ATC Tool Instructions

Issue 01

Date 2020-08-03

HUAWEI TECHNOLOGIES CO., LTD.

Trademarks and Permissions


and other Huawei trademarks are trademarks of Huawei Technologies Co., Ltd.
All other trademarks and trade names mentioned in this document are the property of their respective holders.

Notice

The purchased products, services and features are stipulated by the contract made between Huawei and the customer. All or part of the products, services and features described in this document may not be within the purchase scope or the usage scope. Unless otherwise specified in the contract, all statements, information, and recommendations in this document are provided "AS IS" without warranties, guarantees or representations of any kind, either express or implied.

The information in this document is subject to change without notice. Every effort has been made in the preparation of this document to ensure accuracy of the contents, but all statements, information, and recommendations in this document do not constitute a warranty of any kind, express or implied.

Contents

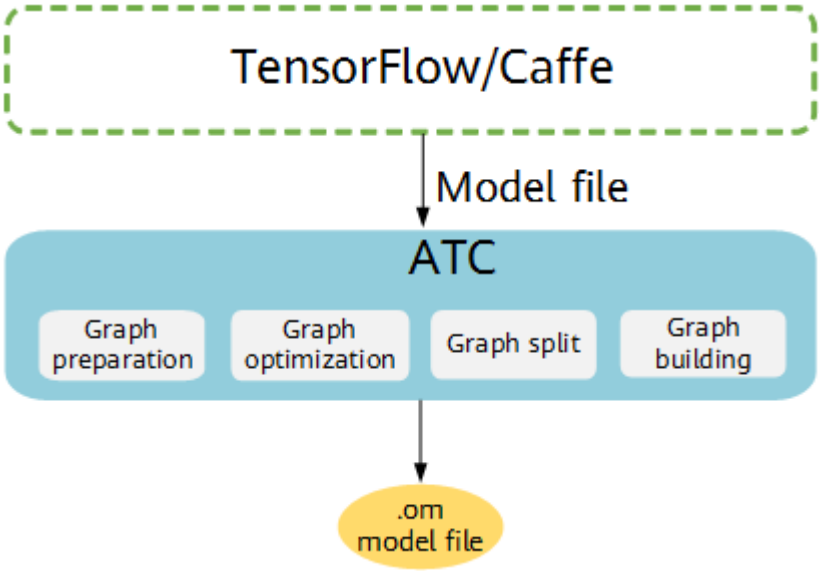
1 Introduction

This document describes how to use the Ascend Tensor Compiler (ATC) to convert models of open-source frameworks (such as Caffe and TensorFlow) and single-operator .json files into offline models supported by the Ascend AI Processor. During model conversion, operator scheduling optimization, weight data rearrangement, and memory optimization can be implemented, and model preprocessing can be completed without depending on the device.

ATC Architecture

Figure 1-1 shows the ATC architecture.

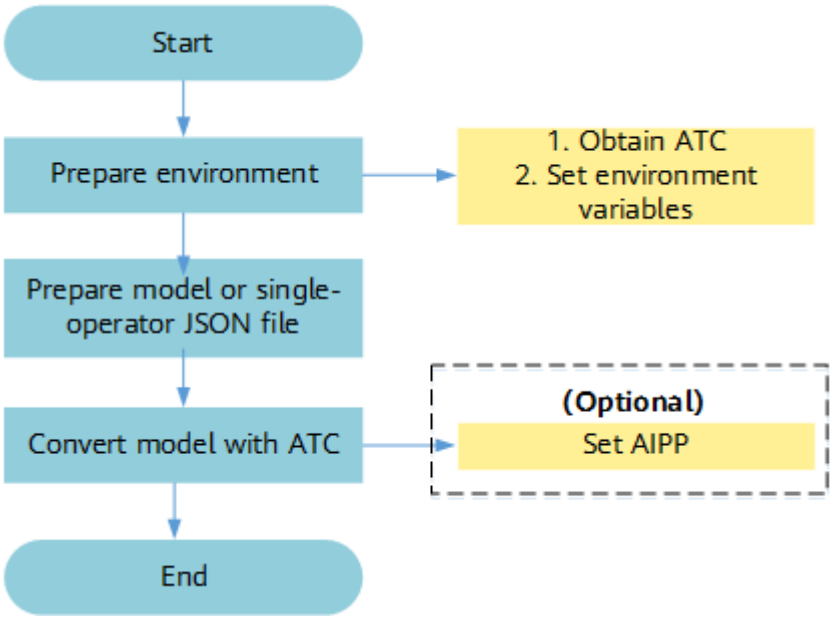
Figure 1-1 ATC architecture



ATC Flow

Figure 1-2 shows the flow of model conversion with the ATC.

Figure 1-2 ATC flowchart



The flow is described as follows:

1. Install the ATC in the development environment and open the ATC from the installation path. For details, see "Preparations" in **Preparations**.
2. Upload the model or single-operator .json file to be converted to the development environment. For details, see **Example (Converting a Caffe Model into an Offline Model)** or **Example (Converting a Single-Operator JSON File into an Offline Model)**.
3. Convert the model using the ATC. **Configure AI preprocessing (AIPP)** as required.

   Ascend AI Processor introduces the artificial intelligence preprocessing (AIPP) module for hardware-based image preprocessing including color space conversion (CSC), image normalization (by subtracting the mean value or multiplying a factor), image cropping (by specifying the crop start and cropping the image to the size required by the neural network), and much more. Images output by the digital vision preprocessing (DVPP) module are YUV420SP images with rounded-up heights and widths. The DVPP module does not output RGB images. Therefore, the AIPP module is introduced to convert the YUV420SP images with rounded-up heights and widths and then crop them to the size required by the model.

Key Terms

- AIPP

  The Ascend AI Processor introduces the AIPP module for hardware-based image preprocessing including CSC, image normalization (by subtracting the mean value or multiplying a factor), image cropping (by specifying the crop start and cropping the image to the size required by the neural network), and much more.
- YUV420SP

  It is a lossy image color encoding format, which can be YUV420SP_UV or YUV420SP_VU.

- Repository

  It stores the tuned tiling policy for direct call in operator build.

- Cost model

  It is an evaluator that evaluates the quality of tiling in the tiling space for selecting the optimal tiling policy.

- Format

  In the deep learning framework, *n*-dimensional data is stored in an *n*-dimensional array. For example, a feature graph of a convolutional neural network is stored in a four-dimensional array. The four dimensions are N, H, W and C, which stand for batch size, height, width, and channels, respectively.

  Data can be stored only in linear mode because the dimensions have a fixed order. Different deep learning frameworks store feature maps with different layouts. For example, Caffe uses the layout [Batch, Channels, Height, Width], that is, NCHW, while TensorFlow uses the layout [Batch, Height, Width, Channels], that is, NHWC. Data in TensorFlow is stored in the order of [Batch, Height, Width, Channels], that is, NHWC.

  As shown in **Figure 1-3**, an RGB image is used as an example. In the NCHW order, the pixel values of each channel are clustered in sequence as RRRGGGBBB. In the NHWC order, the pixel values are interleaved as RGBRGBRGB.

  Figure 1-3 NCHW and NHWC

  To improve data access efficiency, the tensor data is in the Ascend AI Software Stack is stored in the 5D format NC1HWC0. C0, closely related to the micro architecture, is the size of the cube unit in the AI Core. C0 is **16** for FP16 or **32** for INT8. C0 needs to be stored contiguously. C1 = (C + C0 – 1)/C0. If the result is not an integer, round it up.

  The NHWC-to-NC1HWC0 conversion process is as follows:

  a. Split the NHWC data into C1 pieces of NHWC0 along the C dimension.

  b. Arrange the C1 pieces of NHWC0 in the memory contiguously, obtaining NC1HWC0.

  The following is an example of converting NHWC to NC1HWC0:

  a. Convert RGB images at the first layer into the NC1HWC0 format by using AIPP.

  b. Rearrange the NC1HWC0 output from intermediate layers of the feature map during data movement.

2 Restrictions and Parameters

Restrictions

Before model conversion, pay attention to the following restrictions:

- To convert a network model such as Faster R-CNN to an offline model supported by the Ascend AI Processor, you must modify the model file (.prototxt) by referring to **7.1.1 Prototxt Customization**.
- Ensure that the **caffe.proto** file used in the Caffe training network is consistent with the built-in **caffe.proto** in the **atc/include/proto** directory and the customized **custom.proto** file in the **atc/sample/op/caffe** directory in the ATC installation path. Otherwise, you need to integrate the **caffe.proto** file used in the Caffe training network into the **custom.proto** file. When you add the definition of the custom operator to **custom.proto**, ensure that the ID is unique. It is recommended that the ID starts at 1000 in ascending order.
- Only Caffe and TensorFlow models can be converted. For Caffe, the input data type can be fp32, fp16 (by setting the **--input_fp16_nodes** argument), or uint8 (by configuring AIPP). For TensorFlow, the input data type can be fp16, fp32, unit8, int32, int64, or bool.
- For a Caffe model, **op name** and **op type** in the model file (.prototxt) must be consistent with those in the weight file (.caffemodel) must be the consistent (case sensitive).
- For a TensorFlow model, only the **FrozenGraphDef** format is supported.
- Inputs with dynamic shapes are not supported, for example, NHWC = [?, ?, ?, 3]. The dimension sizes must be static.
- For a Caffe model, the input data is up to 4-dimensional and operators such as reshape and expanddim do not support 5D output. For a TensorFlow model, the restrictions of each operator apply.
- Except the const operator, the input and output at all layers in the model must meet the condition of **dim! = 0**.
- Only the operators listed in **7.1.1.4 Caffe Operator Specifications** are supported. The defined operator restrictions must be met.
- Only the operators listed in **7.2.1 TensorFlow Operator Specifications** are supported.

Parameters

Before using the ATC to convert models, you must understand the following parameters.

NOTICE

- If the parameters queried by running the **atc --help** command are not described in the following table, they are reserved or applicable to other chip versions and therefore can be ignored.
- You can run the **atc** command to convert the model in either of the following formats:
  1. **atc param1=value1 param2=value2 ...** (No space is allowed before **value**. Otherwise, it will be truncated, and the value of **param** is empty.)
  2. **atc param1 value1 param2 value2 ...**

Parameter	Description	Required or Not (Depending on the Value of mode)	Default Value
--mode	Work mode. ● 0: Generate an offline model adapted to the Ascend AI Processor. ● 1: Generate an offline model or .json file. ● 3: Perform precheck only to check the validity of the model file. ● 5: Convert a .pbtxt file to a .json file. Currently, a .pbtxt file is generated only in the training scenario. The dump structure (a .pbtxt file) generated by the GE during training will be parsed and converted to a .json file, facilitating fault locating. This parameter is irrelevant to inference.	Not	0
--model	Path of the original model file. **NOTE** The path allows only uppercase letters, lowercase letters, digits, and underscores (_). The file name allows only uppercase letters, lowercase letters, digits, underscores (_), and periods (.).	Yes	N/A
--weight	Path of the weight file. This parameter needs to be specified for a Caffe model. **NOTE** The path allows only uppercase letters, lowercase letters, digits, and underscores (_). The file name allows only uppercase letters, lowercase letters, digits, underscores (_), and periods (.).	Not	N/A

Parameter	Description	Required or Not (Depending on the Value of mode)	Default Value
--framework	Framework of the original model. ● 0: Caffe ● 1: MindSpore ● 3: TensorFlow **NOTE** ● This parameter is optional when **--mode** is set to **1**. Caffe or TensorFlow can be specified. If this parameter is not specified, the offline model is converted to a JSON file by default. If this parameter is specified, ensure that the values of **--om** and **--framework** are the same. For example: **--framework=0 --om=*$HOME/test/resnet18.prototxt*** or **--framework=3 --om=*$HOME/test/resnet18_tensorflow.pb*** ● This parameter is required when **mode** is set to **0** or **3**. ● When using the MindSpore framework, pay attention to the following restrictions: **--mode** can only be set to **0**. The following parameters do not apply to the MindSpore framework: **--input_format**, **--out_nodes**, **--is_output_adjust_hw_layout**, **--input_fp16_nodes**, **--is_input_adjust_hw_layout**, and **--op_name_map**.	Yes	N/A

Parameter	Description	Required or Not (Depending on the Value of mode)	Default Value
--output	• For a model under an open-source framework: Path and file name of the converted offline model, for example, **$HOME/test/out/caffe_resnet18** or **$HOME/test/out/tf_resnet18**. The name of the converted model file is automatically suffixed with .om, for example, **caffe_resnet18.om**. • For a single-operator .json file: Path of the generated single-operator model, for example, ***$HOME/test/out/op_model***. The naming rule of the generated offline model file is ***SN_opType_InputDescription*** **(dateType_format_shape)** ***_OutputDescription*** **(dateType_format_shape)**. **NOTE** The path allows only uppercase letters, lowercase letters, digits, and underscores (_). The file name allows only uppercase letters, lowercase letters, digits, underscores (_), and periods (.).	Yes	N/A

Parameter	Description	Required or Not (Depending on the Value of mode)	Default Value
--check_report	Path of the precheck result file. If the graph fails to be parsed with **--mode** set to **0** or **1**, or if **--mode** is set to **3** to perform pre-check only, this argument specify the path of the pre-check result file **check_result.json**. If not specified, the pre-check result file is output to the current path. Example: **--check_report=*/home/hisisoc/test/out/*check_result.json** **NOTE** The path allows only uppercase letters, lowercase letters, digits, and underscores (_). The file name allows only uppercase letters, lowercase letters, digits, underscores (_), and periods (.).	Not	check_result.json
--h or --help	Help information.	Not	N/A

Parameter	Description	Required or Not (Depending on the Value of mode)	Default Value
--input_format	Input data format. ● If the original framework is Caffe, **NCHW** and **ND** (any format with N ≤ 4) are supported. Defaults to **NCHW**. ● If the original framework is TensorFlow, the supported formats include **NCHW**, **NHWC** (default), **ND**, **NCDHW**, and **NDHWC**. – For a TensorFlow model converted using the Open Neural Network Exchange (ONNX) model conversion tool, this parameter is required and must be set to **NCHW**. – If the original model contains an operator with the argument **data_format**, this parameter is required and **ND** is recommended. During model conversion, the format is inferred based on the operator with **data_format**. If the input data format cannot be determined, **ND** is recommended. **NOTE** If AIPP is enabled for model conversion, the converted model accepts NHWC input only. In this scenario, the data format specified by the **--input_format** argument in the **atc** command does not take effect.	Not	**NCHW** for Caffe **NHWC** for TensorFlow

