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：	v0.9.7
	作　　者：	饶洪
	完成日期：	2018-12-29
	审　　核：	卓鸿添
	完成日期：	2018-12-29

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
v0.9.0	饶洪	2018-08-29	RKNN-Toolkit v0.9.0 版本相关内容	卓鸿添
v0.9.1-beta1	饶洪	2018-09-07	增补 v0.9.1-beta1 版本更新内容	卓鸿添
v0.9.1-rc1	饶洪	2018-09-19	增补 v0.9.1-rc1 版本更新内容	卓鸿添
v0.9.1-rc1	饶洪	2018-09-29	解决 ssd_mobilenet_fpn inference 阶段 core dump 问题	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	添加 npy 文件作为量化校正数据的使用说明；添加 vdata 编译选项；完善 reorder_channel 参数的使用说明	卓鸿添
V0.9.6	饶洪	2018-11-24	新增接口 get_run_duration 和 get_perf_detail_on_hardware 使用说明	卓鸿添
V0.9.7	饶洪	2018-12-29	接口优化	卓鸿添

目 录

1 主要功能说明

RKNN-Toolkit 是为用户提供在 PC 上进行模型转换、模拟运行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，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）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（或 RK3399Pro Linux）上运行模型，并获取模型在硬件上运行时的总时间和每一层的耗时信息。

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'

3 使用说明

3.1 安装准备

注：由于 TensorFlow 有 CPU 和 GPU 两个版本，在安装时，请根据 PC 是否支持 GPU 选择对应 TensorFlow 的版本。requirements.txt 文件提供两个版本的安装方式，使用时二选一，对于不需要的版本，在执行'pip install'命令前请删掉。

接着执行以下命令进行安装：

```
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
pip install -r package/requirements.txt
pip install package/rknn_toolkit-0.9.7-cp35-cp35m-linux_x86_64.whl
```

请根据不同的 python 版本选择不同的安装包文件（位于 package/目录）：

- **Python3.5**：rknn_toolkit-0.9.7-cp35-cp35m-linux_x86_64.whl
- **Python3.6**：rknn_toolkit-0.9.7-cp36-cp36m-linux_x86_64.whl

3.2 DOCKER 镜像使用说明

在 docker 文件夹下提供了一个打包了所有开发环境的 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-docker-0.9.7.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        0.9.7               1ed7b26caf71        About an hour ago   1.98GB
```

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。

