Classification Level: Top secret ( ) Secret ( ) Internal ( ) Public (√)

RKNN-Toolkit2 User Guide

(Technology Department, Graphic Computing Platform Center)

Mark:	Version	V1.1.0
[ ] Editing	Author	HPC
[√] Released	Completed Date	2021-6-30
	Reviewer	Vincent
	Reviewed Date	2021-6-30

瑞芯微电子股份有限公司

Rockchip Electronics Co., Ltd

Revision History

Version	Modifier	Date	Modify description	Reviewer
V0.5.0	HPC	2021-1-13	Initial version	Vincent
V0.6.0	HPC	2021-2-24	1. Update Hybrid Quantization 2. Update RKNN initialization 3. Update Caffe load API	Vincent
V0.7.0	HPC	2021-3-30	1. Update Mmse support 2. Update some interface 3. Update explanation and suggestion of quantization algorithm	Vincent
V1.0.0	HPC	2021-4-21	1. Update config (remove reorder_channel， custom_string add quant_img_RGB2BGR, custom_string) 2. add list_devices and get_sdk_version description 3. add eval_perf and eval_memory description	Vincent
V1.1.0	HPC	2021-6-30	1. Update Hybrid Quantization interface 2. Update MMSE interface 3. Update Quantitative Accuracy Analysis interface 4. Add Accuracy Troubleshooting chapter	Vincent

Version	Modifier	Date	Modify description	Reviewer

Table of Contents

1 Overview

1.1 Main function description

RKNN-Toolkit2 is a development kit that provides users with model conversion, inference and performance evaluation on PC and Rockchip NPU platforms. Users can easily complete the following functions through the Python interface provided by the tool:

1） Model conversion: support to convert Caffe / TensorFlow / TensorFlow Lite / ONNX / Darknet / PyTorch model to RKNN model, support RKNN model import/export, which can be used on Rockchip NPU platform later.

2） Quantization: support to convert float model to quantization model, currently support quantized methods including asymmetric quantization (asymmetric_quantized-8, asymmetric_quantized-16). and support hybrid quantization. Asymmetric_quantized-16 not supported yet.

3） Model inference: Able to simulate Rockchip NPU to run RKNN model on PC and get the inference result. This tool can also distribute the RKNN model to the specified NPU device to run, and get the inference results.

4） Performance evaluation: distribute the RKNN model to the specified NPU device to run, and evaluate the model performance in the actual device.

5） Memory evaluation: Evaluate memory consumption at runtime of the model. When using this function, the RKNN model must be distributed to the NPU device to run, and then call the relevant interface to obtain memory information.

6） Quantitative error analysis: This function will give the Euclidean or cosine distance of each layer of inference results before and after the model is quantized. This can be used to analyze how quantitative error occurs, and provide ideas for improving the accuracy of quantitative models.

Note: Some features are limited by the operating system or chip platform and cannot be used on some operating systems or platforms. The feature support list of each operating system (platform) is as follows:

	Ubuntu 18.04	Windows 7/10	Debian 9.8 / 10 (ARM 64)	MacOS Mojave / Catalina
Model conversion	yes			
Quantization	yes			
Model inference	yes			
Performance evaluation	yes			
Memory evaluation	yes			
Multiple inputs	yes			
Batch inference	yes(part)			
List devices	yes			
Query SDK version	yes			
Quantitative error analysis	yes			
Visualization				
Model optimization level	yes			

1.2 Applicable chip model

- RK3566
- RK3568

1.3 Applicable Operating System

RKNN Toolkit2 is a cross-platform development kit. The supported operating systems are as follows:

- Ubuntu: 18.04 (x64) or later

2 Requirements/Dependencies

It is recommended to meet the following requirements in the operating system environment:

Table 1 Operating system environment

Operating system version	Ubuntu18.04(x64)or later
Python version	3.6
Python library dependencies	numpy==1.16.6 onnx==1.7.0 onnxoptimizer==0.1.0 onnxruntime==1.6.0 tensorflow==1.14.0 tensorboard==1.14.0 protobuf==3.12.0 torch==1.6.0 torchvision==0.7.0 psutil==5.6.2 ruamel.yaml==0.15.81 scipy==1.2.1 tqdm==4.27.0 requests==2.21.0 tflite==2.3.0 opencv-python==4.4.0.46 PuLP==2.4 scikit_image==0.17.2

Note:

1. This document mainly uses Ubuntu 18.04 / Python3.6 as an example. For other operating systems, please refer to the corresponding quick start guide: <Rockchip_Quick_Start_RKNN_Toolkit2_EN.pdf>.

3 User Guide

3.1 Installation

There are two ways to install RKNN-Toolkit2: one is through the Python package installation and management tool pip, the other is running docker image with full RKNN-Toolkit2 environment. The specific steps of the two installation ways are described below.

3.1.1 Install by pip command

1. Create virtualenv environment. If there are multiple versions of the Python environment in the system, it is recommended to use virtualenv to manage the Python environment.

```
sudo apt install virtualenv
sudo apt-get install python3 python3-dev python3-pip
sudo apt-get install libxslt1-dev zlib1g zlib1g-dev libglib2.0-0 \
libsm6 libgl1-mesa-glx libprotobuf-dev gcc

virtualenv -p /usr/bin/python3 venv
source venv/bin/activate
```

2. Install dependent libraries:

```
pip3 install -r doc/requirements.txt
```

Note: RKNN-Toolkit2 itself does not rely on opencv-python, but the example will use this library to load image, so the library is also installed here.

3. Install RKNN-Toolkit2

```
pip3 install package/rknn_toolkit2*.whl
```

Please select corresponding installation package (located at the *packages/* directory) according to different python versions and processor architectures:

- **Python3.6 for x86_64:** rknn_toolkit2-x.x.x-cp36-cp36m-linux_x86_64.whl

3.1.2 Install by the Docker Image

In docker folder, there is a Docker image that has been packaged for all development requirements, Users only need to load the image and can directly use RKNN-toolkit2, detailed steps are as follows:

1. Install Docker

   Please install Docker according to the official manual:

   https://docs.docker.com/install/linux/docker-ce/ubuntu/

2. Load Docker image

   Execute the following command to load Docker image:

```
docker load --input rknn-toolkit2-x.x.x-docker.tar.gz
```

After loading successfully, execute "docker images" command and the image of rknn-toolkit2 appears as follows:

```
REPOSITORY        TAG       IMAGE ID          CREATED        SIZE
rknn-toolkit2     x.x.x     4f6bae6686d8      1 hours ago    5.34GB
```

3. Run image

   Execute the following command to run the docker image. After running, it will enter the bash environment.

```
docker run -t -i --privileged -v /dev/bus/usb:/dev/bus/usb rknn-toolkit2:x.x.x /bin/bash
```

If you want to map your own code, you can add the "-v <host src folder>:<image dst folder>" parameter, for example:

```
docker run -t -i --privileged -v /dev/bus/usb:/dev/bus/usb -v /home/rk/test:/test
rknn-toolkit2:x.x.x /bin/bash
```

4. Run demo

```
cd /example/tflite/mobilenet_v1
python3 test.py
```

3.2 Usage of RKNN-Toolkit2

Next, the use process of RKNN Toolkit2 under each use scenario will be given in detail.

3.2.1 Scenario 1: Inference for Simulation on PC

In this scenario, RKNN Toolkit2 runs on the PC, and runs the model through the simulator.

Depending on the type of model, this scenario can be divided into two sub-scenarios: one scenario is that the model is a non-RKNN model, i.e. Caffe, TensorFlow, TensorFlow Lite, ONNX, DarkNet, PyTorch model, and the other scenario is that the model is an RKNN model which is a proprietary model of Rockchip with the file suffix "rknn".

Note: Simulator only supported on x86_64 Linux.

