密级状态：绝密(　　)　　秘密(　　)　　内部(　)　　公开(√)

RK3399Pro_Linux&Android_RKNN_API_V0.9.3_20190124

（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：	V0.9.3
	作　　者：	杜坤明
	完成日期：	2019-01-24
	审　　核：	卓鸿添
	完成日期：	2019-01-24

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
V0.9.1	杜坤明	2018-11-27	初始版本	卓鸿添
V0.9.2	杜坤明	2018-12-19	主要修改 input 和 output 的 API 定义	卓鸿添
V0.9.3	杜坤明	2019-01-24	增加 v0.9.1 到 v0.9.2 的 API 迁移说明	卓鸿添

目 录

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 Android8.1 开发，需要在 RK3399Pro 的 Android8.1 及 Android8.1 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 详细说明

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 和 rknn_api/lib/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。

下面的章节是 rknn api 的函数说明。

3.1.1 rknn_init

API	int rknn_init(rknn_context* context, void* model, uint32_t size, uint32_t flag)
功能	创建 context 并加载 rknn 模型，并根据 flag 执行特定的初始化行为。
参数	rknn_context* context： context 对象指针。用于返回创建的 context 对象。
	void* model： 指向 rknn 模型的指针。
	uint32_t size： rknn 模型的大小。
	uint32_t flag： 扩展 flag： **RKNN_FLAG_PRIOR_HIGH**：创建高优先级的 Context。 **RKNN_FLAG_PRIOR_MEDIUM**：创建中优先级的 Context。 **RKNN_FLAG_PRIOR_LOW**：创建低优先级的 Context。 **RKNN_FLAG_ASYNC_MASK**：打开异步模式。打开之后，rknn_outputs_get 将不会阻塞太久，因为它直接返回的上一帧的推理结果（第一帧的推理结果除外），这将显著提高单线程模式下的推理帧率，但代价是 rknn_outputs_get 返回的不是当前帧的推理结果。但当 rknn_run 和 rknn_outputs_get 不在同一个线程时，则无需打开该异步模式。 **RKNN_FLAG_COLLECT_PERF_MASK**：打开性能收集调试开关。打开之后能够通过 rknn_query 接口查询网络每层运行时间。需要注意，该标志被设置后，因为需要同步每层的执行操作，所以推理一帧的总耗时会比不使用 RKNN_FLAG_COLLECT_PERF_MASK 标志时更长。
返回值	int 错误码（见 rknn 返回值错误码）。

示例代码如下：

```
rknn_context ctx;
int ret = rknn_init(&ctx, model_data, model_data_size, 0);
```

3.1.2 rknn_destroy

API	int rknn_destroy(rknn_context context)
功能	卸载 rknn 模型并销毁 context 及其相关资源。
参数	rknn_context context： context 的对象。
返回值	int 错误码（见 rknn 返回值错误码）。

示例代码如下：

```
int ret = rknn_destroy (ctx);
```

3.1.3 rknn_query

API	int rknn_query(rknn_context context, rknn_query_cmd cmd, void* info, uint32_t size)
功能	查询模型与 SDK 的相关信息。
参数	rknn_context context： context 的对象。
	rknn_query_cmd cmd： 查询命令。
	void* info： 存放返回结果的结构体变量。
	uint32_t size： info 对应的结构体变量的大小。
返回值	int 错误码（见 rknn 返回值错误码）

当前 SDK 支持的查询命令如下表所示：

查询命令	返回结果结构体	功能
RKNN_QUERY_IN_OUT_NUM	rknn_input_output_num	查询 input 和 output 的 Tensor 个数。
RKNN_QUERY_INPUT_ATTR	rknn_tensor_attr	查询 Input Tensor 属性。
RKNN_QUERY_OUTPUT_ATTR	rknn_tensor_attr	查询 Output Tensor 属性。
RKNN_QUERY_PERF_DETAIL	rknn_perf_detail	查询网络各层运行时间。 该查询需要在 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。另外，RKNN_QUERY_PERF_DETAIL 查询返回的 rknn_perf_detail 结构体的 perf_data 成员不需要用户进行主动释放。 同时该查询需要在 rknn_outputs_get 函数调用后才能返回正确的查询结果。
RKNN_QUERY_PERF_RUN	rknn_perf_run	查询单帧推理的硬件执行时间。 同时该查询需要在 rknn_outputs_get 函数调用后才能返回正确的查询结果。
RKNN_QUERY_SDK_VERSION	rknn_sdk_version	查询 SDK 版本。

