Ingenic®

Magik

Post-Training-Quantization User Guide

Date: Feb. 2022

北京君正集成电路股份有限公司
Ingenic Semiconductor Co., Ltd.

Ingenic®

Magik Post-Training-Quantization User Guide

Release history

Date	Author	Revision	Change
Feb. 2022	Owen	1.0.1	First release

Disclaimer

Ingenic Semiconductor Co., Ltd.

Ingenic Headquarters, East Bldg. 14, Courtyard #10
Xibeiwang East Road, Haidian District, Beijing, China,
Tel: 86-10-56345000
Fax:86-10-56345001
Http: //www.ingenic.com

目录

1 Introduction.......... 1

2 Conversion commands.......... 1

2.1 common command line parameters.......... 1

2.2 Post-training-quantization command line arguments.......... 2

3 Json file.......... 2

3.1 Basic information of json.......... 2

3.2 json debug option (optional).......... 3

4 yolov5 post-training-quantization demo.......... 4

4.1 Prepare the model.......... 4

4.2 Prepare the calibration picture.......... 5

4.3 Json file.......... 5

4.4 Check command line writing.......... 5

5 Run on-board.......... 6

5.1 Code compling.......... 6

5.1.1 Preparation.......... 6

5.1.2 Input of network.......... 7

5.1.3 Model loading.......... 7

5.1.4 Hyperparameter settings.......... 7

5.1.5 Compile.......... 7

5.2 Run.......... 8

5.2.1 Release(defalut).......... 8

5.2.2 Debug（Check data）.......... 9

5.2.3 Profile(network visualization).......... 9

5.2.4 Nmem(nmem occupancy).......... 10

6 Check data.......... 10

6.1 On PC.......... 10

6.2 On chip.......... 11

7 Frequently Asked Questions.......... 13

8 json debug process.......... 14

8.1 The phenomenon of poor effect.......... 14

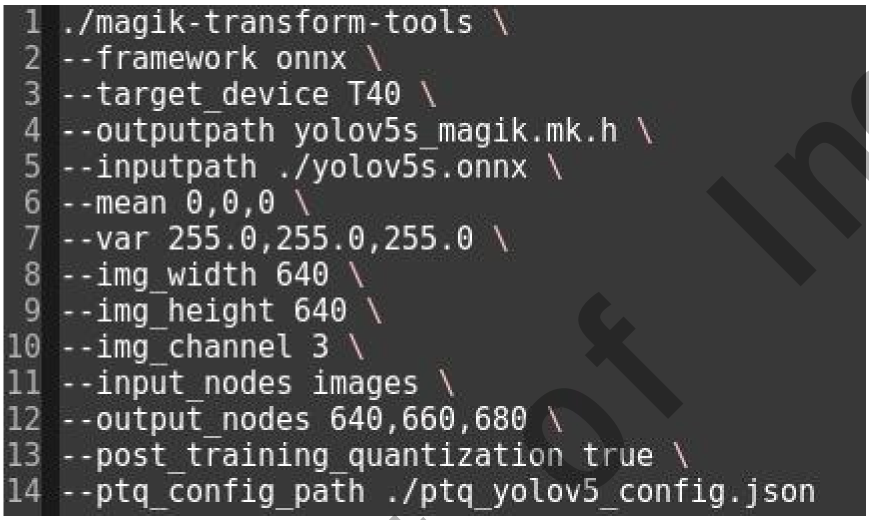
8.2 Solution.......... 14

1 Introduction

The MagikTransformKit tool supports post-quantization of the onnx/tensorflow format model to obtain the magik format model, which can be deployed on the chip side.

The document is based on the yolov5s network to show how to generate the magik format model deployed on the chip, and how to locate the problem when the generated model effect is not ideal.

2 Conversion commands



```
./magik-transform-tools \
--framework onnx \
--target_device T40 \
--outputpath yolov5s_magik.mk.h \
--inputpath ./yolov5s.onnx \
--mean 0,0,0 \
--var 255.0,255.0,255.0 \
--img_width 640 \
--img_height 640 \
--img_channel 3 \
--input_nodes images \
--output_nodes 640,660,680 \
--post_training_quantization true \
--ptq_config_path ./ptq_yolov5_config.json
```

The MagikTransformKit tool completes the post-trainning-quantization work of the onnx or tensorflow model through command-line parameters, and supports training and quantization model conversion work. The command-line parameters are divided into general parts and post-trainning-quantization parts.

2.1 common command line parameters

framework – Input model format, now supports onnx/tensorflow format model

(The caffe/mxnet/pytorch format model can uniformly convert the onnx format model)

target_device – part number of the deployment board

inputpath – Input model to be converted

outputpath – The output of the conversion. After the conversion is completed, the required *.bin file will be generated in the specified directory

mean –The mean of the preprocessing at training time

var — The variance of the preprocessing at training time

img_width — The width of the input when running on the chip side

img_height — The height of the input when running on the chip side

img_channel — The channel of the input when running on the chip side

input_nodes — Model input node

output_node — Model output node

2.2 Post-training-quantization command line arguments

post_training_quantization — The post-training-quantization switch options

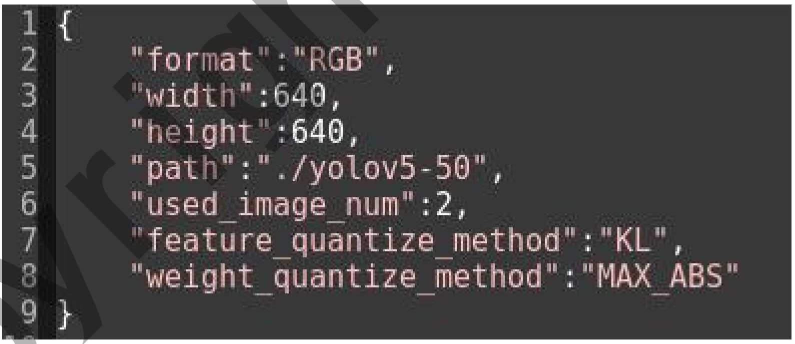
ptq_config_path — “json” file path that is provided to the post-training-quantization part.

3 Json file

The json file consists of two parts, the basic information required by the MagikTransformKit tool for post-training-quantization and the debug information of the debugging model.

When the quantization model generated by the MagikTransformKit tool is not satisfactory, you can add debug information to the Json file for debugging.

3.1 Basic information of json



```
{
    "format":"RGB",
    "width":640,
    "height":640,
    "path":"./yolov5-50",
    "used_image_num":2,
    "feature_quantize_method":"KL",
    "weight_quantize_method":"MAX_ABS"
}
```

format — image processing format: RGB,BGR,GRAY

width — Input width of the network inference

height — Input height of the network inference

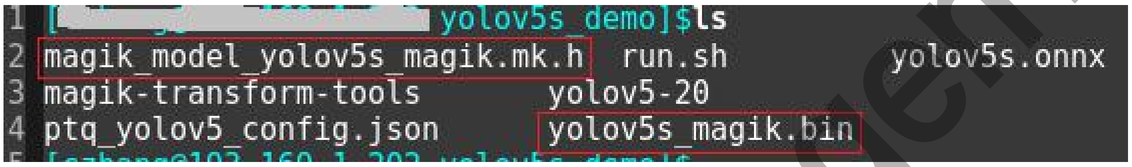
path ─ Directory path of the calibration image set.

used_image_num ─ Number of the calibration image set.

feature_quantize_method ─ feature quantization method: KL, ADMM, MSE

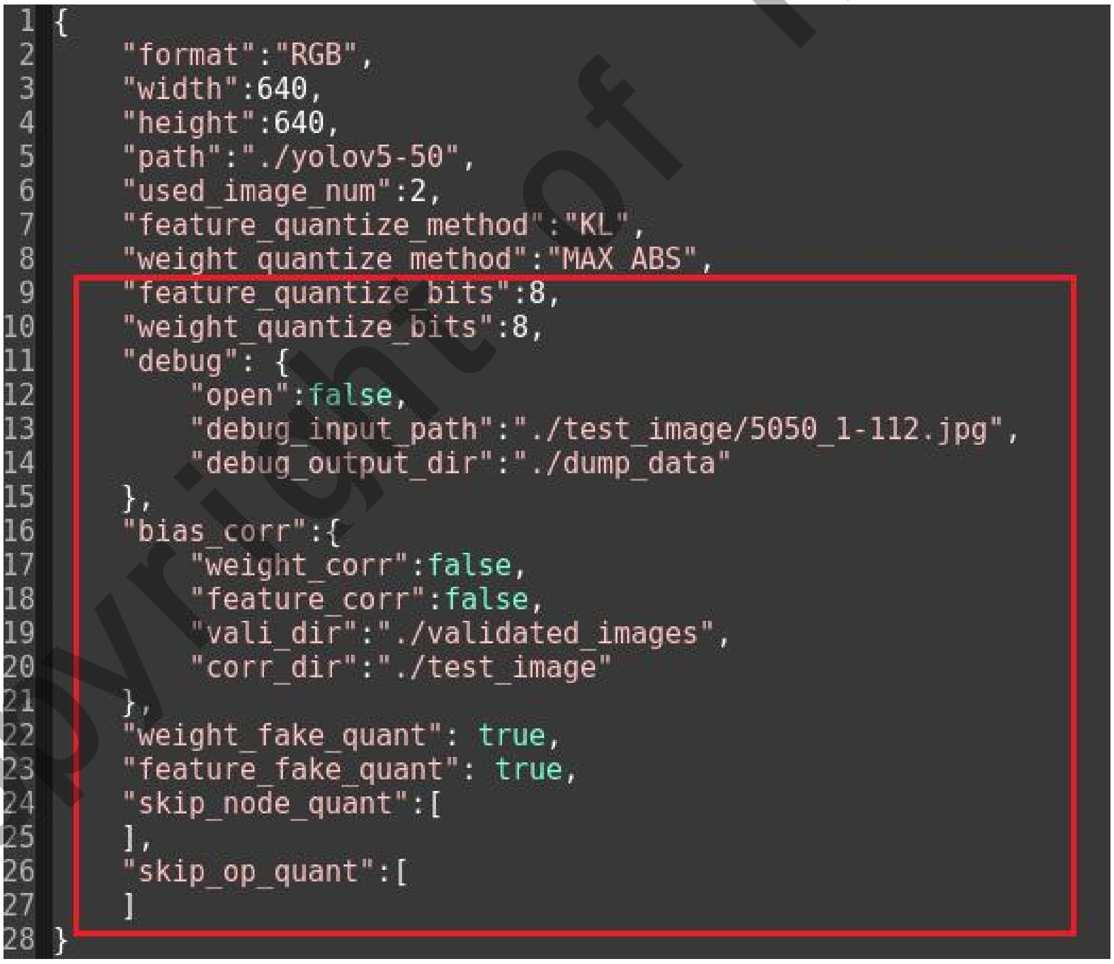
weight_quantize_method ─ weight quantization method: MAX,ADMM