3.2.1.1 run the non-RKNN model

When running a non-RKNN model, the RKNN-Toolkit2 usage flow is shown below:

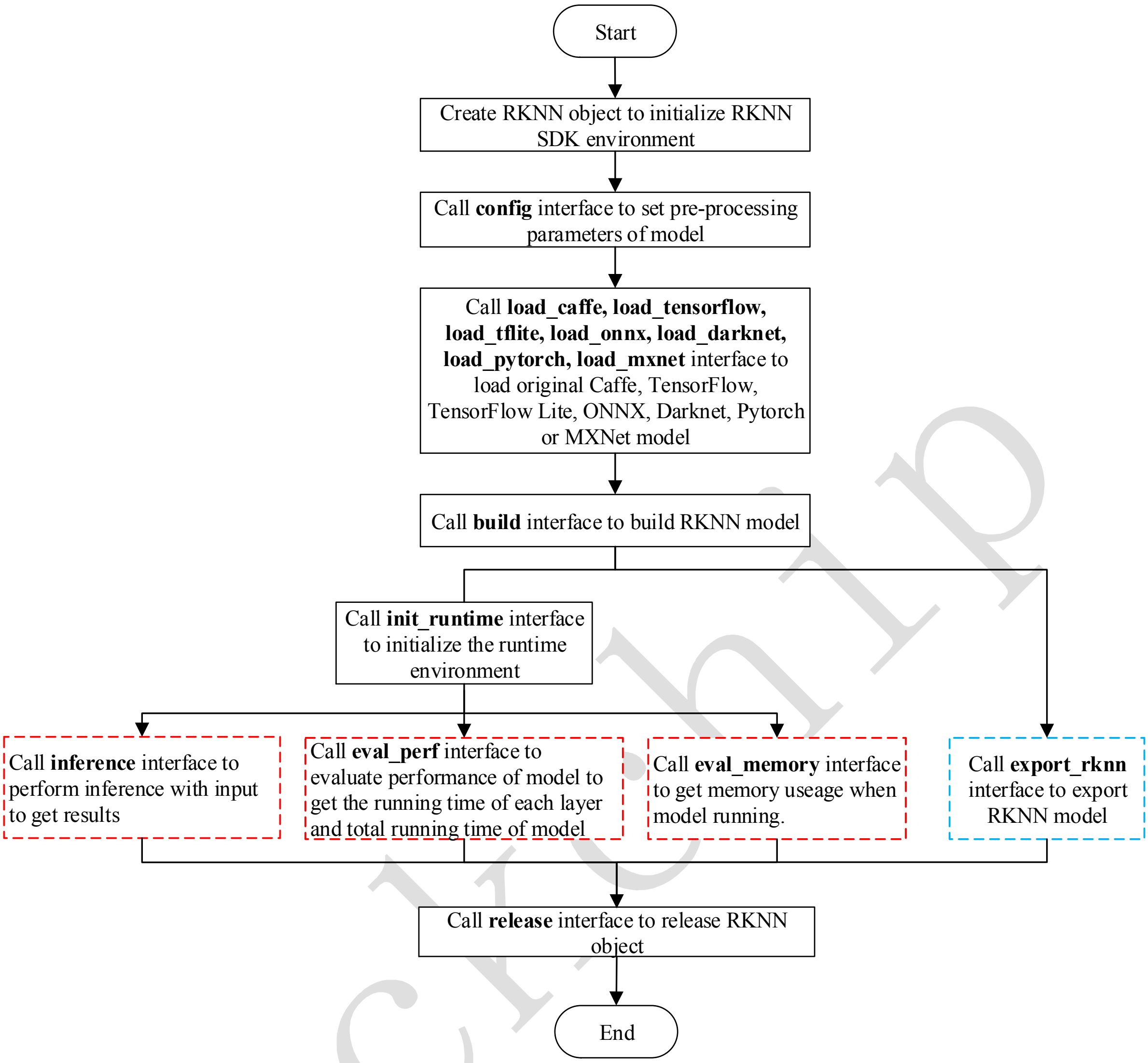

Figure 1 Usage flow of RKNN-Toolkit2 when running a non-RKNN model on PC

Note:

1. The above steps should be performed in order.

2.The model exporting step marked in the blue box is not necessary. If you exported, you can use load_rknn to load it later on.

3. The order of model inference, performance evaluation and memory evaluation steps marked in red box is not fixed, it depends on the actual demand.

4. Only when the target hardware platform is Rockchip NPU, we can call eval_perf / eval_memory interface.

3.2.2 Scenario 2: Run on Rockchip NPU connected to the PC.

Rockchip NPU platforms currently supported by RKNN Toolkit2 include RK3566 / RK3568.

In this Scenario, In this scenario, RKNN Toolkit2 runs on the PC and connects to the NPU device through the PC's USB. RKNN Toolkit2 transfers the RKNN model to the NPU device to run, and then obtains the inference results, performance information, etc. from the NPU device

First, we need to complete the following two steps:

1. Make sure the USB OTG of development board is connected to PC, and call list_devices interface will show the device. More information about "list_devices" interface can see Scction 3.5.15.

2. "Target" parameter and "device_id" parameter need to be specified when calling "init_runtime" interface to initialize the runtime environment, where "target" indicates the type of hardware, optional values are "rk3566" and "rk3568". When multiple devices are connected to PC, "device_id" parameter needs to be specified. It is a string which can be obtained by calling "list_devices" interface, for example:

```
all device(s) with adb mode:
VD46C3KM6N
```

Runtime initialization code is as follows:

```
# RK3566
ret = init_runtime(target='rk3366', device_id='VGEJY9PW7T')


# RK3568
ret = init_runtime(target='rk3568', device_id='515e9b401c060c0b')
```

3.2.2.1 run the non-RKNN model

If the model is a non-RKNN model (Caffe, TensorFlow, TensorFlow Lite, ONNX, DarkNet, PyTorch), the usage flow and precautions of RKNN-Toolkit2 are the same as the sub-scenario 1 of the scenario 1(see Section 3.2.1.1).

3.2.2.2 run the RKNN model

When running an RKNN model, users do not need to set model pre-processing parameters, nor do they need to build an RKNN model, the usage flow is shown in the following figure.

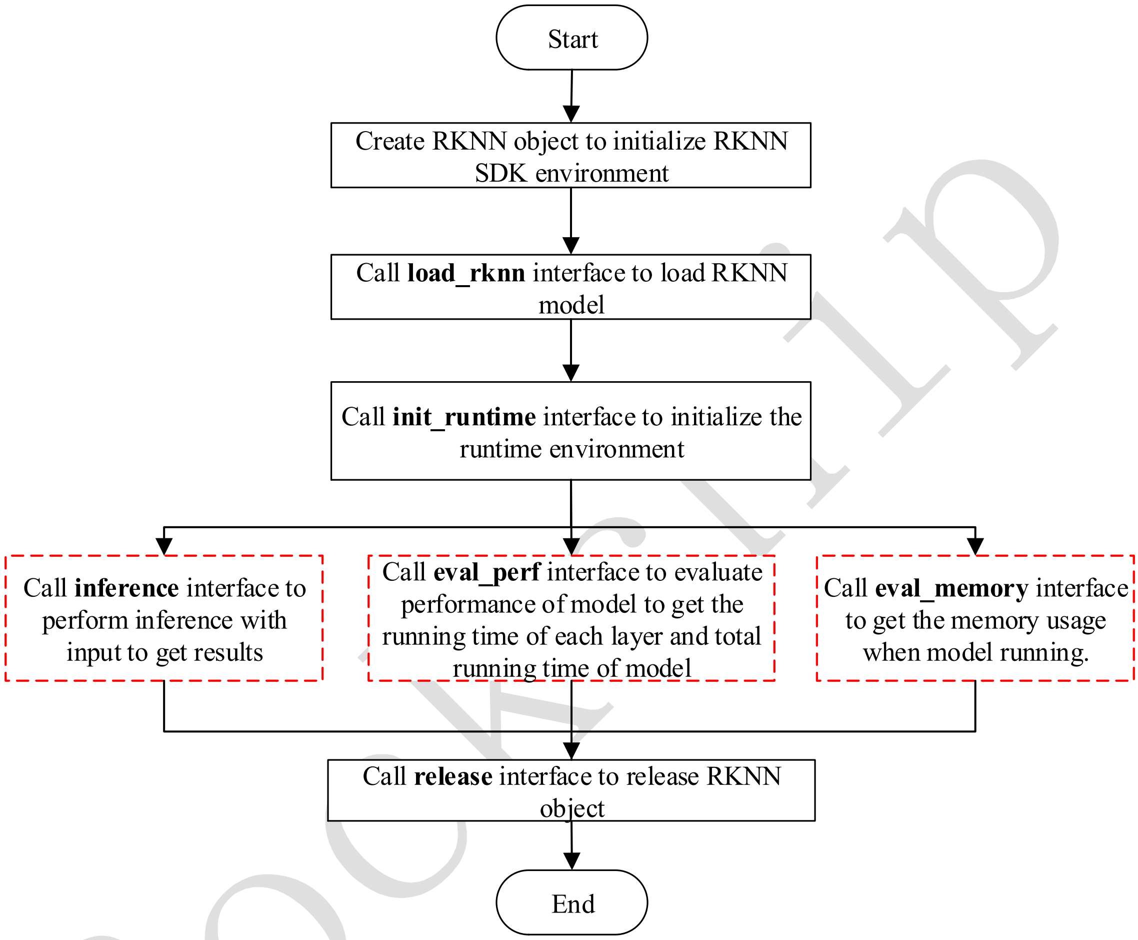

Figure 2 Usage flow of RKNN-Toolkit2 when running an RKNN model on PC

Note:

1. The above steps should be performed in order.

2. The order of model inference, performance evaluation and memory evaluation steps marked in red box is not fixed, it depends on the actual demand.

3. We can call inference / eval_perf / eval_memory only when the target is hardware platform.

4. The import method through load_rknn is only used for the use of hardware platform-related functions, and functions such as accuracy_analysis cannot be used.

3.2.3 Scenario 3: Inference on RK356x Linux development board

Not supported yet.

In this scenario, RKNN-Toolkit2 is installed in RK356x Linux system directly. The built or imported RKNN model runs directly on RK356x to obtain the actual inference results or performance information of the model.