接下来的小节将依次详解各个查询命令如何使用。

3.1.3.1 查询 Input 和 Output 的 Tensor 个数

传入 RKNN_QUERY_IN_OUT_NUM 命令可以查询模型 Input 和 Output 的 Tensor 个数。其中需要先创建 rknn_input_output_num 结构体对象。

示例代码如下：

```
rknn_input_output_num io_num;
ret = rknn_query(ctx, RKNN_QUERY_IN_OUT_NUM, &io_num, sizeof(io_num));
printf("model input num: %d, output num: %d\n", io_num.n_input,
                io_num.n_output);
```

3.1.3.2 查询 Input 的 Tensor 属性

传入 RKNN_QUERY_INPUT_ATTR 命令可以查询模型 Input 的 Tensor 的属性。其中需要先创建 rknn_tensor_attr 结构体对象。

示例代码如下：

```
rknn_tensor_attr input_attrs[io_num.n_input];
memset(input_attrs, 0, sizeof(input_attrs));
for (int i = 0; i < io_num.n_input; i++) {
    input_attrs[i].index = i;
    ret = rknn_query(ctx, RKNN_QUERY_INPUT_ATTR, &(input_attrs[i]),
                     sizeof(rknn_tensor_attr));
}
```

3.1.3.3 查询 Output 的 Tensor 属性

传入 RKNN_QUERY_OUTPUT_ATTR 命令可以查询模型 Output 的 Tensor 的属性。其中需要先创建 rknn_tensor_attr 结构体对象。

示例代码如下：

```
rknn_tensor_attr output_attrs[io_num.n_output];
memset(output_attrs, 0, sizeof(output_attrs));
for (int i = 0; i < io_num.n_output; i++) {
    output_attrs[i].index = i;
    ret = rknn_query(ctx, RKNN_QUERY_OUTPUT_ATTR, &(output_attrs[i]),
                     sizeof(rknn_tensor_attr));
}
```

3.1.3.4 查询网络各层运行时间

如果在 rknn_init 函数调用时有设置 RKNN_FLAG_COLLECT_PERF_MASK 标志，那么在执行 rknn_outputs_get 调用完成之后，可以传入 RKNN_QUERY_PERF_DETAIL 命令来查询网络每层运行时间。其中需要先创建 rknn_perf_detail 结构体对象。

另外，RKNN_QUERY_PERF_DETAIL 查询返回的 rknn_perf_detail 结构体的 perf_data 成员不需要用户进行主动释放。

同时该查询需要在 rknn_outputs_get 函数调用后才能返回正确的查询结果。

示例代码如下：

```
rknn_perf_detail perf_detail;
ret = rknn_query(ctx, RKNN_QUERY_PERF_DETAIL, &perf_detail,
                        sizeof(rknn_perf_detail));
printf("%s", perf_detail.perf_data);
```

3.1.3.5 查询单帧推理的时间

传入 RKNN_QUERY_PERF_RUN 命令可以查询单帧推理的硬件执行时间。其中需要先创建 rknn_perf_run 结构体对象。

同时该查询需要在 rknn_outputs_get 函数调用后才能返回正确的查询结果。

示例代码如下：

```
rknn_perf_run perf_run;
ret = rknn_query(ctx, RKNN_QUERY_PERF_RUN, &perf_run,
                        sizeof(rknn_perf_run));
printf("%ld", perf_run.run_duration);
```

3.1.3.6 查询 SDK 版本

传入 RKNN_QUERY_SDK_VERSION 命令可以查询 RKNN API 以及 Driver 的版本。其中需要先创建 rknn_sdk_version 结构体对象。

