

RK1808_RKNN_SDK 开发指南

V1.2.0

Toybrick Confidential

目 录

1	主要功能说明.....	4
2	系统依赖说明.....	4
3	使用说明.....	4
3.1	RKNN SDK 开发流程	4
3.2	RKNN C API.....	5
3.2.1	RK1808 RKNN API 库	5
3.2.2	EXAMPLE 使用说明.....	5
3.2.3	API 详细说明	6
3.2.3.1	<i>rknn_init</i>	6
3.2.3.2	<i>rknn_destroy</i>	7
3.2.3.3	<i>rknn_query</i>	7
3.2.3.4	<i>rknn_inputs_set</i>	9
3.2.3.5	<i>rknn_run</i>	10
3.2.3.6	<i>rknn_outputs_get</i>	11
3.2.3.7	<i>rknn_outputs_release</i>	12
3.2.4	RKNN 数据结构定义	12
3.2.4.1	<i>rknn_input_output_num</i>	12
3.2.4.2	<i>rknn_tensor_attr</i>	13
3.2.4.3	<i>rknn_input</i>	14
3.2.4.4	<i>rknn_output</i>	14
3.2.4.5	<i>rknn_perf_detail</i>	15
3.2.4.6	<i>rknn_sdk_version</i>	15
3.2.5	RKNN 返回值错误码	15
3.3	RKNN PYTHON API	16

3.3.1	RKNN PYTHON API 开启.....	16
3.3.2	EXAMPLE 使用说明.....	17
3.3.3	API 详细说明	17
3.3.3.1	<i>RKNN create and release</i>	17
3.3.3.2	<i>load_rknn</i>	18
3.3.3.3	<i>init_runtime</i>	18
3.3.3.4	<i>inference</i>	19
3.3.3.5	<i>eval_perf</i>	19

1 主要功能说明

RK1808 RKNN SDK 为 RK1808 平台 NPU 提供编程接口，能够帮助用户在 RK1808 Linux 平台上部署运行 RKNN-Toolkit 导出的 RKNN 模型，客户能够基于 RKNN SDK 快速开发基于 RK1808 平台使用 NPU 加速的 AI 应用。本 SDK 主要包括以下两部分：

- 1) RKNN C API: 支持 C/C++编程，提供 RKNN 模型的加载、运行、数据输入输出以及调试等接口。
- 2) RKNN Python API: 支持 Python3 编程，提供 RKNN 模型的加载、运行、数据输入输出以及调试等接口。

2 系统依赖说明

RKNN SDK 所需依赖如下：

- 1) RK1808 RKNN SDK 所包含的库支持运行于 RK1808 Linux AArch64 位平台。
- 2) RK1808 RKNN Python SDK 需要依赖 Python3.6 和 python-numpy 包。

3 使用说明

3.1 RKNN SDK 开发流程

在使用 RK1808 RKNN SDK 之前，用户首先需要在 PC 上使用 RKNN-Toolkit 工具将用户的模型转换为 RKNN 模型，用户可以在<rk1808-linux-sdk>/external/rknn-toolkit/目录获取工具的完整安装包，RKNN-Toolkit 的使用请参考《RKNN-Toolkit 使用指南》文档。

成功转换生成 RKNN 模型之后，用户可以先通过 RKNN-Toolkit 连接 RK1808 开发板进行联机调试，确保模型的精度性能符合要求。

得到 RKNN 模型文件之后，用户可以选择使用 C 或 Python 接口在 RK1808 平台开发应用，

后续章节将说明如何在 RK1808 平台上基于 RKNN SDK 进行开发。

3.2 RKNN C API

3.2.1 RK1808 RKNN API 库

RK1808 RKNN SDK 所提供的库和头文件位于 `<rk1808-linux-sdk>/external/rknpu/rknn/rknn_api/librknn_api` 目录下，开发者可以在自己应用中引用即可开发应用。

需要注意的是，RK1808 和 RK3399Pro 平台的 RKNN API 是兼容的，两者开发的应用程序可以很方便地移植。但是使用过程中需要注意要区分两个平台的 `librknn_api.so`，如果开发者使用 RK3399Pro 的 `librknn_api.so` 将无法在 RK1808 平台上运行。开发者可以使用以下方法来区分 `librknn_api.so` 的平台：

```
$ strings librknn_api.so |grep version
RK1808 librknn_api version 0.9.8 (58c7577 build: 2019-06-05 17:00:44)
```

