

密级状态：绝密( ) 秘密( ) 内部( ) 公开(√)

# RK3399Pro\_Linux&Android\_RKNN\_API\_V0.9.1\_ 20181127

(技术部，图形显示平台中心)

|                               |       |            |
|-------------------------------|-------|------------|
| 文件状态：<br>[ ] 正在修改<br>[√] 正式发布 | 当前版本： | V0.9.1     |
|                               | 作 者：  | 杜坤明        |
|                               | 完成日期： | 2018-11-27 |
|                               | 审 核：  | 卓鸿添        |
|                               | 完成日期： | 2018-11-27 |

福州瑞芯微电子股份有限公司

Fuzhou Rockchips Semiconductor Co., Ltd

(版本所有, 翻版必究)

## 更新记录

| 版本     | 修改人 | 修改日期       | 修改说明 | 核定人 |
|--------|-----|------------|------|-----|
| V0.9.1 | 杜坤明 | 2018-11-27 | 初始版本 | 卓鸿添 |
|        |     |            |      |     |
|        |     |            |      |     |
|        |     |            |      |     |
|        |     |            |      |     |
|        |     |            |      |     |
|        |     |            |      |     |
|        |     |            |      |     |
|        |     |            |      |     |
|        |     |            |      |     |

## 目 录

|       |                    |    |
|-------|--------------------|----|
| 1     | 主要功能说明 .....       | 3  |
| 2     | 系统依赖说明 .....       | 3  |
| 2.1   | LINUX 平台依赖 .....   | 3  |
| 2.2   | ANDROID 平台依赖 ..... | 3  |
| 3     | API 使用说明 .....     | 3  |
| 3.1   | RKNN API 说明 .....  | 4  |
| 3.2   | LINUX DEMO .....   | 11 |
| 3.2.1 | 编译说明 .....         | 11 |
| 3.2.2 | 运行说明 .....         | 12 |
| 3.3   | ANDROID DEMO ..... | 12 |
| 3.3.1 | 编译说明 .....         | 12 |
| 3.3.2 | 运行说明 .....         | 13 |

## 1 主要功能说明

本 API SDK 为基于 RK3399Pro Linux/Android 的神经网络 NPU 硬件的一套加速方案，可为采用 RKNN API 开发的 AI 相关应用提供通用加速支持。

本文主要包含 3 个部分：

- 1) RKNN API： RKNN API 详细的 API 定义和使用说明。
- 2) Linux Demo： 编译出 Linux 平台上使用硬件加速的基于 MobileNet 的分类器 Demo 和基于 MobileNet-SSD 目标检测的 Demo。
- 3) Android Demo： 编译出 Android 平台上使用硬件加速的基于 MobileNet-SSD 目标检测的 Demo。

## 2 系统依赖说明

### 2.1 Linux 平台依赖

本 API SDK 基于 RK3399Pro 的 64 位 Linux 开发，需要在 RK3399Pro 的 64 位 Linux 系统上使用。

### 2.2 Android 平台依赖

本 API SDK 基于 RK3399Pro 的 Android 8.1 开发，需要在 RK3399Pro 的 Android 8.1 及 Android 8.1 以上系统上使用。

## 3 API 使用说明

RKNN API 是一套基于 RK3399Pro 的 NPU 硬件加速的应用编程接口（API），开发者可以使用该 API 开发相关的 AI 应用，该 API 会自动调用 NPU 硬件加速器来进行加速。

目前该 RKNN API 在 Linux 和 Android 平台下的接口是一致的。

Linux 平台上，API SDK 提供了两个使用 RKNN API 的 Demo，一个是基于 MobileNet 模型图像

分类器 Demo，另一个是基于 MobileNet-SSD 模型的目标检测 Demo；

Android 平台上，API SDK 提供了一个使用 RKNN API 的基于 MobileNet-SSD 模型的目标检测 Demo。

### 3.1 RKNN API 说明

RKNN API 为 Rockchips 为 RK3399Pro 的 NPU 硬件加速器设计的一套通用 API，该 API 需要配合 Rockchips 提供的 RKNN 模型转换工具一起使用，RKNN 模型转换工具可以将常见的模型格式转换成 RKNN 模型，例如 Tensorflow 的 pb 模型和 tflite 模型，caffe 的模型等。