The MagikTransformKit tool gets the correct command line information and json file, and can generate a quantized model with a suffix *.mk.h in magik format and a quantized model with a suffix *.bin, which can be deployed on the board.



```
[[redacted] yolov5s_demo]$ls
magik_model_yolov5s_magik.mk.h  run.sh            yolov5s.onnx
magik-transform-tools           yolov5-20
ptq_yolov5_config.json          yolov5s_magik.bin
```

3.2 json debug option (optional)



```
{
    "format":"RGB",
    "width":640,
    "height":640,
    "path":"./yolov5-50",
    "used_image_num":2,
    "feature_quantize_method":"KL",
    "weight_quantize_method":"MAX_ABS",
    "feature_quantize_bits":8,
    "weight_quantize_bits":8,
    "debug": {
        "open":false,
        "debug_input_path":"./test_image/5050_1-112.jpg",
        "debug_output_dir":"./dump_data"
    },
    "bias_corr":{
        "weight_corr":false,
        "feature_corr":false,
        "vali_dir":"./validated_images",
        "corr_dir":"./test_image"
    },
    "weight_fake_quant": true,
    "feature_fake_quant": true,
    "skip_node_quant":[
    ],
    "skip_op_quant":[
    ]
}
```

feature_quantize_bits ─ feature quantization bit width, optional, default 8 bits

weight_quantize_bits ─ feature quantization bit width, optional, default 8 bits

debug ─ Option to locate the poor inference effect of the quantitative model, optional{

  debug_input_path ─ path of the debug calibration image set

  debug_output_dir ─ path of the debug results

}

weight_fake_quant ─ Simulate weight for quantization effect, optional

feature_fake_quant ─ Simulate feature for quantization effect, optional

bias_corr ─ Adjust the bias information of the network, optional {

  weight_corr ─ quantize the weight in order to adjust the bias information of the network

  feature_corr ─ quantize the feature in order to adjust the bias information of the network

  vali_dir ─ path of the validation image set

  corr_dir ─ path of the correction bias image set

}

skip_node_quant ─ Skip specifying operator name quantization, optional

skip_op_quant ─ Skip specifying operator type quantization, optional

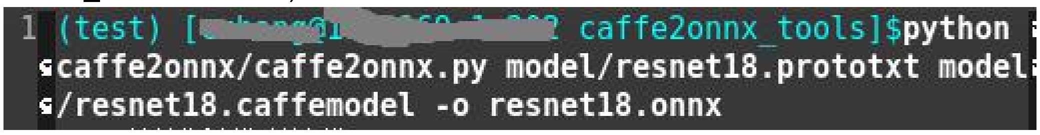
4 yolov5 post-training-quantization demo

Demo path：***magik-toolkit/Models/post/soc_sample/T40/yolov5s/***

4.1 Prepare the model

The MagikTransformKit tool directly supports tensorflow/onnx format models for post-quantization work, and indirectly supports Pytorch/mxnet/caffe format models. These three formats need to be uniformly converted to onnx format.

For the caffe format model, we provide the caffe2onnx tool for conversion work (see caffe2onnx_tools for details)



```
(test) [[redacted] caffe2onnx_tools]$python
caffe2onnx/caffe2onnx.py model/resnet18.prototxt model
/resnet18.caffemodel -o resnet18.onnx
```

caffe2onnx path: ***magik-toolkit/Models/post/caffe2onnx_tools***

The currently used yolov5s model can be downloaded from Google network disk:
Url:

https://drive.google.com/file/d/1AKZy5rqkq5kalJz6tUqNg_kLV2uf43Nt/view?usp=sh

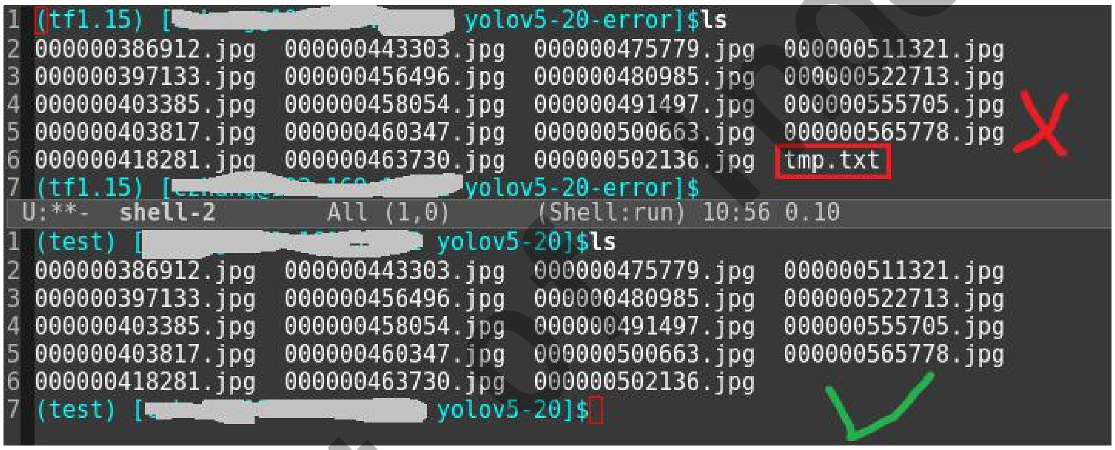
aring

model download to path: ***magik-toolkit/Models/post/yolov5s/***

4.2 Prepare the calibration picture

The calibration pictures in the post-training-quantization process are usually extracted from the training set or the validation set, and the number is between 10 and 100.

Make sure that there are only picture files in the calibration picture directory, and no other files, otherwise the quantization process will go wrong.



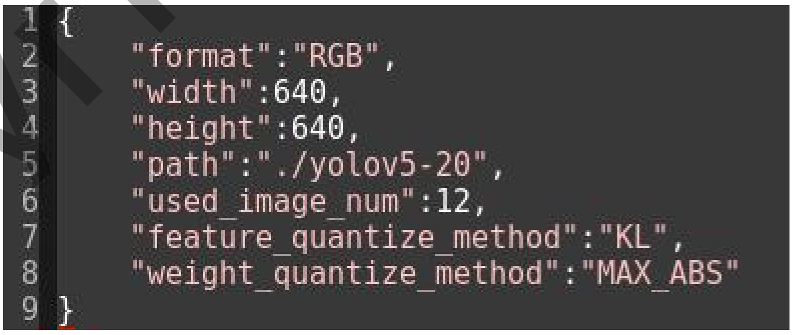
```
(tf1.15) [[redacted] yolov5-20-error]$ls
000000386912.jpg  000000443303.jpg  000000475779.jpg  000000511321.jpg
000000397133.jpg  000000456496.jpg  000000480985.jpg  000000522713.jpg
000000403385.jpg  000000458054.jpg  000000491497.jpg  000000555705.jpg
000000403817.jpg  000000460347.jpg  000000500663.jpg  000000565778.jpg
000000418281.jpg  000000463730.jpg  000000502136.jpg  tmp.txt
(tf1.15) [[redacted] yolov5-20-error]$
 U:**-  shell-2        All (1,0)      (Shell:run) 10:56 0.10
(test) [[redacted] yolov5-20]$ls
000000386912.jpg  000000443303.jpg  000000475779.jpg  000000511321.jpg
000000397133.jpg  000000456496.jpg  000000480985.jpg  000000522713.jpg
000000403385.jpg  000000458054.jpg  000000491497.jpg  000000555705.jpg
000000403817.jpg  000000460347.jpg  000000500663.jpg  000000565778.jpg
000000418281.jpg  000000463730.jpg  000000502136.jpg
(test) [[redacted] yolov5-20]$
```

