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

RKNN Toolkit Trouble Shooting

(Technology Department, Graphic Computing Platform Center)

Mark:	Version	1.7.3
[ ] Editing	Author	HPC
[√] Released	Completed Date	2022-08-08
	Auditor	Vincent
	Reviewed Date	2022-08-08

瑞芯微电子股份有限公司

Rockchip Electronics Co., Ltd

Revision History

Version no.	Author	Revision Date	Revision description	Auditor
V0.9	HPC	2019-04-01	Initial version release.	Vincent
V1.0	HPC	2019-07-18	Add some questions.	Vincent
V1.1	HPC	2019-08-22	Add some convolution acceleration tips.	Vincent
V1.2	HPC	2019-10-11	Add some questions.	Vincent
V1.3	HPC	2019-12-25	Add some questions, rearrange directory structure.	Vincent
V1.3.2	HPC	2020-04-13	Add some questions.	Vincent
V1.4.0	HPC	2020-08-13	Add some questions.	Vincent
V1.6.0	HPC	2020-12-31	Add some questions.	Vincent
V1.6.1	HPC	2021-05-21	Add some questions.	Vincent
V1.7.0	HPC	2021-08-08	Add some questions.	Vincent
v1.7.1	HPC	2021-11-17	1.Add some questions. 1.Adjust doc structure.	Vincent
v1.7.3	HPC	2022-08-08	1.Add some questions. 2.Adjust doc structure.	Vincent

目 录

1 Overview

The following documents are only applicable to RKNN Toolkit unless otherwise specified. At this time, the model converted by the tool is suitable for RK3399PRO, RK1806, RK1808, RV1126, and 1109 platforms.

When referring to this document, please confirm the version of RKNN Toolkit in your environment, otherwise some content may not apply.

2 Relationship between RKNN Toolkit and RKNPU

RKNN Toollkit is a model conversion tool based on Python language. It can convert models exported from other training frameworks to RKNN, and provides a limited inference interface to help users test the effect of model conversion. The project link is:

https://github.com/rockchip-linux/rknn-toolkit

RKNPU is a board-side component that provides NPU driver and provides functions such as model loading and model inference based on C language. Compared with the RKNN Toolkit tool, the C API inference interface of RKNPU is more flexible and has better performance. The project link is:

https://github.com/rockchip-linux/rknpu

RKNN Toolkit, RKNPU have consistent version. In order to avoid unnecessary trouble, users are recommended to use the same version of RKNN Toolkit and RKNPU; version updates usually include bug fixes and performance optimizations, and it’s advised to use the latest version .

3 RKNN Toolkit environment dependency problem

Contents:

3.1 Undefined symbol error occurs when install RKNN Toolkit

The error log is as follows:

```
Undefined symbol: PyFPE_jbuf
```

The reason of the error is Python environment is not clean, for example, numpy is installed in two different paths. Please re-build a clean Python environment and try again.

3.2 The environment that RKNN Toolkit depends on is too restrictive, making it impossible to install successfully

When all dependent libraries have been installed, but the versions and requirements of some libraries do not match, you can try adding the --no-deps parameter after the installation command to bypass the environment check during installation. Such as `pip install rknn-toolkit --no-deps.`

3.3 TensorFlow dependency description

The model quantization function of RKNN Toolkit relies on the TensorFlow library. Our recommended TensorFlow version is 1.14.0 or 2.2.0. Due to the poor compatibility between versions of TensorFlow, other versions may cause exceptions or work normally. In addition, when loading the TensorFlow model, It is recommended that the TensorFlow version for exporting the original model should be the same as the TensorFlow version that the RKNN Toolkit depends on..

For problems caused by the Tensorflow version, it is usually reflected in the load_tensorflow and build phases, and the error message will point to the tensorflow-lib path.

3.4 **PyTorch dependency description**

The function of loading PyTorch models of RKNN Toolkit depends on the pytorch version. PyTorch's models can be divided into floating-point models and quantized models (including QAT and PTQ quantized models). For floating-point models, there is a big difference between the models exported by PyTorch 1.5.1 and PyTorch 1.6.0, and compatibility problems may occur when they are mixed. Some parameters are loaded abnormally. For quantized models (QAT, PTQ), it`s recommend using PyTorch 1.10 to export the model, and upgrade the PyTorch version that RKNN Toolkit depends on to 1.10. In addition, when loading the PyTorch model, it`s recommend exporting the pytorch version of the original model, which should be consistent with the pytorch version that RKNN Toolkit depends on.

In general, it`s recommend using Pytorch 1.6.0, 1.9.0 or Pytorch 1.10.0.

3.5 **ONNX dependency description**

The loading onnx model function of RKNN-Toolkit depends on the onnx version. Due to the good compatibility between versions of onnx, no problems caused by versions have been found.

4 User guide for setting parameters while model conversion.

Contents:

4.1 Main parameters description

During model conversion, 'rknn.config' and 'rknn.build' will affect the conversion effect. 'rknn.load_onnx', 'rknn.load_tensorflow' specify input and output nodes, which will affect the conversion result. For 'rknn.load_pytorch', 'rknn.load_tensorflow', 'rknn.load_mxnet', the size of the specified input will affect the conversion result.

Usually, you can refer to the following basic ideas for configuration:

1. Prepare data for quantization, provide dataset file.

2. Determine the platform to be used by the model, such as RK1808, RV1109, and determine the target_platform parameter in 'rknn.config'.

3. If the input of the model is a 3-channel image, and the quantized data is in image format (such as jpg, png format), you need to pay attention to whether the model input is in RGB or BGR format, and confirm the reorder parameter in 'rknn.config'.

4. Confirm the normalization parameters during model training, and confirm the value of the mean/std parameter in 'rknn.config'.

5. Check the input size of the model, fill in the 'rknn.load' interface, such as the 'rknn.load_pytorch' interface.

6. Confirm the quantized_dtype parameter in 'rknn.config' (It can be ignored when the model is not quantized).

7. Use the built-in quantization algorithm when confirming the model quantization, and confirm the quantized_algorithm parameter in 'rknn.oonfig' (It can be ignored when the model is not quantized).

8. Confirm whether to quantize the model and confirm the do_quantization parameter in 'rknn.build'. When quantizing, you need to fill in the dataset parameter in 'rknn.build' at the same time.

9. When the model is precompiled offline, confirm the pre_compile parameter in 'rknn.build'.

4.2 Compatibility issues on RK platform.

For the platform parameters set by target_platform, the compatibility relationship is as follows:

- The models used by the RK3399PRO and RK1808 platforms are compatible with each other.
- The models used by the RV1126 and RV1109 platforms are compatible with each other.

4.3 Format and requirements of data for quantization

The format of the data is allowed to be in image (jpg, png) format, and RKNN-Toolkit relies on opencv for reading; it is also allowed to be in npy format, and numpy is called for reading at this time.

When using numpy format. If the input to the model is 4D dimensional, the data needs to be saved as an NHWC distribution. For example, when the input of the pytorch model is 1x3x112x224, the shape of the input saved by numpy is 1x112x224x3. If the input is not 4D, it can be processed normally.

For models with non-RGB/BGR image input, we recommend using numpy's npy format to provide quantized data

4.4 How to fill in dataset.txt file when multiple input

For dataset.txt file, one line is an example of one input; when there are multiple inputs, multiple inputs are written on the same line and separated by spaces

For single input, such as

```
sampleA.npy
sampleB.npy
```

For thriple inputs, such as

```
sampleA_in0.npy sampleA_in1.npy sampleA_in2.npy
sampleB_in0.npy sampleB_in1.npy sampleB_in2.npy
```

4.5 How to confirm the reorder parameter

When using pictures as data for quantization, you need to consider setting the reorder parameter.

When the model is trained with RGB images, the reorder parameter is set to '0 1 2'. And when using rknn.inference, or RKNPU capi for inference, input RGB images.

When the model is trained with BGR images, the reorder parameter is set to '2 1 0'. And when using rknn.inference, or RKNPU capi for inference, input RGB images.

If the data for quantization is in numpy's npy format, it is recommended not to use the reorder parameter to avoid confusion.

4.6 Calculation order of mean/std and reorder

For RKNN Toolkit and RKNPU C API, the calculation sequence is as follows: first perform the reorder operation on the input data, then perform the mean subtraction (mean) operation, and then divide by the standard deviation (std).

If the order of action of the original framework is the same as that of RKNN Toolkit, you can also fill in it without changing the order.

If the order of action of the original framework is to first subtract the mean and divide the standard deviation, and then perform channel conversion (such as RGB to BGR operation), you need to convert the numerical order of mean and std.