RKNN 模型转换工具的详细说明请参见《RKNN-Toolkit 使用指南》。

RKNN 模型转换工具可以输出文件后缀为 rknn 的模型文件，如 mobilenet\_v1-tf.rknn。

Linux 平台上，进入 Linux/rknn\_api\_sdk 目录，RKNN API 的定义在 rknn\_api\_sdk/rknn\_api/include/rknn\_api.h 的头文件里。RKNN API 的动态库路径为 rknn\_api\_sdk/rknn\_api/lib64/librknn\_api.so。应用程序只需要包含该头文件和动态库，就可以编写相关的 AI 应用。

Android 平台上，进入 Android/rknn\_api 目录，RKNN API 的定义在 rknn\_api/include/rknn\_api.h 的头文件里。RKNN API 的动态库路径为 rknn\_api/lib64/librknn\_api.so。应用程序只需要包含该头文件和动态库，就可以编写相关的 AI 应用的 JNI 库。目前 Android 上只支持采用 JNI 的开发方式。

**RKNN API 的函数定义如下：**

1) int rknn\_init(void\* model, int len, uint32\_t flag)

功能：

加载 rknn 模型并初始化 context。

参数：

|                            |   |
|----------------------------|---|
| <code>void* model</code>   | 指向 rknn 模型的指针。                                    |
| <code>int len</code>       | rknn 模型的长度。                                       |
| <code>uint32_t flag</code> | 扩展 flag, 详见 rknn_api.h 里的 RKNN_FLAG_XXX_XXX 的宏定义。 |

返回值:

|                  |                      |  |
|------------------|----------------------|--|
| <code>int</code> | <code>&gt;= 0</code> | 初始化成功, 返回 context 的句柄。                   |
|                  | <code>&lt; 0</code>  | 错误码, 错误码定义详见 rknn_api.h 里的 RKNN_ERR_XXX。 |

## 2) `int rknn_destroy(int context)`

功能:

卸载 rknn 模型并销毁 context。

参数:

|                          |              |
|--------------------------|--------------|
| <code>int context</code> | context 的句柄。 |
|--------------------------|--------------|

返回值:

|                  |                     |       |
|------------------|---------------------|-------|
| <code>int</code> | <code>= 0</code>    | 执行成功。 |
|                  | <code>&lt; 0</code> | 错误码。  |

## 3) `int rknn_query(int context, rknn_query_cmd cmd, void* info, int info_len)`

功能:

查询模型或其他的信息, 查询项详见 rknn\_api.h 的 enum rknn\_query\_cmd。

参数:

|                                 |                  |
|---------------------------------|------------------|
| <code>int context</code>        | context 的句柄。     |
| <code>rknn_query_cmd cmd</code> | 查询命令。            |
| <code>void* info</code>         | 查询结果的 buffer 地址。 |
| <code>int info_len</code>       | 查询结果的 buffer 长度。 |

返回值:

int == 0 执行成功。  
 < 0 错误码。

注:

如果 cmd 为 RKNN\_QUERY\_PERF\_DETAIL, 则需要在 rknn\_init 的 flag 与上 RKNN\_FLAG\_COLLECT\_PERF\_MASK, 否则获取不到详细的各层性能信息, 同时在使用 RKNN\_FLAG\_COLLECT\_PERF\_MASK 标志之后, 由于需要同步每层的执行操作, 因此总耗时会比不使用 RKNN\_FLAG\_COLLECT\_PERF\_MASK 标志时更长。另外, RKNN\_QUERY\_PERF\_DETAIL 查询返回的 struct rknn\_perf\_detail 结构体的 perf\_data 成员不需要用户进行主动释放。