4.3 Json file

1）Ensure that the format/weight/height parameters are consistent with the training time

2）Specify the path of the quantized pictures (19 in total) and the number of quantized pictures 12

3）Commonly used quantitative(feature:KL,weight:MAX_ABS)

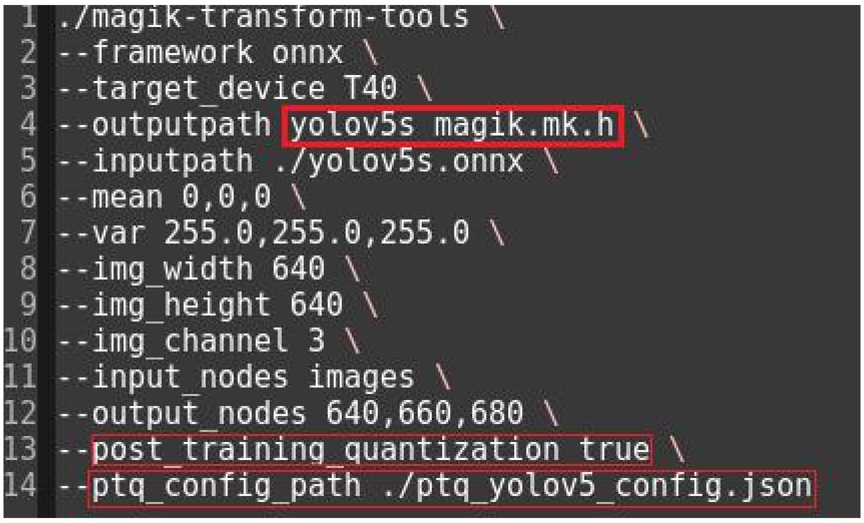


```
{
    "format":"RGB",
    "width":640,
    "height":640,
    "path":"./yolov5-20",
    "used_image_num":12,
    "feature_quantize_method":"KL",
    "weight_quantize_method":"MAX_ABS"
}
```

4.4 Check command line writing

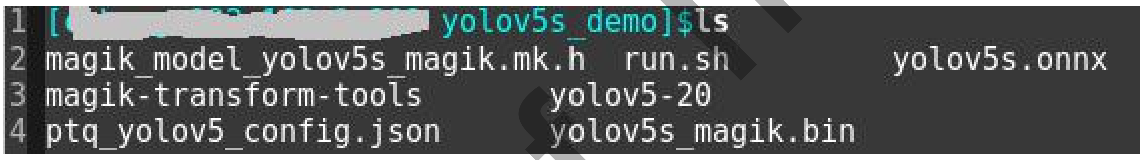
1）Check that the output model parameter suffix is *.mk.h

2）Set the post_training_quantization option to true, and confirm that the input file path of the ptq_config_path option is correct.



```
./magik-transform-tools \
--framework onnx \
--target_device T40 \
--outputpath yolov5s_magik.mk.h \
--inputpath ./yolov5s.onnx \
--mean 0,0,0 \
--var 255.0,255.0,255.0 \
--img_width 640 \
--img_height 640 \
--img_channel 3 \
--input_nodes images \
--output_nodes 640,660,680 \
--post_training_quantization true \
--ptq_config_path ./ptq_yolov5_config.json
```

3）The quantization process of the yolov5 network is performed normally by the MagikTransformKit tool, and the yolov5s_magik.bin file running on the chip will be generated.



```
[[redacted] yolov5s_demo]$ls
magik_model_yolov5s_magik.mk.h  run.sh        yolov5s.onnx
magik-transform-tools      yolov5-20
ptq_yolov5_config.json     yolov5s_magik.bin
```

yolov5s.onnx — Model file to be quantified

yolov5-20 — post-training-quantization using calibration image set

ptq_yolov5_config.json — post-training-quantization using json file

run.sh — Script of convert commands

yolov5s_magik.bin — run file on chip

magik_model_yolov5s_magik.mk.h — run file on chip

5 Run on-board

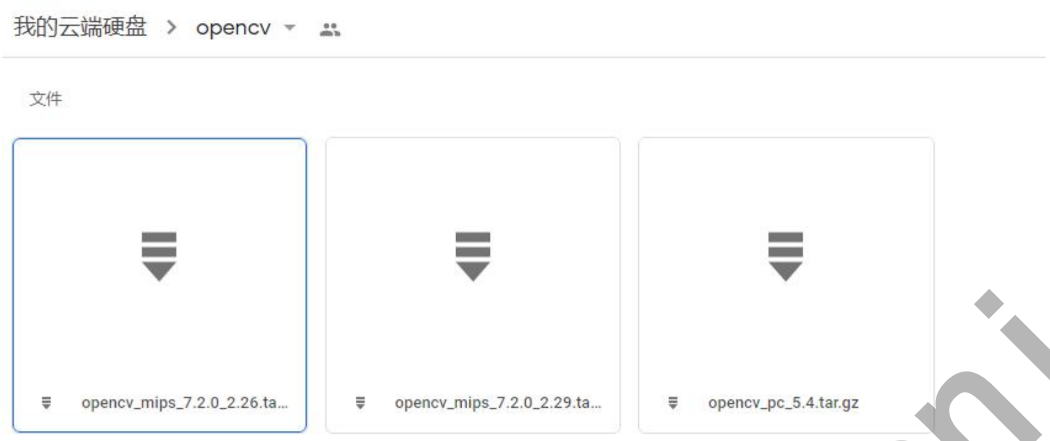
5.1 Code compling

5.1.1 Preparation

First, download the resources compiled by our mips

Link:https://drive.google.com/drive/folders/1Gdj4vF2TpaPzbPNT0H1hIEKDTXEj92Jg?

usp=sharing



Please download and decompress it into magik-toolkit/ThirdParty.

```
$   tar -xvf opencv_mips_7.2.0_2.29.tar.gz
```

Secondly, we also provide inference.cpp, Makefile and test data, which are necessary while running the model on board. In addition, we also need venus library and mips compilation tool. The venus library is located in: "***magik-toolkit/InferenceKit/nna1/***", and the mips compilation tool is provided during debugging process.

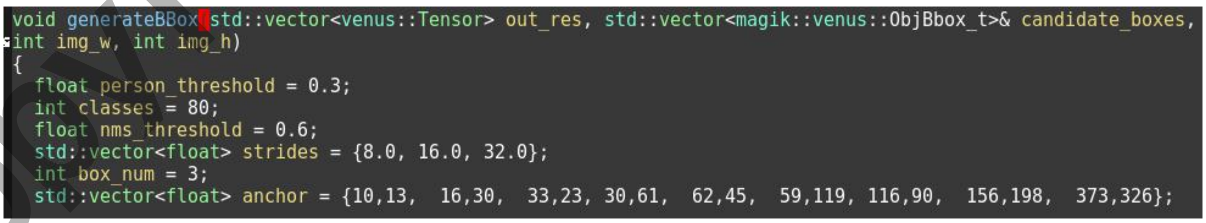
5.1.2 Input of network

Here, in order to demonstrate the process quickly, we've added opencv library functions here to facilitate calling "imread()" and other functions, so we can test jpg diagram directly on board.

5.1.3 Model loading

The model in inference.cpp is passed in through parameters. So, please pay attention to synchronously copying it to the corresponding directory on the board side and passing it in at run time.

5.1.4 Hyperparameter settings



```
void generateBBox(std::vector<venus::Tensor> out_res, std::vector<magik::venus::ObjBbox_t>& candidate_boxes,
int img_w, int img_h)
{
  float person_threshold = 0.3;
  int classes = 80;
  float nms_threshold = 0.6;
  std::vector<float> strides = {8.0, 16.0, 32.0};
  int box_num = 3;
  std::vector<float> anchor = {10,13,  16,30,  33,23, 30,61,  62,45,  59,119, 116,90,  156,198,  373,326};
```

These settings should be consistent with the test code on the PC side.

5.1.5 Compile

TOPDIR -- directory of venus

opencv_path -- path of the compiled opencv library

Libtype -- muclibc or other(glibc)

build_type -- release (Default setting.)

-- profile (Visualization of network structure at runtime and statistics of running time and GOPs at each layer of the network.)

-- debug (Save the results of quantization features of each layer.)

-- nmem (Count the memory usage of nmem when the model is running, and it's in /tmp/nmem_memory.txt.)

Directly compile inference.cpp, then it will generate venus_ yolov5s_ bin_ uclibc_*， which is the executable file we need on the board.

We also provide the code inference_nv12.cpp and input data 10_w1024_ h714. nv12 when you want to use nv12 type input. If necessary, just modify Makefile for compilation and use test.