For RK356x Linux development board, the usage flow of RKNN-Toolkit2 depends on the type of model. If the model is a non-RKNN model, the usage flow is the same as that in the sub-scenario 1 of scenario 1(see Section 3.2.1.1), otherwise, please refer to the usage flow in the sub-scenario 2 of scenario1(see Section 3.2.2.2).

3.3 Hybrid Quantization

The quantization feature can ensure the accuracy of model based on improved model inference speed. But for some models, the accuracy has dropped a bit. In order to better balance performance and accuracy, we add new feature hybrid quantization. Users can decide which layers to quantize or not manually, the quantization parameters also can been modified.

Note:

1. The examples/functions directory provides a hybrid quantization example named hybrid_quant. Users can refer to this example for hybrid quantification practice.

3.3.1 Instructions of hybrid quantization

Currently, RKNN Toolkit2 has three kind of ways to use hybrid quantization:

1. Convert quantized layer to non-quantized (e.g. float16) layer. Due to the low non-quantized computing power on the NPU, the inference speed will be reduced.

3.3.2 Hybrid quantization profile

When using the hybrid quantization feature, the first step is to generate a hybrid quantization profile, which is briefly described in this section.

When the hybrid quantization interface hybrid_quantization_step1 is called, a configuration file of {model_name}.quantization.cfg is generated in the current directory. The configuration file format is as follows:

```
custom_quantize_layers: {}
quantize_parameters:
    Preprocessor/sub:0:
        qtype: asymmetric_quantized
        qmethod: layer
        dtype: int8
        min:
        -    -1.0
        max:
        -    1.0
        scale:
        -    0.00784313725490196
        zero_point:
        -    0
        ori_min:
        -    -1.0
        ori_max:
        -    1.0
……
```

The first line of the body of the configuration file is a dictionary of customized quantize layers, add the layer names and their corresponding quantized type (choose from **float16 / int16**) to be changed to customized quantize layers. int16 not supported yet.

Next is the quantization parameter of each operand in the model, and each operand is a dictionary. The key of each dictionary is tensor_name, the value of dictionary is quantization parameter, if it is not quantized, the "dtype" value is float16.

3.3.3 Usage flow of hybrid quantization

When using the hybrid quantization function, it can be done in four steps.

Step1, load the original model and generate a quantize configuration file, a model structure file and a model weight bias file. The specific interface call process is as follows:

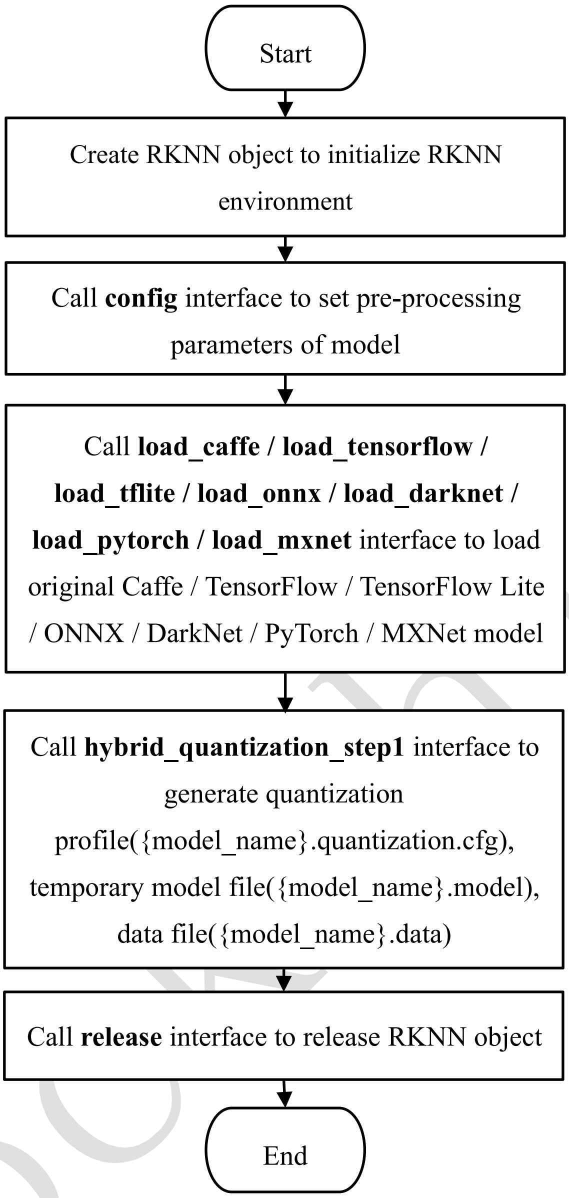

Figure 3 call process of hybrid quantization step 1

Step 2, Modify the quantization configuration file generated in the first step.

- If some quantization layers is changed to a non-quantization layer, find the output operand of layer that is not to be quantized, and add these operands name and float16 to custom_quantize_layers, such as "<operands name>: float16".

Step 3, generate hybrid quantized RKNN model. The specific interface call flow is as follows:

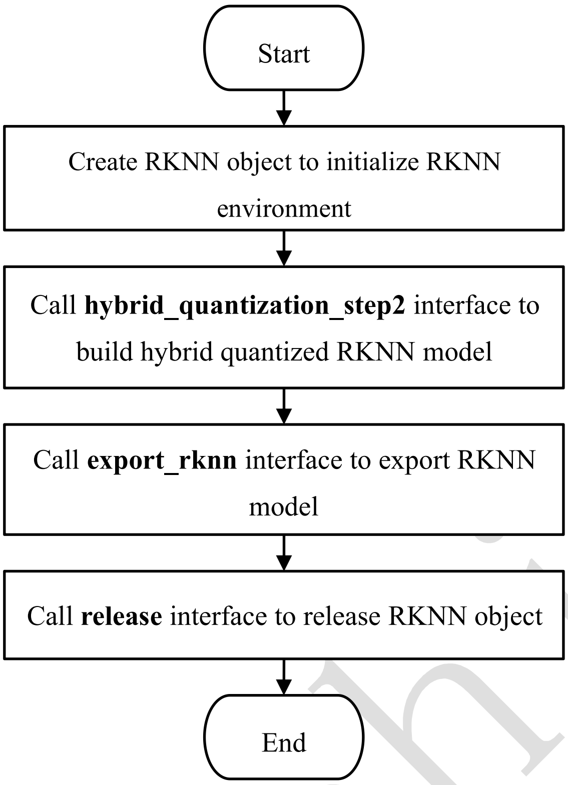

Figure 4 call process of hybrid quantization step 3

Step 4, use the RKNN model generated in the previous step to inference.

3.4 Example

The following is the sample code for loading TensorFlow Lite model (see the *example/tflite/mobilenet_v1* directory for details), if it is executed on PC, the RKNN model will run on the simulator.

```
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)


if __name__ == '__main__':

    # Create RKNN object
    rknn = RKNN()

    # pre-process config
    print('--> config model')
    rknn.config(mean_values=[128, 128, 128], std_values=[128, 128, 128])
    print('done')

    # Load tensorflow model
    print('--> Loading model')
    ret = rknn.load_tflite(model='mobilenet_v1_1.0_224.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)
    img = np.expand_dims(img, 0)

    # 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')

    rknn.release()
```

Where dataset.txt is a text file containing the path of the test image. For example, if a picture of dog_224x224.jpg in the *example/tflite/mobilenet_v1* directory, then the corresponding content in dataset.txt is as follows:

```
dog_224x224.jpg
```

When performing model inference, the result of this demo is as follows:

```
-----TOP 5-----
[156]: 0.8544921875
[155]: 0.080322265625
[205]: 0.0129241943359375
[284]: 0.0084075927734375
[194]: 0.0025787353515625
```

3.5 RKNN-Toolkit2 API description

3.5.1 RKNN object initialization and release

The initialization/release function group consists of API interfaces to initialize and release the RKNN object as needed. The **RKNN()** must be called before using all the API interfaces of RKNN-Toolkit2, and call the **release()** method to release the object when task finished.

When the RKNN object is initing, the users can set ***verbose*** and ***verbose_file*** parameters, used to show detailed log information of model loading, building and so on. The data type of verbose parameter is bool. If the value of this parameter is set to True, the RKNN Toolkit2 will show detailed log information on screen. The data type of verbose_file is string. If the value of this parameter is set to a file path, the detailed log information will be written to this file (**the verbose also need be set to True**).

The sample code is as follows:

```
# Show the detailed log information on screen, and saved to
# mobilenet_build.log
rknn = RKNN(verbose=True, verbose_file='./mobilenet_build.log')
# Only show the detailed log information on screen.
rknn = RKNN(verbose=True)
…
rknn.release()
```

3.5.2 RKNN model configuration

Before the RKNN model is built, the model needs to be configured first through the **config** interface.