4) int rknn\_input\_set(int context, int index, void\* buf, int len, uint8\_t order)

功能:

设置 input 的 buffer。

参数:

int context context 的句柄。  
 int index 在 rknn 模型中对应的 input 索引号。  
 void\* buf input buffer 指针  
 int len input buffer 的长度。  
 uint8\_t order input buffer 的 order 调整, 详见 rknn\_api.h 的 rknn\_input\_order\_type 的定义。该设置需要与 rknn-toolkit 里配置的 input 参数一致。

返回值:

int == 0 执行成功。  
 < 0 错误码。

5) int rknn\_run(int context, struct rknn\_run\_extend\* extend)

功能:

执行模型的推理操作。该函数任何情况下都不会阻塞。

参数:

int context            context 的句柄。

struct rknn\_run\_extend\* extend 扩展信息的指针，用于输出当前 run 对应的帧的信息，如 frame\_id，详见 rknn\_api.h 的 struct rknn\_run\_extend 定义。

返回值:

int == 0 执行成功。

< 0 错误码。

```
6) int rknn_outputs_get(int context, int num, struct rknn_output outputs[],
                        struct rknn_output_extend* extend)
```

功能:

等待推理操作结束并获取 `outputs` 结果。该函数在推理结束前会一直阻塞（除非有异常出错）。输出结果会被存至 `outputs[]` 数组。

参数:

int context                      context 的句柄。

int num          outputs 数组的个数,该个数要与 rknn 模型的输出个数一致。

struct rknn\_output outputs[]      outputs 的数组指针。

struct rknn\_output\_extend\* extend 扩展信息的指针，用于输出当前 output 对应的帧的信息，如 frame id，详见 rknn api.h 的 struct rknn output extend 定义。

返回值:

int            >= 0      执行成功，返回 outputs 的句柄。

$< 0$  错误码。

```
7) int rknn_outputs_release(int context, int buf handle)
```

功能:

释放由 rknn outputs get 获取的 outputs 句柄。当该句柄被释放后，由



rknn\_outputs\_get 获取的 rknn\_output[x].buf 地址也会被释放。

参数:

int context                context 的句柄。

int buf\_handle            outputs 的句柄。

返回值:

Int                        == 0      执行成功。

                          < 0      错误码。

8) int rknn\_output\_to\_float(const struct rknn\_output &output, void\* dst, int size)

功能:

将由 rknn\_outputs\_get 获取的 rknn\_output 转为 float32。因为 rknn\_output 里的数据可能是量化的数据，如需要浮点的结果，可通过该函数进行转换。

参数:

int context                context 的句柄。

struct rknn\_output &output            由 rknn\_outputs\_get 获取的单项 output。

void\* dst                  float32 的目标 buffer。

int size                    目标 buffer 的大小。

返回值:

Int                        == 0      执行成功。

                          < 0      错误码。

**RKNN API 的基本调用流程如下:**

- 1) 读取 rknn 模型文件到内存，这边的 rknn 模型文件就是用前面介绍的 RKNN 模型转换工具生成的文件后缀为.rknn 的模型文件，如 mobilenet\_v1-tf.rknn。
- 2) 调用 rknn\_init 加载 rknn 的模型并初始化 context。代码如下:

```
int ctx = rknn_init(model, model_len, RKNN_FLAG_PRIOR_MEDIUM);
if(ctx < 0) {
    printf("rknn_init fail! ret=%d\n", ctx);
    goto Error;
}
```

其中，model 为 rknn 模型再内存里的指针；model\_len 为模型大小；RKNN\_FLAG\_PRIOR\_MEDIUM 为优先级标志位。其他标志位详见 rknn\_api.h 的 RKNN\_FLAG\_XXX。