示例代码如下：

```
rknn_sdk_version version;
ret = rknn_query(ctx, RKNN_QUERY_SDK_VERSION, &version,
                    sizeof(rknn_sdk_version));
printf("api version: %s\n", version.api_version);
printf("driver version: %s\n", version.drv_version);
```

3.1.4 rknn_inputs_set

API	int rknn_inputs_set(rknn_context context, uint32_t n_inputs, rknn_input inputs[])

功能	设置 inputs 的 buffer 以及参数。 Buffer 及参数需存储在 rknn_input 中。该函数能够支持多个 input，其中每个 input 是 rknn_input 结构体对象，在传入之前用户需要设置该对象。
参数	rknn_context context： context 的对象。
	uint32_t n_inputs： inputs 的个数。
	rknn_input inputs[]： inputs 的数组指针，数组每个元素是 rknn_input 结构体对象。
返回值	int 错误码（见 rknn 返回值错误码）

示例代码如下：

```
rknn_input inputs[1];
memset(inputs, 0, sizeof(inputs));
inputs[0].index = 0;
inputs[0].type = RKNN_TENSOR_UINT8;
inputs[0].size = img_width*img_height*img_channels;
inputs[0].pass_through = FALSE;
inputs[0].fmt = RKNN_TENSOR_NHWC;
inputs[0].buf = in_data;

ret = rknn_inputs_set(ctx, 1, inputs);
```

更详细的使用方法见【RKNN API 基本调用流程】章节的步骤 4。

3.1.5 rknn_run

API	int rknn_run(rknn_context context, rknn_run_extend* extend)
功能	执行一次模型推理，调用之前需要先通过 rknn_inputs_set 函数设置 input 数据。 该函数正常不会阻塞，但是当有超过 3 次推理结果没有通过 rknn_outputs_get 获取时则会阻塞，直至 rknn_outputs_get 被调用。
参数	rknn_context context：context 的对象。
	rknn_run_extend* extend：扩展信息的指针，用于设置或输出当前 rknn_run 对应的帧的信息，如 frame_id（详见 rknn_api.h 的 rknn_run_extend 定义）。如不用，可赋 NULL。
返回值	int 错误码（见 rknn 返回值错误码）

示例代码如下：

```
ret = rknn_run(ctx, NULL);
```

3.1.6 rknn_outputs_get

API	int rknn_outputs_get(rknn_context context, uint32_t n_outputs, rknn_output outputs[], rknn_output_extend* extend)
功能	等待推理操作结束并获取 outputs 结果。 该函数能够一次获取多个 output 数据。其中每个 output 是 rknn_output 结构体对象，在函数调用之前需要依次创建并设置每个 rknn_output 对象。另外，在推理结束前该函数会一直阻塞（除非有异常出错）。output 结果最后会被存至 outputs[]数组。 对于 output 数据的 buffer 存放可以采用两种方式：一种是用户自行申请和释放，此时 rknn_output 对象的 is_prealloc 需要设置为 TRUE，并且将 buf 指针指向用户申请的 buffer；另一种是由 rknn 来进行分配，此时 rknn_output 对象的 is_prealloc 设置为 FALSE 即可，函数执行之后 buf 将指向 output 数据。
参数	rknn_context context： context 的对象。
	uint32_t n_outputs： outputs 数组的个数，该个数要与 rknn 模型的 output 个数一致。（rknn 模型的 output 个数可以通过 rknn_query 查询得到。）
	rknn_output outputs[]： output 数据的数组，其中数组每个元素为 rknn_output 结构体对象，代表模型的一个 output。
	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 rknn_output_extend 定义）。如不用，可赋 NULL。
返回值	int 错误码（见 rknn 返回值错误码）

示例代码如下：

```
rknn_output outputs[io_num.n_output];
memset(outputs, 0, sizeof(outputs));
for (int i = 0; i < io_num.n_output; i++) {
    outputs[i].want_float = TRUE;
    outputs[i].is_prealloc = FALSE;
}
ret = rknn_outputs_get(ctx, io_num.n_output, outputs, NULL);
```

