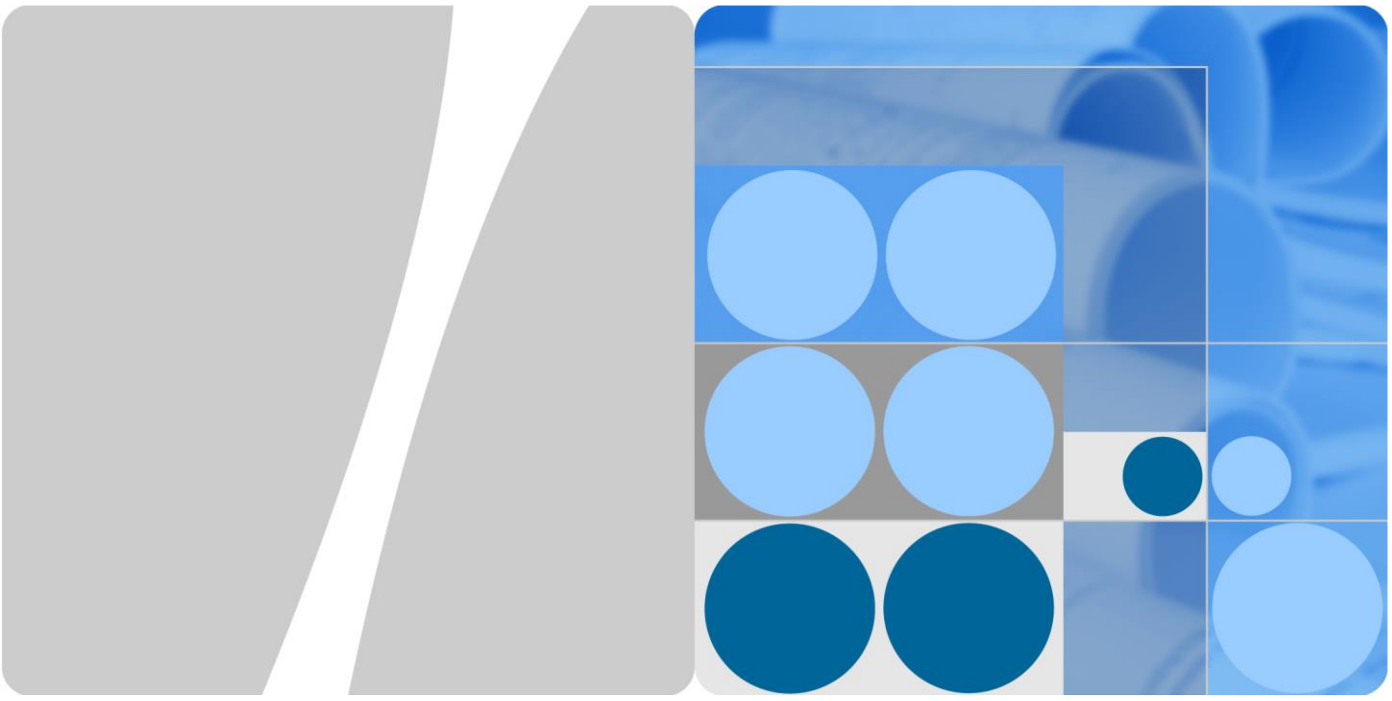

HiAI DDK V320

OMG Tool Instructions

Issue 02

Date 2019-12-31

HUAWEI TECHNOLOGIES CO., LTD.

Huawei HiAI Application

Send an application email to developer@huawei.com.

Email subject: HUAWEI HiAI + Company name + Product name

Email body: Cooperation company + Contact person + Phone number + Email address

We will reply to you within 5 working days.

Official website: https://developer.huawei.com/consumer/en/

About This Document

Purpose

This document introduces the Offline Model Generator (OMG) tool.

Change History

Changes between document issues are cumulative. The latest document issue contains all the changes made in earlier issues.

Date	Version	Change Description
2019-12-31	02	Added the description of HiAI DDK V320.
2019-09-04	01	Added the description of HiAI DDK V310.

Contents

Tables

1 Introduction

Before using the HiAI DDK, you can use the offline model generator (OMG) to convert Caffe and TensorFlow models into offline models (OMs), so that the mobile AI app can directly read the OM for inference. The OMG is located in the **tools** directory of the DDK. It can run on both Linux (64-bit).

2 Model Conversion Example

2.1 Caffe Model Conversion

```
./omg --model xxx.prototxt --weight yyy.caffemodel --framework 0 --output ./modelname
```

Instance:

```
./omg --model deploy.prototxt --weight squeezenet_v1.1.caffemodel --framework 0 --
output ./squeezenet
```

If the log message "OMG generate offline model success" is printed, the model is successfully converted. A **squeezenet.om** file is generated in the current directory.

2.2 TensorFlow Model Conversion

```
./omg --model xxx.pb --framework 3 --output ./modelname --input_shape "xxx:n,h,w,c" --
out_nodes "node_name1:0"
```

Instance:

```
./omg --model mobilenet_v2_1.0_224_frozen.pb --framework 3 --output ./mobilenet_v2 --
input_shape "input:1,224,224,3" --out_nodes "MobilenetV2/Predictions/Reshape_1:0"
```

If the log message "OMG generate offline model success" is printed, the model is successfully converted. A **mobilenet_v2.om** file is generated in the current directory.

