密级状态：绝密(　　)　　秘密(　　)　　内部(　)　　公开(√)

RKNN-Toolkit 使用指南

（技术部，图形显示平台中心）

文件状态： [　] 正在修改 [√] 正式发布	当前版本：	V0.9.8
	作　　者：	饶洪
	完成日期：	2019-01-30
	审　　核：	卓鸿添
	完成日期：	2019-01-30

福州瑞芯微电子股份有限公司

Fuzhou　Rockchips　Semiconductor　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

版本	修改人	修改日期	修改说明	核定人
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

目 录

1 主要功能说明

RKNN-Toolkit 是为用户提供在 PC、RK3399Pro 或 RK1808 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。

2）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）、RK1808 上运行模型并获取推理结果。

3）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 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。

2 系统依赖说明

本开发套件支持运行于 Ubuntu 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）以上
Python 版本	3.5/3.6
Python 库依赖	'numpy >= 1.14.3' '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 >= 1.3.0' 'flask >= 1.0.2' 'tensorflow >= 1.11.0' 'dill==0.2.8.2' 'opencv-python>=3.4.3.18'

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。

3.1.1 通过 pip install 命令安装

由于 TensorFlow 有 CPU 和 GPU 两个版本，所以当前提供 requirements-cpu.txt 和 requirements-gpu.txt 两个依赖文件，分别对应 TensorFlow 为 CPU 和 GPU 时的依赖包，安装 python 依赖包时选择一个即可。

接着执行以下命令进行安装（以 python3.5 为例）：

```
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
# install tesnorflow cpu
pip install -r package/requirements-cpu.txt
# or install tesnorflow-gpu, use command below:
# pip install -r package/requirements-gpu.txt
pip install package/rknn_toolkit-0.9.8-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-0.9.8-cp35-cp35m-linux_x86_64.whl
- **Python3.6 for x86_64**：rknn_toolkit-0.9.8-cp36-cp36m-linux_x86_64.whl
- **Python3.6 for arm_x64**: rknn_toolkit-0.9.8-cp36-cp36m-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-0.9.8-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        0.9.8               58dc618231e3        4 hours ago         1.93GB
```

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:0.9.8
/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:0.9.8 /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 RK3399Pro 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，RockChips 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：