3.2.2 EXAMPLE 使用说明

SDK 提供了 Linux 平台的 MobileNet 图像分类、MobileNet SSD 目标检测以及 Yolo v3 目标检测 Demo。这些 Demo 能够为客户基于 RKNN SDK 开发自己的 AI 应用提供参考。Demo 代码位于 `<rk1808-linux-sdk>/external/rknpu/rknn/rknn_api/examples` 目录。下面以 `rknn_mobilenet_demo` 为例来讲解如何快速上手运行。

1) 编译 Demo

```
cd examples/rknn_mobilenet_demo
mkdir build && cd build
cmake ..
make && make install
cd -
```

2) 部署到 RK1808 设备

```
adb push install/rknn_mobilenet_demo /userdata/
```

3) 运行 Demo

```
adb shell
cd /userdata/rknn_mobilenet_demo/
./rknn_mobilenet_demo mobilenet_v1.rknn dog_224x224.jpg
```

3.2.3 API 详细说明

3.2.3.1 rknn_init

rknn_init 初始化函数将创建 rknn_context 对象、加载 RKNN 模型以及根据 flag 执行特定的初始化行为。

API	rknn_init
功能	初始化 rknn
参数	rknn_context *context: rknn_context 指针。函数调用之后，context 将会被赋值。
	void *model: RKNN 模型的二进制数据。
	uint32_t size: 模型大小。
	uint32_t flag: 特定的初始化标志。目前 RK1808 平台仅支持以下标志： 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 rknn_run 的运行时间将会变长。
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);
```

3.2.3.2 rknn_destroy

rknn_destroy 函数将释放传入的 rknn_context 及其相关资源。

API	rknn_destroy
功能	销毁 rknn_context 对象及其相关资源。
参数	rknn_context context: 要销毁的 rknn_context 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);
```

3.2.3.3 rknn_query

rknn_query 函数能够查询获取到模型输入输出、运行时间以及 SDK 版本等信息。

API	rknn_query
功能	查询模型与 SDK 的相关信息。
参数	rknn_context context: rknn_context 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	查询输入输出 Tensor 个数

RKNN_QUERY_INPUT_ATTR	rknn_tensor_attr	查询输入 Tensor 属性
RKNN_QUERY_OUTPUT_ATTR	rknn_tensor_attr	查询输出 Tensor 属性
RKNN_QUERY_PERF_DETAIL	rknn_perf_detail	查询网络各层运行时间
RKNN_QUERY_SDK_VERSION	rknn_sdk_version	查询 SDK 版本

接下来的将依次详解各个查询命令如何使用。

1) 查询输入输出 Tensor 个数

传入 RKNN_QUERY_IN_OUT_NUM 命令可以查询模型输入输出 Tensor 的个数。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);
```

2) 查询输入 Tensor 属性

传入 RKNN_QUERY_INPUT_ATTR 命令可以查询模型输入 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) 查询输出 Tensor 属性

传入 RKNN_QUERY_OUTPUT_ATTR 命令可以查询模型输出 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));
}
```

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_run` 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。

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);
```

注意，用户不需要释放 `rknn_perf_detail` 中的 `perf_data`，SDK 会自动管理该 Buffer 内存。

5) 查询 SDK 版本

传入 `RKNN_QUERY_SDK_VERSION` 命令可以查询 RKNN SDK 的版本信息。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("sdk api version: %s\n", version.api_version);
printf("driver version: %s\n", version.driv_version);
```

3.2.3.4 rknn_inputs_set

通过 `rknn_inputs_set` 函数可以设置模型的输入数据。该函数能够支持多个输入，其中每个输入是 `rknn_input` 结构体对象，在传入之前用户需要设置该对象。

API	rknn_inputs_set
功能	设置模型输入数据。
参数	rknn_context context: rknn_context 对象。
	uint32_t n_inputs: 输入数据个数。
	rknn_input inputs[]: 输入数据数组，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].fmt = RKNN_TENSOR_NHWC;
inputs[0].buf = in_data;