API	**config**
Description	Set model parameters
Parameter	batch_size: The size of each batch of data sets. The default value is 100. When quantifying, the amount of data to imported in each batch will be determined according to this parameter to correct the quantization results.
	mean_values: The mean values of the input. The parameter format is a list. The list contains one or more mean sublists. The multi-input model corresponds to multiple sublists. The length of each sublist is consistent with the number of channels of the input. For example, if the parameter is [[128,128,128]], it means an input subtract 128 from the values of the three channels. If quant_img_RGB2BGR is set to True, the RGB2BGR conversion will be done first, and then the average value will be subtracted.
	std_values: The normalized value of the input. The parameter format is a list. The list contains one or more normalized value sublists. The multi-input model corresponds to multiple sublists. The length of each sublist is consistent with the number of channels of the input. For example, if the parameter is [[128,128,128]], it means the value of the three channels of an input minus the average value and then divide by 128. If quant_img_RGB2BGR is set to True, the RGB2BGR conversion will be performed first,

	followed by subtracting the mean value and dividing by the normalized value.
	epochs: Number of iterations in quantization. Quantization parameter calibration is performed with specified data at each iteration. Default value is -1, in this situation, the number of iteration is automatically calculated based on the amount of data in the dataset. Not support yet.
	quant_img_RGB2BGR: Indicates whether the RGB2BGR operation needs to be done first when loading the quantized image. The default value is False. If there are multiple inputs, the corresponding parameters for each input is split with ',', such as [True, True, False]. This configuration is generally used on the Caffe model. Most of the Caffe model training will perform RGB2BGR conversion on the dataset image firstly. At this time, the configuration needs to be set to True. In addition, this configuration is only valid for the quantized image format of jpg/jpeg/png/bmp. This configuration is ignored when the npy format is read. Therefore, when the model input is BGR, npy also needs to be in BGR format. **This configuration is only used to read the quantize image in the quantization stage (build interface) or in quantitative accuracy analysis (accuracy_analysis interface), and will not be recorded in the final RKNN model. Therefore, if the input of the model is BGR, you need to ensure that the imported image data is also in BGR format before calling the inference of the toolkit or the run function of the C-API.**
	quantized_dtype: Quantization type, the quantization types currently supported are asymmetric_quantized-8, asymmetric_quantized-16. The default value is asymmetric_quantized-8. asymmetric_quantized-16 is not supported yet.
	quantized_algorithm: The quantization algorithm used when calcaulating the quantization parameters of each layer. Currently support: **normal**, **mmse**. Default is **normal**. The characteristic of **normal** quantization algorithm is fast. The recommended

	quantization data is generally about 20-100 pieces. with more data, the accuracy may not be further improved. The **mmse** quantization algorithm is slower due to the violent iteration method, but usually has higher accuracy than **normal**. The recommended quantization data is generally about 20-50 pieces. Users can also increase or decrease the amount of data appropriately according to the length of the quantization time.
	quantized_method: Currently support layer or channel, That is each layer has only one set of quantization parameters or each channel of weight has its own set of quantization parameters. Usually the channel will be more accurate than the layer, default is layer.
	optimization_level: Model optimization level. By modifying the model optimization level, you can turn off some or all of the optimization rules used in the model conversion process. The default value of this parameter is 3, and all optimization options are turned on. When the value is 2 or 1, turn off some optimization options that may affect the accuracy of some models. Turn off all optimization options when the value is 0.
	target_platform: Specify which target chip platform the RKNN model is based on. RK3566 and RK3568 are currently supported.
	custom_string: Add custom string information to rknn model, then can query the information at runtime.
Return Value	None

The sample code is as follows:

```
# model config
rknn.config(mean_values=[[103.94, 116.78, 123.68]],
                    std_values=[[58.82, 58.82, 58.82]],
                    quant_img_RGB2BGR=True, target_platform='rk3566')
```

3.5.3 Loading non-RKNN model

RKNN-Toolkit2 currently supports load non-RKNN models of Caffe, TensorFlow, TensorFlow Lite, ONNX, DarkNet, PyTorch. There are different calling interfaces when loading models, the loading interfaces are described in detail below.

3.5.3.1 Loading Caffe model

API	**load_caffe**
Description	Load Caffe model
Parameter	model: The path of Caffe model structure file (suffixed with ".prototxt").
	proto: Caffe model format (valid value is 'caffe' or 'lstm_caffe'). Plaese use 'lstm_caffe' when the model is RNN model. 'lstm_caffe' is not supported yet.
	blobs: The path of Caffe model binary data file (suffixed with ".caffemodel"). The value can be None, RKNN Toolkit2 will randomly generate parameters such as weights.
	inputname: When the caffe model has multiple inputs, you can specify the order of the input layer names through this parameter, such as ['input1','input2','input3'],note that the name needs to be consistent with the model input name；It can also be set default. The sequence is automatically given by the caffe model file (file suffix with .prototxt).
Return Value	0: Import successfully
	-1: Import failed

The sample code is as follows:

```
# Load the mobilenet_v2 Caffe model in the current path
ret = rknn.load_caffe(model='./mobilenet_v2.prototxt',
                      proto='caffe',
                      blobs='./mobilenet_v2.caffemodel')
```

3.5.3.2 Loading TensorFlow model

API	**load_tensorflow**
Description	Load TensorFlow model
Parameter	tf_pb: The path of TensorFlow model file (suffixed with ".pb").
	inputs: The input node (layer name) of model, input with multiple nodes is supported now. All the input node string are placed in a list.
	input_size_list: The size and number of channels of the image corresponding to the input node. As in the example of mobilenet_v1 model, the input_size_list parameter should be set to [[224,224,3]].
	outputs: The output node (operand name) of model, output with multiple nodes is supported now. All the output nodes are placed in a list.
	predef_file: In order to support some controlling logic, a predefined file in npz format needs to be provided. This predefined fie can be generated by the following function call: np.savez('prd.npz', [placeholder name]=prd_value).If there are / in placeholder name, use # to replace. Not supported yet.
Return value	0: Import successfully
	-1: Import failed

The sample code is as follows:

```
# Load ssd_mobilenet_v1_coco_2017_11_17 TF model in the current path
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]])
```

3.5.3.3 Loading TensorFlow Lite model

API	**load_tflite**
Description	Load TensorFlow Lite model.
Parameter	model: The path of TensorFlow Lite model file (suffixed with ".tflite").
Return Value	0: Import successfully
	-1: Import failed

The sample code is as follows:

```
# Load the mobilenet_v1 TF-Lite model in the current path
ret = rknn.load_tflite(model = './mobilenet_v1.tflite')
```

3.5.3.4 Loading ONNX model

API	**load_onnx**
Description	Load ONNX model
Parameter	model: The path of ONNX model file (suffixed with ".onnx")
Return Value	0: Import successfully
	-1: Import failed

The sample code is as follows:

```
# Load the arcface onnx model in the current path
ret = rknn.load_onnx(model = './arcface.onnx')
```

3.5.3.5 Loading DarkNet model

API	**load_darknet**
Description	Load DarkNet model
Parameter	model: The path of DarkNet model structure file (suffixed with ".cfg").
	weight: The path of weight file (suffixed with ".weight").

Return Value	0: Import successfully
	-1: Import failed

The sample code is as follows:

```
# Load the yolov3-tiny DarkNet model in the current path
ret = rknn.load_darknet(model = './yolov3-tiny.cfg',
                        weight= './yolov3.weights')
```

3.5.3.6 Loading PyTorch model

API	**load_pytorch**
Description	Load PyTorch model
Parameter	model:The path of PyTorch model structure file (suffixed with ".pt"), and need a model in the torchscript format. Required.
	input_size_list:The size and number of channels of each input node. For example, [[1,1,224,224],[1,3,224,224]] means there are two inputs. One of the input shapes is [1,1, 224, 224], and the other input shape is [1,3, 224, 224]. Required.
Return Value	0: Import successfully
	-1: Import failed

The sample code is as follows:

```
# Load the PyTorch model resnet18 in the current path
ret = rknn. load_pytorch(model = './resnet18.pt',
                         input_size_list=[[1,3,224,224]])
```

3.5.3.7 Loading MXNet model

Not support yet.

API	**load_mxnet**
Description	Load MXNet model

Parameter	symbol:Network structure file of MXNet model, suffixed with "json". Required.
	params:Network parameters file of MXNet model, suffixed with "params". Required.
	input_size_list:The size and number of channels of each input node. For example, [[1,1,224,224],[1,3,224,224]] means there are two inputs. One of the input shapes is [1,1, 224, 224], and the other input shape is [1,3, 224, 224]. Required.
Return Value	0: Import successfully
	-1: Import failed

The sample code is as follows:

```
# Load the MXNet model resnext50 in the current path
ret = rknn.load_mxnet(symbol='resnext50_32x4d-symbol.json',
                      params='resnext50_32x4d-4ecf62e2.params',
                      input_size_list=[[1,3,224,224]])
```

3.5.4 Building RKNN model

API	**build**
Description	Build corresponding RKNN model according to imported model.
Parameter	do_quantization: Whether to quantize the model, optional values are True and False.
	dataset: A input data set for rectifying quantization parameters. Currently supports text file format, the user can place the path of picture(jpg or png) or npy file which is used for rectification. A file path for each line. Such as: a.jpg b.jpg or a.npy b.npy If there are multiple inputs, the corresponding files are divided by space. Such as:

	a.jpg a2.jpg b.jpg b2.jpg or a.npy a2.npy b.npy b2.npy Note: It is generally recommended to select the quantization image which is consistent with the prediction scene.
	rknn_batch_size: batch size of input, default is 1. If greater than 1, NPU can inference multiple frames of input image or input data in one inference. For example, original input of MobileNet is [1, 224, 224, 3], output shape is [1, 1001]. When rknn_batch_size is set to 4, the input shape of MobileNet becomes [4, 224, 224, 3], output shape becomes [4, 1001]. **Note:** **1. The adjustment of rknn_batch_size does not improve the performance of the general model on the NPU, but it will significantly increase memory consumption and increase the delay of single frame.** **2. The adjustment of rknn_batch_size can reduce the consumption of the ultra-small model on the CPU and improve the average frame rate of the ultra-small model. (Applicable to the model is too small, CPU overhead is greater than the NPU overhead)** **3. The value of rknn_batch_size is recommended to be less than 32, to avoid the memory usage is too large and the reasoning fails.** **4. After the rknn_batch_size is modified, the shape of input and output will be modified. So the inputs of inference should be set to correct size. It`s also needed to process the returned outputs on post processing.** Not support yet.
Return	0: Build successfully

value	-1: Build failed

The sample code is as follows:

```
# Build and quantize RKNN model
ret = rknn.build(do_quantization=True, dataset='./dataset.txt')
```

3.5.5 Export RKNN model

The RKNN model built by 'build' interface can be saved as a file, it can used to model deployment.