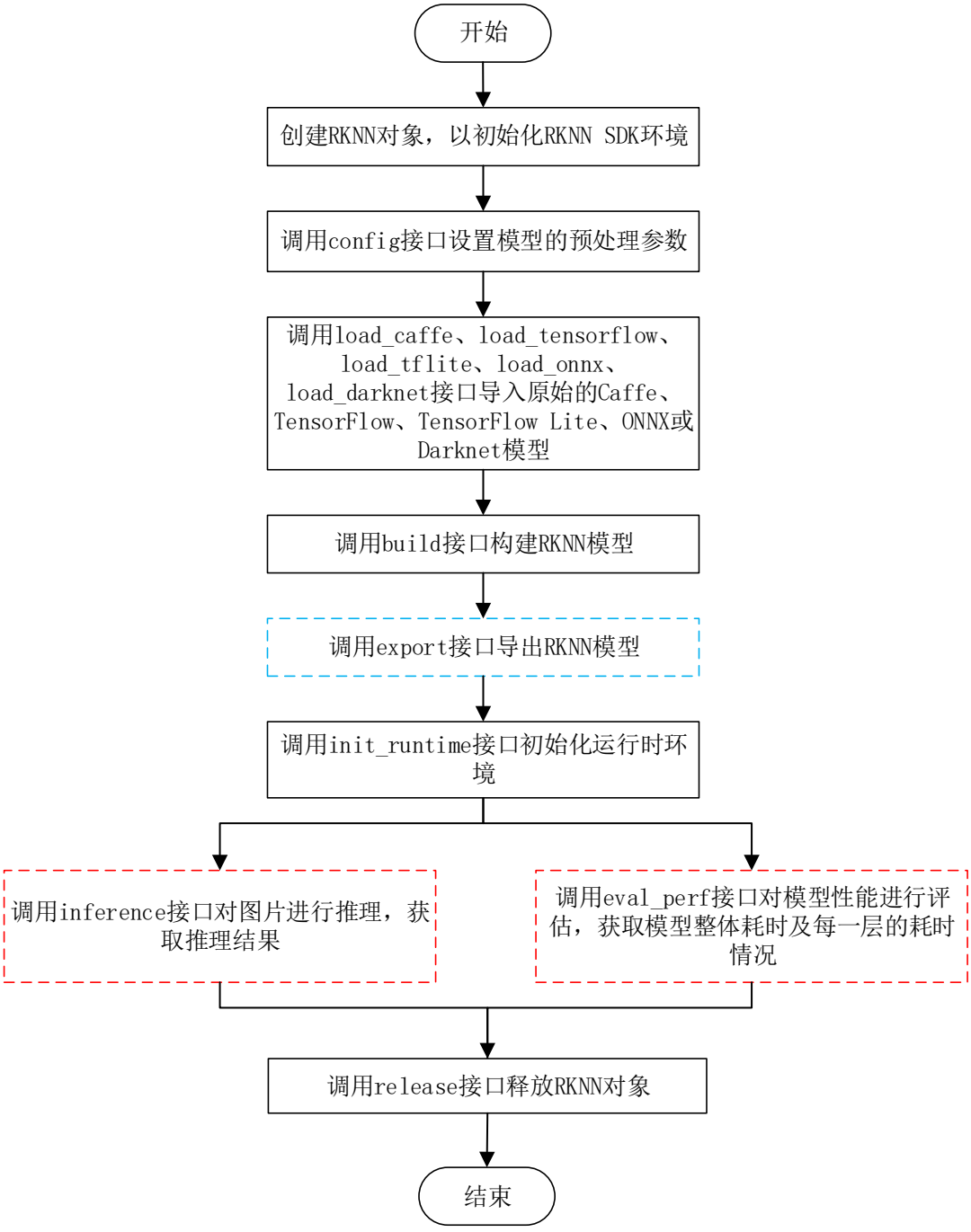

图 3-2-1-1-1 PC 上运行非 RKNN 模型时工具的使用流程

注：

1、 以上步骤请按顺序执行。

2、 蓝色框标注的模型导出步骤不是必要的，如果后续不再需要使用这个 RKNN 模型，可以不做这一步。

3、 红色框标注的模型推理和性能评估步骤先后顺序不固定，根据实际使用情况决定。

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：

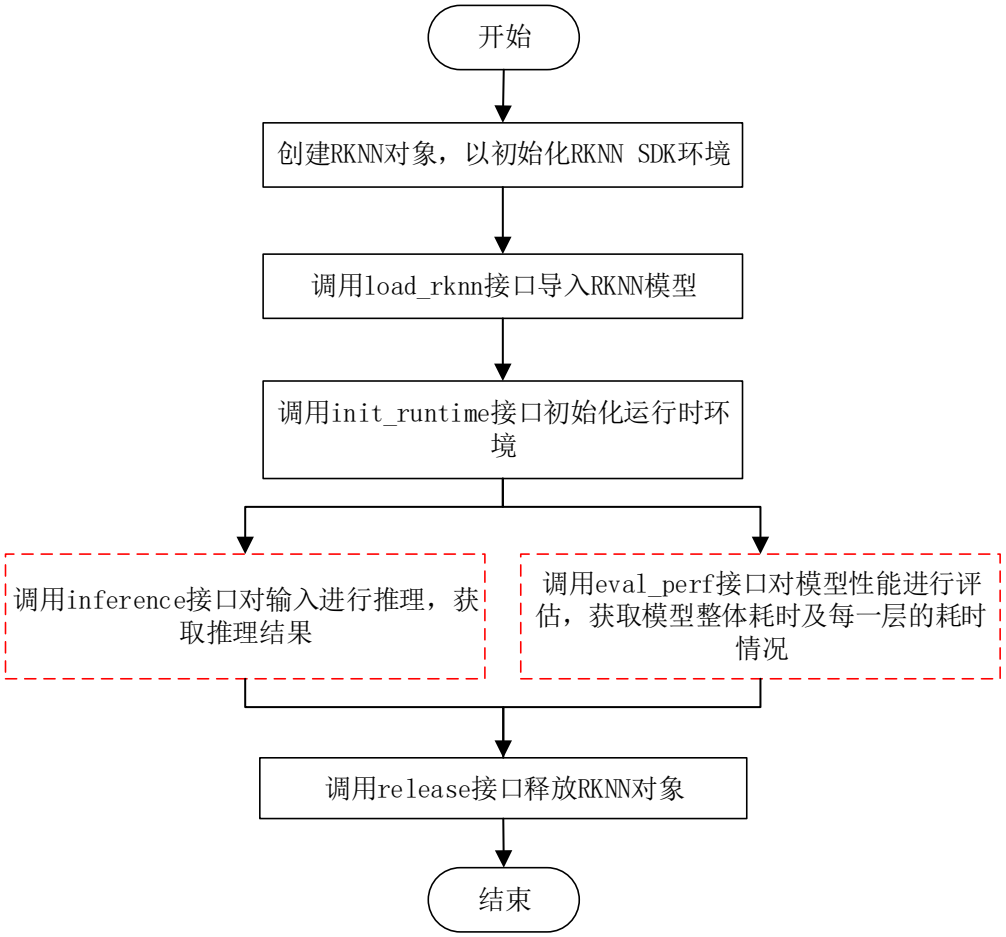

图 3-2-1-2-1 PC 上运行 RKNN 模型时工具的使用流程

注：

1、 以上步骤请按顺序执行。

2、 红色框标注的模型推理和性能评估步骤先后顺序不固定，根据实际使用情况决定。

3.2.2 场景二：模型运行在与 PC 相连的 RK3399Pro 或 RK1808 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 或 RK1808 上运行，并从 RK3399Pro 或 RK1808 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 ADB 能够正确识别到设备，即在 PC 上执行“adb devices -l”命令能看到目标设备。

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“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，可以通过“adb devics -l”命令查看，举例如下：

```
raul@raul:~$ adb devices -l
List of devices attached
0123456789ABCDEF       device usb:3-4 product:occam model:Nexus_4 device:mako transport_id:13
U00QCP85IW             device usb:3-3 product:rk3399pro model:rk3399pro device:rk3399pro transport_id:20
```

初始化运行时环境**代码示例**如下：

```
# RK3399Pro
ret = init_runtime(target='rk3399pro', device_id='U00QCP85IW')

......

# RK1808
ret = init_runtime(target='rk1808', device_id='0123456789ABCDEF)
```

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，安装流程同 PC 一致（见 3.1.1 章节）。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 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：

```
mobilenet_v1
-----TOP 5-----
[156]: 0.8388671875
[155]: 0.0472412109375
[188]: 0.0254974365234375
[205]: 0.01525115966796875
[263]: 0.0074310302734375
```

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            convolution.relu.pooling.layer2_2   362
1            convolution.relu.pooling.layer2_2   158