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

3.2.3.5 rknn_run

rknn_run 函数将执行一次模型推理，调用之前需要先通过 rknn_inputs_set 函数设置输入数据。

API	rknn_run
功能	执行一次模型推理。
参数	rknn_context context: rknn_context 对象。
	rknn_run_extend* extend: 保留扩展，当前没有使用，传入 NULL 即可。
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);
```

3.2.3.6 rknn_outputs_get

`rknn_outputs_get` 函数可以获取模型推理的输出数据。该函数能够一次获取多个输出数据。其中每个输出是 `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。

对于输出数据的 `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 1，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 0 即可，函数执行之后 `buf` 将指向输出数据。

API	<code>rknn_outputs_get</code>
功能	获取模型推理输出
参数	<code>rknn_context context</code> : <code>rknn_context</code> 对象。
	<code>uint32_t n_outputs</code> : 输出数据个数
	<code>rknn_output outputs[]</code> : 输出数据的数组，其中数组每个元素为 <code>rknn_output</code> 结构体对象，代表模型的一个输出。
	<code>rknn_output_extend* extend</code> : 保留扩展，当前没有使用，传入 <code>NULL</code> 即可。
返回值	<code>int</code> 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 = 1;
}
ret = rknn_outputs_get(ctx, io_num.n_output, outputs, NULL);
```

3.2.3.7 rknn_outputs_release

rknn_outputs_release 函数将释放之前获取得到的输出的相关资源。

API	rknn_outputs_release
功能	释放 rknn_output 对象
参数	rknn_context context: rknn_context 对象
	uint32_t n_outputs: 输出数据个数
	rknn_output outputs[]: 要销毁的 rknn_output 数组
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);
```

3.2.4 RKNN 数据结构定义

3.2.4.1 rknn_input_output_num

结构体 rknn_input_output_num 表示输入输出 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	输入 Tensor 个数
n_output	uint32_t	输出 Tensor 个数

3.2.4.2 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	表示输入输出 Tensor 的索引位置。
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。

3.2.4.3 rknn_input

结构体 `rknn_input` 表示模型的一个数据输入，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	该输入的索引位置。
buf	void*	输入数据 Buffer 的指针。
size	uint32_t	输入数据 Buffer 所占内存大小。
pass_through	uint8_t	设置为 1 时会将 buf 存放的输入数据直接设置给模型的输入节点，不做任何预处理。
type	rknn_tensor_type	输入数据的类型。
fmt	rknn_tensor_format	输入数据的格式。

3.2.4.4 rknn_output

结构体 `rknn_output` 表示模型的一个数据输出，用来作为参数传入给 `rknn_outputs_get` 函数，在函数执行后，结构体对象将会被赋值。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	标识是否需要将输出数据转为 float 类型输出。
is_prealloc	uint8_t	标识存放输出数据的 Buffer 是否是预分配。
index	uint32_t	该输出的索引位置。
buf	void*	输出数据 Buffer 的指针。
size	uint32_t	输出数据 Buffer 所占内存大小。

3.2.4.5 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。

3.2.4.6 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[]	SDK 的版本信息。
drv_version	char[]	SDK 所基于的驱动版本信息。

3.2.5 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 PYTHON API

3.3.1 RKNN PYTHON API 开启

RK1808 Linux SDK 中默认没有开启 RKNN Python API，如需使用需要添加相应配置。

可以直接通过修改 `buildroot/configs/rockchip_rk1808_defconfig` 文件来添加配置，修改后重新编译固件。

```
diff --git a/configs/rockchip_rk1808_defconfig
b/configs/rockchip_rk1808_defconfig
index 1659453be8..fb5303bd00 100644
--- a/configs/rockchip_rk1808_defconfig
+++ b/configs/rockchip_rk1808_defconfig
@@ -19,3 +19,15 @@ BR2_PACKAGE_MINIGUI_ENABLE_PNG=y
 BR2_PACKAGE_RKNPU=y
 BR2_PACKAGE_RKNN_DEMO=y
 BR2_PACKAGE_RKWIFIBT_BTUART="ttyS4"
+
+BR2_PACKAGE_PYTHON3=y
+BR2_PACKAGE_PYTHON3_PY_PYC=y
+BR2_PACKAGE_PYTHON_NUMPY_ARCH_SUPPORTS=y
+BR2_PACKAGE_PYTHON_NUMPY=y
+BR2_PACKAGE_OPENCV3=y
+BR2_PACKAGE_OPENCV3_LIB_PYTHON=y
+BR2_PACKAGE_OPENCV3_LIB_IMGCODECS=y
+BR2_PACKAGE_OPENCV3_LIB_IMGPROC=y
+BR2_PACKAGE_OPENCV3_WITH_JPEG=y
+
+BR2_PACKAGE_PYTHON_RKNN=y
```

或者也可以通过 `make menuconfig` 命令将上述的配置一一打开，开启后同样需要重新编译固件。

编译完成并重新烧写固件后，可以在 RK1808 设备的 shell 命令行中执行以下命令确认是否正常开启：

