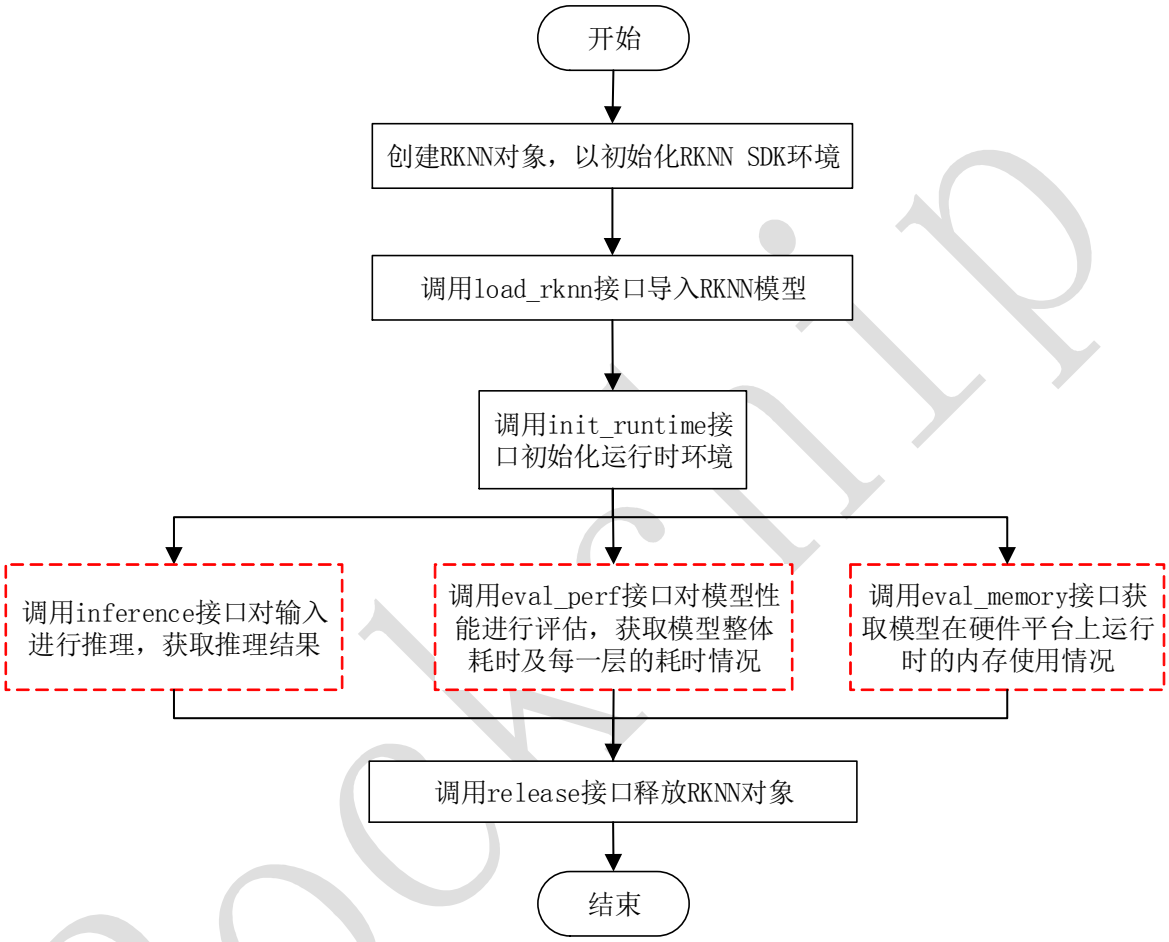

图 3-2-1-2-1 PC 上运行 RKNN 模型时工具的使用流程

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 inference / eval_perf / 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。

4、 通过 load_rknn 导入的方式仅用于硬件平台相关功能的使用，无法使用如精度分析 accuracy_analysis 等功能。

3.2.3 场景三：模型运行在 RK356x Linux 开发板上

目前该功能暂不支持。

这种场景下，RKNN-Toolkit2 直接安装在 RK356x Linux 系统里。构建或导入的 RKNN 模型直接在 RK356x 上运行，以获取模型实际的推理结果或性能信息。