Parameter	Description	Required or Not (Depending on the Value of mode)	Default Value
--out_nodes	Output nodes. If the output nodes (operators) are not specified, the last operator of the model is output by default. If specified, information of the specified operators is output. To check the parameters of a specific operator layer, mark the operator layer so that this operator layer is specified as the output node. After the model is converted, you can view the parameter settings of the specified operator at the end of the .om model file or the .json file converted from the .om model file. Example: **"node_name1:0;node_name1:1;node_name2:0"** Enclose each specified node in double quotation marks ("") and separate nodes by semicolons (;). **node_name** must be the node name in the original model. The number after the colon indicates the output index. For example, **node_name1:0** indicates output 0 of the node named **node_name1**.	Not	N/A

Parameter	Description	Required or Not (Depending on the Value of mode)	Default Value
--is_output_adjust_hw_layout	Whether the output type and format of the previous network are **FP16** and **NC1HWC0**, respectively. Use this parameter in conjunction with **out_nodes**. Example: **false,true,false,true** An example is as follows: atc --model=*$HOME/test/resnet18.prototxt* --weight=*$HOME/test/resnet18*.caffemodel --framework=0 --output=*$HOME/test*/out/caffe_resnet50 --is_output_adjust_hw_layout=true --out_nodes="prob:0" --soc_version=${soc_version} **NOTE** If set to **true**, **out_nodes** outputs NC1HWC0 tensors of type FP16.	Not	false
--input_fp16_nodes	Name of an input node with input data of type FP16. Example: **"node_*name1*;node_*name2*"**. Enclose each node in double quotation marks ("") and separate the nodes by semicolons (;). This parameter is mutually exclusive with **--insert_op_conf**.	Not	N/A
--is_input_adjust_hw_layout	Whether the input type and format of the previous network are **FP16** and **NC1HWC0**, respectively. Use this parameter in conjunction with **--input_fp16_nodes**. Example: **false,true,false,true** An example is as follows: atc --model=*$HOME/test/resnet18.prototxt* --weight=*$HOME/test/resnet18*.caffemodel --framework=0 --output=*$HOME/test*/out/caffe_resnet50 --is_input_adjust_hw_layout=true --input_fp16_nodes="data" --soc_version=${soc_version} **NOTE** If this parameter is set to **true**, **input_fp16_nodes** accepts NC1HWC0 tensors of type FP16.	Not	false

Parameter	Description	Required or Not (Depending on the Value of mode)	Default Value
--input_shape	Input shape. Example: **"input_name1:n1,c1,h1,w1;input_name2:n2,c2,h2,w2"** Enclose each specified node in double quotation marks ("") and separate nodes by semicolons (;). **input_name** must be the node name in the original model. If the original model has a dynamic shape, for example, **input_*name1*:? ,h,w,c**, this parameter is required. The question mark (?) indicates the batch size (number of images processed at a time). In this case, you have two options: 1. Set **?** to a fixed value (such as 1, 2, 3...) to convert the original model with a dynamic shape into an offline model with a fixed shape. 2. Set **?** to **–1** and use this parameter in conjunction with **--dynamic_batch_size** to set the dynamic batch size. For details, see the **--dynamic_batch_size** parameter.	Not	N/A

Parameter	Description	Required or Not (Depending on the Value of mode)	Default Value
--json	Path and name of the JSON file converted from the offline model or original model file. Use this parameter in conjunction with parameters **--mode=1** and **--om** (as well as **--weight** for a Caffe framework model file). Example: ***$HOME/test/out/resnet18.json*** **NOTE** The path allows only uppercase letters, lowercase letters, digits, and underscores (_). The file name allows only uppercase letters, lowercase letters, digits, underscores (_), and periods (.).	Not	N/A
--dump_mode	Whether to generate a JSON file with shape information. ● **0**: no (default) ● **1**: yes Use this parameter in conjunction with parameters **--json**, **--mode=1**, and **--om** (as well as **--weight** for a Caffe original model file). An example is as follows: atc --mode=1 --om=*$HOME/test/resnet18.prototxt* --json=*$HOME/test/out/resnet18.json* --framework=0 --weight=*$HOME/test/resnet18*.caffemodel --dump_mode=1	Not	0

Parameter	Description	Required or Not (Depending on the Value of mode)	Default Value
--om	This parameter is required when **--mode** is set to **1**. Path of the offline model or model file to be converted to a JSON file. Examples: ● For Caffe: ***$HOME/test/resnet18.prototxt*** or ***$HOME/test/out/caffe_resnet18.om*** ● For TensorFlow: ***$HOME/test/resnet18_tensorflow.pb*** or ***$HOME/test/out/tf_resnet18.om*** **NOTE** The path allows only uppercase letters, lowercase letters, digits, and underscores (_). The file name allows only uppercase letters, lowercase letters, digits, underscores (_), and periods (.).	Not	N/A

Parameter	Description	Required or Not (Depending on the Value of mode)	Default Value
--op_name_map	Path of the operator mapping configuration file. This parameter is required when the DetectionOutput algorithm is used on the network. Example: The function of the DetectionOutput operator varies in different networks. You can specify the mapping between the DetectionOutput operator (an operator in the offline model) and the following operators: ● FSRDetectionOutput (of the Faster R-CNN network) ● SSDDetectionOutput (of the SSD network) **NOTE** ● The path allows only uppercase letters, lowercase letters, digits, and underscores (_). The file name allows only uppercase letters, lowercase letters, digits, underscores (_), and periods (.). ● The following is an example of the operator mapping configuration file: DetectionOutput:SSDDetectionOutput	Not	N/A

Parameter	Description	Required or Not (Depending on the Value of mode)	Default Value

--insert_op_conf	Configuration file of the preprocessing operator (such as, operator aipp).
	This parameter is mutually exclusive with **--input_fp16_nodes**.
	NOTE
	● The path allows only uppercase letters, lowercase letters, digits, and underscores (_). The file name allows only uppercase letters, lowercase letters, digits, underscores (_), and periods (.).
	● The following is an example of the configuration file:
	aipp_op {
	aipp_mode:static
	input_format:YUV420SP_U8
	csc_switch:true
	var_reci_chn_0:0.00392157
	var_reci_chn_1:0.00392157
	var_reci_chn_2:0.00392157
	}
	● For details about the AIPP configuration file, see the description of AIPP configuration.
		Not	N/A

Parameter	Description	Required or Not (Depending on the Value of mode)	Default Value
--output_type	Output data type of a network or an output node. ● Output data type of a network: – FP32: recommended for classification and detection networks – **UINT8**: recommended for image super-resolution networks for better inference performance – **FP16**: recommended for classification and detection networks After the model conversion is complete, the preceding data types are displayed as **DT_FLOAT**, **DT_UINT8**, and **DT_FLOAT16** in the corresponding .om model file. **NOTE** If the output data type is not specified by this parameter during model conversion, the output data type of the last layer of the original network model is used. If specified, the specified data type is output and prevails over that specified by **is_output_adjust_hw_layout**. ● Output data type of an output node: For example, **--output_type="node1:0:FP16;node2:0:FP32"**, indicating that the output data type of node1 is set to **FP16** and that of node2 is set to **FP32**. Enclose each node in double quotation marks ("") and separate the nodes by semicolons (;). In this scenario, use this parameter in conjunction with the **--out_nodes** parameter. An example is as follows: atc --model=*$HOME/test/resnet18.prototxt* --weight=*$HOME/test/resnet18.caffemodel* --framework=0 --output=*$HOME/test/out/*	Not	N/A

Parameter	Description	Required or Not (Depending on the Value of mode)	Default Value
	caffe_resnet18 --soc_version=${soc_version} --output_type="*conv1:0:FP16*" --out_nodes="*conv1:0*"		
--singleop	Single-operator definition file, which is used to convert the JSON file of a single operator into an offline model that adapts to the Ascend AI Processor. For details about the .json file format and parameter configuration of a single operator, see **5 Single-Operator JSON File Configuration**. Only the following parameters can be used in conjunction with this parameter. **--output** and **--soc_version** are required. ● --output ● --soc_version ● --aicore_num ● --auto_tune_mode ● --precision_mode ● --op_select_implmode ● --enable_small_channel ● --log	Not	N/A

Parameter	Description	Required or Not (Depending on the Value of mode)	Default Value
--precision_mode	Operator precision mode. ● **force_fp16** (default): forcibly selects fp16 even if the operator supports both fp16 and fp32. ● **allow_mix_precision**: allows mixed precisions. ● **allow_fp32_to_fp16**: selects fp16 if the operator does not support fp32. ● **must_keep_origin_dtype**: retains the original image precision.	Not	force_fp16
--op_select_implmode	Operator implementation mode when the input data is of type fp16. In high-precision mode, Newton's Method and Taylor's Formula is used to improve operator precision. In high-performance mode, the optimal performance is implemented without affecting the network precision. ● **high_precision**: precision first ● **high_performance** (default): performance first If an operator supports both high-precision and high-performance, the mode specified by --**op_select_implmode** is selected during running. If only high-precision or high-performance is supported, the supported mode is used during running. For example, an operator supports only high-precision and **--op_select_implmode** is set to **high_performance**. In this case, the **--op_select_implmode** parameter does not take effect and the high-precision mode is used.	Not	high_performance

Parameter	Description	Required or Not (Depending on the Value of mode)	Default Value
--optypelist_for_implmode	List of operator types. The operators in the list use the mode specified by the **--op_select_implmode** parameter. Currently, only the Pooling operator is supported. Use this parameter in conjunction with the **--op_select_implmode** parameter. For example, **--op_select_implmode=high_precision --optypelist_for_implmode=pooling**.	Not	N/A
--disable_reuse_memory	Memory reuse enable. ● **1**: disabled ● **0** (default):enabled **NOTE** If the model is large and memory reuse is disabled, the memory may be insufficient for model conversion, leading to model conversion failures.	Not	0

Parameter	Description	Required or Not (Depending on the Value of mode)	Default Value
--auto_tune_mode	Operator automatic tuning mode. Controls whether to tune the TBE operator to maximize the Ascend AI Processor performance. ● Genetic Algorithm (GA): used for cube operator tuning (only Conv2D, depthwise, and GEMM are supported). ● Reinforcement Learning (RL): used for vector operator tuning. Multiple modes are supported. Enclose them in double quotation marks ("") and separate them by commas (,). For example, **"RL,GA"**. This function applies only to the scenario where the development environment and operating environment are co-deployed. To use this parameter for tuning, you need to set environment variables. For details, see **2.b.i**. ● GA tuning result: – The **${install_path_atc}/atc/data/tiling/${*soc_version*}/built-in/** directory stores the built-in repository and cost model. If a shape in the model does not hit the built-in repository, tuning is initiated. – The **${install_path_atc}/atc/data/tiling/${*soc_version*}/custom/** directory stores the repository generated after tuning. If a repository already exists in the path, information is added to the repository. If no repository exists in the path, a new repository is created.	Not	N/A

Parameter	Description	Required or Not (Depending on the Value of mode)	Default Value
	When the **atc** command is used for model conversion, the tuning policy in the custom repository is read by default during model build only when **--auto_tune_mode=GA** or **--auto_tune_mode="RL,GA"**. Otherwise, **Auto Schedule** is used for tuning during model conversion. ● RL tuning result: – The **${install_path_atc}/atc/data/rl/${*soc_version*}/built-in/** directory stores the built-in repository after common shape tuning. – The **${install_path_atc}/atc/data/rl/${*soc_version*}/custom/** directory stores the repository that has a better tuning performance than the repository in the **built-in** directory or does not exist in **built-in** directory during tuning. For details about the principle and usage of the Auto Tune tool, see Auto Tune Instructions.		

Parameter	Description	Required or Not (Depending on the Value of mode)	Default Value
	NOTE ● If a tuned repository exists in the **custom** directory and the operator logic is changed (for example, support for ND input is added to the GEMM operator), you need to set the following environment variable and perform tuning again. `export REPEAT_TUNE=True` ● When auto tuning is enabled, memory allocation is needed for board evaluation. The Auto Tune tool may use more memory than the ATC tool. The required memory is related to the number of devices used at the same time and can be estimated as follows: **2 x Number of devices x Input data size**. If the memory size exceeds the size of the ATC running memory, tuning fails. ● During GA tuning, the device resource is exclusively occupied. Therefore, other operations that require the device resource are not allowed. If the tuning fails, stop other processes and perform the tuning again. If there are build services running on the host, the tuning duration is affected.		
--aicore_num	Number of AI Cores used for building. The value range is (0, 2]. The default value is the maximum value.	Not	2
--buffer_optimize	Buffer optimization switch. ● **l2_optimize**: enabled. ● **off_optimize**: disabled.	Not	l2_optimize

Parameter	Description	Required or Not (Depending on the Value of mode)	Default Value
--enable_small_channel	Small channel optimization enable. If enabled, performance benefits are generated at the convolutional layers with channel <= 4. You are advised to enable this function in inference scenarios. • **0** (default): disabled. • **1**: enabled. **NOTE** Performance benefits can be obtained only for GoogleNet, ResNet-50, ResNet-101, and ResNet-152 . If this function is enabled in other scenarios, the performance may deteriorate.	Not	0
--fusion_switch_file	Path and name of the fusion switch configuration file. This parameter is required if you want to use the model quantized by the model miniaturization tool for operator accuracy comparison. Configure this file to disable the fusion function. The following is an example of the configuration file **fusion_switch.cfg**. For details about the configuration file, see **6 Fusion Switch Configuration File**. RequantFusionPass:off TbeConvDequantVaddReluQuantFusionPass:off TbeConvDequantQuantFusionPass:off TbePool2dQuantFusionPass:off **NOTE** If the **group** attribute of the Convolution operator is consistent with the **num_output** attribute in the prototxt model file, **RequantFusionPass** in the configuration file must be enabled.	Not	N/A

Parameter	Description	Required or Not (Depending on the Value of mode)	Default Value
--soc_version	Chip version specified during model conversion: Ascend310	Yes	N/A

Parameter	Description	Required or Not (Depending on the Value of mode)	Default Value
--dynamic_batch_size	Dynamic batch configuration for the scenario where image count per inference batch is unfixed. This parameter must be used in conjunction with **--input_shape** and is mutually exclusive with **--dynamic_image_size**. A maximum of 100 batch size choices are supported. Separate the batch size choices by commas (,). The value range of each batch size is [1, 2048]. Example: **--input_shape="data:-1,3,416,416;img_info:-1,4" --dynamic_batch_size="1,2,4,8"** where, **-1** in **--input_shape** indicates dynamic batch enabled.	Not	N/A

Parameter	Description	Required or Not (Depending on the Value of mode)	Default Value
	NOTE • If this parameter is specified for model conversion, you need to add the **aclmdlSetDynamicBatchSize** API before the **aclmdlExecute** API to set the actual batch size before model inference using an application project. For details about how to use the **aclmdlSetDynamicBatch-Size** API, see **See Also > ACL API Reference > Model Loading and Execution** in Application Software Development Guide. • If the number of images to be processed each batch is unfixed, you can set this parameter to dynamically allocate the number of images to be processed each batch. For example, to process two, four, or eight images per inference batch, set this parameter to **2,4,8**. The memory will be allocated based on the maximum preset batch size. • Too large batch sizes or too many batch size choices will cause model conversion failures. • If you set too large batch sizes or too many batch size choices for inference in the operating environment, you are advised to run the **swapoff -a** command to disable the memory swap function. This prevents slow running due to memory insufficiency.		