4.7 The value of mean/std should be filled with the value under the scale of 0~255

When using images for quantization, RKNN Toolkit uses TensorFlow to load data, and the value range of the data is always between 0 and 255. Therefore, if the data range when the original framework loads the data is between 0 and 1, the values of mean and std need to be multiplied by 255.

4.8 The setting problem of mean/std when non-3-channel input (such as grayscale image) and multi-input

The setting format of mean, std is consistent. Here is an example of mean.

Assuming that the input has N channels, the value of channel_mean_value is [[channel_1, channel_2, channel_3, channel_4, ..., channel_n]].

When there are multiple inputs, the value of channel_mean_value is [[channel_1, channel_2, channel_3, channel_4, ..., channel_n], [channel_1, channel_2, channel_3, channel_4, ..., channel_n]]

4.9 Selection of quantized data type

quantized_dtype has 'asymmetric_affine-u8', 'dynamic_fixed_point-i16' can be selected. Among them, u8 has better quantization performance and faster inference speed; while i16 has higher quantization accuracy, little loss of quantization accuracy(if there is any loss, it can be fed back to Rockchip NPU departments). When this parameter is not specified, u8 is used by default.

We recommend using u8 for quantization first, and then use i16 quantization if the accuracy does not meet the requirements.

4.10 Selection of quantization parameter correction algorithm and dataset size

quantized_algorithm has 'normal' and 'mmse' to choose from. When this parameter is not specified, normal is used by default. The quantization algorithm of mmse can help u8 quantization achieve better quantization accuracy on some models, but the disadvantage is that quantization takes a long time.

It`s recommend to use the 'normal' method for quantization first. If the quantization effect is not good, you can try to use the 'mmse' quantization algorithm.

When using normal quantization, it is recommended to give 200-500 sets of data for quantization. When using mmse quantification, it is recommended to use 20 sets of data for quantification.

4.11 After the model is quantized, what will be change for the input and output during inference

When using the regular RKNPU C API operation (referring to calling C API without pass_through and zero_copy), the data type of the input data (such as uint8 data, float data) has nothing to do with model quantization. The data type of the output data can always be automatically processed into float32 format. According to the quantization type of the model, you can also choose to output the model directly. In this case, the data type is the data type output by the last node of the model. The python inference interface will be slightly different, the specific relationship is as follows:

Table 4-1 Input and output relationship of quantized model and float model

After quantization	Python inference (rknn.inference)	C API inference (rknn.run)
Change of input dtype	No change. rknn.inference has a data_type parameter, which can specify the data type of int8, uint8, int16, float16, and float32 according to the actual input. The default is uint8. After specifying, the input will be automatically converted into the data format required by the rknn model.	No change. The rknn_tensor_type parameter of rknn inputs can specify RKNN_TENSOR_FLOAT32, RKNN_TENSOR_FLOAT16, RKNN_TENSOR_INT8, RKNN_TENSOR_UIN8, RKNN_TENSOR_INT16 according to the actual input. After specifying, the input will be automatically converted into the data format required by the rknn model.
Change of output dtype	No change. Regardless of whether the model is quantized or not, python's rknn.inference interface always returns an output of type float. Python's rknn.inference interface cannot provide output in the last node's data type.	New output dtype allowed. For the rknn outputs of RKNPU Capi, you can always set want_float=True or 1 to get the output of float type. After quantization, you can set want_float=False or 0, and you can output the original output type of the last node. For example, when u8 is quantized, output uint8 data.

Change of input layout (NCHW, NHWC)	No change. Regardless of quantization, data_format can set nchw or nhwc arrangement according to the input	No change. Regardless of quantization, rknn_tensor_format of rknn inputs can always be set to NCHW or NHWC arrangement

4.12 Offline precompile and online precompile

The precompiled model can effectively reduce the time of model initialization (corresponding to rknn.init_runtime or rknn_init). The precompiled model cannot use the simulator for inference. There may be incompatibilities between precompiled models on different NPU driver versions.

Offline precompile: When building an RKNN model using the rknn.build interface, set the pre_compile parameter to True. At this point, RKNN Toolkit uses the built-in simulator to complete the precompile function.

Online precompile: After using the rknn.build interface to build a common RKNN model, call the export_rknn_precompile_model interface to push the model to the development board, and use the driver on the board to complete the model precompile function. For details, please refer to examples/common_function_demos/export_rknn_precompile_model.

Since the simulator may be different from the actual driver, it`s recommend to use online precompile.

5 Problems when loading model

Contents:

1) Framework support and version problem

2) OP support

3) Common issues of Caffe model conversion

4) Common issues of Darknet model conversion

5) Common issues of ONNX model conversion

6) Common issues of PyTorch model conversion

7) Common issues of Tensorflow model conversion

8) Troubleshooting steps when loading model errors

9) When I load model, the numpy module raises error: Object arrays cannot be loaded when allow pickle=False.

5.1 Framework support and version problem

Deep learning frameworks supported by the RKNN Toolkit include TensorFlow, TensorFlow Lite, Caffe, ONNX and Darknet.

It corresponds to the version of each deep learning framework as follows:

Table 5-1 The first part of the support of each deep learning framework

RKNN Toolkit	Caffe	Darknet	Keras	MXNet
1.6.0	1.0	Commit ID：810d7f7	>=2.1.6-tf	>=1.4.0, <=1.5.1
1.6.1	1.0	Commit ID：810d7f7	>=2.1.6-tf	>=1.4.0, <=1.5.1
1.7.0	1.0	Commit ID：810d7f7	>=2.1.6-tf	>=1.4.0, <=1.5.1
1.7.1	1.0	Commit ID：810d7f7	>=2.1.6-tf	>=1.4.0, <=1.5.1
1.7.3	1.0	Commit ID：810d7f7	>=2.1.6-tf	>=1.4.0, <=1.5.1

Table 5-2 The second part of the support of each deep learning framework

RKNN Toolkit	ONNX	Pytorch	TensorFlow	TF Lite
1.6.0	1.6.0	>=1.0.0, <=1.6.0	>=1.10.0, <=2.0.0	Schema version = 3
1.6.1	1.6.0	>=1.0.0, <=1.6.0	>=1.10.0, <=2.0.0	Schema version = 3
1.7.0	1.6.0	>=1.0.0, <=1.6.0	>=1.10.0, <=2.0.0	Schema version = 3
1.7.1	1.6.0	>=1.5.1, <=1.9.0	>=1.10.0, <=2.0.0	Schema version = 3
1.7.3	1.6.0	>=1.5.1, <=1.10.0	>=1.10.0, <=2.2.0	Schema version = 3

Note:

1. In compliance with semver, SavedModels written with one version of TensorFlow can be loaded and evaluated with a later version of TensorFlow with the same major release. So in theory, the pb file generated by TensorFlow before version 1.14.0, RKNN Toolkit 1.0.0 and later are supported. For more information on TensorFlow version compatibility, please refer to the official link:
   https://www.tensorflow.org/guide/versions
2. RKNN Toolkit uses the TF Lite schema commits in link:
   https://github.com/tensorflow/tensorflow/commits/master/tensorflow/lite/schema/schema.fbs
   Commit hash: 0c4f5dfea4ceb3d7c0b46fc04828420a344f7598.
   Because TF Lite schema may not compatible with each other, TF Lite models with older or newer schema may not be loaded successfully.
3. There are two caffe protocols RKNN Toolkit uses, one based on the officially modified protocol of berkeley, and one based on the protocol containing the LSTM layer. The protocol based on the official revision of berkeley comes from this link: https://github.com/BVLC/caffe/tree/master/src/caffe/proto, commit hash is 21d0608. On this basis RKNN Toolkit have added some OPs. The protocol containing the LSTM layer refers to: https://github.com/xmfbit/warpctc-caffe/tree/master/src/caffe/proto, commit hash is bd6181b. These two protocols are specified by the proto parameter in the load_caffe interface.

4. The relationship between ONNX release version and opset version, IR version refers to the official website description:
   https://github.com/microsoft/onnxruntime/blob/v1.6.0/docs/Versioning.md

Table 5-3 Correspondence between ONNX release, opset and IR version

ONNX release version	ONNX opset version	Supported ONNX IR version
1.3.0	8	3
1.4.1	9	3
1.6.0	11	6

5. Darknet official Github link: https://github.com/pjreddie/darknet. RKNN Toolkit current conversion rules are based on the latest commit of the master branch (commit value: 810d7f7).
6. RKNN Toolkit currently mainly supports the Keras version with TensorFlow as the backend. The tested Keras version is the Keras built-in TensorFlow.
7. When trying to loading a PyTorch model, it`s recommend that the model is exported by the same version pytorch lib as RKNN Toolkit depends on.
8. Please refer to the "Rockchip_User_Guide_RKNN_Toolkit_EN" document for the support of various deep learning frameworks in earlier versions of RKNN Toolkit.