API	**export_rknn**
Description	Save RKNN model in the specified file (suffixed with ".rknn").
Parameter	export_path: The path of generated RKNN model file.
Return Value	0: Export successfully
	-1: Export failed

The sample code is as follows:

```
# save the built RKNN model as a mobilenet_v1.rknn file in the current    # path
ret = rknn.export_rknn(export_path = './mobilenet_v1.rknn')
```

3.5.6 Loading RKNN model

API	**load_rknn**
Description	Load RKNN model. The loading model is limited to connecting to the NPU hardware for inference or performance data acquisition. It can not be used for simulator or accuracy analysis.
Parameter	path: The path of RKNN model file.
	load_model_in_npu: Whether to load RKNN model in NPU directly. The path parameter should fill in the path of the model in NPU. It can be set to True only when RKNN-Toolkit2 run on RK356x Linux or NPU device(RK3566, rk3568) is connected. Default value is False. Not supported yet.

Return Value	0: Load successfully
	-1: Load failed

The sample code is as follows:

```
# Load the mobilenet_v1 RKNN model in the current path
ret = rknn.load_rknn(path='./mobilenet_v1.rknn')
```

3.5.7 Initialize the runtime environment

Before inference or performance evaluation, the runtime environment must be initialized. This interface determines the type of runtime (hardware platform or software simulator).

API	**init_runtime**
Description	Initialize the runtime environment. Set the device information (hardware platform, device ID). Determine whether to enable debug mode to obtain more detailed performance information for performance evaluation.
Parameter	target: Target hardware platform, now supports "RK3566", "RK3568". The default value is "None", which indicates model runs on simulator.
	device_id: Device identity number, if multiple devices are connected to PC, this parameter needs to be specified which can be obtained by calling "***list_devices***" interface. The default value is "None ".
	perf_debug: Debug mode option for performance evaluation. In debug mode, the running time of each layer can be obtained, otherwise, only the total running time of model can be given. The default value is False.
	eval_mem: Whether enter memory evaluation mode. If set True, the eval_memory interface can be called later to fetch memory usage of model running. The default value is False.
	async_mode: Whether to use asynchronous mode. When calling the inference interface, it involves setting the input picture, model running, and fetching the inference result. If the

	asynchronous mode is enabled, setting the input of the current frame will be performed simultaneously with the inference of the previous frame, so in addition to the first frame, each subsequent frame can hide the setting input time, thereby improving performance. In asynchronous mode, the inference result returned each time is the previous frame. The default value for this parameter is False. Not Supported yet.
Return Value	0: Initialize the runtime environment successfully
	-1: Initialize the runtime environment failed

The sample code is as follows:

```
# Initialize the runtime environment
ret = rknn.init_runtime(target='rk3566', device_id='012345789AB')
if ret != 0:
        print('Init runtime environment failed')
        exit(ret)
```

3.5.8 Inference with RKNN model

This interface kicks off the RKNN model inference and get the result of inference.

API	**inference**
Description	Use the model to perform inference with specified input and get the inference result. Detailed scenarios are as follows: 1. If RKNN Toolkit2 is running on PC and the target is set to Rockchip NPU when initializing the runtime environment, the inference of model is performed on the specified hardware platform. 2. If RKNN Toolkit2 is running on PC and the target is not set when initializing the runtime environment, the inference of model is performed on the simulator.
Parameter	inputs: Inputs to be inferred, such as images processed by cv2. The object type is ndarray list.

	data_format: The shape format of input data. Optional values are "nchw", "nhwc". The default value is 'nhwc'.
	inputs_pass_through: Pass the input transparently to the NPU driver. In non-transparent mode, the tool will reduce the mean, divide the variance, etc. before the input is passed to the NPU driver; in transparent mode, these operations will not be performed. The value of this parameter is an array. For example, to pass input0 and not input1, the value of this parameter is [1, 0]. The default value is None, which means that all input is not transparent.
Return Value	results: The result of inference, the object type is ndarray list.

The sample code is as follows:

For classification model, such as mobilenet_v1, the code is as follows (refer to *example/tfilte/mobilenet_v1* for the complete code):

```
# Preform inference for a picture with a model and get a top-5 result
……
outputs = rknn.inference(inputs=[img])
show_outputs(outputs)
……
```

The result of top-5 is as follows:

```
-----TOP 5-----
[156]: 0.85107421875
[155]: 0.09173583984375
[205]: 0.01358795166015625
[284]: 0.006465911865234375
[194]: 0.002239227294921875
```

For object detection model, such as ssd_mobilenet_v1, the code is as follows (refer to *example/tensorflow/ssd_mobilenet_v1* for the complete code):

```
# Perform inference for a picture with a model and get the result of object
# detection
……
```

```
outputs = rknn.inference(inputs=[image])
……
```

After the inference result is post-processed, the final output is shown in the following picture (the color of the object border is randomly generated, so the border color obtained will be different each time):

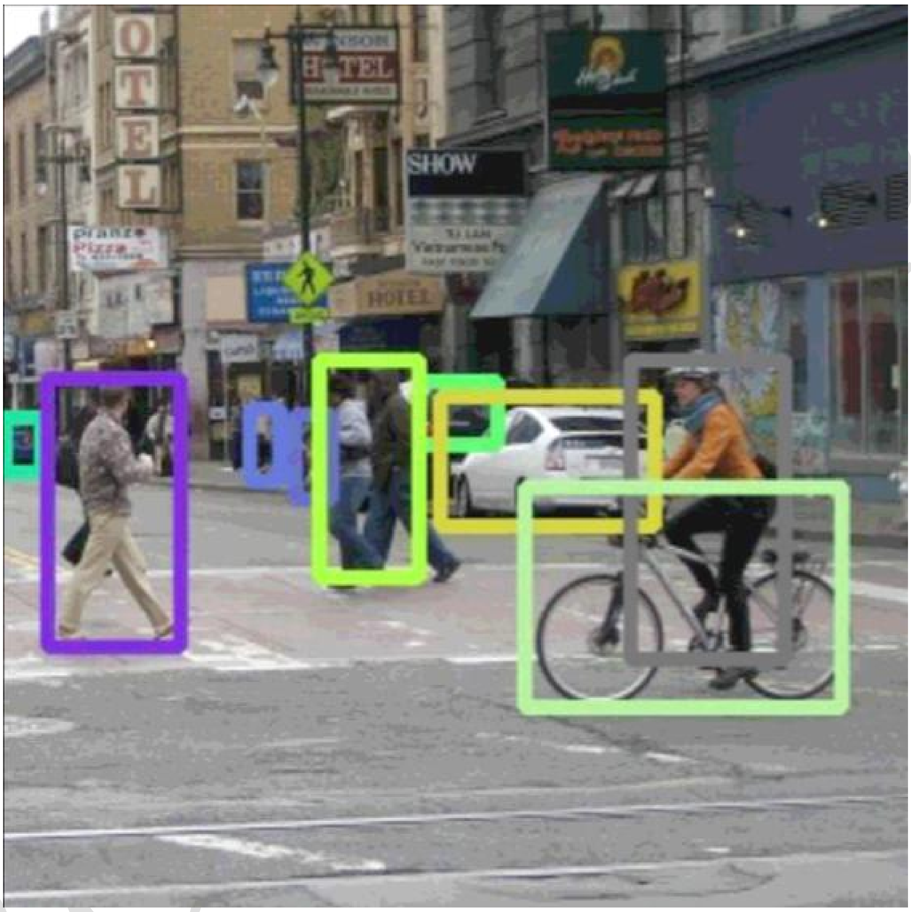

Figure 3 ssd_mobilenet_v1 inference result

3.5.9 Evaluate model performance

API	**eval_perf**
Description	Evaluate model performance. Model must run on RK3566 or RK3568 which connected to PC.If setting perf_debug to False when initializing runtime environment, the performance information is obtained from hardware, which only contains the total running time of model. And if the

	perf_debug is set to True, the running time of each layer will also be captured in detail.