Parameter	Description	Required or Not (Depending on the Value of mode)	Default Value
--dynamic_image_size	Dynamic image size configuration for the scenario where the image size per inference batch is unfixed. This parameter must be used in conjunction with **--input_shape** and is mutually exclusive with **--dynamic_batch_size**. A maximum of 100 image size choices are supported. Separate the image size choices by commas (,). The format is **"imagesize1_height,imagesize1_width;imagesize2_height,imagesize2_width"**. Enclose specified parameters in double quotation marks ("), and separate the parameters by semicolons (;). The following is an example, where, the **-1** argument for **--input_shape** indicates that the dynamic image size function is enabled. --input_shape="data:8,3,-1,-1;img_info:8,4" --dynamic_image_size="416,416;832,832"	Not	N/A

Parameter	Description	Required or Not (Depending on the Value of mode)	Default Value
	NOTE ● If this parameter is specified for model conversion, you need to add the **aclmdlSetDynamicHWSize** API before the **aclmdlExecute** API to set the actual image size before model inference using an application project. For details about how to use the **aclmdlSetDynamicHWSize** API, see **See Also > ACL API Reference > Model Loading and Execution** in Application Software Development Guide. ● Too large image sizes or too many image size choices will cause model conversion failures. ● If the dynamic image size function is enabled, the size of the dataset image used for inference must match the actual image size. ● If you set too large image sizes or too many image size choices for inference in the operating environment, you are advised to run the **swapoff -a** command to disable the memory swap function. This prevents slow running due to memory insufficiency.		

Parameter	Description	Required or Not (Depending on the Value of mode)	Default Value
--log	Level of logs to be displayed during ATC model conversion. ● **debug**: generates debug, info, warning, error, and event logs. ● **info**: generates info, warning, error, and event logs. ● **warning**: generates warning, error, and event logs. ● **error**: generates error and event logs. ● **null**: logging disabled If not specified: ● If the **/var/log/npu/conf/slog/slog.conf** configuration file exists on the server where the ATC tool is deployed, the log level specified by **global_level** in this configuration file is used. If **global_level** is set to **null**, the event-level logs can be displayed. ● If the preceding configuration files do not exist on the server where the ATC tool is deployed, the error-level logs are displayed by default.	Not	error

3 Conversion Example

The Ascend 310 chip is capable of accelerating inference under the Caffe and TensorFlow framework models. During model conversion, operator scheduling tuning, weight data rearrangement, quantization compression, and memory usage tuning can be implemented, and model preprocessing can be completed without using devices. After model training is complete, you need to convert the trained model to the model file (.om file) supported by Ascend 310 by using ATC, compile service code, and call APIs provided by ACL to implement service functions.

Preparations

1. Obtain the ATC software package.

   Before using the ATC, install the development kit Ascend-Toolkit

2. Set environment variables.

   a. Mandatory environment variables: (**${install_path}** in the following environment variables uses the default installation path /usr/local/Ascend/ascend-toolkit/latest/*xxx*-linux_gcc*x.x.x* of the development kit Ascend-Toolkit as an example.)

   ```
   export PATH=/usr/local/python3.7.5/bin:${install_path}/atc/ccec_compiler/bin:${install_path}/atc/bin:$PATH
   export PYTHONPATH=${install_path}/atc/python/site-packages/te:${install_path}/atc/python/site-packages/topi:$PYTHONPATH
   export LD_LIBRARY_PATH=${install_path}/atc/lib64:$LD_LIBRARY_PATH
   export ASCEND_OPP_PATH=${install_path}/opp
   ```

   b. Set optional environment variables:

      i. To use the auto tuning function **--auto_tune_mode="RL,GA"**, modify the environment variables **PYTHONPATH** and **LD_LIBRARY_PATH**:

      ```
      export PYTHONPATH=${install_path}/atc/python/site-packages/te:${install_path}/atc/python/site-packages/topi:${install_path}/atc/python/site-packages/auto_tune.egg/auto_tune/:${install_path}/atc/python/site-packages/schedule_search.egg:${install_path}/opp/op_impl/built-in/ai_core/tbe/:$PYTHONPATH
      export LD_LIBRARY_PATH=${install_path}/acllib/lib64:${install_path}/atc/lib64:$LD_LIBRARY_PATH
      ```

      ii. If the slog process exists in the current environment (which can be checked by running the **ps -ef | grep slog** command), logs are generated and recorded in log files by default. To display logs on the screen or redirect logs to files, set the following environment variable:


```
export SLOG_PRINT_TO_STDOUT=1
```

If the slog process does not exist in the current environment, logs are displayed on the screen by default. To redirect logs to a file, set the environment variable **SLOG_PRINT_TO_STDOUT** to **1**.

iii. If you want to convert only TBE operators using single-operator .json files, set the following environment variable:

```
export ASCEND_ENGINE_PATH=${install_path}/atc/lib64/plugin/opskernel/libfe.so:${install_path}/atc/lib64/plugin/opskernel/libge_local_engine.so
```

After the preceding command is executed, if you need to run the command for converting a third-party model to an offline model again, as shown in **Example (Converting a Caffe Model into an Offline Model)**, run the **unset ASCEND_ENGINE_PATH** command to make the **ASCEND_ENGINE_PATH** environment variable take effect first.

iv. If the model is large, you can set the following environment variable to enable parallel operator building during model conversion:

```
export TE_PARALLEL_COMPILER=xx
```

The value of **TE_PARALLEL_COMPILER** indicates the number of parallel operator building processes. The value must be an integer. The default value is **8**. As long as the value is greater than **0**, parallel building is enabled. In the single-chip scenario, you are advised to set this parameter to (80% * Host CPU core count). In the multi-chip scenario, you are advised to set this parameter to (80% * Host CPU core coun/Chip count).

NOTE

- Environment variables set by using **export** commands are valid only in the current window. If the environment variable of the ATC installation path has been set in the .bashrc file, you need to manually delete it before running the preceding commands.
- If it takes too long to convert a model in the Arm (AArch64) development environment, fix it by referring to **8.1 What Do I Do If Model Conversion Takes Too Long When the OS and Architecture Configuration of the Development Environment Is Arm (AArch64)?** .

Example (Converting a Caffe Model into an Offline Model)

Step 1 Log in to the development environment as the ATC running user and upload the model file (*.prototxt) and weight file (*.caffemodel) to be used for model conversion to any path in the development environment, for example, **$HOME/*test*/**.

Step 2 Generate a model file (the directory and file arguments in the command are for reference only):

```
atc --model=$HOME/test/resnet50.prototxt --weight=$HOME/test/resnet50.caffemodel --framework=0 --output=$HOME/test/out/caffe_resnet50 --soc_version=Ascend310
```

Step 3 If the following message is displayed, the model is successfully converted:

```
ATC run success
```

After the command is executed successfully, you can view the model file (for example, **caffe_resnet50.om**) in the path specified by the **output** parameter.

NOTE

If the user model contains custom operators, develop and deploy custom operators by referring to the *TBE Custom Operator Development Guide*. During model conversion, the custom operator library will be preferentially looked up than the built-in operator library for operators in the user model.

Step 4 (Optional) If the output node is specified by using the **--out_nodes** parameter during model conversion, the output information of the operator will not be available after the model is converted into an .om model. You can run the following command to convert the .om model into a .json file and view the output information:

```
atc --mode=1 --om=$HOME/test/caffe_resnet50.om  --json=$HOME/test/out/resnet.json
```

In the example shown in **Figure 3-1**, the **--out_nodes** parameter specifies the res4f operator as the output node. The left part shows the code without specifying the **--out_nodes** parameter, and the right part shows the code with the **--out_nodes** parameter specified.

Figure 3-1 Code with and without the --out_nodes parameter specified

----End

Example (Converting a Single-Operator JSON File into an Offline Model)

Step 1 Log in to the development environment as the ATC running user and upload the single-operator JSON file to the ***$HOME/singleop*** directory in the development environment. This section uses the single-operator GEMM with format ND as an example.

Step 2 Generate a model file (the directory and file arguments in the command are for reference only):

```
atc --singleop=$HOME/singleop/gemm.json --output=$HOME/test/out/op_model --soc_version=Ascend310
```

Step 3 If the following message is displayed, the model is successfully converted:

```
ATC run success
```

After the command is executed successfully, you can view the model file, for example, **0_*GEMM*_1_2_16_16_1_2_16_16_1_2_16_16_1_2_1_2_1_2_16_16.om**, in the path specified by the **output** parameter.

The naming rule of the generated offline model file is **SN_opType_*InputDescription* (dataType_format_shape)_*OutputDescription* (dataType_format_shape)**. The enumeration of **dataType** in the .json file is as follows:

```
typedef enum {
ACL_DT_UNDEFINED = -1, // Unknown data type (default)
  ACL_FLOAT = 0,
  ACL_FLOAT16 = 1,
  ACL_INT8 = 2,
  ACL_INT32 = 3,
  ACL_UINT8 = 4,
  ACL_INT16 = 6,
  ACL_UINT16 = 7,
  ACL_UINT32 = 8,
  ACL_INT64 = 9,
  ACL_UINT64 = 10,
  ACL_DOUBLE = 11,
  ACL_BOOL = 12,
} aclDataType;
```

----End

Example (Converting a TensorFlow Model into an Offline Model)

Step 1 Log in to the development environment as the ATC running user and upload the .pb model file to any path in the development environment, for example, **$HOME/*test*/**.

Step 2 Generate a model file (the directory and file arguments in the command are for reference only):

- If the original model has a fixed shape, that is, the values of all NHWC parameters in **input_name** are fixed, the conversion command is as follows:

  ```
  atc --model=$HOME/test/resnet18_tensorflow.pb --framework=3 --output=$HOME/test/out/tf_resnet18 --soc_version=Ascend310
  ```

- If the original model has a dynamic shape, for example, **input_*name1*:? ,h,w,c**. In this scenario, **--input_shape** is required, and set **?** to a fixed value as required. The conversion command is as follows:

  ```
  atc --model=$HOME/test/module_tensorflow.pb  --input_shape="input_name1:n,h,w,c" --framework=3  --output=$HOME/test/out/module_tf  --soc_version=Ascend310
  ```


Step 3 If the following message is displayed, the model is successfully converted:

```
ATC run success
```

After the command is executed successfully, you can view the model file (for example, **tf_resnet18.om**) in the path specified by the **output** parameter.

----End

Example (Converting a MindSpore Model to an Offline Model)

Step 1 Log in to the development environment as the ATC running user and upload the .pb model file to any path in the development environment, for example, **$HOME/*test*/**.

Step 2 Generate a model file (the directory and file arguments in the command are for reference only):

```
atc --model=$HOME/test/ReLU.pb --framework=1 --output=$HOME/test/out/ReLU_mindspore --
soc_version=Ascend310
```

Step 3 If the following message is displayed, the model is successfully converted:

```
ATC run success
```

After the command is executed successfully, you can view the model file (for example, **ReLU_mindspore.om**) in the path specified by the **output** parameter.

----End

4 AIPP Configuration

4.1 Overview

The AIPP module is introduced for image preprocessing including CSC (by converting the image format), data normalization (by subtracting the mean value or multiplying a factor), image cropping (by specifying the crop start and cropping the image to the size required by the neural network), and much more.

Static AIPP and dynamic AIPP modes are provided. However, the two modes cannot be both configured at a time.

- Static AIPP: During model conversion, set the AIPP mode to static and set the AIPP parameters. After the model is generated, the AIPP parameter values are saved in the offline model. The same AIPP parameter configurations are used in each model inference phase.

  If the static AIPP mode is used, batches share the same AIPP parameter configurations.
- Dynamic AIPP: During model conversion, set the AIPP mode to dynamic. The AIPP parameters must be set in the code of the inference engine before each model inference. In this way, different sets of AIPP parameters can be used for different model inference phases. In dynamic AIPP mode, the preprocessing settings vary according to the service requirements in the scenario where, for example, cameras use different normalization parameters or both YUV420 and RGB input formats need to be supported. For details about the APIs for setting dynamic AIPP parameters, see **See Also > ACL API Reference > Model**

Loading and Execution > aclmdlSetInputAIPP in the Application Software Development Guide.

In dynamic AIPP mode, batches use different parameter configurations (such as crop) defined by the dynamic parameter structure. For details about the dynamic parameter structure, see **4.7 Parameter Structure for Dynamic AIPP**.

AIPP supports the following image input formats: **YUV420SP_U8**, **RGB888_U8**, **XRGB8888_U8**, and **YUV400_U8**.