5.2 Run

Opencv library path: ***magik-toolkit/ThirdParty/opencv_mips_7.2.0_2.29/uclibc/lib***

Venus library path: ***magik-toolkit/InferenceKit/nna1/mips720-glibc229/lib/uclibc/***

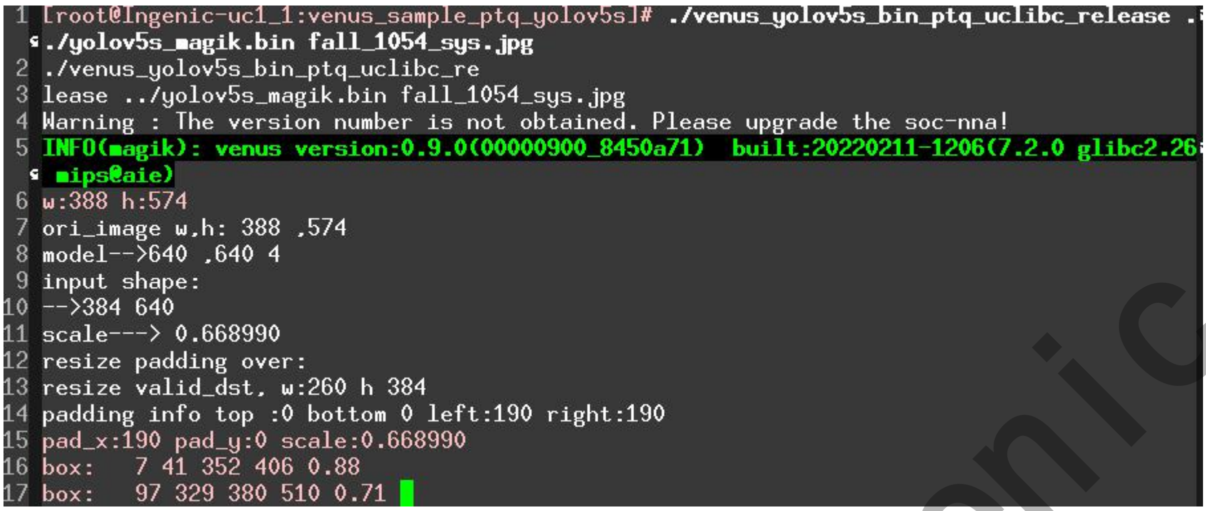
5.2.1 Release(default)

$ **make build_type=release**

(use "**make build_type=release clean**" if you want to clean)

We can get executable file venus_yolov5s_bin_uclibc_release after the compiling process, then copy libvenus.so, venus_yolov5s_bin_uclibc_release, opencv libraries (which consists of libopencv_core.so.3.3, libopencv_highgui.so.3.3, libopencv_imgcodecs.so.3.3, libopencv_imgproc.so.3.3, libopencv_videoio.so.3.3), yolov5s_magik.bin and fall_1054_sys.jpg to board.

When you access the board, please add the library path first (**export LD_LIBRARY_PATH=$lib_path:$LD_LIBRARY_PATH**), after that , you just need to run it on board: "**./venus_yolov5s_bin_uclibc_release yolov5s_magik.bin fall_1054_sys.jpg**". The operation results are shown in the figure below



```
[root@Ingenic-uc1_1:venus_sample_ptq_yolov5s]# ./venus_yolov5s_bin_ptq_uclibc_release .
./yolov5s_magik.bin fall_1054_sys.jpg
./venus_yolov5s_bin_ptq_uclibc_re
lease ../yolov5s_magik.bin fall_1054_sys.jpg
Warning : The version number is not obtained. Please upgrade the soc-nna!
INFO(magik): venus version:0.9.0(00000900_8450a71)  built:20220211-1206(7.2.0 glibc2.26
 mips@aie)
w:388 h:574
ori_image w.h: 388 ,574
model-->640 ,640 4
input shape:
-->384 640
scale---> 0.668990
resize padding over:
resize valid_dst, w:260 h 384
padding info top :0 bottom 0 left:190 right:190
pad_x:190 pad_y:0 scale:0.668990
box:   7 41 352 406 0.88
box:   97 329 380 510 0.71
```

5.2.2 Debug（Check data）

See step 7.2 of the “Magik Training Quantization User Guide” for details, the processing of the input is a little different.

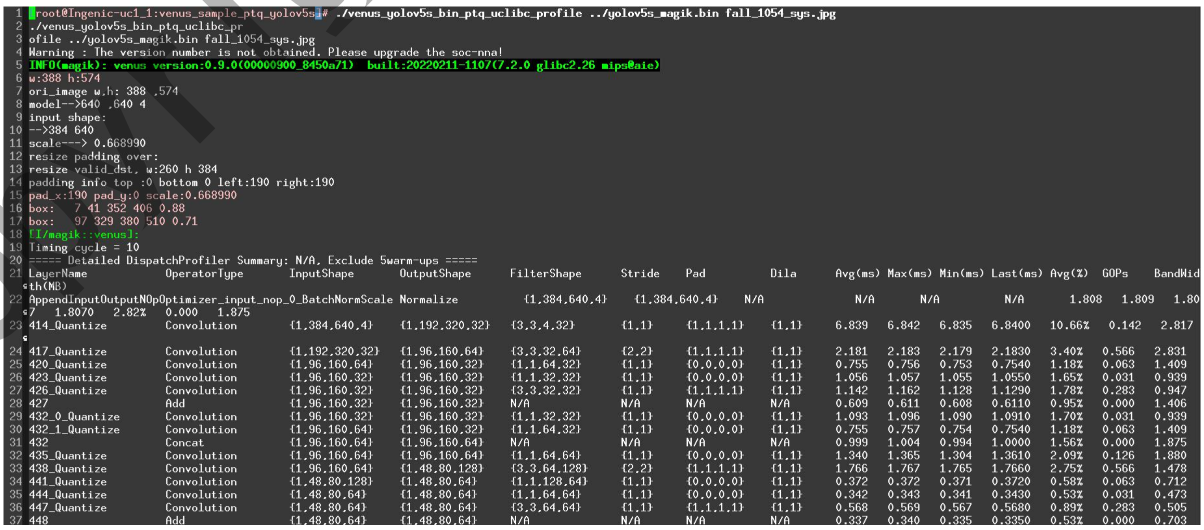
5.2.3 Profile (network visualization)

$ **make build_type=profile**

(use “**make build_type=profile clean**” if you want to clean)

We can get executable file ***venus_yolov5s_bin_uclibc_profile*** after compiling process, then copy libvenus.p.so, venus_yolov5s_bin_uclibc_profile, opencv library(which consists of libopencv_core.so.3.3, libopencv_highgui.so.3.3, libopencv_imgcodecs.so.3.3, libopencv_imgproc.so.3.3, libopencv_videoio.so.3.3), yolov5s_magik.bin and fall_1054_sys.jpg to board.

When you access the board, please add library path first (**export LD_LIBRARY_PATH=$lib_path:$LD_LIBRARY_PATH**), then run the program on the board with: “**./venus_yolov5s_bin_uclibc_profile yolov5s_magik.bin fall_1054_sys.jpg**”. The operation results are shown in the figure below.



```
[root@Ingenic-uc1_1:venus_sample_ptq_yolov5s]# ./venus_yolov5s_bin_ptq_uclibc_profile ../yolov5s_magik.bin fall_1054_sys.jpg
./venus_yolov5s_bin_ptq_uclibc_pr
ofile ../yolov5s_magik.bin fall_1054_sys.jpg
Warning : The version number is not obtained. Please upgrade the soc-nna!
INFO(magik): venus version:0.9.0(00000900_8450a71)  built:20220211-1107(7.2.0 glibc2.26 mips@aie)
w:388 h:574
ori_image w.h: 388 ,574
model-->640 ,640 4
input shape:
-->384 640
scale---> 0.668990
resize padding over:
resize valid_dst, w:260 h 384
padding info top :0 bottom 0 left:190 right:190
pad_x:190 pad_y:0 scale:0.668990
box:   7 41 352 406 0.88
box:   97 329 380 510 0.71
[I/magik::venus]:
Timing cycle = 10
===== Detailed DispatchProfiler Summary: N/A, Exclude 5warm-ups =====
LayerName            OperatorType       InputShape       OutputShape      FilterShape      Stride    Pad          Dila       Avg(ms) Max(ms) Min(ms) Last(ms) Avg(%)  GOPs     BandWid
th(MB)
AppendInputOutputNOpOptimizer_input_nop_0_BatchNormScale Normalize        {1,384,640,4}    {1,384,640,4}    N/A          N/A        N/A        N/A        1.808   1.809   1.80
7    1.8070    2.82%    0.000    1.875
414_Quantize         Convolution        {1,384,640,4}    {1,192,320,32}   {3,3,4,32}       {1,1}     {1,1,1,1}    {1,1}      6.839   6.842   6.835   6.8400   10.66%   0.142    2.817

417_Quantize         Convolution        {1,192,320,32}   {1,96,160,64}    {3,3,32,64}      {2,2}     {1,1,1,1}    {1,1}      2.181   2.183   2.179   2.1830   3.40%    0.566    2.831
420_Quantize         Convolution        {1,96,160,64}    {1,96,160,32}    {1,1,64,32}      {1,1}     {0,0,0,0}    {1,1}      0.755   0.756   0.753   0.7540   1.18%    0.063    1.409
423_Quantize         Convolution        {1,96,160,32}    {1,96,160,32}    {1,1,32,32}      {1,1}     {0,0,0,0}    {1,1}      1.056   1.057   1.055   1.0550   1.65%    0.031    0.939
426_Quantize         Convolution        {1,96,160,32}    {1,96,160,32}    {3,3,32,32}      {1,1}     {1,1,1,1}    {1,1}      1.142   1.162   1.128   1.1290   1.78%    0.283    0.947
427                  Add                {1,96,160,32}    {1,96,160,32}    N/A              N/A       N/A          N/A        0.609   0.611   0.608   0.6110   0.95%    0.000    1.406
432_0_Quantize       Convolution        {1,96,160,32}    {1,96,160,32}    {1,1,32,32}      {1,1}     {0,0,0,0}    {1,1}      1.093   1.096   1.090   1.0910   1.70%    0.031    0.939
432_1_Quantize       Convolution        {1,96,160,64}    {1,96,160,32}    {1,1,64,32}      {1,1}     {0,0,0,0}    {1,1}      0.755   0.757   0.754   0.7540   1.18%    0.063    1.409
432                  Concat             {1,96,160,64}    {1,96,160,64}    N/A              N/A       N/A          N/A        0.999   1.004   0.994   1.0000   1.56%    0.000    1.875
435_Quantize         Convolution        {1,96,160,64}    {1,96,160,64}    {1,1,64,64}      {1,1}     {0,0,0,0}    {1,1}      1.340   1.365   1.304   1.3610   2.09%    0.126    1.880
438_Quantize         Convolution        {1,96,160,64}    {1,48,80,128}    {3,3,64,128}     {2,2}     {1,1,1,1}    {1,1}      1.766   1.767   1.765   1.7660   2.75%    0.566    1.478
441_Quantize         Convolution        {1,48,80,128}    {1,48,80,64}     {1,1,128,64}     {1,1}     {0,0,0,0}    {1,1}      0.372   0.372   0.371   0.3720   0.58%    0.063    0.712
444_Quantize         Convolution        {1,48,80,64}     {1,48,80,64}     {1,1,64,64}      {1,1}     {0,0,0,0}    {1,1}      0.342   0.343   0.341   0.3430   0.53%    0.031    0.473
447_Quantize         Convolution        {1,48,80,64}     {1,48,80,64}     {3,3,64,64}      {1,1}     {1,1,1,1}    {1,1}      0.568   0.569   0.567   0.5680   0.89%    0.283    0.505
448                  Add                {1,48,80,64}     {1,48,80,64}     N/A              N/A       N/A          N/A        0.337   0.340   0.335   0.3350   0.53%    0.000    0.703
```