5.2 OP support

It depends on the framework where the model comes from. For details, please refer to the following documents:

https://github.com/rockchip-linux/rknn-toolkit/blob/master/doc/

5.3 Common issues of Caffe model conversion

5.3.1 Deprecated caffe input usage error occurs during model conversion

The error log is as follows:

```
Deprecated caffe input usage
```

It means this model is old version of caffe mode. Need to change input layer into below format.

```
layer {
  name: "data"
  type: "Input"
  top: "data"
  input_param {
    shape {
      dim: 1
      dim: 3
      dim: 224
      dim: 224
    }
  }}
```

5.3.2 The caffe.PoolingParameter has no field named round_mode error occurs during model conversion

The error log is as follows:

```
Message type "caffe.PoolingParameter" has no field named "round_mode"
```

The round_mode field of Pool layer cannot be recognized, please change it to ceil_model. For example, if originally it is round_mode: CEIL, then you can delete it (ceil_mode is True by default) or change to ceil_mode:True.

5.3.3 Not a valid scope name error occurs during caffe or other model conversion

The error log is as follows:

```
T       raise ValueError("'%s' is not a valid scope name" % name)
T   ValueError: '_plus0_17' is not a valid scope name
```

In this case, it is because layer name '_plusxxx' is not allowed to use _ at the beginning. Need to follow the naming rule of tensorflow:

```
[A-Za-z0-9.][A-Za-z0-9_.\\-/]* (for scopes at the root)
[A-Za-z0-9_.\\-/]* (for other scopes)
```

5.3.4 Invalid tensor id for mbox_conf_flatten_188 error occurs when Caffe version SSD conversion fails

The error log is as follows:

```
"Invalid tensor id(1), tensor(@mbox_conf_flatten_188:out0)"
```

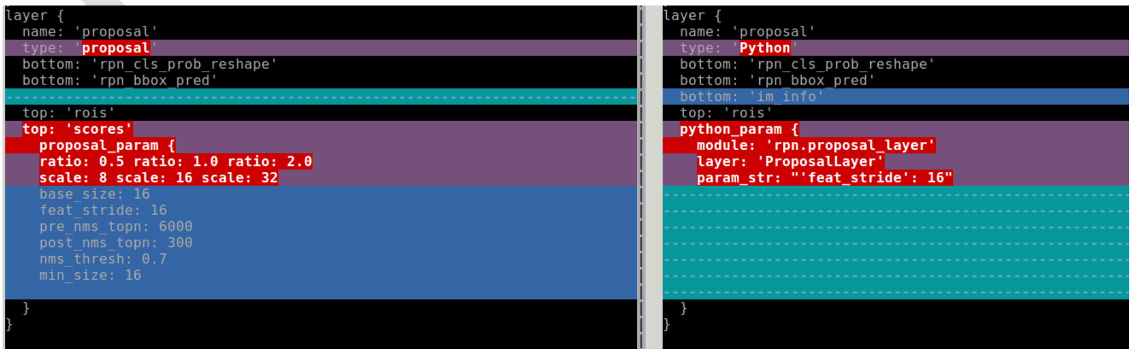
Not support DetectionOutput layer, you can delete and then change it to CPU.

5.3.5 There should be three output tensor after Caffe version SSD model deletes detectionoutput, but actually only return two tensor by RKNN inference

The missing tensor is priori box. It is the same during training and inference stage, and for all inputs. In order to improve performance, RKNN Toolkit optimized the relative layer in the model. If want to get the tensor of priori box, you can save the tensor of priori box, or use Caffe to do inference once in training stage.

5.3.6 Invalid tensor id for rpn_bbox_pred_18 error occurs during py-faster-rcnn model conversion

Comparing with official code, need to change 'proposal' layer of prototxt as below:



```
layer {
  name: 'proposal'
  type: 'proposal'
  bottom: 'rpn_cls_prob_reshape'
  bottom: 'rpn_bbox_pred'
  top: 'rois'
  top: 'scores'
    proposal_param {
    ratio: 0.5 ratio: 1.0 ratio: 2.0
    scale: 8 scale: 16 scale: 32
    base_size: 16
    feat_stride: 16
    pre_nms_topn: 6000
    post_nms_topn: 300
    nms_thresh: 0.7
    min_size: 16
  }
}
```

```
layer {
  name: 'proposal'
  type: 'Python'
  bottom: 'rpn_cls_prob_reshape'
  bottom: 'rpn_bbox_pred'
  bottom: 'im_info'
  top: 'rois'
  python_param {
    module: 'rpn.proposal_layer'
    layer: 'ProposalLayer'
    param_str: "'feat_stride': 16"
  }
}
```

Figure 8-1 Comparison before and after proposal layer modification

The modified proposal layer is defined as follows:

```
layer {
    name: 'proposal'
    type: 'proposal'
    bottom: 'rpn_cls_prob_reshape'
    bottom: 'rpn_bbox_pred'
    top: 'rois'
    top: 'scores'
        proposal_param {
        ratio: 0.5 ratio: 1.0 ratio: 2.0
        scale: 8 scale: 16 scale: 32
        base_size: 16
        feat_stride: 16
        pre_nms_topn: 6000
        post_nms_topn: 300
        nms_thresh: 0.7
        min_size: 16
    }
}
```

5.3.7 Does RKNN Toolkit support upsample of Caffe?

Upsample before 1.4.0 needs to be replaced by Resize, after 1.4.0, the Upsample layer is directly supported.

5.3.8 Not supported caffenet model version error occurs during model conversion

The error log is as follows:

```
E Not supported caffenet model version(v0 layer or v1 layer)
E Catch exception when loading caffe model: ../model/vgg16.prototxt!
```

The main reason is that the version of the caffe model is too old and needs to be updated. The update method is as follows (take VGG16 as an example):

1) Download Caffe source code from https://github.com/BVLC/caffe.git

2) Compile Caffe

3) Convert the model to a new format

```
/build_release/tools/upgrade_net_proto_text    vgg16_old/vgg16.prototxt \
vgg16_new/vgg16.prototxt

./build_release/tools/upgrade_net_proto_binary    vgg16_old/vgg16.caffemodel \
vgg16_new/vgg16.caffemodel
```

5.4 Common issues of Darknet model conversion

5.4.1 An error occurred when running examples/darknet/yolov3, and the error message prompted MultiDiGraph object has no attribute node

The error log is as follows:

```
MultiDiGraph object has no attribute node
```

The reason for this error is that the networkx version is too high. The solution is to downgrade the networkx version to version 1.11.

5.5 Common issues of ONNX model conversion

5.5.1 Encountered the error that the IR version is too high during conversion

The error log is as follows:

```
Your model ir_version is higher than the checker`s
```

RKNN Toolkit 1.1.0 and earlier versions only support models exported by ONNX from version 1.3.2 and below. RKNN Toolkit 1.4.0 and earlier versions support models exported by ONNX 1.4.1 and below; RKNN Toolkit 1.6.0 supports models exported by ONNX version 1.6.0.

5.5.2 Resize Issue

For RKNN Toolkit 1.7.0 and earlier versions, if Resize(upsample) OP makes an error during the parsing process, please update to RKNN Toolkit 1.7.3 version or the latest one, and choose to set Opset-version=11 or 12 when exporting ONNX.

5.5.3 Slice Issue

For RKNN Toolkit 1.7.0 and earlier versions, if Slice OP makes an error during parsing process, please update to RKNN Toolkit 1.7.3 version or the latest one.

5.6 Common issues of PyTorch model conversion

Before version 1.3.0, RKNN Toolkit indirectly supports Pytorch through ONNX, so need to convert Pytorch to ONNX first. Start from version 1.3.0, RKNN Toolkit supports loading Pytorch model directly. If issue occurs during conversion, please update RKNN Toolkit to the latest version first.

5.6.1 How to solve the problem of torch._C has no attribute _jit_pass_inline when loading the Pytorch model?

The error log is as follows:

```
‘torch._C’ has no attribute ‘_jit_pass_inline’
```

Please update version of PyTorch to 1.6.0 or later.

5.6.2 Save format of PyTorch model

Only models saved in torch.jit.trace can be used. The torch.save saves only the parameters of the model, and does not have a network structure. With only one network parameter, RKNN Toolkit cannot construct the corresponding network.