- For RGB888_U8, the AIPP output format varies according to the value of **rbuv_swap_switch**.
  - If **rbuv_swap_switch** in **4.2 Configuration File Template** is set to **false**, the AIPP output format is RGB888_U8.
  - If **rbuv_swap_switch** in **4.2 Configuration File Template** is set to **true**, the AIPP output format is BGR888_U8.

NOTICE

- In dynamic AIPP mode, parameters are computed for each inference operation, which is time consuming. Therefore, dynamic AIPP delivers poorer performance than static AIPP.
- When AIPP is enabled, the model input is in **RGB888_U8** or **BGR888_U8** format. The two formats correspond to different CSC matrices.
- Pay attention to the following requirements on the input images:
  - If AIPP is enabled for model conversion, the converted model accepts NHWC input only. In this scenario, the data format specified by the **--input_format** argument in the **atc** command does not take effect.
  - If AIPP is disabled for model conversion, the converted model accepts NCHW input only for inference. Therefore, you need to manually convert the NHWC data to NCHW data.
- AIPP input format mapping:
  - **YUV420SP_U8** corresponds to the YUV420SP format.
  - **RGB888_U8** represents the RGB Package format.

4.2 Configuration File Template

Due to the restrictions of hardware processing logic, parameters in the configuration file must be processed in sequence: cropping > CSC > data normalization (by subtracting the mean value and multiplying a factor) > padding. The configuration file is described as follows:

```
#Enclose each set of AIPP parameters within aipp_op { }, which is the configuration format of an AIPP operator. Dynamic AIPP allows only one set of AIPP parameters.
aipp_op {
#
# AIPP provides the following features: CSC, crop, mean subtraction, multiplication coefficient, data exchange between channels, and single-line mode.
# The input images must be of type RAW or uint8.
# To use this configuration file, delete the comments of parameters to be configured and set the parameters as required.
```



```
# The parameters in the template are set to their default values. The input_format attribute is required, while other attributes are optional.
#======================== Global settings (start) ========================================================================================================================
# aipp_mode specifies the AIPP mode. This parameter is required.
# Type: enum
# Value range: dynamic or static (dynamic indicates dynamic AIPP, and static indicates static AIPP.)
# aipp_mode:

# 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: int
# Value range: >= 0
# related_input_rank: 0

#======================== Global settings (end) ========================================================================================================================

#========================Dynamic AIPP settings (start) ========================================================================================================================
# The maximum size of the input image must be greater than or equal to the size of the original image. The dynamic AIPP parameters must be configured. (In the dynamic batch scenario, N is set to the maximum number of batch size choices.)
# Type: int
# max_src_image_size: 0
# If the input image format is YUV400_U8, max_src_image_size >= N * src_image_size_w * src_image_size_h * 1.
# If the input image format is YUV420SP_U8, max_src_image_size >= N * src_image_size_w * src_image_size_h * 1.5.
# If the input image format is XRGB8888_U8, max_src_image_size >= N * src_image_size_w * src_image_size_h * 4.
# If the input image format is RGB888_U8, max_src_image_size >= N * src_image_size_w * src_image_size_h * 3.
# Rotation enable. This parameter is reserved. Rotation is not supported currently.
# Type: bool
# Value: true (enabled) or false (disabled)
# support_rotation: false
#======================== Dynamic AIPP settings (end) ========================================================================================================================

#======================== Static AIPP settings (start) ========================================================================================================================
# Input image format
# Type: enum
# Value range: YUV420SP_U8, RGB888_U8, XRGB8888_U8, YUV400_U8
# input_format:
# Note: After model conversion is complete, the preceding arguments are displayed as the following enumerated values in the .om model file: 1, 3, 2, 4, receptively.

# Width and height of the source image
# Type: int32
# Value range: [0, 4096]. For YUV420SP_U8 images, the argument 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 specified or specified as 0, the width and height defined in the network input are used.
# src_image_size_w: 0
# src_image_size_h: 0
# If the input image format is ARGB8888_U8, XRGB8888_U8, or AYUV444_U8, (src_image_size_w x 4) %16=0 must be met. That is, (width of the original image x 4) must be rounded up to the nearest multiple of 16.

# Padding value for the C dimension. This field is reserved because the function is not supported currently.
# Type: float16
# Value range: [–65504, +65504]
```

```
# cpadding_value: 0.0

#========= Crop settings (For the configuration example, see section AIPP Configuration > Crop/
Padding Configuration.) =========
# Crop enable
# Type: bool
# Value: true (enabled) or false (disabled)
# crop: false

# Start coordinates of the cropping area. W and H defined in the network input are used as the size of the
cropped image.
# Type: int32
# Value range: [0, 4096). For YUV420SP_U8 images, the argument must be an even number.
# Note: load_start_pos_w < src_image_size_w, load_start_pos_h < src_image_size_h
# load_start_pos_w: 0
# load_start_pos_h: 0

# Size of the cropped image
# Type: int32
# Value range: even number within [0, 4096], load_start_pos_w + crop_size_w <= src_image_size_w, and
load_start_pos_h + crop_size_h <= src_image_size_h
# crop_size_w: 0
# crop_size_h: 0

# The following conditions must be met:
# If input_format = YUV420SP_U8, the values of crop_size_w, crop_size_h, load_start_pos_w, and
load_start_pos_h must be even numbers.
# If input_format = YUV422SP_U8 or input_format = YUYV_U8, the values of crop_size_w and
load_start_pos_w must be even numbers. There is no restriction on crop_size_h and load_start_pos_h.
# If input_format is set to other values, there is no restriction on crop_size_w, crop_size_h,
load_start_pos_w, and load_start_pos_h.
# If cropping is enabled, the following condition must be met: src_image_size[W|H] >= crop_size[W|H] +
load_start_pos[W|H]
# For AIPP preprocessing with resizing and padding disabled, the following conditions must be met:
# If cropping is enabled, the default size of the cropped image (crop_size[W|H]) is obtained from --
input_shape in the model file.


#======================== Resize settings ========================
# Resize enable. This parameter is reserved. This function is not supported currently.
# Type: bool
# Value: true (enabled) or false (disabled)
resize: false

# Width and height of the resized image. This parameter is reserved. This function is not supported
currently.
# Type: int32
# Value range: [16, 4096]; resize_output_w/resize_input_w ∈ [1/16, 16], resize_output_h/resize_input_h ∈
[1/16, 16]
resize_output_w: 0
resize_output_h: 0


#========= Padding settings (For the configuration example, see section AIPP Configuration > Crop/
Padding Configuration.) =========
# Padding enable
# Type: bool
# Value: true (enabled) or false (disabled)
# padding: false

#Padding values in the H and W dimensions for static AIP. The following conditions must be met:
left_padding_size <= 32, right_padding_size <= 32, top_padding_size <= 32, bottom_padding_size <= 32
# Type: int32
# left_padding_size: 0
# right_padding_size: 0
# top_padding_size: 0
# bottom_padding_size: 0
#The output H and W must meet the model requirements.
```



```
#============================== Rotation settings ================================
# Rotation angle. This parameter is reserved. Rotation is not supported currently.
# Type: uint8
# Value range: {0, 1, 2, 3}, where, 0 indicates rotation disabled, 1 indicates 90° clockwise, 2 indicates 180° clockwise, and 3 indicates 270° clockwise.
# rotation_angle :0


#========= CSC settings (For the configuration example, see section AIPP Configuration > CSC Configuration.) =============
# CSC enable
# Type: bool
# Value: true (enabled) or false (disabled)
# csc_switch: false

# R/B or U/V channel swap switch
# Type: bool
# Value: true (enabled) or false (disabled)
# rbuv_swap_switch: false

# RGBA->ARGB and YUVA->AYUV swap switch
# Type: bool
# Value: true (enabled) or false (disabled)
# ax_swap_switch: false

# Single-line mode enable (in this mode, only the first line of the cropped image is processed)
# Type: bool
# Value: true (enabled) or false (disabled)
# single_line_mode: false

# If the CSC is disabled (false), this function is bypassed.
# If the input image has 4 channels, channel A or X 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 the 3 x 3 CSC matrix
# Type: int16
# Value range: [–32677, +32676]
# 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 bias for RGB2YUV conversion
# Type: uint8
# Value range: [0, 255]
# output_bias_0: 16
# output_bias_1: 128
# output_bias_2: 128

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

#Mean value of each channel
# Type: uint8
# Value range: [0, 255]
# mean_chn_0: 0
# mean_chn_1: 0
# mean_chn_2: 0
# mean_chn_3: 0

#Minimum value of each channel
# Type: float16
# Value range: [0, 255]
# min_chn_0: 0.0
# min_chn_1: 0.0
# min_chn_2: 0.0
# min_chn_3: 0.0

# Variance value of each channel
# Type: float16
# Value range: [–65504, +65504]
# var_reci_chn_0: 1.0
# var_reci_chn_1: 1.0
# var_reci_chn_2: 1.0
# var_reci_chn_3: 1.0
}
#========================= Static AIPP settings (end)
================================================================================================
================================================
```

4.3 CSC Configuration

If the input image format is inconsistent with that required by the model, CSC needs to be performed prior to AIPP processing. The value of **matrix_r*c*** is fixed and does not need to be adjusted.

The following two typical sets of CSC configuration is provided for images or videos (in various color encoding modes such as YUV420SP_U8 and RGB888_U8) input to the model:

- For JPEG image files (such as jpg, jpeg, JPG, and JPEG), you can select from the CSC configurations in the **JPEG** columns in the following tables.
- For the decoded video data, the CSC parameters can be configured according to different color video digital standards (such as BT.601 and BT.709).

YUV420SP_U8 to YUV444

```
aipp_op {
    aipp_mode: static
    input_format : YUV420SP_U8
    csc_switch : false
    rbuv_swap_switch : false
}
```

YUV420SP_U8 to YVU444

```
aipp_op {
    aipp_mode: static
```

```
    input_format : YUV420SP_U8
    csc_switch : false
    rbuv_swap_switch : true
}
```

YUV420SP_U8 to RGB

Table 4-1 YUV420SP_U8 to RGB

JPEG	BT-601NARROW	BT-601WIDE	BT-709NARROW	BT-709WIDE

aipp_op {
aipp_mode: static
input_format : YUV420SP_U8
csc_switch : true
 	
rbuv_swap_switch : false
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
}
	aipp_op {
	aipp_mode: static
	input_format : YUV420SP_U8
	csc_switch : true
	
	rbuv_swap_switch : false
	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
	}
		aipp_op {
		aipp_mode: static
		input_format : YUV420SP_U8
		csc_switch : true
		
		rbuv_swap_switch : false
		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
		}
			aipp_op {
			aipp_mode: static
			input_format : YUV420SP_U8
			csc_switch : true
			
			rbuv_swap_switch : false
			matrix_r0c0 : 298
			matrix_r0c1 : 0
			matrix_r0c2 : 459
			matrix_r1c0 : 298
			matrix_r1c1 : -55
			matrix_r1c2 : -136
			matrix_r2c0 : 298
			matrix_r2c1 : 541
			matrix_r2c2 : 0
			input_bias_0 : 16
			input_bias_1 : 128
			input_bias_2 : 128
			}
				aipp_op {
				aipp_mode: static
				
				input_format : YUV420SP_U8
				csc_switch : true
				
				rbuv_swap_switch : false
				
				matrix_r0c0 : 256
				
				matrix_r0c1 : 0
				
				matrix_r0c2 : 403
				
				matrix_r1c0 : 256
				
				matrix_r1c1 : -48
				
				matrix_r1c2 : -120
				
				matrix_r2c0 : 256
				
				matrix_r2c1 : 475
				
				matrix_r2c2 : 0
				
				input_bias_0 : 0
				
				input_bias_1 : 128
				
				input_bias_2 : 128
				}

YUV420SP_U8 to BGR

Table 4-2 YUV420SP_U8 to BGR

JPEG	BT-601NARROW	BT-601WIDE	BT-709NARROW	BT-709WIDE

aipp_op {
aipp_mode: static
input_format : YUV420SP_U8
csc_switch : true
 	
rbuv_swap_switch : false
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
}
	aipp_op {
	aipp_mode: static
	input_format : YUV420SP_U8
	csc_switch : true
	
	rbuv_swap_switch : false
	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
	input_bias_0 : 16
	input_bias_1 : 128
	input_bias_2 : 128
	}
		aipp_op {
		aipp_mode: static
		input_format : YUV420SP_U8
		csc_switch : true
		
		rbuv_swap_switch : false
		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
		}
			aipp_op {
			aipp_mode: static
			input_format : YUV420SP_U8
			csc_switch : true
			
			rbuv_swap_switch : false
			matrix_r0c0 : 298
			matrix_r0c1 : 541
			matrix_r0c2 : 0
			matrix_r1c0 : 298
			matrix_r1c1 : -55
			matrix_r1c2 : -136
			matrix_r2c0 : 298
			matrix_r2c1 : 0
			matrix_r2c2 : 459
			input_bias_0 : 16
			input_bias_1 : 128
			input_bias_2 : 128
			}
				aipp_op {
				aipp_mode: static
				
				input_format : YUV420SP_U8
				csc_switch : true
				
				rbuv_swap_switch : false
				
				matrix_r0c0 : 256
				
				matrix_r0c1 : 475
				
				matrix_r0c2 : 0
				
				matrix_r1c0 : 256
				
				matrix_r1c1 : -48
				
				matrix_r1c2 : -120
				
				matrix_r2c0 : 256
				
				matrix_r2c1 : 0
				
				matrix_r2c2 : 403
				
				input_bias_0 : 0
				
				input_bias_1 : 128
				
				input_bias_2 : 128
				}

RGB888_U8 to RGB

```
aipp_op {
    aipp_mode: static
    input_format : RGB888_U8
    csc_switch : false
    rbuv_swap_switch : false
}
```


RGB888_U8 to BGR

```
aipp_op {
    aipp_mode : static
    input_format : RGB888_U8
    csc_switch : false
    rbuv_swap_switch : true
}
```