更详细的使用方法见【RKNN API 基本调用流程】章节的步骤 6。

3.1.7 rknn_outputs_release

API	int rknn_outputs_release(rknn_context context, uint32_t n_ouputs, rknn_output outputs[])
功能	释放由 rknn_outputs_get 获取的 outputs。 在 outputs 不再使用时需要调用该函数进行 outputs 的释放（不管 rknn_output[x].is_prealloc 是 TRUE 还是 FALSE 都需要调用该函数进行最终的释放）。 该函数被调用后，当 rknn_output[x].is_prealloc = FALSE 时，由 rknn_outputs_get 获取的 rknn_output[x].buf 地址也会被自动释放；当 rknn_output[x].is_prealloc = ture 时，rknn_output[x].buf 则需要用户自己主动释放。
参数	rknn_context context： context 的对象
	uint32_t n_outputs：outputs 数组的个数，该个数要与 rknn 模型的 output 个数一致。（rknn 模型的 output 个数可以通过 rknn_query 查询得到）
	rknn_output outputs[]： outputs 的数组指针
返回值	int 错误码（见 rknn 返回值错误码）

示例代码如下

```
ret = rknn_outputs_release(ctx, io_num.n_output, outputs);
```

3.2 RKNN 数据结构定义

3.2.1 rknn_input_output_num

结构体 rknn_input_output_num 表示 input 和 output 的 Tensor 个数，其结构体成员变量如下表所示：

成员变量	数据类型	含义
n_input	uint32_t	Input Tensor 个数
n_output	uint32_t	Output Tensor 个数

3.2.2 rknn_tensor_attr

结构体 rknn_tensor_attr 表示模型的 Tensor 的属性，结构体的定义如下表所示：

成员变量	数据类型	含义
index	uint32_t	表示 input 或 output 的 Tensor 的索引。 当使用 rknn_query 查询前，该 index 需要进行设置。
n_dims	uint32_t	Tensor 维度个数。
dims	uint32_t[]	Tensor 各维度值。
name	char[]	Tensor 名称。
n_elems	uint32_t	Tensor 数据元素个数。
size	uint32_t	Tensor 数据所占内存大小。
fmt	rknn_tensor_format	Tensor 维度的格式，有以下格式： RKNN_TENSOR_NCHW RKNN_TENSOR_NHWC
type	rknn_tensor_type	Tensor 数据类型，有以下数据类型： RKNN_TENSOR_FLOAT32

		RKNN_TENSOR_FLOAT16 RKNN_TENSOR_INT8 RKNN_TENSOR_UINT8 RKNN_TENSOR_INT16
qnt_type	rknn_tensor_qnt_type	Tensor 量化类型，有以下的量化类型： RKNN_TENSOR_QNT_NONE：未量化； RKNN_TENSOR_QNT_DFP：动态定点法量化； RKNN_TENSOR_QNT_AFFINE_ASYMMETRIC：非对称量化。
fl	int8_t	RKNN_TENSOR_QNT_DFP 量化类型的参数。
zp	uint32_t	RKNN_TENSOR_QNT_AFFINE_ASYMMETRIC 量化类型的参数。
scale	float	RKNN_TENSOR_QNT_AFFINE_ASYMMETRIC 量化类型的参数。

3.2.3 rknn_input

结构体 rknn_input 表示模型的一个数据 input，用来作为参数传入给 rknn_inputs_set 函数。结构体的定义如下表所示：

成员变量	数据类型	含义
index	uint32_t	该 input 的索引。
buf	void*	input 数据 Buffer 的指针。
size	uint32_t	input 数据 Buffer 所占内存大小。
pass_through	uint8_t	input 数据直通模式。 TRUE：input 数据不做任何转换直接传至 rknn 模型的 input 节点，因此下面的 type 和 fmt 不需要进行设置。

		FALSE：input 数据会根据下面的 type 和 fmt 转换成跟模型的 input 节点一致的数据，因此下面的 type 和 fmt 需要进行设置。
type	rknn_tensor_type	input 数据的类型。
fmt	rknn_tensor_format	input 数据的格式。