对于 RK356x 开发板，RKNN-Toolkit2 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 3.2.2.2 章节）。

3.3 混合量化

目前该功能暂不支持。

RKNN-Toolkit2 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-Toolkit2 引入了混合量化功能，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用法：

1. 将指定的量化层改成非量化层（如用 float16 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。

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 配置文件。配置文件格式如下：

```
custom_quantize_layers: {}
quantize_parameters:
    FeatureExtractor/MobilenetV2/Conv/BatchNorm/batchnorm/add_1:0:
        qtype: asymmetric_quantized
        qmethod: layer
        dtype: int8
        min:
        -   0.0
        max:
        -   6.0
        scale:
        -   0.023529411764705882
        zero_point:
        -   -128
        ori_min:
        -   -13.971162796020508
        ori_max:
        -   22.79466438293457
......
```

配置文件正文第一行是一个自定义量化操作数的字典，将操作数名和相应的量化类型（可选值为 float16 / int16）添加到这儿。

接着是模型中每个操作数的量化参数，每一个操作数都是一个字典。每个字典的 key 即 tensor_name，字典的 value 即量化参数，如果没有经过量化，则 dtype 值为 float16。

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：

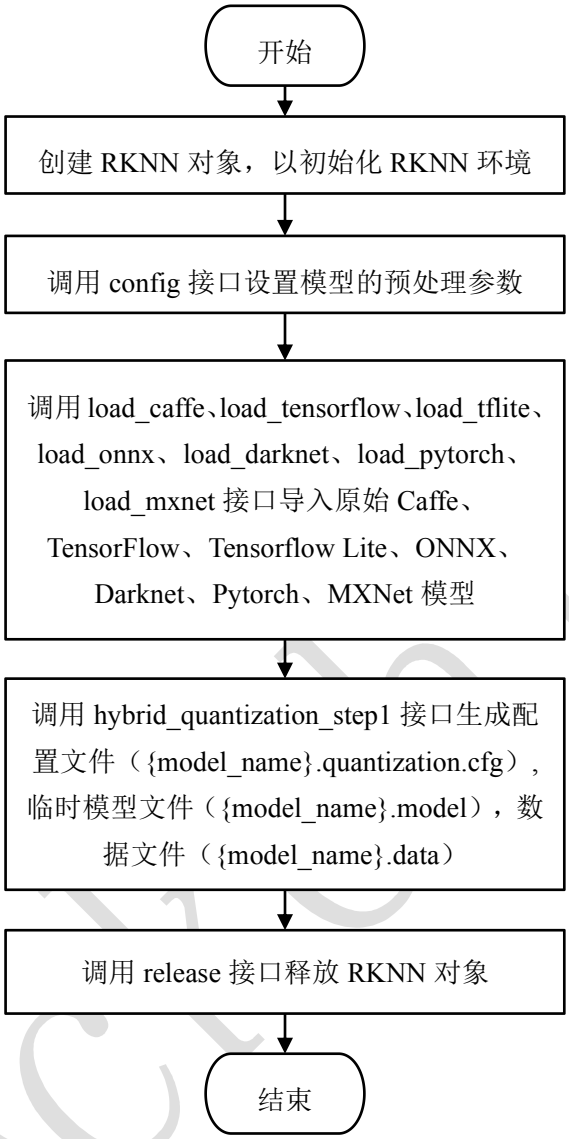

图 3-3-3-1 混合量化第一步接口调用流程

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的输出操作数（如果该层的输出操作数有多个，设置第一个或任意一个即可），将这些操作数加到 custom_quantize_layers 字典中，值为 float16。