2.3 Caffe Model Conversion with Quantization

Most models use the 32-bit float type for computation. Quantizing models can reduce the model size and thereby accelerate model inference.

Model conversion with quantization depends on the lightweight tool. The **compress_conf** parameter is passed to the OMG to generate a quantized model. For details, see the *Huawei HiAI DDK V320 Lightweight Tool Instructions*.

Command:

```
./omg --model xxx.prototxt --weight xxx.caffemodel --framework 0 --output ./modelname   --
compress_conf=param.json
```

Instance:

```
./omg --model deploy.prototxt --weight squeezenet_v1.1.caffemodel --framework 0 --
output ./squeezenet --compress_conf=param.json
```

If the log message "OMG generate offline model success" is printed, the model is successfully converted. A **suqeezenet8bit.om** file is generated in the current directory.

2.4 Caffe Model Conversion with AIPP

The AI pre-processing (AIPP) module is introduced for input pre-processing including image cropping and resizing, color space conversion (CSC), mean subtraction and factor multiplication, and much more. Operators are added to the converted model to replace these operations, improving the efficiency.

NOTE

This feature is newly added in HiAI DDK V310.

Command:

```
./omg --model xxx.prototxt --weight xxx.caffemodel --framework 0 --insert_op_conf
aipp_conf_static.cfg --output ./modelname
```

Instance:

```
./omg --model deploy.prototxt --weight squeezenet_v1.1.caffemodel --framework 0 --
insert_op_conf aipp_conf_static.cfg --output ./squeezenet
```

If the log message "OMG generate offline model success" is printed, the model is successfully converted. A **squeezenet.om** file is generated in the current directory.

NOTE

aipp_conf_static.cfg in the **tools/tools_omg/sample** folder is the AIPP configuration file. For details, see 3.2.4 AIPP Configuration File.

3 Parameter Description

NOTE

- A path can contain uppercase letters, lowercase letters, digits, and underscores (_).
- A file name can contain uppercase letters, lowercase letters, digits, underscores (_), and dots (.).

3.1 General Parameters

Parameter	Description	Mandatory or Not	Default Value
--hiai_version	OMG version to be used: **v300 | v310 | v320 | IR**	No	IR
--model	Path of the source model file	Yes	N/A
--weight	Path of the weight file. This parameter needs to be specified when the source model is a Caffe model. This parameter does not need to be specified when the source model is a TensorFlow model.	No	N/A
--framework	Framework of the source model 0: Caffe 3: TensorFlow **NOTE** • This parameter is not mandatory when **mode** is set to **1**. If this parameter is not set, the offline model is converted to the JSON format by default. • This parameter is mandatory when **mode** is set to **0** or **3**.	Yes	N/A
--output	Path for storing the converted offline model file (including the file name), for example, **out/caffe_resnet18**. The converted model file is	Yes	N/A

Parameter	Description	Mandatory or Not	Default Value
	automatically suffixed with **.om**.		
--stream_num	Number of streams used by the model. Must be **1**.	No	1
--input_shape	Shape of the input data Example: **input_name1:n1,c1,h1,w1;input_name2:n2,c2,h2,w2** **input_name** must be the node name in the network model before model conversion. Note: This parameter must be specified when the source model framework is TensorFlow.	No	N/A
--h/help	Help information display enable	No	N/A
--compress_conf	Path of the lightweighting configuration file. This parameter is automatically generated by the lightweight tool. For details, see the *Huawei HiAI DDK V320 Lightweight Tool Instructions*.	No	N/A
--insert_op_conf	Path of the AIPP configuration file. For details, see 3.2.4 AIPP Configuration File. **NOTE** This parameter is newly added in HiAI DDK V310.	No	N/A
--op_name_map	Path of the operator mapping configuration file. This parameter is mandatory when the DetectionOutput network is used. For example, a DetectionOutput operator can play different roles in different networks. It can specify the mapping from DetectionOutput to FSRDetectionOutput or to SSDDetectionOutput. **NOTE** The following is a content reference of the operator mapping configuration file: DetectionOutput: SSDDetectionOutput	No	N/A
--om	Path of the model file. This parameter is mandatory when **mode** is set to **1**.	No	N/A
--json	Path for the converted JSON file	No	N/A
--mode	Operating mode 0: Generate a Da Vinci model. 1: Convert a model to JSON. Converting a model to JSON allows you to view the model structure in text format.	No	0