3.2.4 rknn_output

结构体 rknn_output 表示模型的一个数据 output，用来作为参数传入给 rknn_outputs_get 函数。结构体的定义如下表所示：

成员变量	数据类型	含义
want_float	uint8_t	标识是否需要将 output 数据转为 float 类型的 output。
is_prealloc	uint8_t	标识存放 output 数据的 Buffer 是否是预分配。
index	uint32_t	该 output 的索引。
buf	void*	output 数据 Buffer 的指针。
size	uint32_t	output 数据 Buffer 所占内存大小。

is_prealloc 为 FALSE 时，在 rknn_outputs_ge 函数执行后，结构体对象的 index/buf/size 成员将会被赋值，因此这三个成员变量不需要预先赋值。

is_prealloc 为 TRUE 时，结构体对象的 index/buf/size 需要预先被赋值，否则 rknn_outputs_get 函数调用会失败并报错。

3.2.5 rknn_perf_detail

结构体 rknn_perf_detail 表示模型的性能详情，结构体的定义如下表所示：

成员变量	数据类型	含义
perf_data	char*	性能详情包含网络每层运行时间，能够直接打印出来查看。

data_len	uint64_t	存放性能详情的字符串数组的长度。

3.2.6 rknn_perf_run

结构体 rknn_perf_run 表示模型的单次推理的执行时间，结构体的定义如下表所示：

成员变量	数据类型	含义
run_duration	int64_t	模型的单次推理的硬件执行时间。

3.2.7 rknn_run_extend

结构体 rknn_run_extend 表示 rknn_run 的扩展信息,用来作为参数传入给 rknn_run 函数，结构体的定义如下表所示：

成员变量	数据类型	含义
frame_id	uint64_t	返回参数，表示当前 run 的帧 id。该 id 与 rknn_output_extend.frame_id 一一对应，在 rknn_run 和 rknn_outputs_get 处于不同线程的情况下，可以用来确定帧的对应关系。

3.2.8 rknn_output_extend

结构体 rknn_output_extend 表示 rknn_outputs_get 的扩展信息，用来作为参数传入给 rknn_outputs_get 函数，结构体的定义如下表所示：

成员变量	数据类型	含义
frame_id	uint64_t	返回参数，表示当前 output 的帧 id。该 id 与 rknn_run_extend.frame_id 一一对应，在 rknn_run 和 rknn_outputs_get 处于不同线程的情况下，可以用来确定帧的对应关系。

3.2.9 rknn_sdk_version

结构体 rknn_sdk_version 用来表示 RKNN SDK 的版本信息，结构体的定义如下：

成员变量	数据类型	含义
api_version	char[]	rknn api 的版本信息。
drv_version	char[]	rknn api 所基于的驱动版本信息。

3.2.10 rknn 返回值错误码

RKNN API 函数的返回值错误码定义如下表所示

错误码	错误详情
RKNN_SUCC	执行成功
RKNN_ERR_FAIL	执行出错
RKNN_ERR_TIMEOUT	执行超时
RKNN_ERR_DEVICE_UNAVAILABLE	NPU 设备不可用
RKNN_ERR_MALLOC_FAIL	内存分配失败
RKNN_ERR_PARAM_INVALID	传入参数错误
RKNN_ERR_MODEL_INVALID	传入的 RKNN 模型无效
RKNN_ERR_CTX_INVALID	传入的 rknn_context 无效
RKNN_ERR_INPUT_INVALID	传入的 rknn_input 对象无效
RKNN_ERR_OUTPUT_INVALID	传入的 rknn_output 对象无效
RKNN_ERR_DEVICE_UNMATCH	版本不匹配

3.3 RKNN API 基本调用流程

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 初始化 context 并加载 rknn 的模型。代码如下：

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

其中，ctx 为 context 对象；model 为 rknn 模型在内存里的指针；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 属性可能已经和原始模型（pb 或 caffe）不同，因此需要通过 rknn_query 这个 api 获取 input 和 output 的属性，如下：

```
    rknn_input_output_num io_num;
    ret = rknn_query(ctx, RKNN_QUERY_IN_OUT_NUM, &io_num, sizeof(io_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 io_num.n_input 和 io_num.n_output 里。

接下来获取 output 的属性:

```
    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 参数和 input 数据的具体格式，调用 rknn_input_set 对 inputs 进行设置。代码如下：

```
rknn_input inputs[1];
inputs[0].index = input_index;
inputs[0].buf = img.data;
inputs[0].size = img_width * img_height * img_channels;
inputs[0].pass_through = FALSE;
inputs[0].type = RKNN_TENSOR_UINT8;
inputs[0].fmt = RKNN_TENSOR_NHWC;
ret = rknn_inputs_set(ctx, 1, inputs);
if(ret < 0) {
    printf("rknn_input_set fail! ret=%d\n", ret);
    goto Error;
}
```

首先，先创建 rknn_input 数组（这边假设只有一个 input，因此数组大小设为 1），然后填充数组的每个数组项的每个成员。

其中：

inputs[0].index　为 rknn 模型的 input node 的索引。

inputs[0].buf　为 cpu 可以访问的 buffer 指针，一般是指向由 Camera 产生的图像数据，如 RGB888 的数据。

inputs[0].size　为 buffer 的大小。

inputs[0].pass_through　为 input 数据直通模式选择。

TRUE： 如果用户传入的 input 数据的属性（主要是 type 和 fmt 以及量化参数）和通过 rknn_query 查询得到的 input 属性一致，则可以将该变量设为 TRUE，同时下面的 type 和 fmt 无需进行设置。在这种模式下，rknn_inputs_set 会将用户传入的 input 数据直接传至 rknn 模型的 input 节点。

这种模式用于用户已知 rknn 模型的 input 属性，并且已经将原始输入数据自行

转为与 rknn 模型 input 一致的数据。

FALSE： 如果用户传入的 input 数据的属性（主要是 type 和 fmt 以及量化参数）和通过 rknn_query 查询得到的 input 属性不一致，则需要将该变量设为 FASLE，同时下面的 type 和 fmt 也需要根据用户传入的 input 数据进行设置。在这种模式下，rknn_inputs_set 函数会自动进行类型、格式的转换以及量化的处理。注意，目前这种模式下不支持用户传入使用动态定点量化(DFP)或非对称量化（AFFINE ASYMMETRIC）的 input 数据。

inputs[0].type 为 buffer 的数据类型，如果是 RGB888 的数据，则为 RKNN_TENSOR_UINT8。

inputs[0].fmt 为 buffer 的数据格式，也就是 NHWC 或 NCHW，一般 Cemara 获取的数据格式为 RKNN_TENSOR_NHWC。

5） 在 inputs 参数设置完毕后，调用 rknn_run 触发推理的操作，该函数正常情况下会立即返回，并不会阻塞（但是当有超过 3 次推理的结果没有通过 rknn_outputs_get 获取时则会阻塞，直至 rknn_outputs_get 被调用）。代码如下：

```
ret = rknn_run(ctx, NULL);
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，该函数会阻塞直到推理完成，推理完成后可以获取推理的结果。代码如下：

```
rknn_output outputs[1];
outputs[0].want_float = TRUE;
outputs[0].is_prealloc = FALSE;
ret = rknn_outputs_get(ctx, 1, outputs, NULL);
if(ret < 0) {
    printf("rknn_outputs_get fail! ret=%d\n", ret);
    goto Error;
}
```

首先，先创建 rknn_output 数组（这边假设只有一个 output，因此数组大小设为 1）。rknn_output 结构体的前两个成员变量必须赋值，分别是 outputs[0].want_float 和 outputs[0].is_prealloc。

want_float：由于 rknn 模型的 output 可能与转换前的原始模型的 output 不一致，一般情况下，rknn 模型的 output 类型为 UINT8 或 FP16（rknn 模型的 output 具体属性可以通过 rknn_query 查询获得），如果用户希望获得的是 FP32 的浮点 output 数据，则可以将该变量置为 ture；如果希望获得是 rknn 模型原始的 output 数据，则置为 FALSE 即可。

is_prealloc = FALSE：如果用户没有预先分配各个 output 的 buffer，可以将该预分配标志设为 FALSE，剩下的 outputs[0]结构体的其余成员变量则不需要赋值。函数返回后，推理的结果会存储在 output[0]结构体中，其中：

outputs[0].index　为对应 output 的 node 索引。

outputs[0].buf　为存放推理结果的 buf 指针。

outputs[0].size　为推理结果的 size 大小。

另外，该 output[0]推理结果的其他属性可以通过 rknn_query 查询得到。这边需要注意的是，outputs[0].buf 在 rknn_outputs_release 函数被调用时会自动释放，因此不需要用户主动释放。

is_prealloc = TRUE：如果用户有预先分配 output 的 buffer，则需要将预分配标志 outputs[0].is_prealloc 设为 TRUE，同时 outputs[0]结构体的其余成员变量也需要赋

值，如下：