5.6.3 The tsr.op_type is not Constant error occurs during conversion

This issue is introduced after pytorch 0.4.5 version. In your model, if there is something like“x = x.view(x.size(0), -1)”, need to change to “x = x.view(int(x.size(0)), -1)”.

5.6.4 The PytorchStreamReader failed error occurs during conversion

The error log is as follows:

```
    E Catch exception when loading pytorch model: ./mobilenet0.25_Final.pth!
    E Traceback (most recent call last):
    ……
    E       cpp_module = torch._C.import_ir_module(cu, f, map_location, extra_files)
    E RuntimeError: [enforce fail at inline container.cc:137]. PytorchStreamReader failed
reading zip archive: faild finding central directory frame #0 ……
```

This error is because the model you are converting contains only weights but no network structure.

Usually the pth file contains only weights and no network structure information. The correct step is to define a net, then load the pth weights with net.load_state_dict(), and finally use torch.jit.trace() to freeze the network structure and weights into a pt file, and then use rknn.load_pytorch() to convert this pt file. For details, please refer to examples / pytorch / resnet18. Usually it can't be converted if you only has a pth file.

5.6.5 KeyError occurs during conversion

The error log is as follows:

```
    E Catch exception when loading pytorch model: ./mobilenet0.25_Final.pth!
    E Traceback (most recent call last):
    ……
    E KeyError: ‘aten::softmax’
```

When an error message like KeyError: 'aten::xxx' appears, it means that the current version does not support the operation. RKNN Toolkit will fix such bugs in each version upgrade, please use the latest version of RKNN Toolkit.

5.6.6 Syntax error in input error occurs during conversion

The error log is as follows:

```
    WARNING: Token ‘COMMENT’ defined, but not used
    WARNING: There is 1 unused token
    !!!!! Illegal character ‘”’
    Syntax error in input! LexToken(NAMED_IDENTIFIER, ‘fc’, 1, 27)
    !!!!! Illegal character ‘”’
```

There are many reasons for this error, please troubleshoot in the following order:

1. Torch.nn.module is not inherited when create a network. Please inherit torch.nn.module to create a network, and then use torch.jit.trace to generate a pt file.

2. If RKNN Toolkit version is 1.6.0 or 1.6.1, it is recommended to use torch 1.5.0 to 1.6.0.

3. RKNN Toolkit 1.7.0 and later version, it is recommended to use torch 1.6.0, 1.9.0 or 1.10.0.

5.7 Common issues of Tensorflow model conversion

5.7.1 AttributeError occurs during Google official ssd_mobilenet_v2 model conversion

The error log is as follows:

```
AttributeError: 'NoneType' object has no attribute op
```

One possible reason is that input node is not correct. You can modify as below:

```
rknn.load_tensorflow(
tf_pb='./ssd_mobilenet_v2_coco_2018_03_29/frozen_inference_graph.pb',
inputs=['FeatureExtractor/MobilenetV2/MobilenetV2/input'],
outputs=['concat', 'concat_1'],
input_size_list=[[INPUT_SIZE, INPUT_SIZE, 3]])
```

5.7.2 Cannot convert value dtype to a Tensorflow Dtype error occurs during SSD_Resnet50_v1_FPN_640x640 model conversion

The error log is as follows:

```
Cannot convert value dtype (['resource', 'u1']) to a Tensorflow Dtype
```

Need to update RKNN Toolkit to version 0.9.8 or higher.

5.7.3 On RKNN Toolkit 1.0.0，is the output shape of RKNN model converted from TensorFlow changed?

Versions prior to 1.0.0 will convert output shape from "NHWC" to "NCHW". Starting from this version, the shape of the output will be consistent with the original model, and no longer convert from "NHWC" to "NCHW". Please pay attention to the location of the channel when performing post processing.

5.7.4 When converting the model, the error attr explicit_paddings not in Op appears.

The error log is as follows:

```
E ValueError: NodeDef mentions attr 'explicit_paddings' not in Op<name=Conv2D;
```

This is because the version of tensorflow used in the model training is greater than or equal to the 1.14.0 version, and when converted to the rknn model, the tensorflow version used is less than or equal to the 1.13.2 version. The solution is to keep the training model and the converted rknn model using the same tensorflow version. For example, either use version >= 1.14.0, or use version <= 1.13.2.

5.7.5 When loading the model, it prompts the RKNN Toolkit TFLite converter module has no attribute TransposeOptions

The error log is as follows:

```
module “RKNNlib.convertert.lite.tflite” has no attribute ‘TransposeOptions’
```

The RKNN Toolkit version 1.3.2 and earlier does not support the Transpose operation of TFLite. Need to update to version 1.4.0 or later.

5.8 Troubleshooting steps when loading model errors

First confirm whether the original deep learning framework can load the model and perform correct inference.

Secondly, please upgrade RKNN Toolkit to the latest version. If the model has a layer (or OP) that is not supported by RKNN Toolkit, by turning on the debug log switch, you can see which layer or OP is not supported by RKNN Toolkit in the log. This type of error log usually contains "Try match xxx failed" or "Not match xxx", etc. If there is a problem of incorrect shape processing when the model is converted, you can find hints in the log such as concat/add/matmul dimension mismatch, or calculation of output tensor error.

If the first step fails, please check the original model for problems. If it still cannot be converted after upgrading to the latest version of RKNN Toolkit, or if some OPs does not supported, please report the version of the tool used and the detailed log when the conversion problem occurs to the Rockchip NPU development team.

5.9 When I load model, the numpy module raises error: Object arrays cannot be loaded when allow pickle=False.

The error log is as follows:

```
    E Catch exception when building RKNN model!
    T Traceback(most recent call last):
    ……
    T    File "/home/test/python-venv/lib/python3.5/site-packages/numy/lib/npyio.py",
line 447, in load(pickle_kwargs=pckle_kwargs)
    T     File "/home/test/python-venv/lib/python3.5/site-packages/numpy/lib/format.py",
line 692, in read_array
              Raise ValueError("Object arrays cannot be loaded when"
    T ValueError: Object_arrays cannot be loaded when allow_pickle=False
```

This error is caused by the change in the default value of the allow_pickle parameter of the load file interface after numpy is upgraded to 1.16.3. There are two solutions: one is to reduce the numpy version to version 1.16.2 or lower; the other is to update RKNN Toolkit to version 1.0.0 or later.

6 Model quantization problem

Contents:

6.1 Model quantization overview

When use this method, user loads the well-trained float point model, and RKNN Toolkit will do the quantization according to the dataset provided by user. Dataset should try to cover as many input type of model as possible. To make example simple, generally put only one picture. Suggest to put more.

RKNN supports two kinds for model quantization:

- RKNN Toolkit quantize the float model according to the calibration dataset and generate the quantized RKNN model.
  - Support quantize type: int16, int8, uint8
  - Quantize method: Post Training Static Quantization
- Loading quantized model, which exported from the opensource DL framework, RKNN Toolkit can use the already exists quantization information to generate the quantized RKNN model.
  - Support framework(The main supported versions are in brackets, it’s recommanded to export the quantized model by that version):

    Pytoch(v1.9.0)、Onnx(Onnxruntime v1.5.1)、Tensorflow、Tflite
  - Quantized method: Post Training Static Quantization, Quantization Aware

Training(QAT)

6.2 QAT and PTQ quantization

- **Post training static quantization**

Currently RKNN Toolkit supports three kinds of quantization methods:

■ asymmetric_quantized-u8（default）

This is the quantization method supported by tensorflow, which is also recommended by Google. According to the description in the article of Quantizing deep convolutional networks for efficient inference: A whitepaper, the accuracy loss of this quantization method is the smallest for most networks.

Its calculation formula is as follows：

$$quant = round\left(\frac{float\_num}{scale}\right) + zero\_point$$
$$quant = cast\_to\_bw$$

Where 'quant' represents the quantized number; 'float_num' represents float; data type of 'scalse' if float32; data type of 'zero-points' is int32, it represents the corresponding quantized value when the real number is 0. Finally saturate 'quant' to [range_min, range_max].

$$range\_\max = 255$$
$$range\_\min = 0$$

Currently only supports the inverse quantization of u8, the calculation formula is as follows:

$$float\_num = scale(quant - zero\_point)$$

■ dynamic_fixed_point-8

For some models, the quantization accuracy of dynamic_fixed_point-8 is higher than asymmetric_quantized-u8.