RGB888_U8 to YUV444

Table 4-3 RGB888_U8 to YUV444

JPEG	BT-601NARROW	BT-601WIDE	BT-709NARROW	BT-709WIDE

aipp_op {
aipp_mode: static
input_format : RGB888_U8
csc_switch : true
 	
rbuv_swap_switch : false
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
}
	aipp_op {
	aipp_mode: static
	input_format : RGB888_U8
	csc_switch : true
	
	rbuv_swap_switch : false
	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
	}
		aipp_op {
		aipp_mode: static
		input_format : RGB888_U8
		csc_switch : true
		
		rbuv_swap_switch : false
		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
		}
			aipp_op {
			aipp_mode: static
			input_format : RGB888_U8
			csc_switch : true
			
			rbuv_swap_switch : false
			matrix_r0c0 : 47
			matrix_r0c1 : 157
			matrix_r0c2 : 16
			matrix_r1c0 : -26
			matrix_r1c1 : -87
			matrix_r1c2 : 112
			matrix_r2c0 : 112
			matrix_r2c1 : -102
			matrix_r2c2 : -10
			output_bias_0 : 16
			output_bias_1 : 128
			output_bias_2 : 128
			}
				aipp_op {
				aipp_mode: static
				input_format : RGB888_U8
				csc_switch : true
				
				rbuv_swap_switch : false
				matrix_r0c0 : 55
				matrix_r0c1 : 183
				matrix_r0c2 : 19
				matrix_r1c0 : -29
				matrix_r1c1 : -99
				matrix_r1c2 : 128
				matrix_r2c0 : 128
				matrix_r2c1 : -116
				matrix_r2c2 : -12
				output_bias_0 : 0
				output_bias_1 : 128
				output_bias_2 : 128
				}

RGB888_U8 to YVU444

Table 4-4 RGB888_U8 to YVU444

JPEG	BT-601NARROW	BT-601WIDE	BT-709NARROW	BT-709WIDE

aipp_op {
aipp_mode: static
input_format : RGB8888_U8
csc_switch : true
 	
rbuv_swap_switch : false
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
}
	aipp_op {
	aipp_mode: static
	input_format : RGB888_U8
	csc_switch : true
	
	rbuv_swap_switch : false
	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
	}
		aipp_op {
		aipp_mode: static
		input_format : RGB888_U8
		csc_switch : true
		
		rbuv_swap_switch : false
		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
		}
			aipp_op {
			aipp_mode: static
			input_format : RGB888_U8
			csc_switch : true
			
			rbuv_swap_switch : false
			matrix_r0c0 : 47
			matrix_r0c1 : 157
			matrix_r0c2 : 16
			matrix_r1c0 : 112
			matrix_r1c1 : -102
			matrix_r1c2 : -10
			matrix_r2c0 : -26
			matrix_r2c1 : -87
			matrix_r2c2 : 112
			output_bias_0 : 16
			output_bias_1 : 128
			output_bias_2 : 128
			}
				aipp_op {
				aipp_mode: static
				input_format : RGB888_U8
				csc_switch : true
				
				rbuv_swap_switch : false
				matrix_r0c0 : 55
				matrix_r0c1 : 183
				matrix_r0c2 : 19
				matrix_r1c0 : 128
				matrix_r1c1 : -116
				matrix_r1c2 : -12
				matrix_r2c0 : -29
				matrix_r2c1 : -99
				matrix_r2c2 : 128
				output_bias_0 : 0
				output_bias_1 : 128
				output_bias_2 : 128
				}

RGBA8888_U8 to RGBA

```
aipp_op {
aipp_mode : static
input_format : ARGB8888_U8
csc_switch : false
rbuv_swap_switch : false
ax_swap_switch : false
}
```

RGBA8888_U8 to BGRA

```
aipp_op {
aipp_mode : static
input_format : ARGB8888_U8
csc_switch : false
rbuv_swap_switch : true
ax_swap_switch : false
}
```


RGB888_U8 to Gray

```
aipp_op {
    aipp_mode: static
    input_format : RGB888_U8
    csc_switch : true
    rbuv_swap_switch : false
    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
    output_bias_0 : 0
    output_bias_1 : 0
    output_bias_2 : 0
}
```

XRGB8888_U8 to RGB

```
aipp_op {
    aipp_mode: static
    input_format : XRGB8888_U8
    csc_switch : false
    rbuv_swap_switch : false
    ax_swap_switch : true
}
```

XRGB8888_U8 to BGR

```
aipp_op {
aipp_mode : static
input_format : XRGB8888_U8
csc_switch : false
rbuv_swap_switch : true
ax_swap_switch : true
}
```


XRGB8888_U8 to YUV444

Table 4-5 XRGB8888_U8 to YUV444

JPEG	BT-601NARROW	BT-601WIDE	BT-709NARROW	BT-709WIDE

aipp_op {
aipp_mode: static
input_format : XRGB8888_U8
csc_switch : true
 	
rbuv_swap_switch : false
 	
ax_swap_switch : true
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
}
	aipp_op {
	aipp_mode: static
	input_format : XRGB8888_U8
	csc_switch : true
	
	rbuv_swap_switch : false
	
	ax_swap_switch : true
	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
	}
		aipp_op {
		aipp_mode: static
		input_format : XRGB8888_U8
		csc_switch : true
		
		rbuv_swap_switch : false
		
		ax_swap_switch : true
		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
		}
			aipp_op {
			aipp_mode: static
			input_format : XRGB8888_U8
			csc_switch : true
			
			rbuv_swap_switch : false
			
			ax_swap_switch : true
			matrix_r0c0 : 47
			matrix_r0c1 : 157
			matrix_r0c2 : 16
			matrix_r1c0 : -26
			matrix_r1c1 : -87
			matrix_r1c2 : 112
			matrix_r2c0 : 112
			matrix_r2c1 : -102
			matrix_r2c2 : -10
			output_bias_0 : 16
			output_bias_1 : 128
			output_bias_2 : 128
			}
				aipp_op {
				aipp_mode: static
				input_format : XRGB8888_U8
				csc_switch : true
				
				rbuv_swap_switch : false
				
				ax_swap_switch : true
				matrix_r0c0 : 55
				matrix_r0c1 : 183
				matrix_r0c2 : 19
				matrix_r1c0 : -29
				matrix_r1c1 : -99
				matrix_r1c2 : 128
				matrix_r2c0 : 128
				matrix_r2c1 : -116
				matrix_r2c2 : -12
				output_bias_0 : 0
				output_bias_1 : 128
				output_bias_2 : 128
				}

XRGB8888_U8 to YVU444

Table 4-6 XRGB8888_U8 to YVU444

JPEG	BT-601NARROW	BT-601WIDE	BT-709NARROW	BT-709WIDE

aipp_op {
aipp_mode: static
input_format : XRGB8888_U8
csc_switch : true
 	
rbuv_swap_switch : false
 	
ax_swap_switch : true
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
}
	aipp_op {
	aipp_mode: static
	input_format : XRGB8888_U8
	csc_switch : true
	
	rbuv_swap_switch : false
	
	ax_swap_switch : true
	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
	}
		aipp_op {
		aipp_mode: static
		input_format : XRGB8888_U8
		csc_switch : true
		
		rbuv_swap_switch : false
		
		ax_swap_switch : true
		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
		}
			aipp_op {
			aipp_mode: static
			input_format : XRGB8888_U8
			csc_switch : true
			
			rbuv_swap_switch : false
			
			ax_swap_switch : true
			matrix_r0c0 : 47
			matrix_r0c1 : 157
			matrix_r0c2 : 16
			matrix_r1c0 : 112
			matrix_r1c1 : -102
			matrix_r1c2 : -10
			matrix_r2c0 : -26
			matrix_r2c1 : -87
			matrix_r2c2 : 112
			output_bias_0 : 16
			output_bias_1 : 128
			output_bias_2 : 128
			}
				aipp_op {
				aipp_mode: static
				input_format : XRGB8888_U8
				csc_switch : true
				
				rbuv_swap_switch : false
				
				ax_swap_switch : true
				matrix_r0c0 : 55
				matrix_r0c1 : 183
				matrix_r0c2 : 19
				matrix_r1c0 : 128
				matrix_r1c1 : -116
				matrix_r1c2 : -12
				matrix_r2c0 : -29
				matrix_r2c1 : -99
				matrix_r2c2 : 128
				output_bias_0 : 0
				output_bias_1 : 128
				output_bias_2 : 128
				}

XRGB8888_U8 to Gray

```
aipp_op {
   aipp_mode: static
   input_format : XRGB8888_U8
   csc_switch : true
   rbuv_swap_switch : false
   ax_swap_switch : true
   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
   output_bias_0 : 0
   output_bias_1 : 0
```

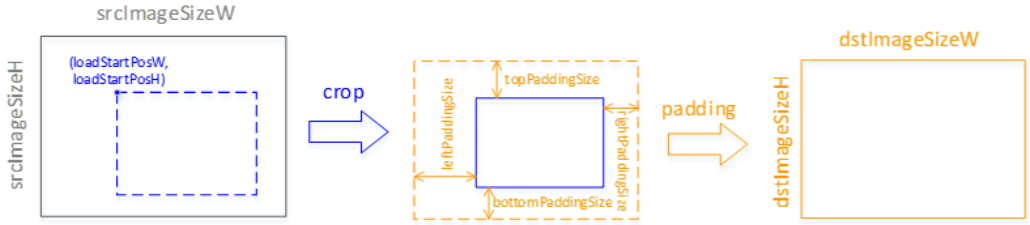


```
    output_bias_2 : 0
}
```

4.4 Cropping and Padding Configuration

Currently, AIPP supports image resizing through cropping and padding.

Figure 4-1 Image resizing



For YUV420SP_U8 images, the values of **load_start_pos_w**, **load_start_pos_h**, **crop_size_w**, and **crop_size_h** must be even numbers. The size of the cropped image must be the same as that defined in the network input.

A configuration example is as follows:

```
aipp_op {
    aipp_mode: static
    input_format : YUV420SP_U8

    src_image_size_w :320
    src_image_size_h :240

    crop :true
    load_start_pos_w :10
    load_start_pos_h :20
    crop_size_w :50
    crop_size_h :60

    padding : true
    left_padding_size :10
    right_padding_size :20
    top_padding_size :10
    bottom_padding_size :20
}
```

4.5 AIPP Configuration for a Multi-Input Model

The AIPP configuration file can define multiple sets of AIPP parameters. Separate AIPP processing is performed for different model inputs. Enclose each set of AIPP parameters within **aipp_op { }**. The following provides two sample sets of AIPP parameters.

```
aipp_op {
      related_input_rank : 0
      src_image_size_w : 608
      src_image_size_h : 608
      crop : false
      input_format : YUV420SP_U8
      aipp_mode: static
      csc_switch : true
      rbuv_swap_switch : false
```



```
    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
    mean_chn_0 : 104
    mean_chn_1 : 117
    mean_chn_2 : 123
}
aipp_op {
    related_input_rank : 1
    src_image_size_w : 608
    src_image_size_h : 608
    crop : false
    input_format : YUV420SP_U8
    aipp_mode: static
    csc_switch : true
    rbuv_swap_switch : false
    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
    mean_chn_0 : 104
    mean_chn_1 : 117
    mean_chn_2 : 123
}
```

In this example, two sets of AIPP parameters are defined. AIPP processing is performed on the first and second inputs of the model. Dynamic AIPP allows only one set of AIPP parameters.

4.6 AIPP Verification of the Model Input Size

If AIPP is configured, whether static AIPP or dynamic AIPP, the input size (**input_size**) of the eventually generated offline model is subject to operations such as cropping and padding. Assume that the batch size is **N** (the maximum batch size in the dynamic batch scenario), the input image width is **src_image_size_w**, and the input image height is **src_image_size_h**, the input image size can be calculated in the formulas in **Table 4-7**.

Table 4-7 input_size verification formulas

input_format	input_size
YUV400_U8	N * src_image_size_w * src_image_size_h * 1
YUV420SP_U8	N * src_image_size_w * src_image_size_h * 1.5

input_format	input_size
XRGB8888_U8	N * src_image_size_w * src_image_size_h * 4
RGB888_U8	N * src_image_size_w * src_image_size_h * 3

For dynamic AIPP, the ATC adds a new model input for passing the AIPP parameters during model conversion. **input_size** of the new input is calculated as follows:

```
sizeof(kAippDynamicPara) – sizeof(kAippDynamicBatchPara) + batch_count *
sizeof(kAippDynamicBatchPara)
```

- NOTE

  For details about the **kAippDynamicPara** and **kAippDynamicBatchPara** parameters, see **4.7 Parameter Structure for Dynamic AIPP**.