Parameter	Description	Mandatory or Not	Default Value
	3: Perform precheck only to check the validity of the model file. Precheck is to check whether the operator is supported. A precheck report will be generated.		
--target	Currently, this parameter can only be set to **Lite**.	No	Lite
--out_nodes	Output node Example: **node_name1:0;node_name1:1;node_name2:0** **node_name** must be the node name in the network model before model conversion. Outputs of the same node are numbered from 0 in ascending order. For example, the second output is numbered **1**. Note: This parameter must be specified when the source model framework is TensorFlow.	No	N/A
--input_format	Input data format, **NCHW** or **NHWC**. Notes: 1. This parameter does not take effect when the source model framework is Caffe. 2. This parameter does not need to be specified in most scenarios when the source model framework is TensorFlow. If required during conversion, specify this parameter based on the site requirements.	No	N/A
--check_report	Path of the precheck result file. If this path is not specified, the precheck result is saved in the current path when the model conversion fails or **mode** is set to **3** (precheck only).	No	N/A
--input_fp16_nodes	Name of the FP16 (NCHW) input node This parameter and **--input_type** are exclusive. Example: node_name1;node_name2 **NOTE** This parameter is not recommended. You are advised to use **--input_type**.	No	N/A
--is_output_fp16	Whether the output data type is FP16 (NCHW) This parameter and **--output_type** are exclusive. Example: **false, true, false, true** **NOTE** This parameter is not recommended. You are advised to use **--output_type**.	No	False

Parameter	Description	Mandatory or Not	Default Value
--net_format	Preferred data format for network operators	No	N/A
--output_type	Model output (multi-output or single-output) data type. The supported data types include FP32, FP16, INT32, and UINT8. Whether an offline model can be successfully generated depends on whether the source model supports the specified output data type. Table 3-3 describes the supported model output scenarios. Example: **output_name1:FP16;output_name2:UINT8** **NOTE** This parameter is newly added in HiAI DDK V310.	No	FP32
--input_type	Model input (multi-input or single-input) data type. The supported data types include FP32, FP16, INT32, and UINT8. Whether an offline model can be successfully generated depends on whether the source model supports the specified input data type. Table 3-1 and Table 3-2 describe the supported model input scenarios. Example: **input_name1:FP16;input_name2:UINT8** **NOTE** This parameter is newly added in HiAI DDK V310.	No	FP32
--weight_data_type	Whether to convert the data type of the model weight from FP32 to FP16. This parameter takes effect only when the existing weight is of the FP32 type. The value is either **FP16** (default) or **FP32**. If **FP16**, the weight is converted from FP32 to FP16. If **FP32**, the data type of the weight is retained. Example: "**weight_data_type:FP16**" **NOTE** This parameter is newly added in HiAI DDK V320.	No	FP16

NOTICE

Scenarios not listed in Table 3-1 to Table 3-3 are not supported.

Table 3-1 Input data types supported in non-AIPP scenarios

Input of the Source Model	Expected Input of the Offline Model (User-Defined)	OMG Parameter
FP32	FP16	--input_type
FP32	FP32	--input_type
FP16	FP16	--input_type
UINT8	UINT8	--input_type
INT32	INT32	--input_type
INT8	INT8	--input_type
INT16	INT16	--input_type
INT32	INT32	--input_type
BOOL	BOOL	--input_type
INT64	INT64	--input_type
DOUBLE	DOUBLE	--input_type

Table 3-2 Input data types supported in AIPP scenarios

Input of the Source Model	Expected Input of the Offline Model	With AIPP or Not	OMG Parameter
FP32	UINT8	Yes	--input_type
FP16	UINT8	Yes	--input_type
UINT8	UINT8	Yes	--input_type

Table 3-3 Supported model output data types

Output of the Source Model	Output of the Offline Model	OMG Parameter
FP32	UINT8	--output_type
FP16	UINT8	--output_type
UINT8	UINT8	--output_type
FP16	FP16	--output_type
FP32	FP32	--output_type

Output of the Source Model	Output of the Offline Model	OMG Parameter
FP32	FP16	--output_type
FP16	FP32	--output_type
INT32	INT32	--output_type

3.2 AIPP Parameters

NOTE
AIPP parameters are newly added in HiAI DDK V310.

3.2.1 Overview

Hardware-based AI pre-processing (AIPP) involves image resize, color space conversion (CSC), and mean subtraction and factor multiplication (pixel changing).

Static AIPP and dynamic AIPP modes are supported, which are mutually exclusive. Table 3-4 describes the differences between the two AIPP modes.

Table 3-4 Static AIPP and dynamic AIPP modes

Mode	AIPP Parameter Configuration	Advantages
Static AIPP	Pre-defined through the configuration file or IR definition during model generation.	Efficiency: AIPP initialization can be completed in the model loading phase.
Dynamic AIPP	Passed externally through APIs during model inference. The model is marked with the AIPP support. For details about the APIs, see section 3.5 "External AIPP APIs" in the *Huawei HiAI DDK V320 Model Inference and Integration Guide.*	Flexibility: Different sets of AIPP parameters can be used for each inference.

3.2.2 AIPP Input Formats

The AIPP supports the following image formats:

- YUV420SP_U8
- XRGB8888_U8
- ARGB8888_U8
- YUYV_U8

- YUV422SP_U8
- AYUV444_U8
- YUV400_U8

The suffix **U8** represents the UINT8 pixel type. The value range is 0–255. When YUV images are input, the AIPP pads the image data to the YUV444 format regardless of the input format is YUV420, YUV422, or YUYV.

In addition to the preceding image types, AIPP can support more image input formats by enabling channel swapping.