```
docker run -t -i --privileged -v /dev/bus/usb:/dev/bus/usb rknn-toolkit:0.9.7 /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:0.9.7 /bin/bash
```

4、 运行 demo

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

3.3 RKNN Toolkit 的使用

RKNN-Toolkit 有两种基本用法：一是使用原始的 Caffe、TensorFlow、TensorFlow Lite、ONNX、Darknet 模型进行推理和模型性能评估；二是使用 RKNN 模型进行推理和模型性能评估。

3.3.1 针对 Caffe、TensorFlow、TensorFlow Lite、ONNX、Darknet 模型的工具使用流程

如果现有的模型类型是 Caffe、TensorFlow、TensorFlow Lite、ONNX、Darknet，可以按照以下流程使用 RKNN Toolkit。

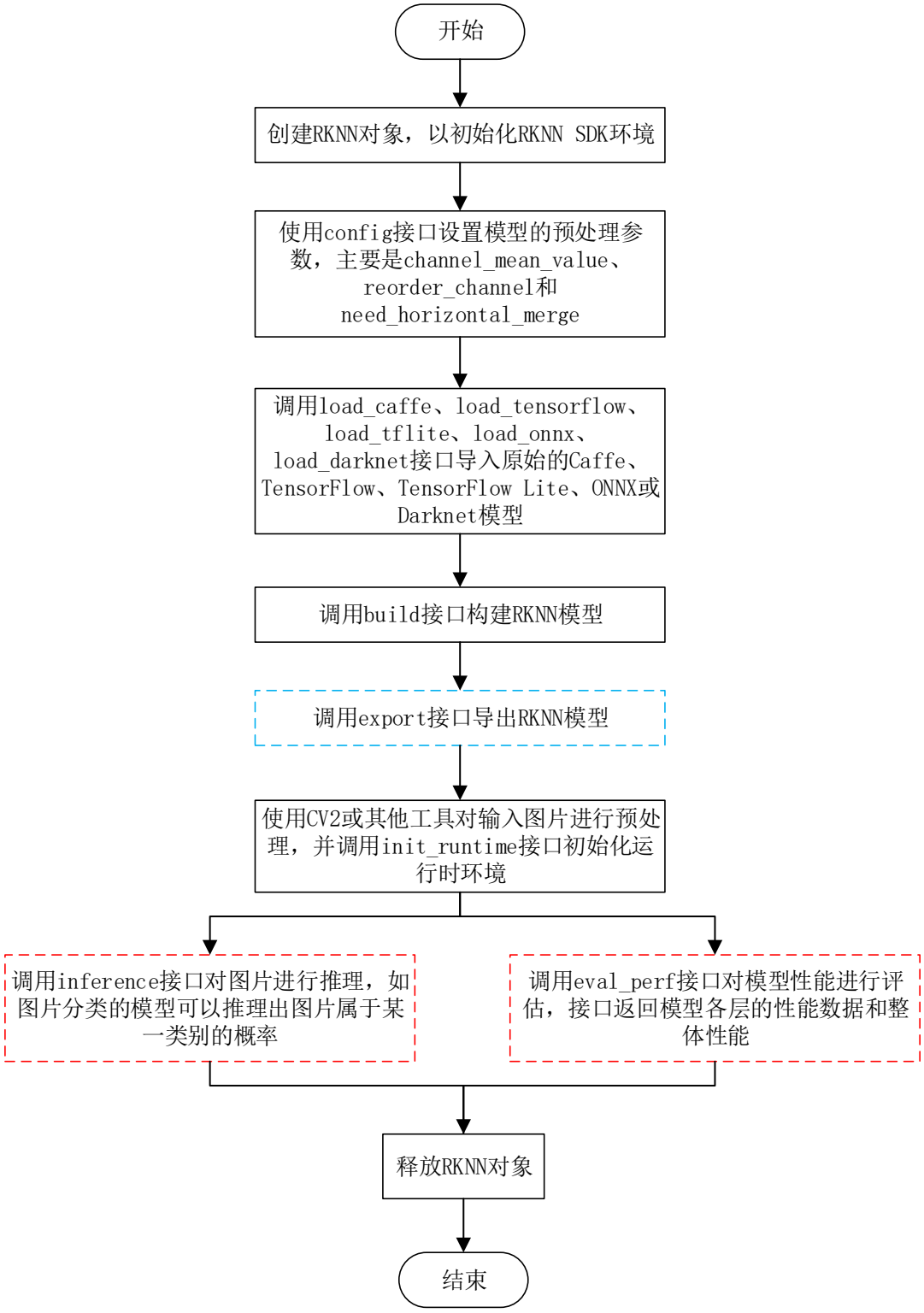

图 3-3-1-1 针对 Caffe、TensorFlow 等原始模型的工具使用流程

注：

1、 在使用模型对图片进行推理或性能评估时，前面的步骤除蓝色虚线框标注的导出模型外，都要执行。

2、 上述步骤要按图中的先后顺序执行。

3、 调用 init_runtime 初始化运行时环境时，如果是连接硬件平台进行调试，则需要设置 target 属性为硬件平台名称（如 RK3399Pro）；如果 PC 连接多台硬件设备，还需要设置 device_id 属性（可以通过 adb devices -l 命令查询）。如果工具运行在 arm64 Linux 系统中，则需要设置 host 属性为”RK3399Pro”。

4、 默认情况下，性能评估接口只返回模型在相应运行环境（模拟器、硬件平台）中的总耗时信息，如果上一步初始化运行时环境时指定 perf_debug 为 True，则评估性能时能够获取模型每一层的运行时间。

3.3.2 针对 RKNN 模型的工具使用流程

如果已有 RKNN 模型，可以根据下面的流程使用 RKNN Toolkit。

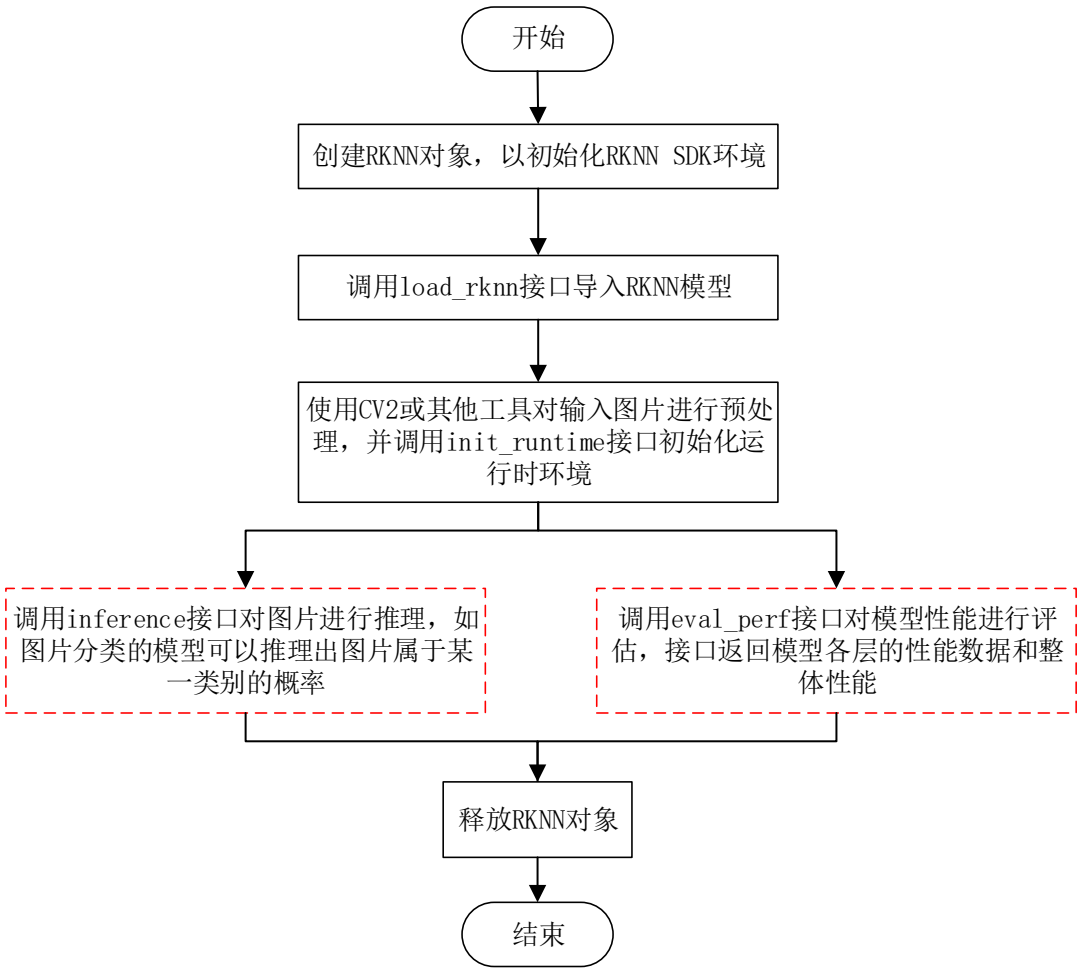

图 3-3-2-1 针对 RKNN 模型的工具使用流程

注：

1、 调用 init_runtime 初始化运行时环境时，如果是连接硬件平台进行调试，则需要设置 target 属性为硬件平台名称（如 RK3399Pro）；如果 PC 连接多台硬件设备，还需要设置 device_id 属性（可以通过 adb devices -l 命令查询）。如果工具运行在 arm64 Linux 系统中，则需要设置 host 属性为”RK3399Pro”。。

3.3.3 RKNN-Toolkit 连接开发板硬件调试

RKNN-Toolkit 能够直接通过 USB 连接开发板硬件，在硬件上运行模型推理或获取模型性能信息。步骤如下：

1、 确保开发板的 USB OTG 连接到 PC，并且 ADB 能够正常识别到设备

2、 修改 Example 并运行

初始化运行环境时需要指定 target 和 device_id