- 3) rknn 模型的 input/output 参数可以由原始模型（pb 或 caffe）得知，也可以通过 rknn\_query 这个 api 获取，如下：

```
struct rknn_input_output_num output_num;
ret = rknn_query(ctx, RKNN_QUERY_IN_OUT_NUM, &output_num, sizeof(output_num));
if(ret < 0) {
    printf("rknn_query fail! ret=%d\n", ret);
    goto Error;
}
```

上述接口用于获取 input/output 的个数，个数会存储在 output\_num.n\_input 和 output\_num.n\_output 里。

```
struct rknn_tensor_attr output0_attr;
output0_attr.index = 0;
ret = rknn_query(ctx, RKNN_QUERY_OUTPUT_ATTR, &output0_attr, sizeof(output0_attr));
if(ret < 0) {
    printf("rknn_query fail! ret=%d\n", ret);
    goto Error;
}
```

上述接口用于获取某个 output 的属性，记得填写 rknn\_tensor\_attr 的 index（该 index 不能大于等于前面获取的 output 的个数）。属性定义详见 rknn\_tensor\_attr。

获取某个 input 的属性方法与获取 output 属性方法类似。

- 4) 根据 rknn 模型的 input 参数，调用 rknn\_input\_set 分别对每个 input buffer 进行设置。

代码如下：

```
ret = rknn_input_set(ctx, input_index, img.data,
                    img_width * img_height * img_channels, RKNN_INPUT_ORDER_012);
if(ret < 0) {
    printf("rknn_input_set fail! ret=%d\n", ret);
    goto Error;
}
```

其中，input\_index 为 rknn 模型的 input node 的索引；img.data 变量为 cpu 可以访问的 buffer 指针；img\_width \* img\_height \* img\_channels 为 buffer 的大小；

RKNN\_INPUT\_ORDER\_012 为该 buffer 的 order, 这个 order 需要与生成 rknn 模型时设置的 reorder\_channel 参数一致, reorder\_channel 请参见《RKNN-Toolkit 使用指南》文档。

- 5) 在所有 input 数据都设置完毕后, 调用 rknn\_run 触发推理的操作, 该函数会立即返回, 并不会阻塞。代码如下:

```
ret = rknn_run(ctx, nullptr);
if(ret < 0) {
    printf("rknn_run fail! ret=%d\n", ret);
    goto Error;
}
```

- 6) 执行完 rknn\_run, 可以调用 rknn\_outputs\_get 等待推理完成, 推理完成后, 可以获取推理的结果。代码如下:

```
h_output = rknn_outputs_get(ctx, 1, outputs, nullptr);
if(h_output < 0) {
    printf("rknn_outputs_get fail! ret=%d\n", ret);
    goto Error;
}
```

推理的结果存储在 outputs 变量里, h\_output 为该次推理结果的句柄。outputs 和 h\_output 的定义如下:

```
int h_output = -1;
struct rknn_output outputs[1];
```

- 7) 如果模型的输出格式不为 float32, 但程序需要 float32 的数据进行后处理, 则可以调用 rknn\_output\_to\_float 将 output 数据转换为 float32 数据。代码如下:

```
if(output0_attr.type != RKNN_OUTPUT_FLOAT32) {
    int output_size = output_elems * sizeof(float);
    void* output_buf = malloc(output_size);
    rknn_output_to_float(ctx, outputs[0], output_buf, output_size);

    // post process code
    ...

    free(output_buf);
}
```

其中, output\_buf 为 float32 数据的存放 buffer, outputs[0] 为从 rknn\_outputs\_get 获取的原始 output。之后就可以拿 output\_buf 进行后处理。

- 8) 当由 rknn\_outputs\_get 获取的所有 outputs 不再需要使用之后, 需要调用 rknn\_outputs\_release 对该 outputs 进行释放, 否则会照成内存泄漏。代码如下:

```
rknn_outputs_release(ctx, h_output);
```