3.2.3 AIPP Functions

The following AIPP functions are supported, listed in the processing sequence of the chip:

- Image cropping
- Channel swapping
- Color space conversion (CSC)
- Image resizing
- Data type conversion (DTC)
- Image padding

3.2.3.1 Cropping

Image cropping in AIPP applies only to input images. The involved parameters are described as follows.

Parameter	Description	Value Range
switch	Cropping enable	false/true
load_start_pos_w	Horizontal coordinate of the crop start	load_start_pos_w < src_image_size_w
load_start_pos_h	Vertical coordinate of the crop start	load_start_pos_h < src_image_size_h
crop_size_w	Width of the cropped out image	load_start_pos_w + crop_size_w <= src_image_size_w
crop_size_h	Height of the cropped out image	load_start_pos_h + crop_size_h <= src_image_size_h

For YUV420SP, YUYV, YUV422SP, or AYUV444 image input, the coordinates of the crop start and the width and height of the cropped out image must be even numbers. Parameter check will be performed accordingly.

3.2.3.2 Channel Swapping

AIPP supports RB/UV channel swapping and AX channel swapping.

RB/UV channel swapping greatly enriches the input image formats.

Configured Format	Input Format
YUV420SP_U8	YUV420, YVU420 + rbuv_swap_switch
XRGB8888_U8	XRGB, XBGR + rbuv_swap_switch
ARGB8888_U8	ARGB, ABGR + rbuv_swap_switch
YUYV_U8	YUYV, YVYU + rbuv_swap_switch
YUV422SP_U8	YUV422, YVU422 + rbuv_swap_switch
AYUV444_U8	AYUV, rbuv_swap_switch

AX channel swapping applies to XRGB, ARGB, and AYUV image inputs. AX channel swapping moves the first channel to the fourth channel. That is, with AX channel swapping, XRGB, ARGB, and AYUV data is converted into RGBX, RGBA, and YUVA data.

When the model training set consists of RGB images and the image input for inference is in XRGB or ARGB format, you can enable AX channel swapping to move the RGB channels forward for compatibility.

3.2.3.3 CSC

CSC refers to the conversion between the YUV444 and RGB888 image formats. The involved parameters are described as follows.

Parameter	Description	Type	Value Range
csc_switch	CSC switch	bool	true/false
matrix_r0c0 matrix_r0c1 matrix_r0c2 matrix_r1c0 matrix_r1c1 matrix_r1c2	CSC matrix elements	int16	[–32677, +32676]

Parameter	Description	Type	Value Range
matrix_r2c0 matrix_r2c1 matrix_r2c2			
output_bias_0 output_bias_1 output_bias_2	Output biases for RGB2YUV conversion	uint8	[0, 255]
input_bias_0 input_bias_1 input_bias_2	Input biases for YUV2RGB conversion	uint8	[0, 255]

Reference 1: YUV and BGR conversion formulas

YUV2BGR:

```
 | B |     | matrix_r0c0 matrix_r0c1 matrix_r0c2 | | Y - input_bias_0 |
 | G | = | matrix_r1c0 matrix_r1c1 matrix_r1c2 | | U - input_bias_1 | >> 8
 | R |     | matrix_r2c0 matrix_r2c1 matrix_r2c2 | | V - input_bias_2 |
```

BGR2YUV:

```
 | Y |     | matrix_r0c0 matrix_r0c1 matrix_r0c2 | | B |            | output_bias_0 |
 | U | = | matrix_r1c0 matrix_r1c1 matrix_r1c2 | | G | >> 8 + | output_bias_1 |
 | V |     | matrix_r2c0 matrix_r2c1 matrix_r2c2 | | R |            | output_bias_2 |
```

Reference 2: BT-601NARROW, JPEG, and BT-709NARROW conversion formulas

BT-601NARROW:

```
Y = (66×R + 129×G + 25×B+128)/256+ 2^4
Cb = ((-38)×R + (-74)×G + 112×B+128)/256+ 2^7
Cr = (112×R + (-94)×G + (-18)×B+128)/256+ 2^7
R = (298×(Y-2^4) + 0×(Cb-2^7) + 409×(Cr-2^7)+128)/256
G = (298×(Y-2^4 )+(-100)×(Cb-2^7) +(-208)×(Cr-2^7)+128)/256
B = (298×(Y-2^4) + 516×(Cb-2^7) + 0×(Cr-2^7)+128)/256
```

JPEG:

```
Y = (77×R + 150×G + 29×B+128)/256
Cb = ((-43)×R + (-85)×G + 128×B+128)/256+ 2^7
Cr = (128×R + (-107)×G + (-21)×B+128)/256+ 2^7
```

```
R = (256×Y + 0×(Cb-2^7) + 359×(Cr-2^7)+128)/256
G = (256×Y+(-88)×(Cb-2^7) +(-183)×(Cr-2^7)+128)/256
B = (256×Y + 454× (Cb-2^7) + 0×(Cr-2^7)+128)/256
```

BT-709NARROW:

```
Y = (47×R + 157×G + 16×B+128)/256+ 2^4
Cb = ((-26)×R +(-87)×G + 112×B+128)/256+ 2^7
Cr = (112×R +(-102)×G +(-10)×B+128)/256+ 2^7
R = (298×(Y-2^4) + 0×(Cb-2^7) + 459×(Cr-2^7)+128)/256
G = (298×(Y-2^4) +(-55)×(Cb-2^7) +(136)×(Cr-2^7)+128)/256
B = (298×(Y-2^4) + 541×(Cb-2^7) + 0×(Cr-2^7)+128)/256
```

To use the configuration file to generate a model with static AIPP, to configure the CSC matrix and the values of **input_bias** and **output_bias** based on the preceding formulas. To define the AIPP CSC operator in IR mode or configure the AIPP CSC parameters using the HiAI API, pass the target color space. The system will fill in the CSC configuration parameters accordingly.