Return Value	perf_result: Performance information. The object type is dictionary.
	If running on device and set perf_debug to False when initializing the runtime environment, the dictionary gives only one field ‘total_time’, example is as follows:
	`{`
	`    ‘total_time’: 1000`
	`}`
	In other scenarios, the obtained dictionary has one more filed called ‘layers’ which is also a dictionary type. The ‘layers’ takes the ID of each layer as the key, and its value is one dictionary which contains 'name' (name of layer), 'operation' (operator), 'target' (execution device), 'time'(time-consuming of this layer). Example is as follows:
	`{`
	`'total_time', 4568,`
	`'layers', {`
	`    '0': {`
	`        'name': 'convolution0',`
	`        'operation': 'ConvRelu',`
	`        'target': 'NPU',`
	`        'time': 362`
	`    }`
	`    '1': {`
	`        'name': 'convolution1',`
	`        'operation': 'ConvRelu',`
	`        'target': 'NPU',`
	`        'time': 158`
	`    }`
	`  }`
	`}`

The sample code is as follows:

```
# Evaluate model performance
……
rknn.eval_perf(inputs=[image], is_print=True)
……
```

For tflite/mobilenet_v1 in example directory, the performance evaluation results are printed as follows(different version of toolkit may be slightly different from the result.):

```
==========================================================================
                              Performance
       #### The performance result is just for debugging, ####
       #### may worse than actual performance!           ####
==========================================================================
Layer ID    Name                                                          Operator        Target    Time(us)
0           InputOperator:input                                           InputOperator   CPU       14
1           Conv:FAF_MobilenetV1/MobilenetV1/Conv2d_0/Relu6               ConvClip        NPU       316
2           Conv:FAF_MobilenetV1/MobilenetV1/Conv2d_1_depthwise/Relu6     ConvClip        NPU       329
3           Conv:FAF_MobilenetV1/MobilenetV1/Conv2d_1_pointwise/Relu6     ConvClip        NPU       510
4           Conv:FAF_MobilenetV1/MobilenetV1/Conv2d_2_depthwise/Relu6     ConvClip        NPU       324
5           Conv:FAF_MobilenetV1/MobilenetV1/Conv2d_2_pointwise/Relu6     ConvClip        NPU       192
6           Conv:FAF_MobilenetV1/MobilenetV1/Conv2d_3_depthwise/Relu6     ConvClip        NPU       233
7           Conv:FAF_MobilenetV1/MobilenetV1/Conv2d_3_pointwise/Relu6     ConvClip        NPU       227
8           Conv:FAF_MobilenetV1/MobilenetV1/Conv2d_4_depthwise/Relu6     ConvClip        NPU       143
9           Conv:FAF_MobilenetV1/MobilenetV1/Conv2d_4_pointwise/Relu6     ConvClip        NPU       132
10          Conv:FAF_MobilenetV1/MobilenetV1/Conv2d_5_depthwise/Relu6     ConvClip        NPU       142
11          Conv:FAF_MobilenetV1/MobilenetV1/Conv2d_5_pointwise/Relu6     ConvClip        NPU       193
12          Conv:FAF_MobilenetV1/MobilenetV1/Conv2d_6_depthwise/Relu6     ConvClip        NPU       71
13          Conv:FAF_MobilenetV1/MobilenetV1/Conv2d_6_pointwise/Relu6     ConvClip        NPU       99
14          Conv:FAF_MobilenetV1/MobilenetV1/Conv2d_7_depthwise/Relu6     ConvClip        NPU       79
15          Conv:FAF_MobilenetV1/MobilenetV1/Conv2d_7_pointwise/Relu6     ConvClip        NPU       171
16          Conv:FAF_MobilenetV1/MobilenetV1/Conv2d_8_depthwise/Relu6     ConvClip        NPU       78
17          Conv:FAF_MobilenetV1/MobilenetV1/Conv2d_8_pointwise/Relu6     ConvClip        NPU       196
18          Conv:FAF_MobilenetV1/MobilenetV1/Conv2d_9_depthwise/Relu6     ConvClip        NPU       78
19          Conv:FAF_MobilenetV1/MobilenetV1/Conv2d_9_pointwise/Relu6     ConvClip        NPU       195
20          Conv:FAF_MobilenetV1/MobilenetV1/Conv2d_10_depthwise/Relu6    ConvClip        NPU       79
21          Conv:FAF_MobilenetV1/MobilenetV1/Conv2d_10_pointwise/Relu6    ConvClip        NPU       170
22          Conv:FAF_MobilenetV1/MobilenetV1/Conv2d_11_depthwise/Relu6    ConvClip        NPU       78
23          Conv:FAF_MobilenetV1/MobilenetV1/Conv2d_11_pointwise/Relu6    ConvClip        NPU       170
24          Conv:FAF_MobilenetV1/MobilenetV1/Conv2d_12_depthwise/Relu6    ConvClip        NPU       62
25          Conv:FAF_MobilenetV1/MobilenetV1/Conv2d_12_pointwise/Relu6    ConvClip        NPU       232
26          Conv:FAF_MobilenetV1/MobilenetV1/Conv2d_13_depthwise/Relu6    ConvClip        NPU       169
27          Conv:FAF_MobilenetV1/MobilenetV1/Conv2d_13_pointwise/Relu6    ConvClip        NPU       494
28          Conv:MobilenetV1/Logits/AvgPool_1a/AvgPool                    Conv            NPU       182
29          Conv:MobilenetV1/Logits/Conv2d_1c_1x1/BiasAdd                 Conv            NPU       206
30          Softmax:MobilenetV1/Predictions/Reshape_1                     Softmax         CPU       335
31          Reshape:MobilenetV1/Logits/SpatialSqueeze                     Reshape         CPU       99
32          OutputOperator:MobilenetV1/Predictions/Reshape_1              OutputOperator  CPU       40
Total Time(us): 6038
FPS: 165.62
==========================================================================
```

3.5.10 Evaluating memory usage

API	**eval_memory**

Description	Fetch memory usage when model is running on hardware platform. Model must run on RK3566 or RK3568 which connected to PC.
Parameter	is_print: Whether to print performance evaluation results in the canonical format. The default value is True.
Return Value	memory_detail：Detail information of memory usage. Data format is dictionary. Data shows as below： `{ 'total_weight_allocation': 4312608 'total_internal_allocation': 1756160, 'total_model_allocation': 6068768 }` ● The 'total_weight_allocation' represents the memory footprint of the weights in the model. ● The 'total_internal_allocation' represents the memory usage of the internal tensor in the model. ● The 'total_model_allocation' represents the memory footprint of the model, that is, the sum of the weight and the memory footprint of the internal tensor.

The sample code is as follows:

```
# eval memory usage
……
memory_detail = rknn.eval_memory()
……
```

For tflite/mobilenet_v1 in examples directory, the memory usage when model running on RK3566 is printed as follows:

```
==============================================
                  Memory Profile Info Dump
==============================================
NPU model memory detail(bytes):
      Total Weight Memory: 4.11 MiB
      Total Internal Tensor Memory: 1.67 MiB
      Total Memory: 5.79 MiB

INFO: When evaluating memory usage, we need consider
```

```
the size of model, current model size is: 4.33 MiB
==================================================
```

3.5.11 Get SDK version

API	**get_sdk_version**
Description	Get API version and driver version of referenced SDK. Note: Before we use this interface, we must load model and initialize runtime first. And this API can only used on RK3566 / RK3568.
Parameter	None
Return Value	sdk_version：API and driver version. Data type is string.

The sample code is as follows：

```
# Get SDK version
……
sdk_version = rknn.get_sdk_version()
……
```

The SDK version looks like below：

```
==================================================
RKNN VERSION:
   RKNNAPI:     API: 1.6.1 (de5c7ec build: 2021-04-25 10:21:45)
   RKNNAPI:     DRV: 1.6.1 (de5c7ec build: 2021-04-25 10:14:11)
==================================================
```

3.5.12 Hybrid Quantization

Not supported yet.