Its calculation formula is as follows：

$$quant = round(float\_num * 2^{fl})$$
$$quant = cast\_to\_bw$$

Where 'quant' represents the quantized number; 'float_num' represents float; 'fl' is the number of digits shifted to the left. Finally saturate 'quant' to [range_min, range_max].

$$range\_\max = 2^{bw-1} - 1$$
$$range\_\min = -\left(2^{bw-1} - 1\right)$$

If 'bw' equals 8, the range is [-127, 127].

- dynamic_fixed_point-16

The quantization formula of dynamic_fixed_point-16 is the same as dynamic_fixed_point-8, except bw=16. For RK3399pro/RK1808, there is 300Gops int16 computing unit inside NPU, for some quantized to 8 bit network with relatively high accuracy loss, you can consider to use this quantization method.

- **Quantization aware training(QAT)**

A model with quantized params/weights can be obtained through the Quantization-aware training. RKNN Toolkit currently supports TensorFlow and PyTorch frameworks for QAT models. Please refer to the following link for the technical details of QAT:

Tensorflow:

https://www.tensorflow.org/model_optimization/guide/quantization/training

PyTorch:

https://pytorch.org/blog/introduction-to-quantization-on-pytorch/

This method requires to use the original framework to train (or fine tune) to obtain a quantized model, and then use RKNN Toolkit to import the quantized model (you need to set do_quantization=False in the build setting during model conversion). At this time, RKNN Toolkit will use the quantized parameters of the model, so theoretically there will be almost no loss of accuracy comparing quantization via RKNN Toolkit.

Note: RKNN Toolkit also supports post-training quantization models of ONNX, PyTorch, TensorFlow, and TensorFlow Lite. The model conversion method is the same as the quantization-aware training model. Do_quantization should be set to False when calling the build interface. Update to RKNN Toolkit v1.7.0 or later for loading quantized onnx model. Update to RKNN Toolkit v1.7.1 or later for loading quantized pytorch model.

6.3 Quantization influence on model size

There are two scenarios. When the loaded model is the quantized model, do_quantization=False will use the quantization parameter of the model, for more details please refer to the answer of 1.4.1. When the loaded model is the non-quantized model, do_quantization=False will not do quantization, but will convert the weight from float32 to float16, which will not cause accuracy loss.

6.4 When convert RKNN model, set do_quantization equals False can build successfully, but set True will fail to build

The error log is as follows:

```
    T Caused by op 'fifo_queue_DequeueMany', defined at:
    T       File "test.py", line 52, in <module>
    T           ret = rknn.build(do_quantization=True, dataset='./dataset.txt')
    ……
    T OutOfRangeError (see above for traceback): FIFOQueue '_0_fifo_queue' is closed
and has insufficient elements (requested 1, current size 0)
```

It is because there is no data in dataset.txt, or the data format is not supported. Recommend to use jpg or npy.

6.5 What is the role of dataset during RKNN quantization

In the quantization process of RKNN Toolkit, it is necessary to find the appropriate quantization parameters according to the maximum and minimum values of the data. At this time, it is necessary to use the dataset for inference and obtain the input and output data of each layer. And then calculate the quantization parameters of each layer's input and output based on these data.

For this reason, the calibration dataset as the representative subset from the training set or validation set is better than others. The recommended data number is between 200 and 500.

6.6 Is the size of the image used for quantization correction the same as the size of the model input

Not required. RKNN Toolkit automatically scales images. Howerver, because zooming can change the image information, it may have some impact on the accuracy, so it is better to use pictures of similar size.

6.7 When calling the rknn.build interface, the program was killed after running for a period of time.

When the batch size in the config is set to a large size, using the data in the calibration data set to calculate the quantitative parameters requires more system memory. Solution: Increase the PC memory; or set the batch_size in the config interface to a smaller value, such as 8 or 16.

7 Description of simulator inference, inference with board-connected, and on-board inference

Contents:

7.1 Basic Instructions

Simulator inference: In the case that the PC side is not connected to the board, it is allowed to call the simulator inference model and obtain the inference results in the Linux environment. In addition, it is also allowed to use the Linux system call simulator to reason in the virtual machine. At this time, the virtual machine operating system must be Ubuntu16.04 or Ubuntu18.04, and the CPU architecture is x86_64.

Inference with board-connected : When the board is connected to the PC [for connection, please refer to Section 2.4 of the Rockchip_User_Guide_RKNN_Toolkit_CN document], use the RKNN-Toolkit python api to obtain inference results.

On-board inference: Refers to using the RKNPU's Capi interface to obtain inference results.

7.2 The inference result of the simulator is inconsistent with the inference result on board

When this happens, please refer to the result of the board side.

Due to hardware differences, the emulator is not guaranteed to get the exact same results as the board. We do not recommend using the emulator for evaluation when on-board debugging is available. The simulator is only an optional debugging method for developers who do not have a board in hand, but we strongly recommend the use of connected board debugging to avoid various problems caused by errors in the simulator.

7.3 The working method of inference with board-connected

When using inference with board-connected, RKNN-Toolkit will start a process of npu_transfer_proxy in the background. This process will communicate with the rknn-server on the board side. During communication, the model and model input will be transferred from the pc side to the board side, and then RKNPU will be called. Capi performs model inference, and then sends the inference results back to the PC.

7.4 Differences between the results of inference with board-connected and inference on board

During the inference with board-connected, the Capi interface of the RKNPU is actually called, so theoretically, when the inference result of board-connected is correct, the inference result of Capi is also correct. If there is a difference between the two, please confirm whether the input preprocessing, data type, data layout format (NCHW, NHWC) and Capi parameter settings are consistent.

It should be pointed out that if the difference is small, the difference is only in the 3 digits after the decimal point and the 4 digits after the decimal point, which is a normal phenomenon, and this difference may occur in the steps of data transmission and reception, data type conversion, etc.

7.5 Inference on-board is faster than inference with board-connected.

Due to the extra data copying and transmission process of board inference, the performance of inference with board-connected is not as good as the inference performance of RKNPU C API on board.

7.6 Getting more detail log

When connecting board debugging and connecting board inference, the initialization and reasoning of the model are mainly completed on the development board, and the log information is mainly generated on the board end.

In order to obtain specific board-side debugging information, you can enter the development board operating system through the serial port. Then execute the following two commands to set the environment variables for obtaining logs. Keep the serial port window closed, and then perform board debugging. At this time, the error message on the board side will be displayed on the serial port window:

```
export RKNN_LOG_LEVEL=5
restart_rknn.sh
```

8 Common issue about model evaluation

Contents:

8.1 Quantized model accuracy is not as good as expected

- **Firstly make sure the accuracy of float type is similar to test result of original platform:**
  1. Make rknn.build(do_quantization=False) when the quantized model is loaded by RKNN Toolkit.
  2. Refer to 1.1 to set **mean_values/std_values**, which should be same as the value used for training model.
  3. Make sure the sequence of the input image channel must be R,G,B while testing.

(Whatever the sequence of the image channel is used for training, it must be input by R,G,B while using RKNN to do testing).

(4) Set **reorder_channel** parameter in **rknn.config** function, ’0 1 2’ stands for RBG, ’2 1 0’ stands for BGR, and it must be consistent with the sequence of the image channel used for training.

➢ **Accuracy test after quantization**

(1) Use multiple pictures to do quantization, to ensure the stability of quantization accuracy.

Set batch_size parameter in rknn.config (recommend to set batch_size = 200) and provide more than 200 images path in dataset.txt for quantization.

If the display memory is not enough, you can set batch_size=10, epochs=20 instead of batch_size = 200 for quantization.

(2) Accuracy comparison, try to use relatively big data set to do testing. Compare the accuracy of top-1, top-5 for classifying network, compare mAP, Recall of data set for checking network, and so on.

(3) If it is face recognition, the results of the float model and the results of the quantitative model cannot be used for feature comparison. For example, two pictures of A1 and A2, the results of using the float model are A1fp, A2fp, the results of running with the quantitative model are A1u8, A2u8. At this time, the Euclidean distance of A1fp and A2fp can be calculated to calculate the similarity of two pictures, and the Euclidean distance of A1u8 and A2u8 can also be calculated to calculate the similarity of two pictures. But can't calculate the Euclidean distance of A1fp and A1u8 to calculate the similarity of two pictures.

➢ **Floating point model results are incorrect**

1) At present, the PC simulator can support dumping the data of each layer of the network. Before executing the inference script, you need to set an environment variable. The command is as follows: export NN_LAYER_DUMP=1.