The following describes how to configure the CSC for the JPEG and BT-601NARROW images:

- YUV input and RGB model training set

JPEG	BT-601NARROW
matrix_r0c0 : 256 matrix_r0c1 : 0 matrix_r0c2 : 359 matrix_r1c0 : 256 matrix_r1c1 : -88 matrix_r1c2 : -183 matrix_r2c0 : 256 matrix_r2c1 : 454 matrix_r2c2 : 0 input_bias_0 : 0 input_bias_1 : 128 input_bias_2 : 128	matrix_r0c0 : 298 matrix_r0c1 : 0 matrix_r0c2 : 409 matrix_r1c0 : 298 matrix_r1c1 : -100 matrix_r1c2 : -208 matrix_r2c0 : 298 matrix_r2c1 : 516 matrix_r2c2 : 0 input_bias_0 : 16 input_bias_1 : 128 input_bias_2 : 128

- YUV input and BGR model training set

JPEG	BT-601NARROW
matrix_r0c0 : 256 matrix_r0c1 : 454 matrix_r0c2 : 0 matrix_r1c0 : 256 matrix_r1c1 : -88 matrix_r1c2 : -183 matrix_r2c0 : 256 matrix_r2c1 : 0 matrix_r2c2 : 359 input_bias_0 : 0 input_bias_1 : 128 input_bias_2 : 128	matrix_r0c0 : 298 matrix_r0c1 : 0 matrix_r0c2 : 409 matrix_r1c0 : 298 matrix_r1c1 : -100 matrix_r1c2 : -208 matrix_r2c0 : 298 matrix_r2c1 : 516 matrix_r2c2 : 0 input_bias_0 : 16 input_bias_1 : 128 input_bias_2 : 128

- YUV input and grayscale (YUV400_U8) model training set

matrix_r0c0 : 256 matrix_r0c1 : 0 matrix_r0c2 : 0 matrix_r1c0 : 0 matrix_r1c1 : 0 matrix_r1c2 : 0 matrix_r2c0 : 0 matrix_r2c1 : 0 matrix_r2c2 : 0	

- RGB input and grayscale (YUV400_U8) model training set

	matrix_r0c0 : 76 matrix_r0c1 : 150 matrix_r0c2 : 30 matrix_r1c0 : 0 matrix_r1c1 : 0 matrix_r1c2 : 0 matrix_r2c0 : 0 matrix_r2c1 : 0 matrix_r2c2 : 0

- RGB input and YUV model training set

JPEG	BT-601NARROW
matrix_r0c0 : 77 matrix_r0c1 : 150 matrix_r0c2 : 29 matrix_r1c0 : -43 matrix_r1c1 : -85 matrix_r1c2 : 128 matrix_r2c0 : 128 matrix_r2c1 : -107 matrix_r2c2 : -21 output_bias_0 : 0 output_bias_1 : 128 output_bias_2 : 128	matrix_r0c0 : 66 matrix_r0c1 : 129 matrix_r0c2 : 25 matrix_r1c0 : -38 matrix_r1c1 : -74 matrix_r1c2 : 112 matrix_r2c0 : 112 matrix_r2c1 : -94 matrix_r2c2 : -18 output_bias_0 : 16 output_bias_1 : 128 output_bias_2 : 128

- RGB input and YVU model training set

JPEG	BT-601NARROW
matrix_r0c0 : 77 matrix_r0c1 : 150 matrix_r0c2 : 29 matrix_r1c0 : 128 matrix_r1c1 : -107 matrix_r1c2 : -21 matrix_r2c0 : -43 matrix_r2c1 : -85 matrix_r2c2 : 128 output_bias_0 : 0 output_bias_1 : 128 output_bias_2 : 128	matrix_r0c0 : 66 matrix_r0c1 : 129 matrix_r0c2 : 25 matrix_r1c0 : 112 matrix_r1c1 : -94 matrix_r1c2 : -18 matrix_r2c0 : -38 matrix_r2c1 : -74 matrix_r2c2 : 112 output_bias_0 : 16 output_bias_1 : 128 output_bias_2 : 128

From the perspective of usage, it is meaningless to convert grayscale images into RGB images. The system does not support CSC for YUV400_U8 inputs.