3.5.12.1 hybrid_quantization_step1

When using the hybrid quantization function, the main interface called in the first phase is hybrid_quantization_step1, which is used to generate the temporary model file ({model_name}.model), the data file ({model_name}.data), and the quantization configuration file ({model_name}.quantization.cfg). Interface details are as follows:

API	**hybrid_quantization_step1**
Description	Corresponding temporary model files, data files, and quantization profiles are generated according to the loaded original model.
Parameter	dataset: A input data set for rectifying quantization parameters. Currently supports text file format, the user can place the path of picture(jpg or png) or npy file which is used for rectification. A file path for each line. Such as: a.jpg b.jpg or a.npy b.npy
	proposal: Generate hybrid quantization config suggestions.
	proposal_dataset_size: The size of dataset used for proposal. Because the proposal function is time-consuming, so the default size is 1.
Return Value	0: success
	-1: failure

The sample code is as follows:

```
# Call hybrid_quantization_step1 to generate quantization config
……
ret = rknn.hybrid_quantization_step1(dataset='./dataset.txt')
……
```

3.5.12.2 hybrid_quantization_step2

When using the hybrid quantization function, the primary interface for generating a hybrid quantized RKNN model phase call is hybrid_quantization_step2. The interface details are as follows:

API	**hybrid_quantization_step2**
Description	The temporary model file, the data file, the quantization profile, and the correction data set are received as inputs, and the hybrid quantized RKNN model is generated.
Parameter	model_input: The temporary model file generated in the first step, which is shaped like "{model_name}.model". The data type is a string. Required parameter.
	data_input: The model data file generated in the first step, which is shaped like "{model_name}.data". The data type is a string. Required parameter.
	model_quantization_cfg: The modified model quantization profile, whick is shaped like "{model_name}.quantization.cfg". The data type is a string. Required parameter.
Return Value	0: success
	-1: failure

The sample code is as follows:

```
# Call hybrid_quantization_step2 to generate hybrid quantized RKNN model
……
ret = rknn.hybrid_quantization_step2(
        model_input='./ssd_mobilenet_v2.model',
        data_input='./ssd_mobilenet_v2.data',
        model_quantization_cfg='./ssd_mobilenet_v2.quantization.cfg',
)
……
```

3.5.13 Quantitative accuracy analysis

The function of this interface is inference with quantized model and generate outputs of each layers for quantitative accuracy analysis.

API	**accuracy_analysis**
Description	Inference with quantized model and generate snapshot, that is dump tensor data of each layers. It will dump a snapshot of both data types include fp32 & quant for calculate quantitative error. **Note:**

<table>
<tr><td></td><td>1. this interface can only be called after build or hybrid_quantization_step2, and the original model should be a non-quantized model, otherwise the call will fail.
2. The quantization method used by this interface is consistent with the setting in config.</td></tr>
<tr><td rowspan="3">Parameter</td><td>inputs: the path list of image (jpg/png/bmp/npy).</td></tr>
<tr><td>output_dir: output directory, all snapshot data will stored here. (default is directory name ‘snapshot’)
If the target is not set, the following content will be output under ‘output_dir’:
● Directory simulator: Save the results of each layer on simulator when the entire quantitative model is fully run (The output has been converted to float32).
● Directory golden: Save the results of each layer on simulator when the entire floating-point model is completely run down.
● error_analysis.txt: Record the the cosine distance (entire_error and per_layer_error) between each layer result on simulator and the floating-point model on simulator during the complete calculation of the quantized model. The different of entire_error/per_layer_error is the input of each layer is come from the quantization model or floating-point model.
If the target is set, more content will output under ‘output_dir’:
● Directory runtime: Save the results of each layer when the entire quantitative model is fully run in NPU (The output has been converted to float32).
● error_analysis.txt: Record the the cosine distance (entire_error) between each layer result on simulator and each layer on NPU during the complete calculation of the quantized model additionally.</td></tr>
<tr><td>target: Target hardware platform, now supports "RK3566", "RK3568". The default value is "None". If target is set, the output of each layer of NPU will be obtained, and analyze it’s accuracy.</td></tr>
</table>

	device_id: Device identity number, if multiple devices are connected to PC, this parameter needs to be specified which can be obtained by calling "list_devices" interface. The default value is "None ".
Return Value	0: success
	-1: failure

The sample code is as follows:

```
……

    # Create RKNN object
    rknn = RKNN(verbose=True)

    print('--> config model')
    rknn.config(mean_values=[128, 128, 128], std_values=[128, 128, 128], )
    print('done')

    # Load model
    print('--> Loading model')
    ret = rknn.load_tensorflow(tf_pb='mobilenet_v1.pb',
                                inputs=['input'],
                                outputs=['MobilenetV1/Logits/SpatialSqueeze'],
                                input_size_list=[[1, 224, 224, 3]])
    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')

    print('--> Accuracy analysis')
    rknn.accuracy_analysis(inputs=['./dog_224x224.jpg'])

……
```

3.5.14 Register Custom OP

Not supported yet.

3.5.15 List Devices

API	**list_devices**
Description	List connected RK3566 / RK3568. Note: There are currently two device connection modes: ADB and NTB. RK3566 / RK3568 support both ADB and NTB. Make sure their modes are the same when connecting multiple devices
Parameter	None
Return Value	Return adb_devices list and ntb_devices list. If there are no devices connected to PC, it will return two empty list.

The sample code is as follows:

```
from rknn.api import RKNN

if __name__ == '__main__':
    rknn = RKNN()
    rknn.list_devices()
    rknn.release()
```

The devices list looks like below:

```
*************************
all device(s) with adb mode:
VD46C3KM6N
*************************
```

3.6 Accuracy troubleshooting

The troubleshooting of model accuracy is generally conducted from two aspects, one is the PC simulation accuracy investigation, and the other is the runtime accuracy check on board-side. The correct PC simulation result is a prerequisite for correct board-side running. Therefore, it is recommended that users prioritize the correct PC simulation results when dealing with the accuracy of the rknn model, and then troubleshoot the board-side running accuracy.Therefore, we will give recommendations and solutions for accuracy problem troubleshooting in terms of PC simulation accuracy investigation and board-side accuracy investigation during runtime.

In addition, the judgment of accuracy can simply use the cosine distance as the basic judgment, but this is not equal to the final model accuracy, it is only used as a reference. And the judgment of accuracy must be verified by running the data set finally. Of course, in the process of troubleshooting the following accuracy problems, you can simply use the cosine distant as a basis for accuracy improving.

3.6.1 PC simulation accuracy troubleshooting

The correctness of the PC simulation results is a prerequisite for the correct inference on board-side, so it is necessary to ensure that the simulator results is correct on the PC-side. Rknn-toolkit2 provides the choice of whether the mode is quantified, so this chapter analyzes the accuracy of the “fp16 model” and the “quantized model” respectively. Because the correct result of the “fp16 model” is the prerequisite for the accuracy of the “quantized model”, when there is a problem with the accuracy of the “quantized model”, it is generally recommended that users first verify the correctness of the “fp16 model”. The troubleshooting strategies for the “fp16 model” and the “quantized model” will be described in detail below.

3.6.1.1 Troubleshooting the accuracy of the “fp16 model”

The correct result of the “fp16 model” is a prerequisite to ensure the accuracy of the subsequent “quantized model”. The user only needs to set the do_quantization parameter to False when using the build interface of rknn to convert the original model to the “fp16 model”. if the output result of “fp16 model” is wrong, you need to perform the following troubleshooting:

1) Configuration issues

The configuration of the model is mainly concentrated in the config interface of rknn. And there are a few configuration in other rknn APIs. But not every configuration will cause the accuracy problems, mainly the parameters that cause the accuracy problems of the “fp16 model” as follows:

mean_values / std_values: The normalized parameters of the model, must ensure that they are the same as the parameters used in the original model.

input_size_list: The input node shape information of load_tensorflow / load_pytorch, if the configurations is wrong, it will also lead the wrong inference results.

i**nputs / outputs**: the name of the input and output nodes of load_tensorflow. If the configuration is wrong, it will also lead the wrong inference results.

parameters of inference interface: The input parameters of rknn's inference interface, mainly including inputs and data_format. Generally, in the python environment, the image data is read through cv2.imread. At this time, it should be noted that the image format read by cv2.imread is BGR. If the input of the original model is BGR (such as most caffe models), Then you can directly transfer the image data to the inference interface of rknn for inference; and if the input of the original model is RGB, you also need to call cv2.cvtColor(img, cv2.COLOR_BGR2RGB) to convert the image data to RGB, and then pass it to the inference interface of rknn inference. In addition, the layout of the image data read by cv2.imread is NHWC, because the default value of data_format is NHWC, so there is no need to set the data_format parameter. If the input data of the model is not read through cv2.imread, the user must clearly know the layout of the input data and set the correct data_format parameter, if it is image data, ensure that its RGB sequence is consistent with the input RGB sequence of the model.

The inspection of parameter configuration is a very important, and it is the main reason why many users have wrong output results of the "fp16 model". Specific steps are as follows:

a. Use the original model to perform inference under the original model's inference framework. For example, the caffe model uses caffe_bvlc or opencv_caffe for inference, the pytorch model uses the pytorch inference framework for inference, pb and tflite use tensorflow for inference, and onnx uses onnxruntime for inference. etc., and then save the inference result.

b. Use the original model to perform inference under the inference framework of rknn-toolkit2. You need to use the same input data as in the previous step, and set the inference mode of fp16 (do_quantization of rknn's build is set to False), and target parameter of init_runtime should not be configured or set to None. At this time, the simulator inside rknn-toolkit2 is used for inference, and the result of the inference is also saved.

c. Comparing the results of the two inferences, if the results are more consistent (cosine distance can be used to judge the consistency), it means that there is no problem with the above configuration.

d. If the results are inconsistent, check whether the above parameters are correct.

If it is confirmed that the above parameter configuration is correct, and the results are still inconsistent, it may be an internal bugs on the emulation-side.