2) After execution, the tensor data file of each layer of the network will be generated in the current directory, so that it can be compared with the data of the original framework layer by layer. Note: Some layers will be merged. For example, conv+bn+scale will be merged into a conv. At this time, it needs to be compared with the output of the scale layer of the original model. Please first check whether the mean_values / std_values /

reorder_channel in the config is set correctly; whether the image is processed in other ways before inference.

3) If the result of the last layer is all 0, pay attention to the serial port log to see if there is a GPU Hang message, and feed the model back to the Rockchip NPU team for analysis. If there is an error in the middle layer, the parameters, input and output information of the layer can be fed back to the Rockchip NPU team for further analysis. If you find that the results of the first layer are very different when comparing the data, it is usually because the input preprocessing is not set correctly.

4) The intermediate results of each layer of model inference can also be saved on development boards such as RK1808/RV1109/RV1126. The specific method is as follows: The serial port is connected to the development board, and a directory(such as dump_data) is created under the /userdata directory (the directory space is generally larger) to save the intermediate results. Then enter the directory and set the environment variable (export NN_LAYER_DUMP=1). If you are debugging through a PC connected to the board, you need to restart rknn_server by executing the restart_rknn.sh command. After making the above settings and then running the model inference program, you can dump the intermediate results of each layer. For the RK3399Pro board, you need to connect the NPU serial port on the board. For the specific location, please refer to the location marked with a red box in the figure below:

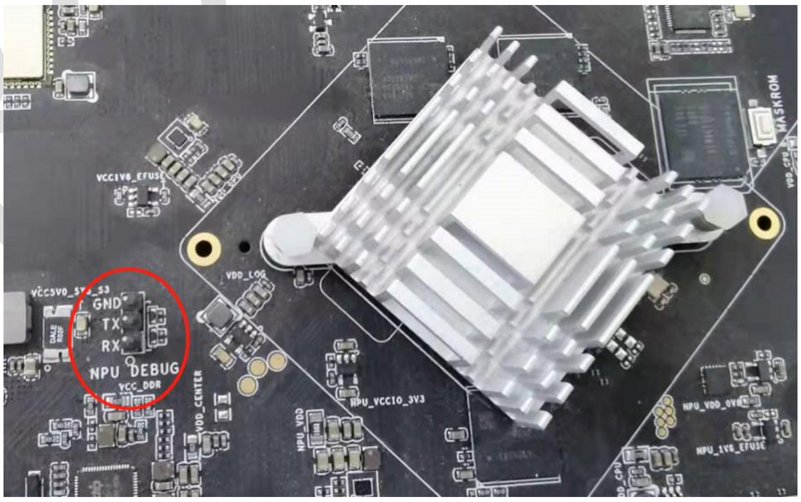


Figure 8-1 Serial port of NPU on RK3399Pro

- **Quantized model results are incorrect**

1) Check if the floating-point model is correct or not, please follow the previous section for

troubleshooting; if correct, proceed to the next step.

2) Use the accuracy_analysis interface to perform accuracy analysis to find out the layer that caused the decrease in accuracy. You can also manually dump the results of each layer, and compare the results to find the layer with reduced accuracy.

3) There are three situations for these layers with reduced accuracy. First, there is a problem with the driver implementation (the quantization result usually has a huge gap with the floating-point result at this time); the second is that the layer itself is not friendly to quantization; it may also be that some model optimizations lead to a decrease in accuracy, such as replacing add or average pool with conv etc. For the first scenario, please try to bypass it with hybrid quantization. For the second scenario, please try to use MMSE fine tune the quantization parameters, or try to use hybrid quantization, using high-precision methods(float or dynamic_fiexed_point-i16) to calculate layers that are not friendly to quantization. If it still fails to meet the requirements, please try quantization aware training of TensorFlow. For the third scenario, you can also try to lower the optimization level (the method of lowering is to set the optimization_level to 2 in the config interface). If the accuracy still fails to meet the requirements, please feedback the relevant model, accuracy analysis results and accuracy test methods to the Rockchip NPU team.

Note: For the usage of accuracy analysis, MMSE and hybrid quantization, please refer to the document <Rockchip_User_Guide_RKNN_Toolkit_EN.pdf>.

8.2 How to dump the output of each layer of network

Refer to the previous question.

8.3 Which frameworks` quantized model are currently supported by the RKNN Toolkit

RKNN Toolkit 1.6.1 and previous versions support quantized models of TensorFlow and TensorFlow Lite. Starting from 1.7.0, the quantized model of the ONNX (quantized with onnxruntime 1.5.2) framework is supported.

8.4 The file type and data format of dump files

It is a one-dimensional numpy array, which can be loaded by numpy.loadtxt interface.

The data is storage in the layout of NHWC. And the layout cannot be specified currently.

8.5 When using adb to query connected devices, it prompts no permission

The error log is as follows:

```
    test@test:~/ $ adb devices
    List of devices attached
    * daemon not running; starting now at tcp:5037
    * daemon started successfully
    1109        no permissions (user hpcci is not in the plugdev group); see
[http://developer.android.com/tools/device.html]
    1126        no permissions (user hpcci is not in the plugdev group); see
[http://developer.android.com/tools/device.html]
```

Solution:

Find script: <sdk>/platform-tools/update_rk_usb_rule/linux/update_rk1808_usb_rule.sh. Execute it. Then try ‘adb devices’ again. If there is no permission, please replug the device or restart the machine. When the permissions are normal, the output is as follows:

```
    test@test:~/ $ adb devices
    List of devices attached
    1109        device
    1126        device
```

8.6 The rknn_init returns RKNN_ERROR_MODEL_INVALID

The error log is as follows:

```
    E RKNNAPI: rknn_init,    msg_load_ack fail, ack = 1, expect 0!
    E Catch exception when init runtime!
    T Traceback (most recent call last):
    T     File "rknn/api/rknn_base.py", line 646, in
rknn.api.rknn_base.RKNNBase.init_runtime
    T     File "rknn/api/rknn_runtime.py", line 378, in
rknn.api.rknn_runtime.RKNNRuntime.build_graph
    T Exception: RKNN init failed. error code: RKNN_ERR_MODEL_INVALID
```

There are some situations in which this error occurs:

1. The option to pre_compile=True was used when generating the rknn model. Different versions of the RKNN Toolkit and drivers have a corresponding relationship. It is recommended to upgrade the firmware of the RKNN Toolkit and the board to the latest version.
2. There is no option to use pre_compile=True when generating the rknn model. At this time, the system firmware is too old. It is recommended to upgrade the firmware of the board to the latest version.
3. The parameter target_platform is not set correctly. For example, when the target_platform in the config interface is not set, the generated RKNN model can only be run on RK1806/RK1808/RK3399Pro, but not on RV1109/RV1126. If you want to run on RV1109/RV1126, you need to set target_platform=[‘rv1109’, ‘rv1126’] when calling the config interface.
4. If the problem occurs when inferring in a Docker container, it may be because the npu_transfer_proxy process on the host machine is not terminated, resulting in abnormal communication. please exit the Docker container first, terminate the npu_transfer_proxy process on the host, and then enter the container to execute the inference script.
5. It may also be that the model transferred from RKNN Toolkit has some problems. At this time, you can obtain the following information and feed it back to the Rockchip NPU team: If you are using a simulator, set verbose=True when initializing the RKNN object, print a detailed log, and record it. There will be a more detailed log description in the model initialization place The reason for the failure of the model check; if the PC is connected to the development board for debugging, or the model is running on the

development board, you can connect the serial port to the development board, and then set the environment variable RKNN_LOG_LEVEL=5, then execute restart_rknn.sh, and then rerun the program. Record the detailed log on the development board.