3.2.3.4 Resizing

The parameters involved in image resizing are as follows:

Parameter	Description	Value Range
switch	Resizing switch	false/true
resize_input_w	Image width prior to resizing	resize_input_w <=8192 resize_output_w / resize_input_w ∈[1/16,16]
resize_input_h	Image height prior to resizing	resize_output_h /resize_input_h∈[1/16,16]
resize_output_w	Width of the resized image	[16, 8192]
resize_output_h	Height of the resized image	>=16

NOTE

- The **resize_input_w** and **resize_input_h** parameters are invisible to users. When the cropping function is disabled, the size of the input image is the image size prior to resizing. When the crop function is enabled, the size of the cropped out image is the image size prior to resizing because image resizing is performed after cropping in AIPP. During configuration, you only need to pay attention to the size after resizing.
- In static AIPP mode, you do not have to configure **crop_size_w** and **crop_size_h** (size of the cropped out image) and **resize_output_w** and **resize_output_h** (size of the resized image) in the configuration file if their values can be obtained through computation. If **resize_output_w** and **resize_output_h** are omitted, their values are obtained by subtracting padded values from the image size of the model training set. If the resizing function is disabled, **crop_size_w** and **crop_size_h** can be omitted.

3.2.3.5 DTC

Data type conversion (DTC) is to convert pixel values in input images into the data type used for model training. AIPP allows users to set DTC parameters so that the converted data falls within the expected range. This avoids forcible conversion.

The formula for UINT8 to INT8 conversion is as follows:

pixel_out_chx(i) = pixel_in_chx(i) – mean_chn_i

The formula for INT8 to FLOAT16 conversion is as follows:

pixel_out_chx(i) = [pixel_in_chx(i) – mean_chn_i–min_chn_i] * var_reci_chn

The following table describes the DTC parameters.

Parameter	Description	Value Range
switch	DTC switch	false/true
mean_chn_0	Mean value of channel 0	[0, 255]
mean_chn_1	Mean value of channel 1	[0, 255]
mean_chn_2	Mean value of channel 2	[0, 255]
mean_chn_3	Mean value of channel 3	[0, 255]
min_chn_0	Minimum value of channel 0	[–65504, +65504]
min_chn_1	Minimum value of channel 1	[–65504, +65504]
min_chn_2	Minimum value of channel 2	[–65504, +65504]

Parameter	Description	Value Range
min_chn_3	Minimum value of channel 3	[–65504, +65504]
var_reci_chn_0	Variance of channel 0	[–65504, +65504]
var_reci_chn_1	Variance of channel 1	[–65504, +65504]
var_reci_chn_2	Variance of channel 2	[–65504, +65504]
var_reci_chn_3	Variance of channel 3	[–65504, +65504]

NOTE

When the DTC switch is set to **false** or the DTC parameters are not passed by calling the HiAI API, the system performs forcible conversion by default, with the mean and minimum values as **0** and the channel variances as **1**.

3.2.3.6 Padding

AIPP padding pads input images. The involved parameters are described as follows.

Parameter	Description	Value Range
switch	Padding switch	false/true
left_padding_size	Left padding size, in pixels	-
right_padding_size	Right padding size, in pixels	-
top_padding_size	Top padding size, in pixels	-
bottom_padding_size	Bottom padding size, in pixels	-

You are advised to keep the AIPP padding sizes within 32 pixels. Too large padding has to be implemented by software code, which is less efficient than hardware implementation.

3.2.4 AIPP Configuration File

NOTE

- The following is an example of the AIPP configuration file optimized for DDK V320. For details about the configuration files of V310 and earlier versions, see 4 Appendix: AIPP Configuration File.
- Da Vinci models generated by using the configuration file of DDK V320 cannot be used in the ROM of versions earlier than HiAI 320.

The AIPP configuration file is used to enable AIPP for the Da Vinci model converted by using the OMG. An example of a complete AIPP configuration file is as follows.

```
# An AIPP configuration starts with the aipp_op flag, which indicates that it is the configuration of an AIPP operator. Multiple aipp_op flags can be configured.
aipp_op {
# The input_name parameter is optional. It specifies the input of the model to be pre-processed.
# Type: string
input_name: "data":

# related_input_rank is optional and corresponds to input_name. But input_name is preferred if the input model name is available. If the model input name is unavailable, related_input_rank can be used to identify the sequence number of the model input for AIPP processing.
# Type: uint32
related_input_rank:0

# node_after_aipp is optional. It is used when an input has multiple branches and different AIPP processing needs to be performed on the branches. You can choose whether to configure all the branches. Configuring some of them is not supported.
# Type: string
node_after_aipp: "conv01"

# input_edge_idx is optional and corresponds to node_after_aipp. If the operators after the Data operator have the same name, input_edge_idx can be used to identify the branch to be AIPP processed.
# Type: uint32
input_edge_idx: 0

input_para {
# Type of the input image
# Type: enum
# Value range: [YUV420SP_U8, XRGB8888_U8, ARGB8888_U8, YUYV_U8, YUV422SP_U8,
AYUV444_U8, YUV400_U8]
format: AYUV444_U8

shape {
# Width and height of the input image
# Type: uint32
# Value range and constraints: [0, 4096]. For YUV images except YUV400, the value must be an
even number.
src_image_size_w: 800
src_image_size_h: 600
}

# max_src_image_size is used in the dynamic AIPP mode. When the image length, width, or
input type is uncertain, set it to the maximum input image size.
# Type: uint32
max_src_image_size: 102400
}

# == Cropping settings. For details about the parameter meanings and value ranges, see section
3.2.3.1. == #
crop_func {
switch: true
load_start_pos_w: 50
load_start_pos_h: 50
crop_size_w: 400
crop_size_h: 400
}

# == Channel swapping settings. For details about the parameter description and value range,
see section 3.2.3.2. == #
swap_func {
rbuv_swap_switch: true
ax_swap_switch: true
}

# == Resizing settings. For details about the parameter description and value range, see section
```

```
3.2.3.4. == #
resize_func {
switch: true
resize_output_w: 200
resize_output_h: 200
}

# == CSC settings. For details about the meanings and value ranges of related parameters, see
section 3.2.3.3. == #
csc_func {
switch: true
matrix_r0c0: 256
matrix_r0c1: 0
matrix_r0c2: 259
matrix_r1c0: 256
matrix_r1c1: -88
matrix_r1c2: -183
matrix_r2c0: 256
matrix_r2c1: 454
matrix_r2c2: 0
output_bias_0: 0
output_bias_1: 0
output_bias_2: 0
input_bias_0: 16
input_bias_1: 128
input_bias_2: 128
}

# == DTC settings. For details about the meanings and value ranges of related parameters, see
section 3.2.3.5. == #
dtc_func {
switch: true
mean_chn_0: 0
mean_chn_1: 0
mean_chn_2: 0
mean_chn_3: 0
min_chn_0: 0
min_chn_1: 0
min_chn_2: 0
min_chn_3: 0
var_reci_chn_0: 1.0
var_reci_chn_1: 1.0
var_reci_chn_2: 1.0
var_reci_chn_3: 1.0
}

# == Padding settings. For details about the meanings and value ranges of related parameters,
see section 3.2.3.6. == #
padding_func {
switch: true
left_padding_size: 12
right_padding_size: 12
top_padding_size: 12
bottom_padding_size: 12
}
}
```