```
rknn_output outputs[1];
outputs[0].want_float = TRUE;
outputs[0].is_prealloc = TRUE;
outputs[0].index = 0;
outputs[0].buf = output0_buf;
outputs[0].size = output0_attr.n_elems * sizeof(float);
ret = rknn_outputs_get(ctx, 1, outputs, NULL);
if(ret < 0) {
    printf("rknn_outputs_get fail! ret=%d\n", ret);
    goto Error;
}
```

其中：

outputs[0].index 为对应 output 的 node 索引。用户需要指定获取的 output 的 index,该 index 需小于 rknn 模型的 output 个数(rknn 模型的 output 个数可由 rknn_query 查询得到)。

outputs[0].buf 为存放推理结果的 buf 指针。该 buf 需要用户提前创建好。

outputs[0].size 为推理结果的 size 大小。也就是用户创建 buf 的 size。该 size 需要根据相应的 output 的属性以及 want_float 变量来计算。

当 want_float 为 FALSE 时，该 size 等于 output0_attr.size;

当 want_float 为 TRUE 时,该 size 等于 output0_attr.n_elems*sizeof(float)。

(output0_attr 为通过 rknn_query 查询 index 为 0 的 output 属性)

rknn_outputs_get 函数返回后，相应 index 的推理结果会存储在用户创建的 output[0].buf 里，由于该 buf 是用户创建的，在不需要再使用时，用户需主动释放以避免内存泄漏。

7）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，否则会造成内存泄漏。代码如下：

```
rknn_outputs_release(ctx, 1, outputs);
```

该函数的传参方式与 rknn_outputs_get 类似。

需要注意的是,不管 rknn_outputs_get 传入的 rknn_output[x].is_prealloc 是 TRUE 还是 FALSE 都需要调用该函数对 output 进行最终的释放。

8） 需要进行多次推理，可跳回步骤 4 进行下一次推理。

9） 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

或 Android 目录下的：

rk_ssd_demo/app/src/main/jni/ssd_image.cc

4 Demo 使用说明

4.1 Linux Demo

4.1.1 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。

注：目前该 Demo 只适用于 64 位的 RK3399Pro Linux 系统，同时只提供 64 位的 rknn_api 库，在 Ubuntu16.04 64 位系统上验证通过。

4.1.2 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。

4.2 Android Demo

4.2.1 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）。

4.2.2 运行说明

直接在 RK3399Pro 的 Android 上运行该 apk 即可。（该 Demo 需要有板载摄像头或外接的 USB 摄像头支持）

5 附录

5.1 API 迁移说明

5.1.1 v0.9.1 到 v0.9.2

由于从 v0.9.1 到 v0.9.2 的 API 变化较大，用户可以根据以下迁移步骤以及上述的 API 说明进行代码的迁移，大致步骤如下：

1） 由于 context 句柄的定义由原先的 int 类型修改为 rknn_context 类型，因此 context 变量的定义以及 rknn_init 的使用略有变化，代码可由原先的：

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

...

if(ctx >= 0)        rknn_destroy(ctx);
```

修改为：

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

...

if(ctx)             rknn_destroy(ctx);
```

注： 红色为变化的部分。

2） 由于 rknn_input_set 函数需要支持除 INT8 以外的数据类型和格式，因此函数定义也进行了调整，代码可由原先的：