```
ret = init_runtime(target='rk3399pro', device_id='xxxxxx')
```

3.3.4 在 RK3399Pro Linux 上运行 RKNN-Toolkit

RKNN-Toolkit 提供了 Linux AArch64 版本的安装包，能够支持在 RK3399Pro 的 Linux 发行版上运行。其安装步骤与 PC 基本一致，在初始化运行环境时需要指定 host 参数。

```
ret = init_runtime(host='rk3399pro')
```

3.4 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）：

```
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):
        #total_cycles = outputs['total_cycles']
        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')

        # 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')
    #print('inference result: ', outputs)

    # 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（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
============================================================
```

3.5 API 详细说明

3.5.1 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。

举例如下：

```
rknn = RKNN()
...
rknn.release()
```

3.5.2 模型加载

RKNN-Toolkit 目前支持 Caffe、TensorFlow、TensorFlow Lite、ONNX、Darknet 五种模型，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。

3.5.2.1 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')
```

3.5.2.2 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。
	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]])
```

3.5.2.3 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(tflite_model = './mobilenet_v1.tflite')
```

3.5.2.4 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')
```

3.5.2.5 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')
```

3.5.3 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：均值，不同的均值对模型推理结果的准确性有影响。
	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：图像通道顺序。’0 1 2’代表 RGB，’2 1 0’代表 BGR。
	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)
```

3.5.4 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')
```