第三步，生成 RKNN 模型。具体的接口调用流程如下：

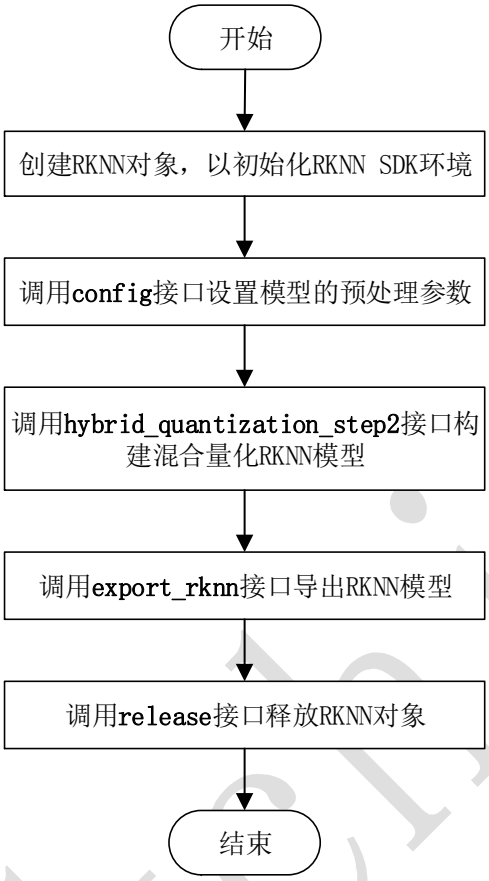

图 3-3-3-2 混合量化第三步接口调用流程

第四步，使用上一步生成的 RKNN 模型进行推理。

3.4 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)


    if __name__ == '__main__':

            # Create RKNN object
            rknn = RKNN()

            # pre-process config
            print('--> config model')
            rknn.config(mean_values=[128, 128, 128], std_values=[128, 128, 128],
reorder_channel=False)
            print('done')

            # Load tensorflow model
            print('--> Loading model')
            ret = rknn.load_tflite(model='mobilenet_v1_1.0_224.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)
            img = np.expand_dims(img, 0)
```

```
    # 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')

    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]: 0.8544921875
[155]: 0.080322265625
[205]: 0.0129241943359375
[284]: 0.0084075927734375
[194]: 0.0025787353515625
```

3.5 API 详细说明

3.5.1 RKNN 初始化及对象释放

在使用 RKNN Toolkit2 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。模型转换过程中还支持 verbose 参数值为”INFO”，会将模型解析过程中的所有日志输出到屏幕，帮助排查哪一层解析失败，方便问题定位。

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()
```

3.5.2 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。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 True，则优先做通道调整，再做减均值。
	std_values：输入的归一化值。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 True，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。目前暂不支持。
	reorder_channel：表示是否需要对图像通道顺序进行调整。False 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；True 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，则用列表包含起来，如[True, True, False]。
	quantized_dtype：量化类型，目前支持的量化类型有 asymmetric_quantized-8、asymmetric_quantized-16，默认值为 asymmetric_quantized-8。asymmetric_quantized-16 暂不支持。
	quantized_algorithm:计算每一层的量化参数时采用的量化算法，目前支持的量化算法有：normal，mmse，默认值为 normal。关于量化算法的基本介绍与使用，需要注意量化算法本身与使用的量化图片的类型数量以及模型本身息息相关，所以文档给出的建议只是参考，并非一定最优，需要用户做适当权衡。Normal 量化算法的特点是速度较快，一般建议选择与预测场景较吻合的量化图片，推荐量化数据量一般为 20-50 张左右，更多的数据量未必有很好的量化精度提升。Mmse 量化算法由于采用暴力迭代的算法，所以速度会慢很多，但通常会比 normal 具有更高的量化精度，一般也建议选择与预测场景较吻合的量化图片，推荐量化数据量一般为 20-50 张左右，用户也可以根据时间长短对数据量进行适当缩减也是可行的。
	mmse_epoch: mmse 量化算法的 epoch，默认值为 3。Mmse 量化算法的迭代次数越多，往往会获得更高的量化精度。
	quantized_method：目前支持 layer 或者 channel，即每层只有一套量化参数或者每个通道都有自己的一套量化参数，通常情况下 channel 会比 layer 精度更高，默认值为 layer。channel 暂不支持。
	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