2            convolution.relu.pooling.layer2_2   184
3            convolution.relu.pooling.layer2_2   264
4            convolution.relu.pooling.layer2_2   94
5            convolution.relu.pooling.layer2_2   143
6            convolution.relu.pooling.layer2_2   148
7            convolution.relu.pooling.layer2_2   114
8            convolution.relu.pooling.layer2_2   89
9            convolution.relu.pooling.layer2_2   95
10           convolution.relu.pooling.layer2_2   166
11           convolution.relu.pooling.layer2_2   95
12           convolution.relu.pooling.layer2_2   101
13           convolution.relu.pooling.layer2_2   107
14           convolution.relu.pooling.layer2_2   195
15           convolution.relu.pooling.layer2_2   107
16           convolution.relu.pooling.layer2_2   195
17           convolution.relu.pooling.layer2_2   107
18           convolution.relu.pooling.layer2_2   195
19           convolution.relu.pooling.layer2_2   107
20           convolution.relu.pooling.layer2_2   195
21           convolution.relu.pooling.layer2_2   107
22           convolution.relu.pooling.layer2_2   195
23           convolution.relu.pooling.layer2_2   107
24           convolution.relu.pooling.layer2_2   163
25           convolution.relu.pooling.layer2_2   202
26           convolution.relu.pooling.layer2_2   334
27           pooling.layer2_1                    36
28           fullyconnected.relu.layer_3         110
29           tensor.transpose_3                  5
30           softmax.layer_1                     88
Total Time(us): 4568
FPS(800MHz): 218.91
============================================
```

3.4 API 详细说明

3.4.1 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()
```