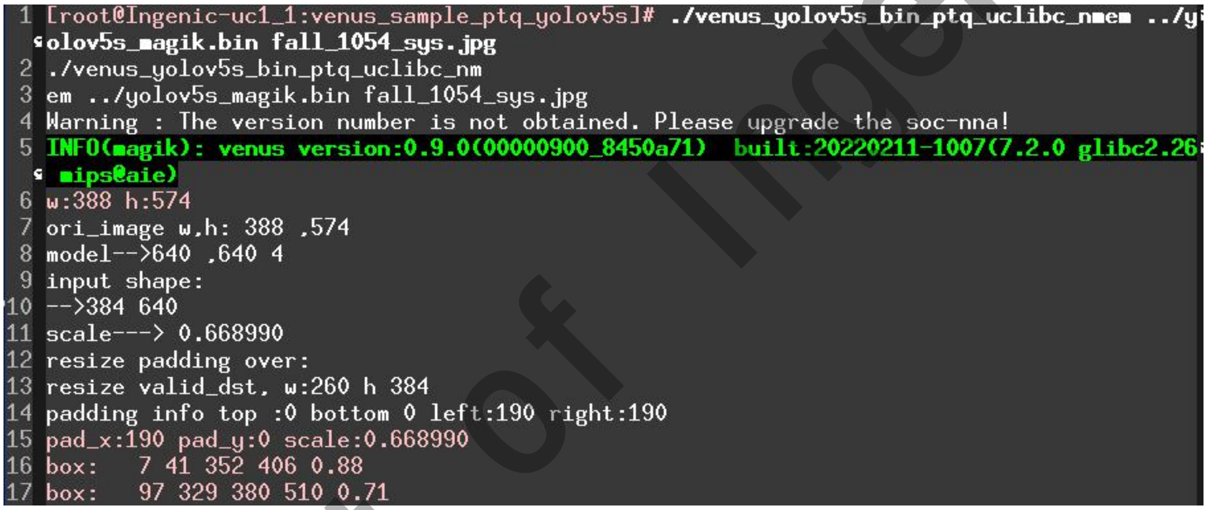
5.2.4 Nmem(nmem occupancy)

$ **make build_type=nmem**

(use "**make build_type=nmemclean**" if you want to clean)

We can get executable file ***venus_yolov5s_bin_uclibc_release*** after compiling process, then copy libvenus.m.so, venus_yolov5s_bin_uclibc_nmem, opencv library(which consists of libopencv_core.so.3.3, libopencv_highgui.so.3.3, libopencv_imgcodecs.so.3.3, libopencv_imgproc.so.3.3, libopencv_videoio.so.3.3), yolov5s_magik.bin and fall_1054_sys.jpg to board.

When you access the board, please add library path first (**export LD_LIBRARY_PATH=$lib_path:$LD_LIBRARY_PATH**), then run the program on the board with: "**./venus_yolov5s_bin_uclibc_nmem yolov5s_magik.bin bus.jpg**". The operation results are shown in the figure below.



```
[root@Ingenic-uc1_1:venus_sample_ptq_yolov5s]# ./venus_yolov5s_bin_ptq_uclibc_nmem ../y
olov5s_magik.bin fall_1054_sys.jpg
./venus_yolov5s_bin_ptq_uclibc_nm
em ../yolov5s_magik.bin fall_1054_sys.jpg
Warning : The version number is not obtained. Please upgrade the soc-nna!
INFO(magik): venus version:0.9.0(00000900_8450a71)  built:20220211-1007(7.2.0 glibc2.26
 mips@aie)
w:388 h:574
ori_image w,h: 388 ,574
model-->640 ,640 4
input shape:
-->384 640
scale---> 0.668990
resize padding over:
resize valid_dst, w:260 h 384
padding info top :0 bottom 0 left:190 right:190
pad_x:190 pad_y:0 scale:0.668990
box:   7 41 352 406 0.88
box:   97 329 380 510 0.71
```

6 Check data

To verify whether the data on the PC side and the chip side can be aligned, follow the steps below:

6.1 On PC

Use the PC-side reasoning library provided by Ingenic, by setting the relevant environment variables. The operator layer-by-layer data can be saved as a binary file and compared with the binary file after the SOC has opened the corresponding mode. The specific operation steps on the PC side are as follows:

In order to keep the input data highly consistent, we use the opencv library as an image parsing tool. If you need to compare the data consistency between python and pc, we recommend that you also use opencv as a tool for parsing images on the python side. Therefore, we have sorted out the necessary libraries and models for running PC-side tools and placed them on Google Cloud.

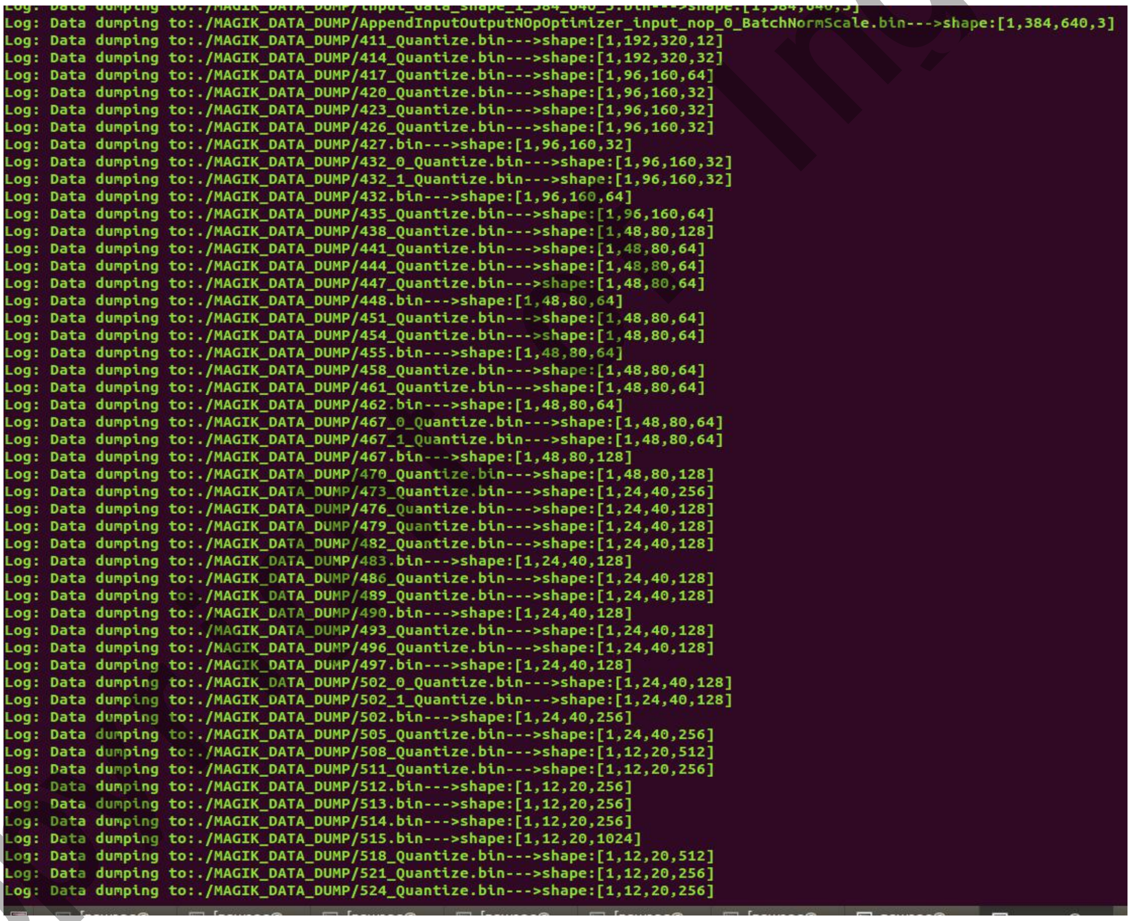
First, you need to switch to the ***magik-toolkit/Models/post/pc_inference/*** folder, and set the environment variable on the PC side

```
export MAGIK_CPP_DUMPDATA=true
```