```
# python
>>> from rknn.api import RKNN
```

3.3.2 EXAMPLE 使用说明

RKNN 的 Python 示例代码位于<rk1808_linux_sdk>/external/rknpu/rknn/python/example 目录，下面以 mobilenet_v1 为例说明如何运行：

- 1) 将 example 推送到 RK1808 设备

```
adb push mobilenet_v1 /userdata/
```

- 2) 运行

```
adb shell
cd /userdata/mobilenet_v1
python test.py
```

执行成功后输出如下所示：

```
/userdata/mobilenet_v1 # python test.py
--> Loading RKNN model
done
--> Running model
mobilenet_v1
-----TOP 5-----
[156]: 0.8232421875
[155]: 0.0513916015625
[205]: 0.0250244140625
[188]: 0.0225830078125
[880]: 0.01351165771484375
done
```

3.3.3 API 详细说明

3.3.3.1 RKNN create and release

在使用 RKNN 的 Python API 接口时，都需要先初始化一个 RKNN 对象，并在用完后调用该对象

的 `release()`方法进行释放。

示例代码如下：

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

3.3.3.2 load_rknn

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

示例代码如下：

```
ret = rknn.load(path='./mobilenet_v1.rknn')
if ret != 0:
    print('Load rknn model failed!')
    exit(ret)
```

3.3.3.3 init_runtime

在模型推理或性能评估之前，必须先初始化运行时环境，并配置所需参数。

API	init_runtime
功能	初始化运行时环境。
参数	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。

示例代码如下：

```
ret = rknn.init_runtime()
if ret != 0:
    print('Init runtime environment failed')
    exit(ret)
```

3.3.3.4 inference

`inference` 接口可以进行模型推理。注意在推理前，必须先调用 `load_rknn` 接口加载 RKNN 模型。

API	<code>inference</code>
功能	使用模型对指定的输入进行推理，得到推理结果。
参数	<code>inputs</code> : 待推理的输入数据，如经过 <code>cv2</code> 处理的图片。格式是 <code>ndarray list</code> 。
	<code>data_type</code> : 输入数据的类型，可填以下值：'float32', 'float16', 'int8', 'uint8', 'ing16'。默认值为 'uint8'。
	<code>data_format</code> : 数据排列格式，可以填以下值：“nchw”, “nhwc”。默认值为 'nhwc'。
	<code>outputs</code> : 指定输出数据的格式，格式是 <code>ndarray list</code> ，用于指定输出的 <code>shape</code> 和 <code>dtype</code> 。默认值为 <code>None</code> ，此时返回的数据 <code>dtype</code> 为 <code>float32</code> 。
返回值	<code>results</code> : 推理结果，类型为 <code>ndarray list</code> 。

示例代码如下：

```
img = cv2.imread('./dog_224x224.jpg')
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
outputs = rknn.inference(inputs=[img])
```

3.3.3.5 eval_perf

API	<code>eval_perf</code>
功能	评估模型性能。 如果初始化运行环境时设置 <code>perf_debug</code> 为 <code>False</code> ，则获得的是模型在硬件上运行的总

	时间; 如果设置 <code>perf_debug</code> 为 <code>True</code> , 除了返回总时间外, 还将返回每一层的耗时情况。
参数	<code>inputs</code> : 输入数据, 如经过 <code>cv2</code> 处理得图片。格式是 <code>ndarray list</code> 。
	<code>data_type</code> : 输入数据的类型, 可填以下值: <code>'float32'</code> , <code>'float16'</code> , <code>'int8'</code> , <code>'uint8'</code> , <code>'int16'</code> 。默认值为 <code>'uint8'</code> 。
	<code>data_format</code> : 数据排列, 可以填以下值: <code>"nchw"</code> , <code>"nhwc"</code> 。默认值为 <code>'nhwc'</code> 。
	<code>is_print</code> : 是否以规范格式输出性能评估结果。默认值为 <code>True</code> 。
返回值	<code>perf_result</code> : 性能信息。类型为字典。

示例代码如下:

```
rknn.eval_perf(inputs=[img])
```