```
    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;
    }
```

修改为：

```
    rknn_input inputs[1];
    inputs[0].index = input_index;
    inputs[0].buf = img.data;
    inputs[0].size = img_width * img_height * img_channels;
    inputs[0].pass_through = false;
    inputs[0].type = RKNN_TENSOR_UINT8;
    inputs[0].fmt = RKNN_TENSOR_NHWC;
    ret = rknn_inputs_set(ctx, 1, inputs);
    if(ret < 0) {
        printf("rknn_input_set fail! ret=%d\n", ret);
        goto Error;
    }
```

注： 红色为变化的部分。 另外，原先代码的 RKNN_INPUT_ORDER_012 参数已经不需要进行设置，rknn_inputs_set 函数也多了个 s。

3） v0.9.2 里将 rknn_outputs_get 和 rknn_output_to_float 函数进行了合并，并增加了新的内存使用方式，因此变化较大，代码可由原先的：

```
    int h_output = -1;
    struct rknn_output outputs[2];
    h_output = rknn_outputs_get(ctx, 2, outputs, nullptr);
    if(h_output < 0) {
        printf("rknn_outputs_get fail! ret=%d\n", ret);
        goto Error;
    }
```

修改为：

```
    rknn_output outputs[2];
    outputs[0].want_float = true;
    outputs[0].is_prealloc = false;
    outputs[1].want_float = true;
    outputs[1].is_prealloc = false;
    ret = rknn_outputs_get(ctx, 2, outputs, nullptr);
    if(ret < 0) {
        printf("rknn_outputs_get fail! ret=%d\n", ret);
        goto Error;
    }
```

注： 红色为变化的部分。 上述的 outputs 个数以 2 为例，其他模型可根据实际情况进行修改。

4） 由于 rknn_outputs_get 已经合并了 rknn_output_to_float 函数的功能（也就是上面的 want_float 标志），因此原先的 rknn_output_to_float 调用步骤可以去掉，代码可由原先的：

```
    float *predictions = (float*)(outputs[0].buf);
    if(outputs_attr[0].type != RKNN_TENSOR_FLOAT32) {
        predictions = (float*)malloc(output_size1);
        rknn_output_to_float(ctx, outputs[0], (void*)predictions,
output_size1);
    }
    float *outputClasses = (float*)(outputs[1].buf);
    if(outputs_attr[1].type != RKNN_TENSOR_FLOAT32) {
        outputClasses = (float*)malloc(output_size2);
        rknn_output_to_float(ctx, outputs[1], (void*)outputClasses,
output_size2);
    }

    ...

    if(outputs_attr[0].type != RKNN_TENSOR_FLOAT32) {
        free(predictions);
    }
    if(outputs_attr[1].type != RKNN_TENSOR_FLOAT32) {
        free(outputClasses);
    }
```

修改为：

```
    float *predictions = (float*)(outputs[0].buf);
    float *outputClasses = (float*)(outputs[1].buf);
```

注： 红色为变化的部分。 上述以 MobileNet-SSD 的后处理为例，其他模型可根据实际情况进行修改。

5） 因为上述 rknn_outputs_get 调用时设置了 want_float 为 true，所以 outputs[x].size 大值和通过 rknn_query 查询到的 outputs_attr[x].size 的值可能不一致，因此判断 outputs[x].size 与查询的属性是否一致的判断条件也需要修改，代码可由原先的：

```
    // Process output
    if(outputs[0].size == outputs_attr[0].size && outputs[1].size ==
outputs_attr[1].size)
    {
        ...
    }
```

修改为：

```
    // Process output
    if(outputs[0].size == outputs_attr[0].n_elems*sizeof(float) &&
outputs[1].size == outputs_attr[1].n_elems*sizeof(float))
    {
        ...
    }
```

注： 红色为变化的部分。 上述的 outputs 个数以 2 为例，其他模型可根据实际情况进行修改。

6） rknn_outputs_release 的使用方式也做了些调整，其传参方式保持与 rknn_outputs_get 一致（h_output 无需再创建），因此代码可由原先的：

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

修改为：

```
rknn_outputs_release(ctx, 2, outputs);
```

注： 红色为变化的部分。