The quantization results of each layer can be saved to the ./MAGIK_DATA_DUMP/ folder. The specific running command is:

```
export   MAGIK_CPP_DUMPDATA=true
make   clean
make   -j8
./pc_inference_bin   ./model_quant.mgk   fall_1054_sys.jpg   640 384
```

when the conversion tool converts the model, ***fall_1054_sys.jpg*** is the input image, and 640 and 384 correspond to the input width and height respectively. Finally save ***./MAGIK_DATA_DUMP/*** (see the figure below for details), it can be seen that the input layer is saved as ***input_data_shape_1_384_640_3.bin***, and the following shape naming rules are n, h, w, c, that is, the height is 384, the width is 640, and the last is the output layer; if you don't want to do it step by step, you can run our provided ***extractor.sh*** script to run it in one click.



```
Log: Data dumping to:./MAGIK_DATA_DUMP/input_data_shape_1_384_640_3.bin--->shape:[1,384,640,3]
Log: Data dumping to:./MAGIK_DATA_DUMP/AppendInputOutputNOpOptimizer_input_nop_0_BatchNormScale.bin--->shape:[1,384,640,3]
Log: Data dumping to:./MAGIK_DATA_DUMP/411_Quantize.bin--->shape:[1,192,320,12]
Log: Data dumping to:./MAGIK_DATA_DUMP/414_Quantize.bin--->shape:[1,192,320,32]
Log: Data dumping to:./MAGIK_DATA_DUMP/417_Quantize.bin--->shape:[1,96,160,64]
Log: Data dumping to:./MAGIK_DATA_DUMP/420_Quantize.bin--->shape:[1,96,160,32]
Log: Data dumping to:./MAGIK_DATA_DUMP/423_Quantize.bin--->shape:[1,96,160,32]
Log: Data dumping to:./MAGIK_DATA_DUMP/426_Quantize.bin--->shape:[1,96,160,32]
Log: Data dumping to:./MAGIK_DATA_DUMP/427.bin--->shape:[1,96,160,32]
Log: Data dumping to:./MAGIK_DATA_DUMP/432_0_Quantize.bin--->shape:[1,96,160,32]
Log: Data dumping to:./MAGIK_DATA_DUMP/432_1_Quantize.bin--->shape:[1,96,160,32]
Log: Data dumping to:./MAGIK_DATA_DUMP/432.bin--->shape:[1,96,160,64]
Log: Data dumping to:./MAGIK_DATA_DUMP/435_Quantize.bin--->shape:[1,96,160,64]
Log: Data dumping to:./MAGIK_DATA_DUMP/438_Quantize.bin--->shape:[1,48,80,128]
Log: Data dumping to:./MAGIK_DATA_DUMP/441_Quantize.bin--->shape:[1,48,80,64]
Log: Data dumping to:./MAGIK_DATA_DUMP/444_Quantize.bin--->shape:[1,48,80,64]
Log: Data dumping to:./MAGIK_DATA_DUMP/447_Quantize.bin--->shape:[1,48,80,64]
Log: Data dumping to:./MAGIK_DATA_DUMP/448.bin--->shape:[1,48,80,64]
Log: Data dumping to:./MAGIK_DATA_DUMP/451_Quantize.bin--->shape:[1,48,80,64]
Log: Data dumping to:./MAGIK_DATA_DUMP/454_Quantize.bin--->shape:[1,48,80,64]
Log: Data dumping to:./MAGIK_DATA_DUMP/455.bin--->shape:[1,48,80,64]
Log: Data dumping to:./MAGIK_DATA_DUMP/458_Quantize.bin--->shape:[1,48,80,64]
Log: Data dumping to:./MAGIK_DATA_DUMP/461_Quantize.bin--->shape:[1,48,80,64]
Log: Data dumping to:./MAGIK_DATA_DUMP/462.bin--->shape:[1,48,80,64]
Log: Data dumping to:./MAGIK_DATA_DUMP/467_0_Quantize.bin--->shape:[1,48,80,64]
Log: Data dumping to:./MAGIK_DATA_DUMP/467_1_Quantize.bin--->shape:[1,48,80,64]
Log: Data dumping to:./MAGIK_DATA_DUMP/467.bin--->shape:[1,48,80,128]
Log: Data dumping to:./MAGIK_DATA_DUMP/470_Quantize.bin--->shape:[1,48,80,128]
Log: Data dumping to:./MAGIK_DATA_DUMP/473_Quantize.bin--->shape:[1,24,40,256]
Log: Data dumping to:./MAGIK_DATA_DUMP/476_Quantize.bin--->shape:[1,24,40,128]
Log: Data dumping to:./MAGIK_DATA_DUMP/479_Quantize.bin--->shape:[1,24,40,128]
Log: Data dumping to:./MAGIK_DATA_DUMP/482_Quantize.bin--->shape:[1,24,40,128]
Log: Data dumping to:./MAGIK_DATA_DUMP/483.bin--->shape:[1,24,40,128]
Log: Data dumping to:./MAGIK_DATA_DUMP/486_Quantize.bin--->shape:[1,24,40,128]
Log: Data dumping to:./MAGIK_DATA_DUMP/489_Quantize.bin--->shape:[1,24,40,128]
Log: Data dumping to:./MAGIK_DATA_DUMP/490.bin--->shape:[1,24,40,128]
Log: Data dumping to:./MAGIK_DATA_DUMP/493_Quantize.bin--->shape:[1,24,40,128]
Log: Data dumping to:./MAGIK_DATA_DUMP/496_Quantize.bin--->shape:[1,24,40,128]
Log: Data dumping to:./MAGIK_DATA_DUMP/497.bin--->shape:[1,24,40,128]
Log: Data dumping to:./MAGIK_DATA_DUMP/502_0_Quantize.bin--->shape:[1,24,40,128]
Log: Data dumping to:./MAGIK_DATA_DUMP/502_1_Quantize.bin--->shape:[1,24,40,128]
Log: Data dumping to:./MAGIK_DATA_DUMP/502.bin--->shape:[1,24,40,256]
Log: Data dumping to:./MAGIK_DATA_DUMP/505_Quantize.bin--->shape:[1,24,40,256]
Log: Data dumping to:./MAGIK_DATA_DUMP/508_Quantize.bin--->shape:[1,12,20,512]
Log: Data dumping to:./MAGIK_DATA_DUMP/511_Quantize.bin--->shape:[1,12,20,256]
Log: Data dumping to:./MAGIK_DATA_DUMP/512.bin--->shape:[1,12,20,256]
Log: Data dumping to:./MAGIK_DATA_DUMP/513.bin--->shape:[1,12,20,256]
Log: Data dumping to:./MAGIK_DATA_DUMP/514.bin--->shape:[1,12,20,256]
Log: Data dumping to:./MAGIK_DATA_DUMP/515.bin--->shape:[1,12,20,1024]
Log: Data dumping to:./MAGIK_DATA_DUMP/518_Quantize.bin--->shape:[1,12,20,512]
Log: Data dumping to:./MAGIK_DATA_DUMP/521_Quantize.bin--->shape:[1,12,20,256]
Log: Data dumping to:./MAGIK_DATA_DUMP/524_Quantize.bin--->shape:[1,12,20,256]
```

6.2 On chip

(1) The input data required by the board side is 4-channel data format. For the “fall_1054_sys.jpg” tested on the PC side above, we use the input “input_data_shape_1_384_640_3.bin” saved at runtime to process it into .h and use it for the board side test to ensure that the inputs of the two

are exactly the same , you can do offline processing first:

Script in:

magik-toolkit//Models/post/soc_sample/T40/yolov5s/venus_sample_ptq_yolov5s/generate_img_input.py:

python generate_img_input.py bin_path w h c bin

bin_path -- The bin file to be converted, that is, the input bin saved by the PC side operation

w, h --The width and height of the bin file correspond to the bin file name, which are 1,h,w,c

bin -- The flag sign indicates that the header file is turned from the bin file, and the other is the img header file.

Note: If you use bin, you can completely ensure that the input on both sides is consistent, which is convenient for data.