4.7 Parameter Structure for Dynamic AIPP

After the dynamic AIPP file is configured based on the **4.2 Configuration File Template**, the following struct will be constructed automatically based on the dynamic AIPP configuration file. No manual operation is required.

```
typedef struct tagAippDynamicBatchPara
{
    int8_t cropSwitch;              //crop switch
    int8_t scfSwitch;               //resize switch
    int8_t paddingSwitch;   // 0: unable padding,
                            // 1: padding config value,sfr_filling_hblank_ch0 ~    sfr_filling_hblank_ch2
                            // 2: padding source picture data, single row/collumn copy
                            // 3: padding source picture data, block copy
                            // 4: padding source picture data, mirror copy
    int8_t rotateSwitch;  //rotate switch, 0: rotation disabled, 1: 90° clockwise, 2: 180° clockwise, 3: 270°
clockwise
    int8_t reserve[4];
    int32_t cropStartPosW;          //the start horizontal position of cropping
    int32_t cropStartPosH;          //the start vertical position of cropping
    int32_t cropSizeW;              //crop width
    int32_t cropSizeH;              //crop height
    int32_t scfInputSizeW;          //input width of scf
    int32_t scfInputSizeH;          //input height of scf
    int32_t scfOutputSizeW;         //output width of scf
    int32_t scfOutputSizeH;         //output height of scf
    int32_t paddingSizeTop;         //top padding size
    int32_t paddingSizeBottom;      //bottom padding size
    int32_t paddingSizeLeft;        //left padding size
    int32_t paddingSizeRight;       //right padding size
    int16_t dtcPixelMeanChn0;       //mean value of channel 0
    int16_t dtcPixelMeanChn1;       //mean value of channel 1
    int16_t dtcPixelMeanChn2;       //mean value of channel 2
    int16_t dtcPixelMeanChn3;       //mean value of channel 3
#ifndef DAVINCI_TINY
    uint16_t dtcPixelMinChn0;       //min value of channel 0
    uint16_t dtcPixelMinChn1;       //min value of channel 1
    uint16_t dtcPixelMinChn2;       //min value of channel 2
    uint16_t dtcPixelMinChn3;       //min value of channel 3
    uint16_t dtcPixelVarReciChn0;   //sfr_dtc_pixel_variance_reci_ch0
    uint16_t dtcPixelVarReciChn1;   //sfr_dtc_pixel_variance_reci_ch1
    uint16_t dtcPixelVarReciChn2;   //sfr_dtc_pixel_variance_reci_ch2
    uint16_t dtcPixelVarReciChn3;   //sfr_dtc_pixel_variance_reci_ch3
    int8_t reserve1[16];            //32B assign, for ub copy
```



```
#endif
}kAippDynamicBatchPara;
typedef struct tagAippDynamicPara
{
    uint8_t inputFormat;       //Input format: YUV420SP_U8, XRGB8888_U8, or RGB888_U8
    //uint8_t outDataType; //output data type: CC_DATA_HALF,CC_DATA_INT8, CC_DATA_UINT8
    int8_t cscSwitch;              //csc switch
    int8_t rbuvSwapSwitch;         //rb/ub swap switch
    int8_t axSwapSwitch;           //RGBA->ARGB, YUVA->AYUV swap switch
    int8_t batchNum;               //batch parameter number
    int8_t reserve1[3];
    int32_t srcImageSizeW;         //source image width
    int32_t srcImageSizeH;         //source image height
    int16_t cscMatrixR0C0;         //csc_matrix_r0_c0
    int16_t cscMatrixR0C1;         //csc_matrix_r0_c1
    int16_t cscMatrixR0C2;         //csc_matrix_r0_c2
    int16_t cscMatrixR1C0;         //csc_matrix_r1_c0
    int16_t cscMatrixR1C1;         //csc_matrix_r1_c1
    int16_t cscMatrixR1C2;         //csc_matrix_r1_c2
    int16_t cscMatrixR2C0;         //csc_matrix_r2_c0
    int16_t cscMatrixR2C1;         //csc_matrix_r2_c1
    int16_t cscMatrixR2C2;         //csc_matrix_r2_c2
    int16_t reserve2[3];
    uint8_t cscOutputBiasR0;   //output bias for RGB to YUV, element of row 0, unsigned number
    uint8_t cscOutputBiasR1;   //output bias for RGB to YUV, element of row 1, unsigned number
    uint8_t cscOutputBiasR2;   //output bias for RGB to YUV, element of row 2, unsigned number
    uint8_t cscInputBiasR0;    //input bias for YUV to RGB, element of row 0, unsigned number
    uint8_t cscInputBiasR1;    //input bias for YUV to RGB, element of row 1, unsigned number
    uint8_t cscInputBiasR2;    //input bias for YUV to RGB, element of row 2, unsigned number
    uint8_t reserve3[2];
    int8_t reserve4[16];           //32B assign, for ub copy
    kAippDynamicBatchPara aippBatchPara;  //allow transfer several batch para.
} kAippDynamicPara;
```


5 Single-Operator JSON File Configuration

The configuration file can define multiple sets of single-operator JSON configurations, each set including the operator type, operator input and output information, and optional attributes. The following provides some configuration code examples. You can modify the examples as required.

1. If format = ND:

```
[
{
 "op": "GEMM",
 "input_desc": [
  {
   "format": "ND",
   "shape": [16, 16],
   "type": "float16"
  },
  {
   "format": "ND",
   "shape": [16, 16],
   "type": "float16"
  },
  {
   "format": "ND",
   "shape": [16, 16],
   "type": "float16"
  },
  {
   "format": "ND",
   "shape": [],
   "type": "float16"
  },
  {
   "format": "ND",
   "shape": [],
   "type": "float16"
  }
 ],
 "output_desc": [
  {
   "format": "ND",
   "shape": [16, 16],
   "type": "float16"
  }
 ],
 "attr": [
 {
  "name": "transpose_a",
  "type": "bool",
  "value": false
```



```
 },
 {
  "name": "transpose_b",
  "type": "bool",
  "value": false
  }
 ]
}
]
```

2. If format = NCHW:

```
[
 {
  "op": "Conv2D",
  "input_desc": [
   {
    "format": "NCHW",
    "shape": [1, 3, 16, 16],
    "type": "float16"
   },
   {
    "format": "NCHW",
    "shape": [3, 3, 3, 3],
    "type": "float16"
   }
  ],
  "output_desc": [
   {
    "format": "NCHW",
    "shape": [1, 3, 16, 16],
    "type": "float16"
   }
  ],
  "attr": [
     {
      "name": "strides",
      "type": "list_int",
      "value": [1, 1, 1, 1]
     },
     {
      "name": "pads",
      "type": "list_int",
      "value": [1, 1, 1, 1]
     },
     {
      "name": "dilations",
      "type": "list_int",
      "value": [1, 1, 1, 1]
     }
  ]
 }
]
```

3. Multiple sets of JSON configurations:

   If multiple groups of operators are configured in the JSON file, multiple .om files are generated after model conversion and are displayed in sequence: **0_*xx***, **1_*xx***, and so on. The following configuration file is only an example. Modify it as required.

```
[
 {
  "op": "MatMul",
  "input_desc": [
   {
    "format": "ND",
    "shape": [
     16,
     16
    ],
    "type": "float16"
```

```
    },
    ...
  ],
  "output_desc": [
    {
      "format": "ND",
      "shape": [
        16,
        16
      ],
      "type": "float16"
    }
  ],
  "attr": [
    {
      "name": "alpha",
      "type": "float",
      "value": 1.0
    },
   ...
  ]
},
{
  "op": "MatMul",
  "input_desc": [
    {
      "format": "ND",
      "shape": [
        256,
        256
      ],
      "type": "float16"
    },
   ...
  ],
  "output_desc": [
    {
      "format": "ND",
      "shape": [
        256,
        256
      ],
      "type": "float16"
    }
  ],
  "attr": [
    {
      "name": "alpha",
      "type": "float",
      "value": 1.0
    },
    ...
  ]
 }
]
```

The .json file consists of **OpDesc** arrays. The parameters are described as follows:

Table 5-1 OpDesc parameters

Attribute Name	Type	Description	Required or Not
op	string	Operator type	Yes
input_desc	TensorDesc array	Operator input description	Yes

Attribute Name	Type	Description	Required or Not
output_desc	TensorDesc array	Operator output description	Yes
attr	Attr array	Operator attributes	Not

Table 5-2 TensorDesc array parameters

Attribute Name	Type	Description	Required or Not
format	string	Tensor format, which is set to the format supported by the original operator framework. The options are as follows: • NCHW • NHWC • **ND**: any format • **NC1HWC0**: the 5D format defined by Huawei **C0** is closely related to the micro-architecture, and the value is equal to a size of a cube unit, for example, **16**. **C1** divides the C dimension by **C0**, that is, C1 = C/C0. When the division is not exact, the last data segment is padded to **C0**. • **FRACTAL_Z**: format of the convolution weight • **FRACTAL_NZ**: fractal format defined by Huawei. The data format of the output matrix is **NW1H1H0W0** during Cube unit computation. The matrix is divided into (H1 x W1) fractals, which are arranged in column major mode, that is, an N-shaped format. Each fractal has (H0*W0) elements, which are arranged in row major mode, that is a z-shaped format. The NW1H1H0W0 data format is referred to as an Nz format. (H0 x W0) indicates the size of a fractal, as shown in the following figure.	Yes

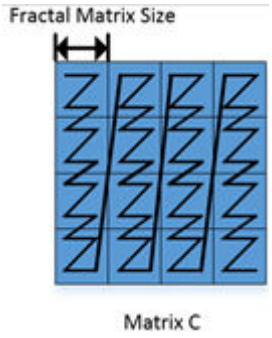
Attribute Name	Type	Description	Required or Not
			
type	string	Tensor data type. The supported types are as follows: • bool • int8 • uint8 • int16 • uint16 • int32 • uint32 • float16 • float • double	Yes
shape	int array	Tensor shape, for example, [1, 224, 224, 3]	Yes
name	string	Tensor name If the input of the operator is dynamic, this field is required. For example, if the dynamic input definition of the ConcatD operator in the operator prototype is **DYNAMIC_INPUT(input_values, ...)**, set this parameter to **"input_values0", "input_values1", ...**	Not

Table 5-3 Attr array parameters

Attribute Name	Type	Description	Required or Not
name	string	Attribute name	Yes

Attribute Name	Type	Description	Required or Not
type	string	Attribute data type. The supported types are as follows: • bool • int • float • string • list_bool • list_int • list_float • list_string • list_list_int	Yes
value	Determined by the value of **type**.	Attribute value. The attribute value varies according to **type**. • bool: true/false • int: 10 • float: 1.0 • string: "NCHW" • list_bool: [false, true] • list_int: [1, 224, 224, 3] • list_float: [1.0, 0.0] • list_string: ["str1", "str2"] • list_list_int: [[1, 3, 5, 7], [2, 4, 6, 8]]	Yes

6 Fusion Switch Configuration File

When the model miniaturization tool is used to quantize the original model, it will insert quantization and dequantization operators. When the ATC is used to convert the model, it will fuse the inserted quantization and dequantization operators. In this case, to measure the computation accuracy of the quantized model by comparing the dump files before and after the model quantization, the fusion function must be disabled by using this configuration file.

The configuration file template is as follows. Only the following **pass** rules can be configured. Disable the following rules altogether when using the configuration file.

```
RequantFusionPass:off                    // In the quantization scenario, optimize the deployment when the dequantization and quantization patterns are met to improve inference performance.
TbeConvDequantVaddReluQuantFusionPass:off   // In the quantization scenario, mark consecutive conv-dequant-vadd-relu-quant nodes with UB fusion to improve inference performance.
TbeConvDequantQuantFusionPass:off        // In the quantization scenario, mark consecutive conv-dequant-quant nodes with UB fusion to improve inference performance.
TbePool2dQuantFusionPass:off             //In the quantization scenario, mark consecutive Pool2d-quant nodes with UB fusion to improve inference performance.
```


7 Supported Operators

7.1 Caffe Network Model

7.1.1 Prototxt Customization

7.1.1.1 Overview

Operators supported by the Ascend AI Processor are classified as follows:

- Standard operators: standard Caffe operators, such as Convolution.
- Extended operators: open-source but non-standard Caffe operators, including:
  - Operators extended based on the Caffe framework, such as ROIPooling in Faster R-CNN and Normalize in SSD.
  - Operators extended based on other deep learning frameworks, such as PassThrough in YOLOv2.

Networks such as Faster R-CNN and SSD include some operator structures not defined in the Caffe framework, such as ROIPooling, Normalize, PSROI Pooling, and Upsample. To support these networks, the Caffe operators are extended for the Ascend AI Processor to reduce developers' workload of operator customization and post-processing. If these extended operators are used in Caffe networks, you need to modify or add the definition of the extension layer in the .prototxt file prior to model conversion.

This chapter provides the rundown of the extended operators supported by the Ascend AI Processor and the instructions of modifying the .prototxt file.

7.1.1.2 Extended Operator List

Table 7-1

Category	Operator Type	Description
Computation operators	**Reverse**	Reverses the dimensions of a tensor.
	ROIPooling	Performs pooling over feature maps of non-uniform input sizes in Faster R-CNN for object detection.
	PSROIPooling	Performs position-sensitive pooling over feature maps of non-uniform input sizes in R-CNN for object detection.
	Upsample	Performs upsampling using pooling mask in the YOLO network.
	Normallize	Normalizes the input tensor along the channel dimension using an L2 norm.
	Reorg	Rearranges blocks of spatial data into depth, or vice versa, in Darknet. **Implemented as a PassThrough operator as defined in the operator specifications.**
	Proposal	Filters bounding boxes (BBoxes) and outputs only those with the highest prediction confidence based on the foreground output of **rpn_cls_prob** and BBox regression output of **rpn_bbox_pred** in Faster R-CNN.
	ROIAlign	A regional feature aggregation method that solves the problem of misalignment caused by two quantifications in ROIPooling operation.
	ShuffleChannel	Permutes data in the channel dimension of the input.
	Yolo (Yolo/Detection/Region)	Replacement to Yolo, Detection, and Region operators to generate coordinates, confidence scores, and category probability of the anchor boxes on the feature map output by the convolutional network.
	PriorBox	Generates prior boxes based on the input parameters in the SSD network.
Post-processing operators	YoloV3DetectionOutput	Generates coordinates, confidence scores, and category probability of the anchor boxes on the feature map output by the convolutional network for the post-processing of YOLOv3.

Category	Operator Type	Description
	YoloV2DetectionOutput	Generates coordinates, confidence scores, and category probability of the anchor boxes on the feature map output by the convolutional network for the post-processing of YOLOv2.
	SSDDetectionOutput	Integrates the BBoxes, BBox offsets, and scores, and outputs object predictions of SSD.
	FSRDetectionOutput	Classifies the results, and outputs the final number, coordinates, category probability, and category indexes of BBoxes of Faster R-CNN.

7.1.1.3 Extended Operator Description

Extended operators can be implemented in Caffe or other deep learning frameworks. Standard .prototxt definitions are available for operators extended based on the Caffe framework, such as ROIPooling, Normalize, PSROIPolling, and Upsample. For operators extended based on frameworks other than Caffe, define them and their parameters in .prototxt format as follows.

Reverse

Reverses the dimensions of a tensor. For example, reverse from [1, 2, 3] to [3, 2, 1].

Define the operator as follows:

1. Add **ReverseParameter** to **LayerParameter**.
```
message LayerParameter {
...
optional ReverseParameter reverse_param = 157;
...
}
```
2. Define the data types and attributes of **ReverseParameter**.
```
message ReverseParameter{
  repeated int32 axis = 1;
}
```

ROIPooling

The major hurdle for going from image classification to object detection is fixed size input requirement to the network because of the existing fully connected (FC) layers. In object detection, different proposals have different shapes. Therefore, it is necessary to convert all the proposals to a fixed shape as required by FC layers.