3.5.5 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
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')
.....
```

3.5.6 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(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（RK3399Pro、RK3399Pro Linux）运行或直接使用模拟器运行。

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。默认值为"simulator"，即在模拟器上运行。
	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"adb devices -l"命令查看。默认值为 None。
	host：工具能够支持在 RK3399Pro 的 Linux 发行版上运行，在这种场景下，设置这个参数为"rk3399pro"。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)
```

3.5.8 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。如果初始化运行环境时设置 target 或 host 为“RK3399Pro”，则得到的是模型在硬件平台上运行得到的推理结果；否则为模拟器上运行得到的推理结果。
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’。
	outputs： 指定输出数据的格式，格式是 ndarray list，可以指定输出的 shape 和 dtype。默认值为 None。
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得到如下结果：

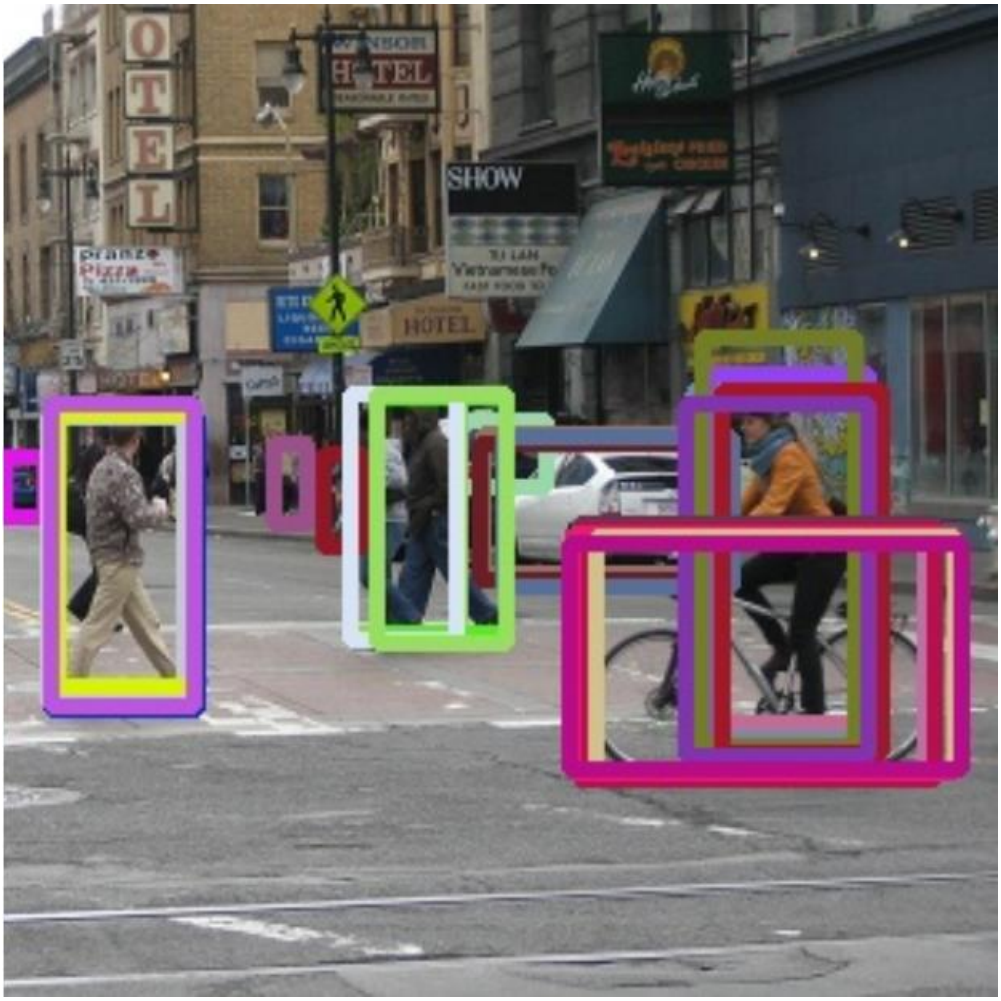

图 3-5-7-1 mobilenet-ssd inference 结果

3.5.9 预估模型性能

API	eval_perf
功能	预估模型的性能。如果初始化 RKNN 对象时没有指定 target 对象，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总的模型运行时间；如果指定 target 对象，但不开启 debug 模式，将只返回模型在硬件上运行的总时间；如果开启 debug 模式，除了返回总时间外，还将返回每一层的运行时间。
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，且不开启 debug 模式时，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，字典 key 为层的 id，举例如下：
	`{ 'total_time', 4568, 'layers', { '0': { 'name': 'convolution.relu.pooling.layer2_2', 'time', 362 } '1': { 'name': 'convolution.relu.pooling.layer2_2', '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
============================================================
```