2) **Internal bugs on the emulation-side**

This chapter may be related to the internal bug of the simulation, but the probability of occurrence is very low. Generally, it is recommended that users use the following two methods to troubleshoot.

One is to set the verbose parameter to 'Debug' when constructing the rknn object, it will turn on the debug mode of rknn-toolkit2, and output the accuracy check log during the construction of the rknn model, according to the result of "check results" in the output log (Cosine similarity and Euclidean distance), you can determine which step has the problem. If the problem is determined, it is best to provide the model and log that reproduce the result to the Rockchip NPU team for analysis and solution. This method uses the interface provided by rknn-toolkit2 for error checking, and is generally suitable for quickly locating the problem. If there is no related "check results" log output or the problem still cannot be located, the next method can also be used.

The prerequisite of the other method is that the user can obtain the ground truth of the output of each layer of the original model under the original framework. At this time, the accuracy analysis interface of rknn can be used to dump the ‘golden’ result of each layer of the floating-point model, and the cosine similarity is compared with the output results of each layer of the model under the original framework. If the ‘golden’ result is not aligned with the output result of the first layer of the model under the original framework (generally it is considered that the cosine similarity is less than 0.99 there is a little inconsistency, and it is almost considered that the result of this layer is wrong if the cosine similarity is lower than 0.98), it may still be the previous parameter configuration If there is an error, you need to go back to the previous step for re-checking. If the output results of the first layer are consistent, but the middle layer or the final results are inconsistent, it may be caused by a PC simulation implementation bug. At this point, the user can locate the layer where the inconsistency started, and can intercept the model around this layer, and provide the reproduction model to the Rockchip NPU team for analysis and solution.

3.6.1.2 Troubleshooting the accuracy of the “quantized model”

After the accuracy of the "fp16 model" is verified, the error of the "fp16 model" is eliminated, the model can be quantified, and the accuracy of the "quantized model" can be further analyzed. If you encounter accuracy problems in the "quantized model", the investigation will be conducted mainly from the following aspects:

3.6.1.2.1 Configuration issues

Similar to the configuration of the "fp16 model", configuration errors can also cause the accuracy of the "quantized model". On the basis of ensuring the correct configuration of the "fp16 model", the following parameter configurations should still be checked.

quantized_dtype: The choice of quantization type. The accuracy of different quantization types is very different, and there is also a big difference in runtime performance. Generally, a compromise quantization type, such as asymmetric_quantized-8, is selected. If asymmetric_quantized-4 is selected, the best runtime performance can be achieved, but the accuracy is also the worst, so it is only suitable for a few models that are not sensitive to 4-bit quantization. If asymmetric_quantized-16 is selected, the accuracy close to the original model can be achieved, but the runtime performance will be relatively poor, so it is only suitable for use in scenarios that are not sensitive to runtime performance but require very high accuracy. However, because generally in the NPU, 16-bit quantization and non-quantization (float16) have little difference in computing performance, it is recommended to choose fp16 (do_quantization of rknn's build interface is set to False) instead of 16-bit quantization. (The current version of asymmetric_quantized-16 is temporarily not supported, and asymmetric_quantized-4 is not supported under rk356x either)

quant_img_RGB2BGR: Indicates whether RGB2BGR needs to be performed first when loading the quantized image. It is generally used for the caffe model. For more detailed information, please refer to the quant_img_RGB2BGR parameter description. This parameter configuration error will also cause a significant decrease in quantization accuracy.

dataset: The configuration of quantitative correction set of rknn's build interface. If you select a calibration set that is not consistent with the actual deployment scenario, the accuracy may be reduced, or too many or too few calibration sets will affect the accuracy (generally choose 50 to 200 sheets).

To check the parameter configuration of the "quantified model" specifically, you can generally follow the steps below:

1) Directly perform "quantized model" inference, and then check the result of the inference and compare it with the result of the original model in the original inference framework. If the result is not very different, it can be considered that the quantized_dtype, quant_img_RGB2BGR and dataset parameters are basically correct.
2) If the result is still very different:
    a. If quantized_dtype is configured as a 4-bit algorithm, you can modify quantized_dtype to a higher-bit quantization algorithm.
    b. If the input image format of the original model is BGR (more common in caffe models),

you can modify quant_img_RGB2BGR to True at this time. In fact, the RGB sequence of the input data can be known from the processing code of the input data in the accuracy verification step of the previous "fp16 model".

c. You can use an image for quantization (leave only one line in dataset.txt), and use this image for inference. If the accuracy of a single image is improved more at this time, it means that the previously used quantization correction set is not well selected, and you can reselect pictures that are more consistent with the deployment scene.

d. If only one image is used for quantization (only one line is left in dataset.txt), you can try to use more images for quantization, which can be increased to about 50~200.

After the above investigation, the accuracy of some models may already meet the requirements, and some models may not be accurate enough, you can try the methods in the following chapters (change the quantization method). But there should not be a situation where the quantized result is completely wrong, if there is a completely wrong situation, please recheck the above configuration.

3.6.1.2.2 Quantitative methods issue

Some models themselves are not friendly to quantization. At this time, you can try to switch between different quantization methods and quantization algorithms. At present, there are two main quantization methods, namely Per-Layer / Per-Channel, corresponding to the quantized_method parameter in rknn's config interface, and the quantization algorithm is mainly divided into two types, namely Normal / MMSE, corresponding to rknn's config interface quantized_algorithm parameter in rknn’s config interface. Steps as follows:

1) If the Per-Layer quantization method was originally used, it can be changed to the Per-Channel quantization method. In general, the accuracy of the Per-Channel quantization method is much higher than that of the Per-Layer quantization method, but it may be brings a slight decrease in execution efficiency (negligible)

2) If the quantization method has been changed to Per-Channel, but the accuracy still cannot meet

the requirements, the quantization algorithm can be changed from Normal to MMSE at this time. This method will greatly increase the quantization time, but will bring better accuracy than Normal, and it will not affect the performance of the runtime.

If the accuracy is still low after using the above method, it may be that some Ops of the model are not friendly to the existing quantization algorithm, and the accuracy will drop more after quantization. For example, when the weight distribution of Conv is very uneven, you can consider using hybrid quantization to further improve the accuracy of the model. Hybrid quantization can allow different OPs in the model to use different quantization types. Specific steps are as follows:

1) First use the accuracy analysis interface to analyze the accuracy and find the layer that causes the accuracy to decrease.
2) Use the hybrid quantization method, and write the name of the output tensor of the suspected layer into the hybrid quantization configuration file.
3) Complete the steps of hybrid quantification and test the accuracy. (You can use the accuracy analysis interface to observe the accuracy changes)

Generally, after hybrid quantization, the accuracy of the model can be improved. If the improvement is not obvious or not enough, you can try to hybrid more layers, but at the same time it will also reduce the runtime speed, so the hybrid quantization requires users weigh the accuracy and speed by themselves. There is also a special way that when the Op with reduced accuracy is in the last layer, you can also choose to run the Op in post-processing, which will also effectively avoid the accuracy problem of this layer.

So far, after investigating the above reasons, we can basically obtain a quantized model with better accuracy, and the accuracy problem investigation on the simulation-side is basically completed.

3.6.2 Runtime accuracy troubleshooting

After the accuracy verification of the "fp16 model" and the "quantized model" on the PC simulation side, the quantization accuracy on the simulator should generally be able to meet the needs of the application, but users may often encounter the problem that the quantization accuracy on the simulator is not bad, but when the inference test is performed on the board through rknn's C API programming, they find that the accuracy is insufficient or not at all correct. There are generally two reasons for this kind of problem. One is caused by the code itself that calls rknn's C API programming, such as incorrect input data, incorrect runtime parameter configuration, or post-processing code error, etc.; the other is caused by runtime bugs on board-side. When encountering this kind of problem, we can first troubleshoot the runtime problem of the board-side through the following steps:

1) In the case of configuring the connecting board debugging environment (the environment configuration method is detailed in the RKNN C API release package), use rknn-toolkit2 to convert the rknn model and set the target parameter of init_runtime, such as target='rk3566' , And connect the board to the PC via USB (refer to chapter 3.2.2), and then perform the inference with board connection and check whether the inference result is correct. (Because the PC simulation does not strictly simulate the NPU hardware, so the result may not be completely consistent with the PC simulation)
2) If the inference result in step 1 differs greatly from the PC simulation result, it can be preliminarily determined that there is a bug in the runtime when running the model on the board-side. At this time, the accuracy analysis interface can be used to check the accuracy of the runtime side, just set the target parameter of accuracy_analysis, such as target='rk3566', after the accuracy_analysis is called, the accuracy analysis results of each layer will be output. If there is a significant difference between the board-side runtime result and the quantized true value of the simulation, the analysis result and the reproduced model can be fed back to the Rockchip NPU team for repair.

If there is no problem with the above verification, the problem lies in the C/C++ code that the user calls rknn's C API for programming. At this time, the user needs to carefully check whether the configuration of rknn's C API is configured correctly, and whether the pre-processing and post-processing processes of the model are correct (need to be exactly the same as the process on the PC simulation side).

For the use and configuration of rknn's C API, please refer to the relevant rknn_api documentation.