Different images produce different feature maps after being processed by the convolution layers. The ROI Pooling layer in the Faster R-CNN network is used to produce feature maps with fixed dimensions from non-uniform inputs.

You need to extend the **caffe.proto** file and define **ROIPoolingParameter** as follows:

- **spatial_scale**: multiplicative spatial scale factor to translate ROI coordinates from their input scale to the scale used when pooling
- **pooled_h** and **pooled_w**: height and width of the ROI output feature map

1. Add **ROIPoolingParameter** to **LayerParameter**.
```
message LayerParameter {
...
optional ROIPoolingParameter roi_pooling_param = 161;
...
}
```
2. Define the data types and attributes of **ROIPoolingParameter**.
```
message ROIPoolingParameter {
    required int32 pooled_h = 1;
    required int32 pooled_w = 2;
    optional float spatial_scale = 3 [default=0.0625];
    optional float spatial_scale_h = 4;
    optional float spatial_scale_w = 5;
}
```

Example .prototxt definition of ROIPooling:

```
layer {
name: "roi_pooling"
type: "ROIPooling"
bottom: "res4f"
bottom: "rois"
bottom: "actual_rois_num"
top: "roi_pool"
roi_pooling_param {
  pooled_h: 14
  pooled_w: 14
  spatial_scale:0.0625
  spatial_scale_h:0.0625
  spatial_scale_w:0.0625
  }
}
```

PSROIPooling

Position Sensitive ROI Pooling (PSROIPooling) works in similar way to ROIPooling. However, unlike ROIPooling, the feature map output from PSROIPooling is obtained from different feature map channels, and average pooling (instead of max-pooling) is performed on each divided bin.

PSROIPooling divides the ROI into $k * k$ bins and outputs a $k * k$ feature map. The number of output channels for pooling is the same as the number of input channels.

You need to extend the **caffe.proto** file and define **PSROIPoolingParameter** as follows:

- **spatial_scale**: multiplicative spatial scale factor to translate ROI coordinates from their input scale to the scale used when pooling
- **output_dim**: number of output channels
- **group_size**: number of groups to encode position-sensitive score maps, that is, k

1. Add **PSROIPoolingParameter** to **LayerParameter**.
```
message LayerParameter {
...
optional PSROIPoolingParameter psroi_pooling_param = 207;
```



```
...
}
```

2. Define the data types and attributes of **PSROIPoolingParameter**.

```
message PSROIPoolingParameter {
  required float spatial_scale = 1;
  required int32 output_dim = 2; // output channel number
  required int32 group_size = 3; // number of groups to encode position-sensitive score maps
}
```

Example .prototxt definition of PSROIPooling:

```
layer {
  name: "psroipooling"
  type: "PSROIPooling"
  bottom: "some_input"
  bottom: "some_input"
  top: "some_output"
  psroi_pooling_param {
    spatial_scale: 0.0625
    output_dim: 21
    group_size: 7
  }
}
```

Upsample

The Upsample layer is the reverse of the Pooling layer. Each decoder upsamples the activations generated by the corresponding encoder.

The Upsample layer needs to extend the **caffe.proto** file and define **UpsampleParameter** as follows. The defined parameters include **scale**, which specifies the ratio of the output size to the input size, for example, **2**.

1. Add **UpsampleParameter** to **LayerParameter**.

```
message LayerParameter {
...
optional UpsampleParameter upsample_param = 160;
...
}
```

2. Define the data types and attributes of **UpsampleParameter**.

```
message UpsampleParameter{
    optional float scale = 1[default = 1];
    optional int32 stride = 2[default = 2];
    optional int32 stride_h = 3[default = 2];
    optional int32 stride_w = 4[default=2];
}
```

Example .prototxt definition of Upsample:

```
layer {
    name: "layer86-upsample"
    type: "Upsample"
    bottom: "some_input"
    top: "some_output"
    upsample_param {
        scale: 1
                stride: 2
    }
}
```

Normallize

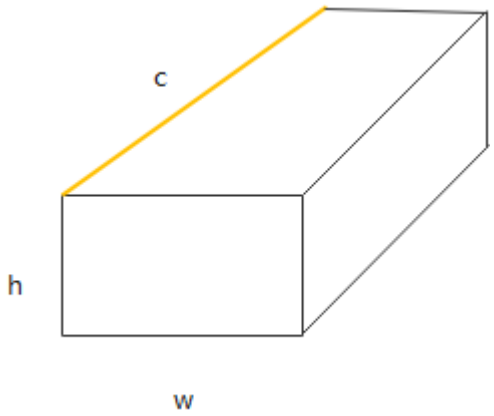
The Normalize layer is a normalization layer in the SSD network, and is mainly used to normalize elements in a space or a channel to the range [0, 1. The

Normalize layer is to output a tensor of a same size for a c*h*w three-dimensional tensor. In the formula, Normalize is calculated based on the square root of the sum of squares in the channel direction for each element. The formula is as follows:

$$x_i' = \frac{x_i}{\sqrt{\sum_{i=1}^{c} x_i^2}}$$

where, the cumulative vector of the square sum in the denominator part is the sum of the channel vectors that share the same height and width, as the orange part shown in **Figure 7-1**.

Figure 7-1 Normalize diagram



After the preceding normalization calculation, the Normalize layer scales each feature map using separate scale factors.

The Normalize layer needs to extend the **caffe.proto** file and define **NormalizeParameter** as follows. The defined parameters are as follows:

- **across_spatial**: a bool. If **True**, normalizes every channel to 1 x c x h x w. If **False**, normalizes every pixel to 1 x c x 1 x 1.
- **channels_shared**: a bool. If **True**, the scale parameters are shared across channels. Defaults to **True**.
- **eps**: a small number to avoid division by zero while normalizing.

The mathematical formulation of Normalize is as follows:

$$x_i = \frac{x_i}{\left(\sum_{1}^{n} x_i^2 + \text{eps}\right)^{\frac{1}{2}}} \times \text{scale}_i$$

Define the operator as follows:

1. Add **NormalizeParameter** to **LayerParameter**.

```
message LayerParameter {
...
optional NormalizeParameter norm_param = 206;
...
}
```


2. Define the data types and attributes of **NormalizeParameter**.

```
message NormalizeParameter {
  optional bool across_spatial = 1 [default = true];
  // Initial value of scale. Default is 1.0 for all
  optional FillerParameter scale_filler = 2;
  // Whether or not scale parameters are shared across channels.
  optional bool channel_shared = 3 [default = true];
  // Epsilon for not dividing by zero while normalizing variance
  optional float eps = 4 [default = 1e-10];
}
```

Example .prototxt definition of Normalize:

```
layer {
  name: "normalize_layer"
  type: "Normalize"
  bottom: ""some_input"
  top: "some_output"
  norm_param {
    across_spatial: false
    scale_filler {
      type: "constant"
      value: 20
    }
    channel_shared: false
  }
}
```

Reorg

The Reorg operator is implemented as a PassThrough operator in Ascend AI Processor, which rearranges blocks of spatial data into depth, or vice versa.

The PassThrough layer is not implemented using the Caffe framework. Therefore, there is no standard definition for this layer. This layer expands the feature map data in the spatial dimension to the channel dimension. The consecutive elements in the channel dimension are still consecutive in the expanded feature map.

Define the operator as follows:

1. Add **ReorgParameter** to **LayerParameter**.

```
message LayerParameter {
...
 optional ReorgParameter reorg_param = 155;
...
}
```

2. Define the data types and attributes of **ReorgParameter**.

```
message ReorgParameter{
    optional uint32 stride = 2 [default = 2];
    optional bool reverse = 1 [default = false];
}
```

Example of Reorg .prototxt definition:

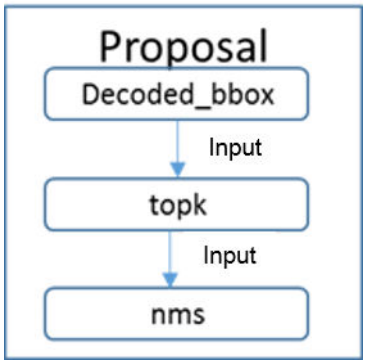
```
layer {
    bottom: "some_input"
    top: "some_output"
    name: "reorg"
    type: "Reorg"
    reorg_param {
        stride: 2
    }
}
```


Proposal

The proposal operator modifies anchors based on foreground of **rpn_cls_prob** and the BBox regression of **rpn_bbox_pred** to obtain accurate proposals.

It consists of three operators: decoded_bbox, topk, and nms, as shown in **Figure 7-2**.

Figure 7-2 Proposal implementation



Define the operator as follows:

1. Add **ProposalParameter** to **LayerParameter**.

```
message LayerParameter {
...
 optional ProposalParameter proposal_param = 201;
...
}
```

2. Define the **ProposalParameter** class and attribute parameters.

```
message ProposalParameter {
    optional float feat_stride = 1 [default = 16];
    optional float base_size = 2 [default = 16];
    optional float min_size = 3 [default = 16];
    repeated float ratio = 4;
    repeated float scale = 5;
    optional int32 pre_nms_topn = 6 [default = 3000];
    optional int32 post_nms_topn = 7 [default = 304];
    optional float iou_threshold = 8 [default = 0.7];
    optional bool output_actual_rois_num = 9 [default = false];
}
```

Example .prototxt definition of Proposal:

```
layer {
name: "faster_rcnn_proposal"
type: "Proposal"                //Operator type


bottom: "rpn_cls_prob_reshape"
bottom: "rpn_bbox_pred"
bottom: "im_info"
top: "rois"
top: "actual_rois_num"       // Added operator output
  proposal_param {

  feat_stride: 16
  base_size: 16
  min_size: 16
  pre_nms_topn: 3000
  post_nms_topn: 304
  iou_threshold: 0.7
```



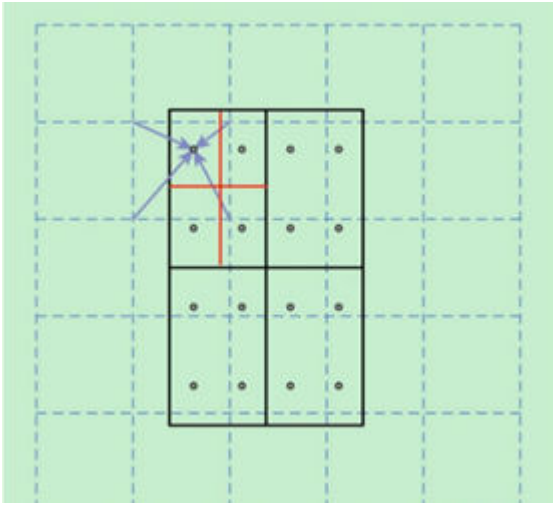
```
  output_actual_rois_num: true
  }
}
```

ROIAlign

ROIAlign is a regional feature aggregation method proposed by Mask-RCNN, which solves the problem of misalignment caused by two quantizations in ROIPooling operation.

The size of the feature map after pooling is **pooled_w * pooled_h**. Each ROI is divided into **sampling_ratio * sampling_ratio** grids of the same size. The grid points are the sampling points. As shown in **Figure 7-3**, the dashed line indicates the feature map, and the solid line indicates the ROI, which is divided into 2 x 2 cells. Assuming that the number of sampling points is 4, it means that four grids are equally divided, each of which takes its center point position. The coordinates of a sampling point are usually floating-point numbers. Therefore, you need to perform bilinear interpolation on the pixel of the sampling point (as shown by the four arrows in **Figure 7-3**) to obtain the value of the pixel. Finally, average the four pixel values as the ROIAlign result.

Figure 7-3 ROIAlign diagram



Define the operator as follows:

1. Add **ROIAlignParameter** to **LayerParameter**.

```
message LayerParameter {
...
 optional ROIAlignParameter  roi_align_param = 154;
...
}
```

2. Define the data types and attributes of **ROIAlignParameter**.

```
message ROIAlignParameter {
  // Pad, kernel size, and stride are all given as a single value for equal
  // dimensions in height and width or as Y, X pairs.
  optional uint32 pooled_h = 1 [default = 0]; // The pooled output height
  optional uint32 pooled_w = 2 [default = 0]; // The pooled output width
  // Multiplicative spatial scale factor to translate ROI coords from their
  // input scale to the scale used when pooling
  optional float spatial_scale = 3 [default = 1];
  optional int32 sampling_ratio = 4 [default = -1];
```



```
  optional int32 roi_end_mode = 5 [default = 0];
}
```

You can customize the .prototxt file based on the preceding data types and attributes.

ShuffleChannel

ShuffleChannel permutes data in the channel dimension of the input.

For example, if **channel = 4** and **group = 2**, ShuffleChannel transposes channel[1] and channel[2].

Define the operator as follows:

1. Add **ShuffleChannelParameter** to **LayerParameter**.

```
message LayerParameter {
...
 optional ShuffleChannelParameter shuffle_channel_param = 159;
...
}
```

2. Define the data types and attributes of **ShuffleChannelParameter**.

```
message ShuffleChannelParameter{
    optional uint32 group = 1[default = 1]; // The number of group
}
```

Example .prototxt definition of ShuffleChannel:

```
layer {
  name: "layer_shuffle"
  type: "ShuffleChannel"
  bottom: "some_input"
  top: "some_output"
  shuffle_channel_param {
    group: 3
  }
}
```

Yolo

The YOLO operator is introduced to the YOLOv2 network and is applied only on the YOLOv2 and YOLOv3 networks. It performs sigmoid and softmax operations on input.

- In YOLOv2, there are four scenarios based on the **background** and **softmax** parameters:
  a. background = false, softmax = true:

     sigmoid is performed on (x, y) in (x, y, h, w), sigmoid is performed on **b**, and softmax is performed on **classes**.
  b. background = false, softmax = false:

     sigmoid is performed on (x, y) in (x, y, h, w), sigmoid is performed on **b**, and sigmoid is performed on **classes**.
  c. background = true, softmax = false:

     sigmoid is performed on (x, y) in (x, y, h, w), **b** is ignored, and sigmoid is performed on **classes**.
  d. background = true, softmax = true:

     sigmoid is performed on (x, y) in (x, y, h, w), and softmax is performed on **b** and **classes**.