3.4.2 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。

3.4.2.1 Caffe 模型加载接口

API	load_caffe
功能	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（默认值为'caffe'，可选值为'lstm_caffe'），由于某些模型可能根据 caffe 模型进行了一定的修改，需要使用特殊的格式，目前支持 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 mobilent_v2 模型
ret = rknn.load_caffe(model='./mobilenet_v2.prototxt',
                      proto='caffe',
                      blobs='./mobilenet_v2.caffemodel')
```

3.4.2.2 TensorFlow 模型加载接口

API	load_tensorflow
功能	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，目前只支持一个输入。输入节点字符串放在列表中。
	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)。
	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]])
```

3.4.2.3 TensorFlow Lite 模型加载接口

API	load_tflite
功能	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')
```

3.4.2.4 ONNX 模型加载

API	load_onnx
功能	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')
```

3.4.2.5 Darknet 模型加载接口

API	load_darknet
功能	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')
```

3.4.3 RKNN 模型配置

在模型加载之前，需要先对模型进行配置，这可以通过 config 接口完成。

API	config
功能	设置模型参数
参数	batch_size：每一批数据集的大小，默认值为 100。

	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)。

	epochs：推理或性能评估时，对同一批数据集处理的次数，默认值为 1。

	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。
	need_horizontal_merge：是否需要合并 Horizontal，默认值为 False。如果模型是 inception v1/v3/v4，建议开启该选项。
	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)
```

3.4.4 构建 RKNN 模型

API	build
功能	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。
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')
```

3.4.5 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，直接通过模型导入接口加载模型即可，无需再用原始模型构建对应的 RKNN 模型。

API	export_rknn
功能	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')
……
```

3.4.6 加载 RKNN 模型

API	load_rknn
功能	加载 RKNN 模型。

参数	path：RKNN 模型文件路径。
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')
```

3.4.7 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）运行或直接通过模拟器运行。

API	init_runtime
功能	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。
	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”***adb devices -l***”命令查看。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。
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)
```

3.4.8 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
功能	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。 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’, ‘ing16’。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 数放置的位置。
	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，次数返回的数据 dtype 为 float32。
返回值	results：推理结果，类型是 ndarray list。

举例如下：

对于推理模型，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：

```
mobilenet_v1
-----TOP 5-----
[156]: 0.8388671875
[155]: 0.0472412109375
[188]: 0.0254974365234375
[205]: 0.01525115966796875
[263]: 0.0074310302734375
```