其中，h\_output 为 rknn\_outputs\_get 返回的句柄。

9) 需要进行多次推理，可跳回步骤 3 进行下一次推理。

程序需要退出时，需要调用 rknn\_destroy 卸载 rknn 模型并销毁 context。代码如下：

```
rknn_destroy(ctx);
```

更具体代码请参见 API SDK 的 Linux 目录下的 rknn\_api\_sdk/rknn\_mobilenet.cpp 和 rknn\_api\_sdk/rknn\_ssd.cpp。

## 3.2 Linux Demo

### 3.2.1 编译说明

API SDK 的 Linux 目录下提供了两个使用 RKNN API 的 Demo，一个是基于 MobileNet 模型图像分类器 Demo，另一个是基于 MobileNet-SSD 模型的目标检测 Demo。

进入 Linux/rknn\_api\_sdk 目录，这两个 Demo 的主源文件为 rknn\_api\_sdk/rknn\_mobilenet.cpp 和 rknn\_api\_sdk/rknn\_ssd.cpp，具体编译方法如下：

1) 安装交叉编译工具，执行：

```
sudo apt install gcc-aarch64-linux-gnu
```

```
sudo apt install g++-aarch64-linux-gnu
```

2) cd rknn\_api\_sdk; mkdir build; cd build; cmake ..

3) make

make 结束后即可在 rknn\_api\_sdk/build/生成 rknn\_mobilenet 和 rknn\_ssd 两个可执行文件。

### 3.2.2 运行说明

rknn\_mobilenet 和 rknn\_ssd 的运行需要将相关依赖库拷贝至/usr/lib64/下，同时将相关资源文件拷贝至/tmp 目录下，具体步骤如下：

- 1) 将 rknn\_api\_sdk/3rdparty/opencv/lib64 和 rknn\_api\_sdk/rknn\_api/lib64 目录下的文件拷贝至 RK3399Pro 目标板的/usr/lib64/目录下。
- 2) 将 API SDK 的 Linux/tmp/目录下的资源文件拷贝至 RK3399Pro 目标板的/tmp/目录下。
- 3) 将上述 rknn\_api\_sdk/build 目录里编译生成的 rknn\_mobilenet 和 rknn\_ssd 也拷贝至 RK3399Pro 目标板的/tmp/目录下。
- 4) 进入 RK3399Pro 目标板的/tmp 目录执行：

```
./rknn_mobilenet
```

执行成功后会有执行时间和检测结果的打印。

进入 RK3399Pro 目标板的/tmp 目录执行：

```
./rknn_ssd
```

执行成功后会有执行时间和检测结果的打印，同时还会在 RK3399Pro 目标板的/tmp 目录下生成包含检测结果的图像 out.jpg，可以导出 out.jpg 查看检测结果。

## 3.3 Android Demo

### 3.3.1 编译说明

API SDK 的 Android 目录下有一个 rknn\_api 目录和一个 rk\_ssd\_demo 目录。

如想直接使用 RKNN API 来开发自己的 JNI 库，则 JNI 库可以直接包含 rknn\_api 里的 include/rknn\_api.h 和 lib64/librknn\_api.so 来调用到 rknn\_api。

Android/rk\_ssd\_demo 目录为使用 RKNN API 的基于 MobileNet-SSD 模型的目标检测 Demo。

该 demo 完整的 ssd 目标检测 demo，包含了 java 和 jni 的部分，其中 jni 目录的路径为：  
rk\_ssd\_demo/app/src/main/jni。该 jni 目录里已经包含了 rknn\_api.h 头文件，另外  
librknn\_api.so 的存放路径为：rk\_ssd\_demo/app/src/main/jniLibs/arm64-v8a。

rk\_ssd\_demo 的具体编译方法如下：

进入 rk\_ssd\_demo 目录，运行 AndroidStudio 打开该目录的工程文件，编译并生成 apk 即可。  
(需要 NDK 的支持，在 android-ndk-r16b 上验证通过)。

### 3.3.2 运行说明

直接在 RK3399Pro 的 Android 上运行该 apk 即可。