	RK3566、RK3568。该参数的值大小写不敏感。
	outputs_quant: 为每个 outputs 设置是否输出量化的数据来代替浮点输出，在类似超分应用时可以避免性能损失。默认为 False。 （暂不支持）
	outputs_skip_transpose: 为每个 outputs 设置是否跳过最后一个 OP（如 Conv）的 transpose 操作，跳过后输出的 layout 会与原始模型不一致，但速度会更快。默认为 False。 （暂不支持）
	custom_string: 添加自定义字符串信息到 rknn 模型，可以在 runtime 时通过 query 查询到该信息。
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=True,
            target_platform='rk3566')
```

3.5.3 模型加载

RKNN-Toolkit2 目前支持 Caffe、TensorFlow、TensorFlow Lite、ONNX、Darknet、Pytorch 七种非 RKNN 模型，它们在加载时调用的接口不同，下面详细说明这七种模型的加载接口。

3.5.3.1 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'。'lstm_caffe'暂不支持。
	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 Toolkit2 将随机生成权重等参数。
	inputname: caffe 模型存在多输入时，可以通过该参数指定输入层名的顺序，形如['input1','input2','input3']，注意名字需要与模型输入名一致；也可不设定，按 caffe 模型文件（.prototxt 后缀文件）自动给定。
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',
                      inputname=['input1'])
```

3.5.3.2 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[[1, 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。该接口暂不支持。
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=[[1, 300, 300, 3]])
```

3.5.3.3 TensorFlow Lite 模型加载接口

API	**load_tflite**
描述	加载 TensorFlow Lite 模型。
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')
```

3.5.3.4 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')
```

3.5.3.5 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')
```

3.5.3.6 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[[1,1,224,224],[1,3,224,224]]表示有两个输入，其中一个输入的 shape 是[1,1,224,224]，另外一个输入的 shape 是[1,3,224,224]。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 resnet18 模型
ret = rknn.load_pytorch(model = './resnet18.pt',
                        input_size_list=[[1,3,224,224]])
```

3.5.3.7 MXNet 模型加载接口 (暂不支持)

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[[1,1,224,224],[1,3,224,224]]表示有两个输入，其中一个输入的 shape 是[1,1,224,224]，另外一个输入的 shape 是[1,3,224,224]。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=[[1,3,224,224]] )
```

3.5.4 构建 RKNN 模型

API	**build**
描述	根据导入的模型，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
	rknn_batch_size：暂不支持 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')
```

3.5.5 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在硬件上进行推理，可直接通过 load_rknn 接口加载模型，或者直接部署到硬件平台上。

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')
……
```

3.5.6 加载 RKNN 模型

API	**load_rknn**
描述	加载 RKNN 模型。加载后的模型仅限于连接 NPU 硬件进行推理或获取性能数据等。不能用于模拟器或精度分析等。
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-Toolkit2 运行在连有 NPU 设备的 PC 上时才可以设为 True。默认值为 False。目前暂不支持。

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')
```

3.5.7 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，支持“rk3566”、“rk3568”。默认为 None，即在 PC 使用工具时，模型在模拟器上运行。
	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。
	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。目前该参数暂不支持。
	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。目前该参数暂不支持。
	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。目前该参数暂不支持。

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='rk3566', device_id='012345789AB')
if ret != 0:
    print('Init runtime environment failed')
    exit(ret)
```

3.5.8 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-Toolkit2 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-Toolkit2 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。
参数	inputs：待推理的输入，如经过 cv2 处理的图片。格式是 ndarray list。
	data_format：数据模式，可以填以下值: “nchw”, “nhwc”。默认值为'nhwc'。
	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，不透传 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）：