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）：

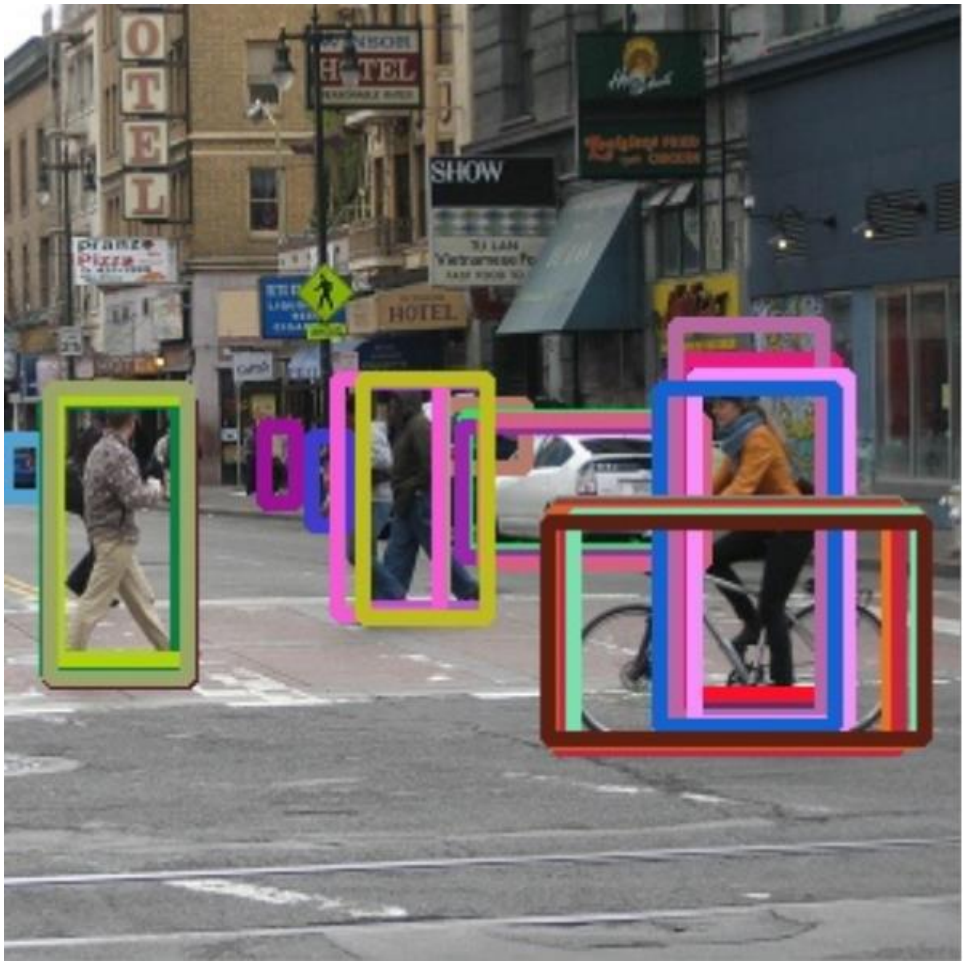

图 3-4-8-1 mobilenet-ssd inference 结果

3.4.9 评估模型性能

API	eval_perf
功能	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 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：是否以规范格式输出性能评估结果。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：

```
============================================
                Performance
============================================
Layer ID    Name                                Time(us)
0           convolution.relu.pooling.layer2_3   324
1           convolution.relu.pooling.layer2_2   338
2           convolution.relu.pooling.layer2_2   434
3           convolution.relu.pooling.layer2_2   463
4           convolution.relu.pooling.layer2_2   244
5           convolution.relu.pooling.layer2_2   330
6           convolution.relu.pooling.layer2_2   437
7           convolution.relu.pooling.layer2_3   528
8           convolution.relu.pooling.layer2_2   154
9           convolution.relu.pooling.layer2_2   241
10          convolution.relu.pooling.layer2_2   289
11          convolution.relu.pooling.layer2_2   241
12          convolution.relu.pooling.layer2_2   150
13          convolution.relu.pooling.layer2_2   255
14          convolution.relu.pooling.layer2_2   290
15          convolution.relu.pooling.layer2_2   255
16          convolution.relu.pooling.layer2_2   290
17          convolution.relu.pooling.layer2_2   255
18          convolution.relu.pooling.layer2_2   290
19          convolution.relu.pooling.layer2_2   255
20          convolution.relu.pooling.layer2_2   290
21          convolution.relu.pooling.layer2_2   255
22          convolution.relu.pooling.layer2_2   290
```

```
23          convolution.relu.pooling.layer2_2          159
24          convolution.relu.pooling.layer2_2          45
25          convolution.relu.pooling.layer2_3          290
26          tensor.transpose_3                         48
27          tensor.transpose_3                         6
28          convolution.relu.pooling.layer2_2          194
29          convolution.relu.pooling.layer2_2          305
30          convolution.relu.pooling.layer2_2          482
31          convolution.relu.pooling.layer2_2          206
32          convolution.relu.pooling.layer2_2          29
33          convolution.relu.pooling.layer2_2          100
34          tensor.transpose_3                         29
35          tensor.transpose_3                         5
36          convolution.relu.pooling.layer2_3          436
37          convolution.relu.pooling.layer2_2          89
38          convolution.relu.pooling.layer2_2          9
39          convolution.relu.pooling.layer2_2          23
40          tensor.transpose_3                         10
41          tensor.transpose_3                         5
42          convolution.relu.pooling.layer2_3          115
43          convolution.relu.pooling.layer2_2          46
44          convolution.relu.pooling.layer2_2          6
45          convolution.relu.pooling.layer2_2          13
46          tensor.transpose_3                         6
47          tensor.transpose_3                         4
48          convolution.relu.pooling.layer2_3          114
49          convolution.relu.pooling.layer2_2          46
50          convolution.relu.pooling.layer2_2          5
51          convolution.relu.pooling.layer2_2          8
52          tensor.transpose_3                         5
53          tensor.transpose_3                         4
54          convolution.relu.pooling.layer2_2          16
55          fullyconnected.relu.layer_3                13
56          fullyconnected.relu.layer_3                8
57          tensor.transpose_3                          5
58          tensor.transpose_3                          4
Total Time(us): 9786
FPS(800MHz): 102.19
================================================
```