3.2.4.1 AIPP Multi-Input Support

Separate AIPP settings for multiple inputs of a multi-input model or for different output branches of a Data operator are supported.

AIPP multi-input support is controlled by two pairs of configuration parameters: **input_name** and **related_input_rank** specify the input to be AIPP processed, while **node_after_aipp** and **input_edge_idx** specify an output of the Data operator to be AIPP processed.

Between **input_name** and **related_input_rank**, parameter **input_name** is preferred. Parameter **related_input_rank** applies to the scenario where the model input name is uncertain. If both parameters are configured, mutual-check is performed. If neither is configured, AIPP is performed on the first input of the model by default.

Between **node_after_aipp** and **input_edge_idx**, parameter **node_after_aipp** is preferred. Parameter **input_edge_idx** applies to the scenario where the downstream operators of the Data operator have duplicated or uncertain names. If both parameters are configured, mutual-check is performed. If neither is configured, AIPP is performed on all output branches of the Data operator.

3.2.4.2 Dynamic AIPP and Static AIPP

If the switches of all AIPP functions are set to **false**, the generated Da Vinci model is a model with dynamic AIPP. In this case, the AIPP parameters need to be passed in the model inference phase. If the switch of any AIPP function is set to **true**, the generated Da Vinci model is a model with static AIPP. In the model inference phase, the AIPP configurations defined in the configuration file are used.

In dynamic AIPP mode, the input image size and image type are uncertain. That is, the **src_image_size_w**, **src_image_size_h** and **input_format** parameters are not configured. In this case, you need to set **max_src_image_size** to specify the maximum image size for dynamic AIPP.