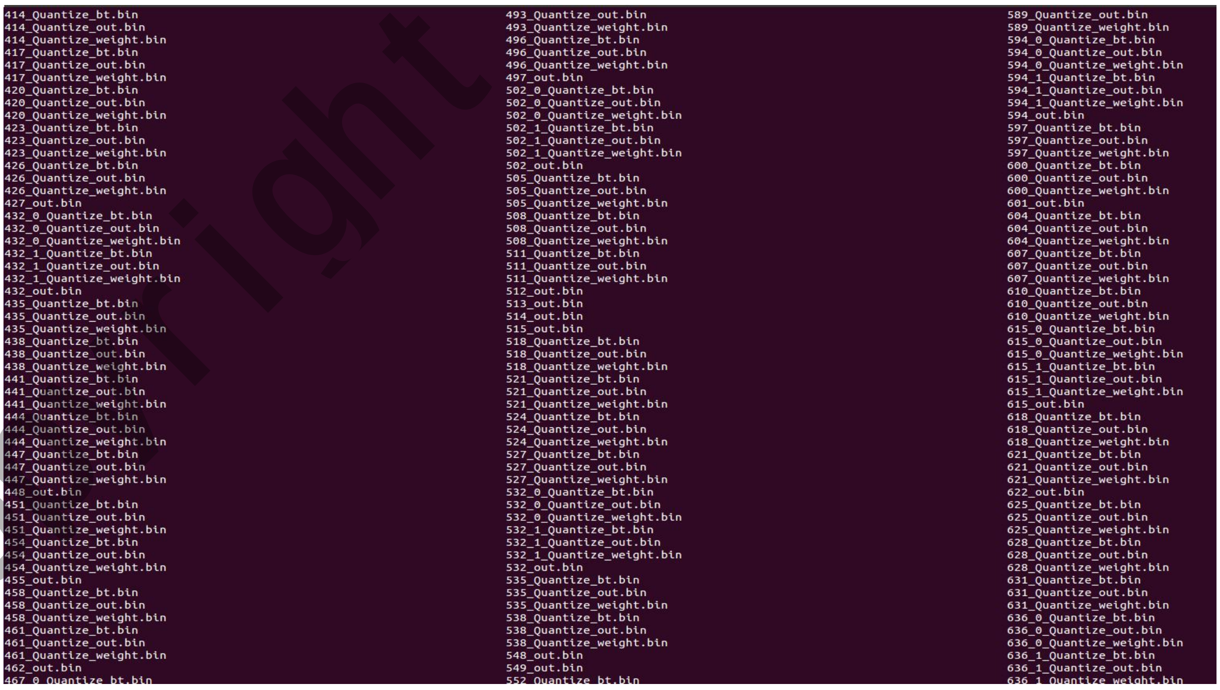
In the provided example, the model is passed in through parameters (argv[1]). When running, pay attention to synchronously copying it to the corresponding directory on the board, and passing it in as needed at run time. The width (w) and height (h) are also passed in from the outside, and the specific values are the same as those in (1).

Compile c code: **make build_type=debug**

The executable file, "venus_yolov5s_bin_ptq_uclibc_debug" will be generated in the current folder, then copy the venus library (libvenus.d.so), the executable file (venus_yolov5s_bin_ptq_uclibc_debug), model file (yolov5s_magik.bin) to the development board, finally run:

./venus_yolov5s_bin_ptq_uclibc_debug yolov5s_magik.bin 640 383

Among them, 640 corresponds to w, 383 corresponds to h, and the output and other information of each layer will be automatically saved during operation (as shown in the following figure)



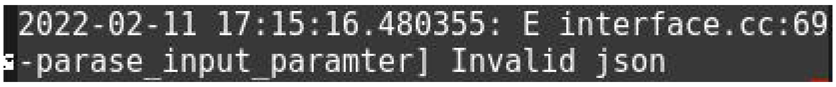
```
414_Quantize_bt.bin               493_Quantize_out.bin               589_Quantize_out.bin
414_Quantize_out.bin              493_Quantize_weight.bin            589_Quantize_weight.bin
414_Quantize_weight.bin           496_Quantize_bt.bin                594_0_Quantize_bt.bin
417_Quantize_bt.bin               496_Quantize_out.bin               594_0_Quantize_out.bin
417_Quantize_out.bin              496_Quantize_weight.bin            594_0_Quantize_weight.bin
417_Quantize_weight.bin           497_out.bin                        594_1_Quantize_bt.bin
420_Quantize_bt.bin               502_0_Quantize_bt.bin              594_1_Quantize_out.bin
420_Quantize_out.bin              502_0_Quantize_out.bin             594_1_Quantize_weight.bin
420_Quantize_weight.bin           502_0_Quantize_weight.bin          594_out.bin
423_Quantize_bt.bin               502_1_Quantize_bt.bin              597_Quantize_bt.bin
423_Quantize_out.bin              502_1_Quantize_out.bin             597_Quantize_out.bin
423_Quantize_weight.bin           502_1_Quantize_weight.bin          597_Quantize_weight.bin
426_Quantize_bt.bin               502_out.bin                        600_Quantize_bt.bin
426_Quantize_out.bin              505_Quantize_bt.bin                600_Quantize_out.bin
426_Quantize_weight.bin           505_Quantize_out.bin               600_Quantize_weight.bin
427_out.bin                       505_Quantize_weight.bin            601_out.bin
432_0_Quantize_bt.bin             508_Quantize_bt.bin                604_Quantize_bt.bin
432_0_Quantize_out.bin            508_Quantize_out.bin               604_Quantize_out.bin
432_0_Quantize_weight.bin         508_Quantize_weight.bin            604_Quantize_weight.bin
432_1_Quantize_bt.bin             511_Quantize_bt.bin                607_Quantize_bt.bin
432_1_Quantize_out.bin            511_Quantize_out.bin               607_Quantize_out.bin
432_1_Quantize_weight.bin         511_Quantize_weight.bin            607_Quantize_weight.bin
432_out.bin                       512_out.bin                        610_Quantize_bt.bin
435_Quantize_bt.bin               513_out.bin                        610_Quantize_out.bin
435_Quantize_out.bin              514_out.bin                        610_Quantize_weight.bin
435_Quantize_weight.bin           515_out.bin                        615_0_Quantize_bt.bin
438_Quantize_bt.bin               518_Quantize_bt.bin                615_0_Quantize_out.bin
438_Quantize_out.bin              518_Quantize_out.bin               615_0_Quantize_weight.bin
438_Quantize_weight.bin           518_Quantize_weight.bin            615_1_Quantize_bt.bin
441_Quantize_bt.bin               521_Quantize_bt.bin                615_1_Quantize_out.bin
441_Quantize_out.bin              521_Quantize_out.bin               615_1_Quantize_weight.bin
441_Quantize_weight.bin           521_Quantize_weight.bin            615_out.bin
444_Quantize_bt.bin               524_Quantize_bt.bin                618_Quantize_bt.bin
444_Quantize_out.bin              524_Quantize_out.bin               618_Quantize_out.bin
444_Quantize_weight.bin           524_Quantize_weight.bin            618_Quantize_weight.bin
447_Quantize_bt.bin               527_Quantize_bt.bin                621_Quantize_bt.bin
447_Quantize_out.bin              527_Quantize_out.bin               621_Quantize_out.bin
447_Quantize_weight.bin           527_Quantize_weight.bin            621_Quantize_weight.bin
448_out.bin                       532_0_Quantize_bt.bin              622_out.bin
451_Quantize_bt.bin               532_0_Quantize_out.bin             625_Quantize_bt.bin
451_Quantize_out.bin              532_0_Quantize_weight.bin          625_Quantize_out.bin
451_Quantize_weight.bin           532_1_Quantize_bt.bin              625_Quantize_weight.bin
454_Quantize_bt.bin               532_1_Quantize_out.bin             628_Quantize_bt.bin
454_Quantize_out.bin              532_1_Quantize_weight.bin          628_Quantize_out.bin
454_Quantize_weight.bin           532_out.bin                        628_Quantize_weight.bin
455_out.bin                       535_Quantize_bt.bin                631_Quantize_bt.bin
458_Quantize_bt.bin               535_Quantize_out.bin               631_Quantize_out.bin
458_Quantize_out.bin              535_Quantize_weight.bin            631_Quantize_weight.bin
458_Quantize_weight.bin           538_Quantize_bt.bin                636_0_Quantize_bt.bin
461_Quantize_bt.bin               538_Quantize_out.bin               636_0_Quantize_out.bin
461_Quantize_out.bin              538_Quantize_weight.bin            636_0_Quantize_weight.bin
461_Quantize_weight.bin           548_out.bin                        636_1_Quantize_bt.bin
462_out.bin                       549_out.bin                        636_1_Quantize_out.bin
467_0_Quantize_bt.bin             552_Quantize_bt.bin                636_1_Quantize_weight.bin
```

Among them, ***_*_QuantizeFeature_out.bin*** and PC-side runtime are saved in ***_*_QuantizeFeature.bin*** under ***./MAGIK_DATA_DUMP/***

There is a one-to-one correspondence, and you can directly check whether the md5 value is consistent. In the case of ensuring that the input is completely consistent, there are layers in the middle that do not correspond to timely feedback.