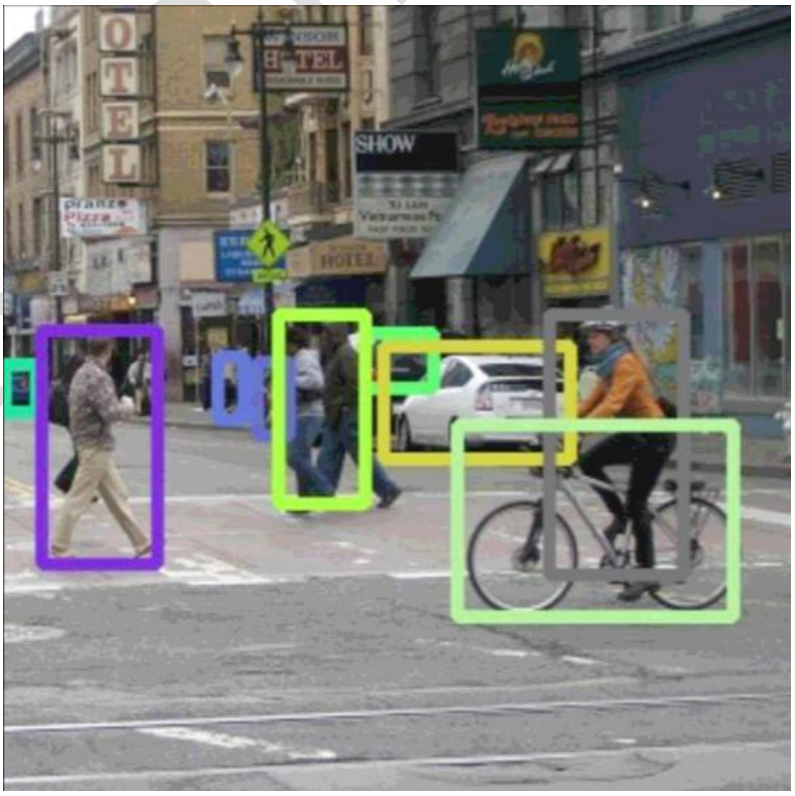

图 3-4-8-1 ssd_mobilenet_v1 inference 结果

3.5.9 评估模型性能

暂不支持。

3.5.10 获取内存使用情况

暂不支持。

3.5.11 查询 SDK 版本

暂不支持。

3.5.12 混合量化

暂不支持。

3.5.12.1 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，用于生成临时模型文件（{model_name}.model）、数据文件（{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，生成对应的临时模型文件、配置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 proposal: 产生混合量化的配置建议值。
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')
……
```

3.5.12.2 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	接收临时模型文件、配置文件、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: 第一步生成的临时模型文件，形如“{model_name}.model”。数据类型为字符串。必填参数。
	data_input: 第一步生成的配置文件，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。
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.model',
        data_input='./ssd_mobilenet_v2.data',
        model_quantization_cfg='./ssd_mobilenet_v2.quantization.cfg',
        )
……
```

3.5.13 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	推理并产生快照，也就是 dump 出每一层的 tensor 数据。会 dump 出包括 fp32 和 qnt 两种数据类型的快照，用于计算量化误差。 **注：** 1. **该接口只能在 build 或 hybrid_quantization_step2 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：图像路径 list 或者 numpy.ndarray list。
	output_dir：输出目录，所有快照都保存在该目录。 如果没有设置 target，输出的目录结构如下： ├── entire_qnt ├── fp32 ├── order.txt .. ├── 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）； ● fp32 目录：保存整个浮点模型完整跑下来时每一层的结果; ● order.txt：记录 dump 出的每一层的 tensor 数据顺序； ● error_analysis.txt：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(entire_error cosine)，以及量化模型取上一层的浮点结果作为输入时，输出与浮点模型的余弦距离(per_layer_error cosine)。
	calc_qnt_error：是否计算量化误差（默认为 True）。
返回值	0：成功
	-1：失败

举例如下：

```
……

    # Create RKNN object
    rknn = RKNN(verbose=True)

    print('--> config model')
    rknn.config(mean_values=[128, 128, 128], std_values=[128, 128, 128], )
    print('done')

    # Load model
    print('--> Loading model')
    ret = rknn.load_tensorflow(tf_pb='mobilenet_v1.pb',
                               inputs=['input'],
                               outputs=['MobilenetV1/Logits/SpatialSqueeze'],
                               input_size_list=[[1, 224, 224, 3]])
    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')

    print('--> Accuracy analysis')
    rknn.accuracy_analysis(inputs=['./dog_224x224.jpg'], output_dir=None)

……
```

3.5.14 注册自定义算子

暂不支持。

3.5.15 查询模型可运行平台

暂不支持。