8.7 The rknn_init returns RKNN_ERR_DEVICE_UNAVAILABLE

The error log is as follows:

```
        E RKNNAPI: rknn_init, driver open fail! ret = -9!
        E Catch exception when init runtime!
        T Traceback (most recent call last):
        T File "rknn/api/rknn_base.py", line 617, in
rknn.api.rknn_base.RKNNBase.init_runtime
        T File "rknn/api/rknn_runtime.py", line 378, in rknn.api.rknn_runtime.RKNNRuntime.
        T build_graph
        T Exception: RKNN init failed. error code: RKNN_ERR_DEVICE_UNAVAILABLE
```

Please check it out as follows:

1) Make sure that the RKNN Toolkit and the firmware of devices have been upgraded to the latest version. The correspondence between each version of the RKNN Toolkit and the components of the system firmware is as follows：

Table 8-1 Version correspondence of RKNN components

RKNN Toolkit	rknn_server	NPU Driver	librknn_runtime
1.0.0	0.9.6/0.9.7	6.3.3.3718	0.9.8/0.9.9
1.1.0	0.9.8	6.3.3.03718	1.0.0
1.2.0	0.9.9	6.4.0.213404	1.1.0
1.2.1	1.2.0	6.4.0.213404	1.2.0
1.3.0	1.3.0	6.4.0.227915	1.3.0
1.3.2	1.3.2	6.4.0.7915	1.3.2
1.4.0	1.4.0	6.4.0.27915	1.4.0
1.6.0	1.6.0	6.4.3.5.293908	1.6.0
1.6.1	1.6.1	6.4.3.5.293908	1.6.1
1.7.0	1.7.0	6.4.6.5.351518	1.7.0
1.7.1	1.7.1	6.4.6.5.351518	1.7.1
1.7.3	1.7.3	6.4.6.5.351518	1.7.3

The version of these components is queried on RK1808 as follows：

```
# execute these commands on RK1808
dmesg | grep -i galcore        # query NPU driver version
strings /usr/bin/rknn_server | grep build        # query rknn_server version
strings /usr/lib/librknn_runtime.so | grep version        # query librknn_runtime version
```

The version information can also be queried through the get_sdk_version interface, where the DRV version corresponds to the version of rknn_server.

2) Make sure the “adb devices” command can get the device, and the target and device_id settings of rknn.init_runtime() are correct.

3) If you use RKNN Toolkit 1.1.0 and above, make sure rknn.list_devices() can get the device.

4) If you are using a compute stick or NTB mode for the RK1808 EVB version, make sure you have called update_rk1808_usb_rule.sh (contained in the RKNN Toolkit distribution) to get read and write access to the USB device.

5) If both ADB equipment and NTB equipment are connected, please make sure that only one type of equipment is in use at the same time. When using the ADB device, the npu_transfer_proxy process needs to be terminated; the ADB device can only be used on the

Linux x86_64 platform.

6) If it is a Mac OS platform, please make sure that only one device is in use. If you want to use another device, please manually finish the npu_transfer_proxy process first.

7) If it is a Windows platform, please turn off programs such as 360 Security Guard/Tencent Computer Manager before use, otherwise this error may also occur.

8) If you are running the AARCH64 version of the RKNN Toolkit directly on the RK3399/RK3399Pro, make sure the system firmware has been upgraded to the latest version.

9) If you are using RV1109/RV1126, please check whether Mini Driver is used, Mini Driver does not support online debugging.

10) If it is RK3399Pro, only the development board with Android system firmware can be debugged online on the Linux_X86 PC, and other firmware can be directly logged in to the corresponding system to install the AARCH64 version of RKNN Toolkit for development and testing.

8.8 Prompt that dlopen failed during the initial operation of the Windows platform

The error log is as follows:

```
    --> Init runtime environment
    E Catch exception when init runtime!
    ……
    E     File “rknn\api\rknn_runtime.py”, line xxx, in
rknn.api.rknn_runtime.RKNNRuntime.__init__,
    E     File “C:\Users\xxx\Python\Python36\lib\ctypes\__init__.py”,
    E           self._handle = _dlopen(self._name, mode)
    E OSError: [WinError 126] Cannot find the specified module.
    E Current device id is: None
    E Devices connected:
    E [‘1808s1’]
    Init runtime environment failed
```

If this error occurs, you need to add the path of librknn_api.so to the system path PATH.

First, find the installation path <RKNN_INSTALL_PATH> of RKNN Toolkit through "pip show rknn-toolkit", then add the following two paths to the system PATH: <RKNN_INSTALL_PATH>/rknn/api/lib/hardware/LION/Windows_x64 and <RKNN_INSTALL_PATH>/rknn/api/lib/hardware/PUMA/Windows_x64.

8.9 Cannot install driver for RV1109/RV1126 development board on Windows platform

Phenomenon: When the driver installation program is executed, the Driver/USB ID, etc. are all empty, and the Installer Driver is gray and cannot be clicked, as shown in the figure below:

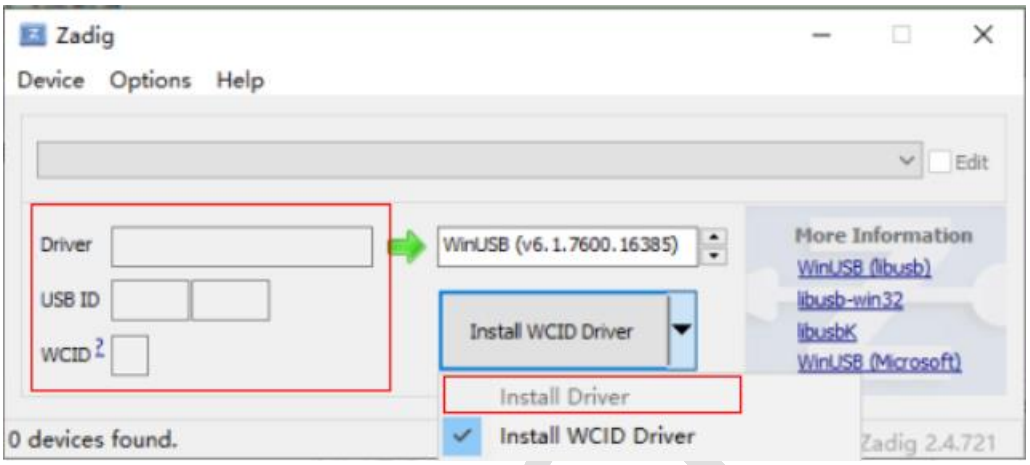


Figure 8-2 Driver installation interface

At this point, check the device list through the Windows Device Manager, and see Android Device instead of Universal Serial Bus device.

The cause of the problem: The firmware of RV1109/RV1126 does not enable NTB by default.

Solution:

1. Enter the development board system through the serial port or adb.
2. Find the file /etc/init.d/.usb_config, open it and add a line: usb_ntb_en.
3. Save the changes and restart the development board.
4. Execute the driver installation program again.

8.10 Cannot detect RV1109/RV1126 development board on Windows platform

Phenomenon: Execute the "python -m rknn.bin.list_devices" command on Windows or call the list_devices interface, but the RV1109 or RV1126 development board cannot be listed.

The cause of the problem: The firmware of RV1109/RV1126 does not enable NTB by default.

Solution:

1. Enable NTB first, and the method is the same as question 1.5.5.
2. Install the development board driver, please refer to <Rockchip_Quick_Start_RKNN_Toolkit_EN> for the installation method.

8.11 The rknn.inference cost too much time

This issue has two kinds of phenomenon:

1) The speed of forward inferring test is slow, and some picture may take over 0.5s while testing mobilenet-ssd.

2) The time difference between model rknn.inference and rknn.eval_perf() is relatively big, such as:

Theoretical computing time(single picture)	1.79ms	8.23ms	7.485ms	30.55ms
Actual computing time(single picture)	21.37ms	39.82ms	33.12ms	76.13ms

There are two reasons for the issue of slow measured frame rate:

1. Using the method of pc + adb to upload picture is quite slow, as it has high frame rate requirement for network such as 1.79ms theoretically.
2. In the implementation of 0.9.8 and earlier, the inference included some extra time, and the 0.9.9 and later versions have been optimized.

For more real measured frame rate, you can directly use c/c++ api to test on the board.

8.12 Does the rknn.inference support multi-inputs and multi-batch inputs

The RKNN Toolkit needs to be upgraded to version 1.2.0 or later. And you need to specify the number of input images when building the RKNN model. For detailed usage, refer to the description of the build interface in <Rockchip_User_Guide_RKNN_Toolkit_V1.2.1_CN.pdf>.

In addition, when rknn_batch_size is greater than 1 (e.g. equal 4), the inference code in python:

```
outputs = rknn.inference(inputs=[img])
```

need modify to:

```
img = np.concatenate((img, img, img, img), axis=0)
outputs = rknn.inference(inputs=[img])
```

8.13 Run with multi RKNN models

When running two or more models, you need to create multiple RKNN objects. One RKNN object corresponds to a model, which is similar to a context. Each model initializes the model in its own context. Each model initializes the model in its own context, makes inferences, and obtains inference results without interfering with each other. These models are inferred serially on the NPU.

8.14 Model inference takes a very long time, and the results obtained are all 0

If the inference takes more than 20s and the result is all 0, this is usually a GPU hang bug in the NPU. If you encounter this problem, you can try to update the NPU driver to version 1.5.0 or later.

8.15 The inference time of the model without quantization is much longer than the model with quantization. Is this normal?