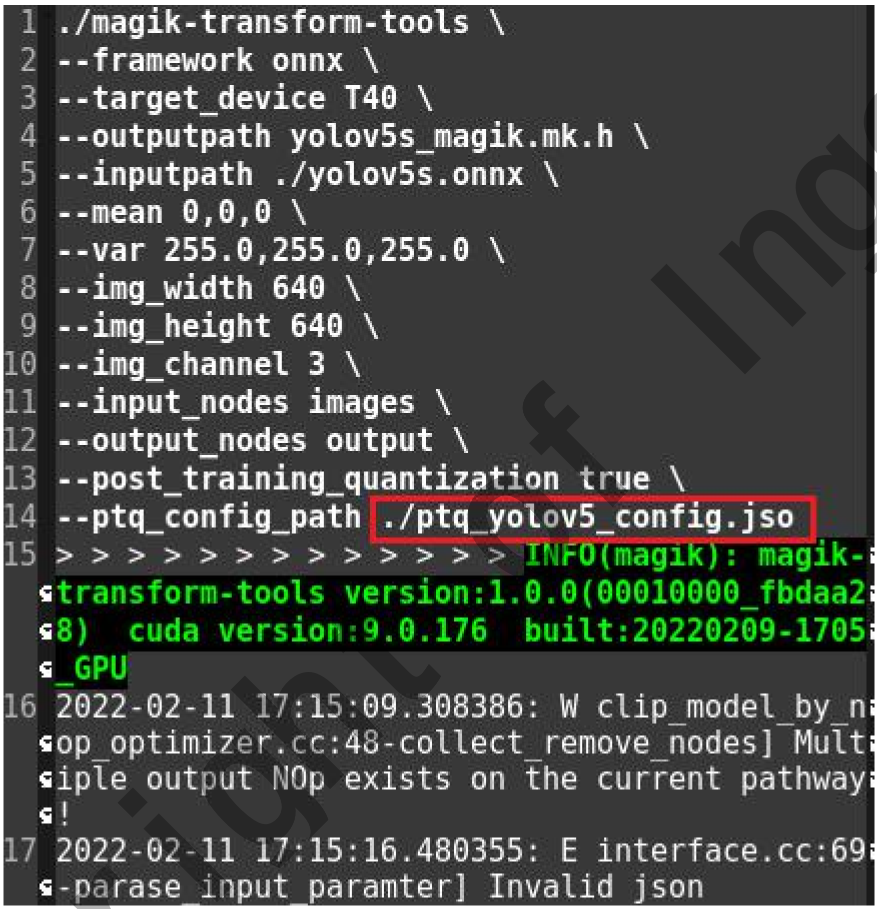
7 Frequently Asked Questions

Q: The conversion process prompts invalid json



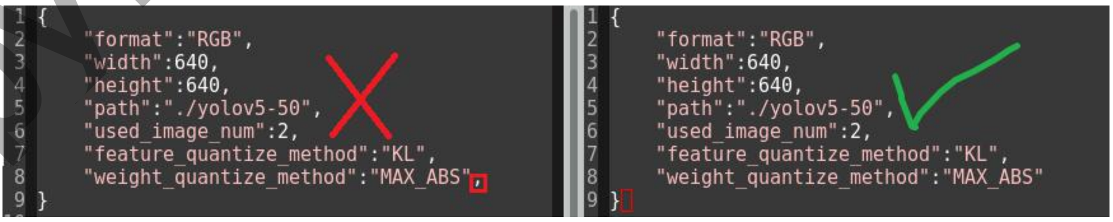
```
2022-02-11 17:15:16.480355: E interface.cc:69
-parase_input_paramter] Invalid json
```

Answer: Check if the json file path and command are correct



```
./magik-transform-tools \
--framework onnx \
--target_device T40 \
--outputpath yolov5s_magik.mk.h \
--inputpath ./yolov5s.onnx \
--mean 0,0,0 \
--var 255.0,255.0,255.0 \
--img_width 640 \
--img_height 640 \
--img_channel 3 \
--input_nodes images \
--output_nodes output \
--post_training_quantization true \
--ptq_config_path ./ptq_yolov5_config.jso
> > > > > > > > > > > > > INFO(magik): magik-
transform-tools version:1.0.0(00010000_fbdaa2
8)  cuda version:9.0.176  built:20220209-1705
_GPU
2022-02-11 17:15:09.308386: W clip_model_by_n
op_optimizer.cc:48-collect_remove_nodes] Mult
iple output NOp exists on the current pathway
!
2022-02-11 17:15:16.480355: E interface.cc:69
-parase_input_paramter] Invalid json
```

, and check if the json format is correct



```
{
    "format":"RGB",
    "width":640,
    "height":640,
    "path":"./yolov5-50",
    "used_image_num":2,
    "feature_quantize_method":"KL",
    "weight_quantize_method":"MAX_ABS",
}
```

```
{
    "format":"RGB",
    "width":640,
    "height":640,
    "path":"./yolov5-50",
    "used_image_num":2,
    "feature_quantize_method":"KL",
    "weight_quantize_method":"MAX_ABS"
}
```

8 json debug process

Turn on the debug option, MagikTransformKit will simulate the network native model (onnx or Tensorflow) inference and chip-side inference, save the inference results of each layer of operators, and calculate the errors of each layer of operators (cosine similarity, Euclidean distance, absolute maximum value).

After the network is quantized, the accuracy loss is within the acceptable range of 0.05% to 1%. Usually, the cosine similarity performance is as follows:

A. The cosine similarity of the first layer to the last layer is 99.0%+;

B. Or there is a little 98.0%+ in the middle layer, and the cosine similarity of the first and last layers is 99.0%+.

8.1 The phenomenon of poor effect

Using the json debug option of MagikTransformKit to locate the problem, the phenomenon can be divided into the following four types:

A. The first layer operator cosine_avg does not reach 99.0%+

B. The quantization of the first layer reaches 99.0%+, a certain layer begins to decline, and the cosine similarity of the last layer is lower than 95.0%

C. The quantization of the first and middle layers is normal, and the quantization cosine similarity of the last few layers decreases

D. The cosine similarity performance is normal, and the chip-side verification effect is poor

8.2 Solution

For the above summary phenomenon information, we organize the steps and methods to locate and solve the problem.

A. The first layer operator cosine_avg does not reach 99.0%+ or max_absolu is very large:

Reason: There is a problem with the network input. Under normal circumstances, the cosine_avg of the first layer operator is greater than 99%

Solution: Check the image format (BRG, RGB) and mean variance in the json file to ensure that these parameters are consistent with training

B. The quantization of the first layer reaches 99.0%+, a certain layer begins to decline, and the cosine similarity of the last layer is lower than 95.0%:

Reason 1: Quantization brings errors, which must exist

Solution: Use the feature_quantize_bit or weight_quantize_bit option to increase the bit width (set to 10, 12, 14, default 8 bits)

Reason 2: There is a problem with the quantization method of a certain layer of operators

Solution: Use the skip_node_quant option to skip the serious layer of cosine_avg drop, after using the mechanism, the cosine similarity of other layers is normal

```
"skip_node_quant":[
      "node_name_0"
],
```

Feed the problem back to the MagikTransformKit docking staff, and at the same time send the dump results of the input and output of the problematic layer under the path "result/path/ptq_model" and the path "result/path/float_model", and send the quantization parameters of the problem layer to the docking staff

Reason 3: Problems simulating native network inference

MagikTransformKit has problems simulating network native model (onnx or tf) inference. Use the "./result/path/input_uint8.bin" data generated by the conversion tool as the input of the native model inference, and save the results of the native model inference; the generated results and the "result/path/float_model" path store the simulation results (float binary form NHWC) to compare and confirm whether there is a problem in simulating the native model inference;

If there is a problem and report it to the docking staff, send the input and output bin files of the problem layer and the correct output file under the path "result/path/float_model" to the docking staff to fix the problem.

C. The cosine similarity of the first layer and the middle layer is normal, and the quantized cosine similarity of the last few layers decreases:

Reason 1: Refer to phenomenon 2)

Reason 2: There is a cumulative error in network quantization, and the error caused by layer-by-layer quantization is obvious in a certain layer

Solution: Confirm the error caused by weight quantization or feature quantization through the "weight_fake_quant" option and the "feature_fake_quant" option, and use the "bias_corr" option to eliminate the error

```
"feature_fake_quant": true,
"weight_fake_quant": true,
"bias_corr":{
      "weight_corr":false,
      "feature_corr":false,
      "vali_dir":"./validated_images",
      "corr_dir":"./test_image"
}
```

"weight_corr" option: Eliminate errors caused by weight quantization

"feature_corr" option：Eliminate errors caused by feature quantization

"corr_dir" option: Pass in the image set for bias calibration

"vali_dir" option: Pass in the verified image set, and after the bias calibration is completed, verify the bias calibration effect

Using the "bias_corr" option still cannot solve the problem, and the quantization error can be eliminated by increasing the quantization bit width option.

```
"feature_quantize_bits":10,
"weight_quantize_bits":10,
```

D. The operator cosine similarity is normal, but the accuracy is poor:

Reason 1: There is a problem with the conversion tool simulating native network reasoning, which is the same as the reason 3 of phenomenon 2), but the manifestation of the phenomenon is different

Solution: Refer to phenomenon 2) Reason 3

Reason 2: There is a problem with inference on the chip side

Solution: Use the "./result/path/input_uint8.bin" data generated by the conversion tool as the input of the onboard inference, and save the result information of the onboard inference output. Compare the generated results with the simulation results stored in the ***"result/path/ptq_model***" path (uint8_t or uint16_t binary form NHWC) to confirm whether there is a problem with the on-board inference;

If there is a problem and feedback the conversion tool to the docking staff, send the "***./result/path/input_uint8.bin***" data generated by MagikTransformKit and the dump result of the input and output of the problematic layer under the path "result/path/ptq_model" to the docking staff.