4 Appendix: AIPP Configuration File for Earlier Versions

```
# An AIPP configuration starts with the aipp_op flag, which indicates that it is the configuration
of an AIPP operator. Multiple aipp_op flags can be configured.
aipp_op {
#
# AIPP provides the following features: CSC, crop, mean reduction, factor multiplication, data
swap between channels, and single-line mode.
# Only UINT8 images can be input.
# To use this configuration file, delete the comments of parameters to be configured and set
the parameter values as required.
# The parameter values in the template are default values. The input_format attribute is
mandatory, while other attributes are optional.
#======================== Global settings =========================
# aipp_mode specifies the AIPP mode. This parameter is mandatory.
# Type: enum
# Value: static, which indicates static AIPP. Currently, only static is supported.
# aipp_mode: static

# related_input_rank is optional. It indicates the processing start of the inputs. The default
value is 0, which indicates that AIPP processing starts from input 0. For example, if the model
has two inputs. To have the AIPP processing start from the second input, set
related_input_rank to 1.
# Type: integer
# Value range: ≥ 0
# related_input_rank: 0
# input_edge_idx is optional. If a model input is shared by multiple operators, that is, the data operator is followed by more operators, set this parameter to perform different AIPP processing for different output dimensions of the Data operator.
# Type: integer
# Value range: ≥ 0
# input_edge_idx: 0

# input_format: input image format
# Type: enum
# Value:
YUV420SP_U8/XRGB8888_U8/ARGB8888_U8/YUYV_U8/YUV422SP_U8/AYUV444_U8/YUV400_U8
# input_format :

# Image width and height
# Type: uint16
# Value range and constraint: (0, 4096]. For a YUV420SP_U8 image, the value must be an even number.
# Note: Set src_image_size_w and src_image_size_h to the actual image width and height respectively. If they are not set or set to 0, the width and height defined in the network input are used.
# src_image_size_w :0
# src_image_size_h :0
Constraints:
(1) The values must be multiples of 16.
(2) If crop, resize, or padding is not required, src_image_size does not need to be configured, or set to the size of the source model. If crop, resize, or padding is required, src_image_size_w and src_image_size_h must be configured and must be greater than 0.

#======================== Crop settings ===================
# Whether to enable crop in AIPP
# Type: bool
# Value: true (yes) or false (no)
# crop :false

# Horizontal and vertical coordinates of the crop start. W and H defined in the network input are used as the size of the cut out image.
# Type: uint16
# Value range and constraint: (0, 4096]. For a YUV420SP_U8 image, the value must be an even number.
# Note: load_start_pos_w plus W defined in the network input must be less than or equal to src_image_size_w. load_start_pos_h plus H defined in the network input must be less than or equal to src_image_size_h.
# load_start_pos_w :0
# load_start_pos_h :0

# Size of the cut out image
# Type: uint16
# Value range and constraint: even number within [0, 4096], load_start_pos_w + crop_size_w ≤ src_image_size_w
load_start_pos_h + crop_size_h ≤ src_image_size_h
# crop_size_w :0
# crop_size_h :0

#==================== Resize settings ==================
# Whether to enable resize in AIPP
# Type: bool
# Value: true (yes) or false (no)
resize :false
# Width and height of the resized image. The fields are reserved.
# Type: uint16s
# Value range and constraint: even number within [0,4096], less than src_image_size
resize_output_w :0
resize_output_h :0

#========================= Padding settings =================
# Whether to enable padding in AIPP. This field is reserved.
# Type: bool
# Value: true (yes) or false (no)
# padding :false
# Padding in the C direction, for static AIPP. This field is reserved.
# Type: float16
# c_padding_value :0.0
# left_padding_size :0
# right_padding_size :0
# top_padding_size :0
# bottom_padding_size :0

#==================CSC settings ===================
# Whether to enable CSC in static AIPP
# Type: bool
# Value: true or false
# csc_switch :false

# Whether to enable R/B or U/V channel swap before CSC
# Type: bool
# Value: true or false
# rbuv_swap_switch :false

# Whether to enable RGBA->ARGB and YUVA->AYUV swap before CSC
# Type: bool
# Value: true or false
# ax_swap_switch :false

# Whether to enable single line mode (in this mode, only the first line of the cut out image is processed)
# Type: bool
# Value: true or false
# single_line_mode :false

# If the CSC is disabled (false), this function is bypassed.
# If the input image has four channels, the first channel is ignored.
# YUV2BGR conversion:
# | B | | matrix_r0c0 matrix_r0c1 matrix_r0c2 | | Y - input_bias_0 |
# | G | = | matrix_r1c0 matrix_r1c1 matrix_r1c2 | | U - input_bias_1 | >> 8
# | R | | matrix_r2c0 matrix_r2c1 matrix_r2c2 | | V - input_bias_2 |
# BGR2YUV conversion:
# | Y | | matrix_r0c0 matrix_r0c1 matrix_r0c2 | | B | | output_bias_0 |
# | U | = | matrix_r1c0 matrix_r1c1 matrix_r1c2 | | G | >> 8 + | output_bias_1 |
# | V | | matrix_r2c0 matrix_r2c1 matrix_r2c2 | | R | | output_bias_2 |
# Elements in a 3 x 3 CSC matrix
# Type: int16
# Value range: [–32768, +32767]
# matrix_r0c0 :298
# matrix_r0c1 :516

# matrix_r0c2 :0

# matrix_r1c0 :298

# matrix_r1c1 :-100

# matrix_r1c2 :-208

# matrix_r2c0 :298

# matrix_r2c1 :0

# matrix_r2c2 :409


# Output biases for RGB2YUV conversion

# Type: uint8

# Value range: [0, 255]

# output_bias_0 :16

# output_bias_1 :128

# output_bias_2 :128


# Input biases for YUV2RGB conversion

# Type: uint8

# Value range: [0, 255]

# input_bias_0 :16

# input_bias_1 :128

# input_bias_2 :128


#======== Mean subtraction and multiplication factor settings =========

# The computation rules are as follows:

# For uint8->uint8, this function is bypassed.

# For uint8->int8: pixel_out_chx(i) = pixel_in_chx(i) – mean_chn_i

# For uint8->fp16: pixel_out_chx(i) = [pixel_in_chx(i) – mean_chn_i – min_chn_i] x var_reci_chn


# Mean value of channel n

# Type: uint8

# Value range: [0, 255]

# mean_chn_0 :0

# mean_chn_1 :0

# mean_chn_2 :0


# Minimum value of channel n

# Type: float16
# Value range: [–65504, +65504]
# min_chn_0 :0.0
# min_chn_1 :0.0
# min_chn_2 :0.0

# Standard deviation of channel n or reciprocal of (Max. – Min.)
# Type: float16
# Value range: [–65504, +65504]
# var_reci_chn_0 :1.0
# var_reci_chn_1 :1.0
# var_reci_chn_2 :1.0
}
```