It is normal. NPU floating-point computing power is relatively weak, and for the quantized model, NPU has been optimized. So the performance of the quantized model will be much better than that of the floating-point model. On NPU, it is recommended to use a quantized model.

8.16 Calling rknn.build with pre_compile=True, it raises an error, it can be successful if it is not set.

The error log is as follows:

```
        E Catch exception when building RKNN model!
        T Traceback (most recent call last):
        T      File "rknn/api/rknn_base.py", line 515, in rknn.api.rknn_base.RKNNBase.build
        T      File "rknn/api/rknn_base.py", line 439, in rknn.api.rknn_base.RKNNBase._build
        T      File "rknn/base/ovxconfiggenerator.py", line 187, in
rknn.base.ovxconfiggenerator.generate_vx_config_from_files
        T      File "rknn/base/RKNNlib/app/exporter/ovxlib_case/casegenerator.py", line 380, in
rknn.base.RKNNlib.app.exporter.ovxlib_case.casegenerator.CaseGenerator.generate
        T      File "rknn/base/RKNNlib/app/exporter/ovxlib_case/casegenerator.py", line 352, in
rknn.base.RKNNlib.app.exporter.ovxlib_case.casegenerator.CaseGenerator._gen_special_cas
e
        T      File "rknn/base/RKNNlib/app/exporter/ovxlib_case/casegenerator.py", line 330, in
rknn.base.RKNNlib.app.exporter.ovxlib_case.casegenerator.CaseGenerator._gen_nb_file
        T AttributeError: 'CaseGenerator' object has no attribute 'nbg_graph_file_path'
```

Please confirm:

1) The system is equipped with the gcc compiler toolchain

2) The name of the model only contains "letters", "numbers", "_". Or you can update RKNN Toolkit to version 1.3.0 or later, the RKNN Toolkit will automatically handle these special characters.

If this is not the case, you can try to export the pre-compiled model from the board using the export_rknn_precompile_model interface.

9 RKNN convolution acceleration tips

9.1 How to design a convolutional neural network to achieve optimal performance on RKNN

Here are some suggestions from us：

1. Optimal Kernel Size is 3x3

Convolution cores can support a large range of kernel sizes. The minimum supported kernel size is [1] and maximum is [11 * stride - 1].

The NN Engine performs most optimally when the Convolution kernel size is 3x3, under which the highest MAC utilization can be achieved.

Non-square kernels are also supported, but with some computation overhead.

2. Fused Operations Reduce Overhead

The Convolution core can fuse ReLU and MAX Pooling operations on the fly to further reduce computation and bandwidth overhead. A ReLU layer following a Convolution layer will always be fused, while MAX pooling layer fusion has the following restrictions, Max pooling must

- have a pool size of 3x3 or 2x2, and stride of 2
- 2x2 pooling must have an even input size and no padding
- 3x3 pooling must have odd input size which is not one and no padding
- Horizontal input size must be less than 64 (8-bit mode) or 32 (16-bit mode) if pool size is 3x3

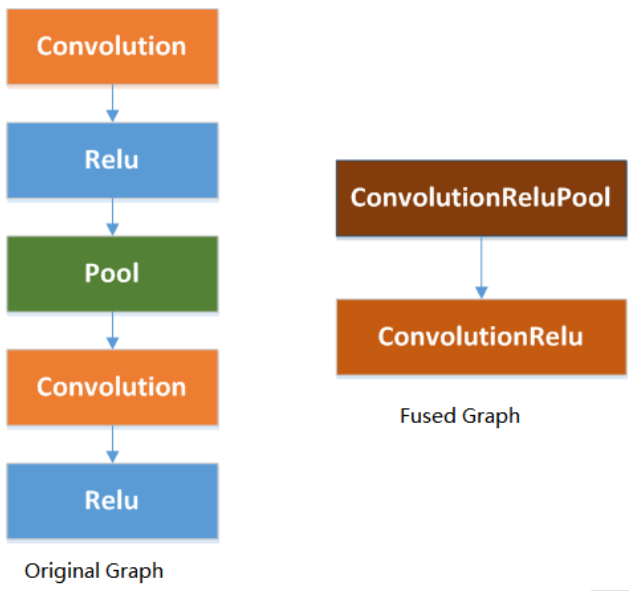


Figure 9-1 Operation fusion example

3. Depthwise Convolutions

Both regular 2D and Depthwise convolutions are supported, while 2D convolutions perform more optimally. Since Depthiwise Convolution-specific structure makes it less friendly to quantized model. It`s recommend to use 2D convolution whenever possible when designing your network.

If you must use a Depthwise convolution, it`s recommend to follow the rules below that can improve the accuracy of the quantized model:

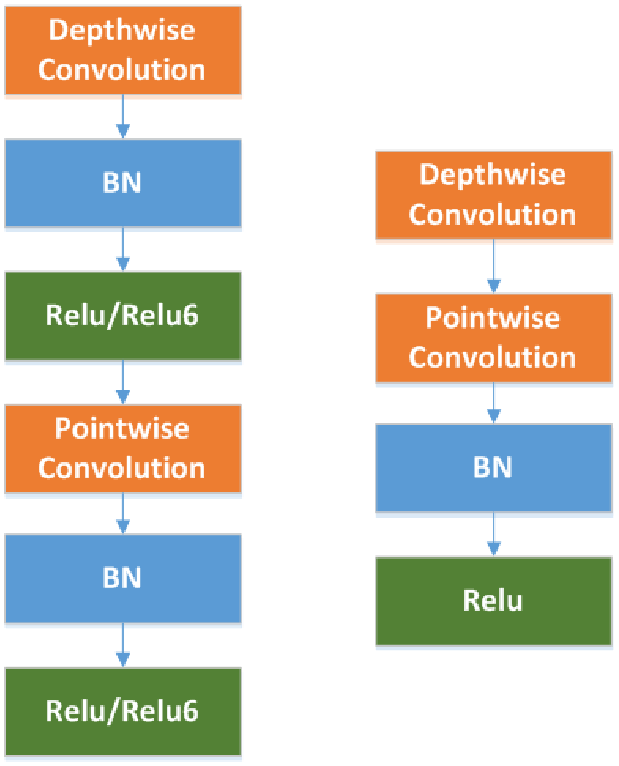


Figure 9-2 Idea to optimize depthwise convolution

- Change the activation function RELU6 to RELU.
- Remove the BN layer and activation layer of the Depthwise convolution layer.
- In training, for the Depthwise convolutional layer, L2 regularization of its weight.

4. Take advantage of Hardware's Sparse Matrix Support

Modern Neural-Networks are known to be over parameterized and have much redundancy in their design. Pruning a network to be sparse has been proven to reduce computation overhead while maintaining accuracy.

RKNN hardware is designed to support sparse matrix operations efficiently by skipping computations and memory fetches on zero values. The sparsity level can be fine grain down to individual weights. Designing a sparse network to take advantage of this technology could further improve performance on RKNN.

5. Dilation Convolution

When using TensorFlow to create a convolution with dilations parameter, use tf.nn.atrous_conv2d to create it. Currently the RKNN Toolkit don't support the direct use of tf.nn.conv2d to create convolution with dilations parameter, otherwise the inference results will be wrong.

In Addition, the version of TensorFlow need to be higher than 1.14.0, and the version of RKNN Toolkit need to be higher than v.1.4.0.

10 Appendix

10.1 Reference documents

OP support list:

RKNN_OP_Support.md

Quick start manual:

Rockchip_Quick_Start_RKNN_Toolkit_EN.pdf

RKNN Toolkit manual:

Rockchip_User_Guide_RKNN_Toolkit_EN.pdf

RKNN Toolkit Lite manual:

Rockchip_User_Guide_RKNN_Toolkit_Lite_EN.pdf

Trouble shooting manual:

Rockchip_Trouble_Shooting_RKNN_Toolkit_EN.pdf

Custom OP define manual:

Rockchip_Developer_Guide_RKNN_Toolkit_Custom_OP_EN.pdf

Visualization function manual:

Rockchip_User_Guide_RKNN_Toolkit_Visualization_EN.pdf

The above documents can be found in the SDK/doc directory. You can also visit the following link to view: https://github.com/rockchip-linux/rknn-toolkit/tree/master/doc

10.2 Issue feedback

All the issue can be feedback via the follow ways:

RKNN QQ group: 1025468710

Github link: https://github.com/rockchip-linux/rknn-toolkit/issues

Rockchip Redmine: https://redmine.rock-chips.com/

Note: Redmine account can only be registered by an authorized salesperson. If your development board is from the third-party manufacturer, please contact them to report the issue.