- In YOLOv3, there is only one scenario: sigmoid is performed on (x,y) in (x,y,h,w), sigmoid is performed on b, and sigmoid is performed on classes.

The input data format is **Tensor(n, coords+backgroup+classes,l.h,l.w)**, where **n** indicates the number of anchor boxes and **corrds** indicates x, y, w, and h, :

Define the operator as follows:

1. Add **YoloParameter** to **LayerParameter**.
```
message LayerParameter {
...
 optional YoloParameter yolo_param = 199;
...
}
```
2. Define the data types and attributes of **YoloParameter**.
```
message YoloParameter {
  optional int32 boxes = 1 [default = 3];
  optional int32 coords = 2 [default = 4];
  optional int32 classes = 3 [default = 80];
  optional string yolo_version = 4 [default = "V3"];
  optional bool softmax = 5 [default = false];
  optional bool background = 6 [default = false];
  optional bool softmaxtree = 7 [default = false];
}
```

Example .prototxt definition of Yolo:

```
layer {
    bottom: "layer82-conv"
    top: "yolo1_coords"
    top: "yolo1_obj"
    top: "yolo1_classes"
    name: "yolo1"
    type: "Yolo"
    yolo_param {
        boxes: 3
        coords: 4
        classes: 80
        yolo_version: "V3"
        softmax: true
        background: false
    }
}
```

PriorBox

The prior box is generated according to the arguments.

The following uses conv7_2_mbox_priorbox as an example. The definition is as follows:

```
layer{
    name:"conv7_2_mbox_priorbox"
    type:"PriorBox"
    bottom:"conv7_2"
    bottom:"data"
    top:"conv7_2_mbox_priorbox"
    prior_box_param{
        min_size:162.0
        max_size:213.0
        aspect_ratio:2
        aspect_ratio:3
        flip:true
        clip:false
        variance:0.1
        variance:0.1
```



```
    variance:0.2
    variance:0.2
    img_size:300
    step:64
    offset:0.5
  }
}
```

1. A prior box is generated when the width and height are both **min_size**.
2. If **max_size** is available, **sqrt(min_size x max_size)** is used to determine the width and height of generated boxes (max_size > min_size).
3. The prior box is generated based on the aspect ratios (1/2 and 1/3 according to the definition).

Therefore, num_priors_ = min_sizes + aspect_ratios * min_size + max_size

Define the operator as follows:

1. Add **PriorBoxParameter** to **LayerParameter**.

```
message LayerParameter {
...
 optional PriorBoxParameter prior_box_param = 203;
...
}
```

2. Define the data types and attributes of **PriorBoxParameter**.

```
message PriorBoxParameter {
  // Encode/decode type.
  enum CodeType {
    CORNER = 1;
    CENTER_SIZE = 2;
    CORNER_SIZE = 3;
  }
  // Minimum box size (in pixels). Required!
  repeated float min_size = 1;
  // Maximum box size (in pixels). Required!
  repeated float max_size = 2;
  // Various of aspect ratios. Duplicate ratios will be ignored.
  // If none is provided, we use default ratio 1.
  repeated float aspect_ratio = 3;
  // If true, will flip each aspect ratio.
  // For example, if there is aspect ratio "r",
  // we will generate aspect ratio "1.0/r" as well.
  optional bool flip = 4 [default = true];
  // If true, will clip the prior so that it is within [0, 1]
  optional bool clip = 5 [default = false];
  // Variance for adjusting the prior bboxes.
  repeated float variance = 6;
  // By default, we calculate img_height, img_width, step_x, step_y based on
  // bottom[0] (feat) and bottom[1] (img). Unless these values are explicitely
  // provided.
  // Explicitly provide the img_size.
  optional uint32 img_size = 7;
  // Either img_size or img_h/img_w should be specified; not both.
  optional uint32 img_h = 8;
  optional uint32 img_w = 9;

  // Explicitly provide the step size.
  optional float step = 10;
  // Either step or step_h/step_w should be specified; not both.
  optional float step_h = 11;
  optional float step_w = 12;

  // Offset to the top left corner of each cell.
  optional float offset = 13 [default = 0.5];
}
```

Example .prototxt definition of PriorBox:


```
layer {
  name: "layer_priorbox"
  type: "PriorBox"
  bottom: "some_input"
  bottom: "some_input"
  top: "some_output"
  prior_box_param {
    min_size: 30.0
    max_size: 60.0
    aspect_ratio: 2
    flip: true
    clip: false
    variance: 0.1
    variance: 0.1
    variance: 0.2
    variance: 0.2
    step: 8
    offset: 0.5
  }
}
```

7.1.1.4 Caffe Operator Specifications

General restrictions:

1. Unless otherwise specified, there is a restriction for the input and output: w <= 4096; h<= 4096

Operator	Classify	Name	TypeRange	Required	Doc	Restrictions
Pooling	INPUT	x	float16	TRUE	Performs pooling on the input **x** and outputs **y**.	float16 only
	OUTPUT	y	float16	TRUE		float16 only
	ATTR	pool	int	FALSE	Pooling method • **0** (default): MAX • **1**: AVE	0 or 1
	ATTR	kernel_size	int	FALSE	Kernel size	Supported. When **global_pooling==False,pool==1**, the restrictions are as follows: 1. x_w (after
	ATTR	kernel_h	int	FALSE	Kernel height. (If **global_pooling==False**, then **kernel_size** and **kernel_h/kernel_w** are mutually exclusive.)	

Operator	Classify	Name	TypeRange	Required	Doc	Restrictions
	ATTR	kernel_w	int	FALSE	Kernel width	padding) > kernel_w; 2. Filter_h*filter_w*32+feature_w*(31*stride_h+filter_w)<=32768, 3. **filter_w** or **filter_h** ∈ [1, 255]
	ATTR	stride	int	FALSE	Stride height and width (default = 1). **stride_h** and **stride_w** are preferred (if specified).	1–63
	ATTR	stride_h	int	FALSE	Stride height (default = 1)	
	ATTR	stride_w	int	FALSE	Stride width (default = 1)	
	ATTR	pad	int	FALSE	Padding height and width (default = 0). **pad_h** is preferred (if specified).	pad<=kernel
	ATTR	pad_h	int	FALSE	Padding height (default = 0)	
	ATTR	pad_w	int	FALSE	Padding width (default = 0)	
	ATTR	global_pooling	int	FALSE	Whether to perform global pooling. Defaults to **false**.	True or False

Operator	Classify	Name	TypeRange	Required	Doc	Restrictions
	ATTR	round_mode	int	FALSE	Rounding type • **0** (default): DOMI_POOLING_CEIL • 1:DOMI_POOLING_FLOOR	0 or 1
Eltwise	DYNAMIC_INPUT	x	float16,float32	TRUE	Input tensor. The maximum tensor size is 32.	float16 only, bottom num <= 32
	OUTPUT	y	float16,float32	TRUE	Output tensor. Has the same data type and shape as the input tensor. Computes element-wise operations, such as product, sum, and max.	float16 only
	ATTR	operation	int	FALSE	Element-wise operation • 0:product • **1** (default): sum • 2:max	0–2
	ATTR	coeff	ListFloat	FALSE	If specified, has the same vector length as bottom num. Coefficient to by multiplied for the sum operation (default = {}).	If specified, has the same vector length as bottom num.
InnerProduct	INPUT	x	float16	TRUE	Input tensor	float16
	INPUT	w	float16	TRUE	Weight tensor	float16

Operator	Classify	Name	TypeRange	Required	Doc	Restrictions
	OPTIONAL_INPUT	b	float16,float32	FALSE	bias	float16
	OUTPUT	y	float16,float32	TRUE	Output tensor	float16
	ATTR	num_output	Int	TRUE	Number of neurons output by InnerProduct	Up to 32768
	ATTR	transpose	bool	FALSE	Whether to transpose (default = False)	True or False
	ATTR	bias_term	bool	FALSE	Whether to have bias (default = True)	True or False
	ATTR	axis	int	FALSE	Axis of InnerProduct.	1 or 2
Softmax	INPUT	x	float16,float32	TRUE	Input tensor	float16
	OUTPUT	y	float16,float32	TRUE	Output tensor	float16

Operator	Classify	Name	TypeRange	Required	Doc	Restrictions
	ATTR	axis	int	FALSE	Axis along which to perform the softmax	vector c<=16384(cs)/ 24576(ES) tensor c<=4096
ReLU, LeakyReLU, RReLU	INPUT	x	float16, float32	TRUE	Input	float16
	OUTPUT	y	float16, float32	TRUE	Output y = max(0, x) +negative_slope * min(0, x)	float16
	ATTR	negative_slope	float	FALSE	Negative slope (default = 0)	float16
Proposal	INPUT	cls_prob	float16,fp32	TRUE	Probability that a proposal is of the foreground or background class. [batch, 2*num_anchors, height, width]. **2** indicates the probabilities that a proposal is of the background class and foreground class respectively, that is **bg prob** and **fg probs** respectively.	float16
	INPUT	bbox_delta	float16,fp32	TRUE	Bounding box location formatted (batch, num_anchors * 4, height, width)	float16
	INPUT	im_info	-	FALSE	Height and width of the input image (default = {375, 1240})	-

Operator	Classify	Name	TypeRange	Required	Doc	Restrictions
	OUTPUT	rois	float16,fp32	TRUE	ROI information, with shape [batch, 5-tuple, post_nms_topn], where **post_nms_topn** is a multiple of 16, and the 5-tuple is formatted (batchID, x1, y1, x2, y2). Each batch is output based on **actual_rois_num**.	float16
	OUTPUT	actual_rois_num	int32	FALSE	Actual number of ROIs output per batch tensor, with shape [batch, 8-tuple], of type int32. Only the first tuple is valid.	int32
	ATTR	feat_stride	float	TRUE	Stride size after extracting features from an image (default = 16)	-
	ATTR	base_size	float	FALSE	Size of the generated anchors (default = 16)	-
	ATTR	min_size	float	FALSE	Minimum size of an anchor (default = {16, 16})	-
	ATTR	ratio	ListFloat	FALSE	Aspect ratio of generated anchors (default = [0.5,1,2])	-
	ATTR	scale	ListFloat	FALSE	Ratio of the height and width of generated anchors to **base_size** (default = {8, 16, 32}).	-
	ATTR	pre_nms_topn	int	FALSE	Number of BBoxes before the NMS operation. In the float16 scenario, up to 3008 for Ascend AI Processor; else, up to 6008. In the FP32 scenario, up to 1502 for Ascend AI Processor; else, up to 3004.	Up to 1502
	ATTR	post_nms_topn	int	FALSE	Number of BBoxes after the NMS operation. Up to 3000 for Ascend AI Processor; else, up to 6000. With FP32 input, up to 1502 for Ascend AI Processor (default = 304); else, up to 3000.	Up to 1502

Operator	Classify	Name	TypeRange	Required	Doc	Restrictions
	ATTR	iou_threshold	float	FALSE	Intersection over Union threshold (default = 0.7). The value range is (0, 1].	(0,1]
	ATTR	output_actual_rois_num	bool	FALSE	Whether to generate **actual_rois_num** **False** (default): not **True**: yes	-
BatchNorm	INPUT	x	float16,float32	TRUE	Input tensor	-
	INPUT	mean	float16,float32	TRUE	Mean value	float16
	INPUT	variance	float16,float32	TRUE	Variance	float16
	INPUT	scale_factor	float16,float32	TRUE	Scale factor to be divided from each mean and variance	float16

Operator	Classify	Name	TypeRange	Required	Doc	Restrictions
	INPUT	scale	float16,float32	FALSE	Reserved	-
	INPUT	offset	float16,float32	FALSE	Reserved	-
	OUTPUT	y	float16,float32	FALSE	Output tensor	float16
	ATTR	eps	float	FALSE	Epsilon for not dividing by zero (default = 1e – 5)	Configurable
	ATTR	use_global_stats	bool	FALSE	Whether to use the mean and variance parameters. Fixed at **True**.	True only

Operator	Classify	Name	TypeRange	Required	Doc	Restrictions
ROIPooling	INPUT	x	float16,float32	TRUE	Input feature map	float16 only. The value ranges of **h** and **w** vary with **pooled_h** and **pooled_w**. 1) If pooled_h=pooled_w: 2, 17, then, h and w <= 50 2) If pooled_h=pooled_w: 4,5,10-16, then, h and w<=70 3) If pooled_h=pooled_w: 7,8, then, h and w <= 80 4) If pooled_h=pooled_w: 3, then, h and w <= 60 5) If pooled_h=pooled_w: 18-20, then, h and w <= 40

Operator	Classify	Name	TypeRange	Required	Doc	Restrictions
	INPUT	rois	float16,float32	TRUE	ROIS information formatted [batch, 5-tuple, N], where, **N** is a multiple of 16.	float16
	INPUT	roi_actual_num	int	FALSE	A tensor formatted [batch, 8-tuple]. The actual number of ROIs output per batch.	int32
	OUTPUT	y	float16,float32	TRUE	Crops the feature map based on the ROI coordinates, and then perform max pooling based on the output size configuration.	float16
	ATTR	pooled_h	int	TRUE	Height of the ROI output feature map, greater than 0	[2,20]
	ATTR	pooled_w	int	TRUE	Width of the ROI output feature map, greater than 0	[2,20]
	ATTR	spatial_scale	float	FALSE	Ratio of the input feature map over the input image size. If **spatial_scale_h** and **spatial_scale_w** are specified, **spatial_scale_h** and **spatial_scale_w** applies. If **spatial_scale_h** and **spatial_scale_w** are not specified, convert **spatial_scale** to **spatial_scale_h** and **spatial_scale_w** in the Caffe plug-in.	-
	ATTR	spatial_scale_h	float	FALSE	Ratio of the input feature map over the input image height. Defaults to **0.0625**.	-
	ATTR	spatial_scale_w	float	FALSE	Ratio of the input feature map over the input image width. Defaults to **0.0625**.	-

Operator	Classify	Name	TypeRange	Required	Doc	Restrictions
AbsVal	INPUT	x	float16,float32,int32	-	Returns the absolute value: y = |x|	float16
	OUTPUT	y	float16,float32,int32	-		float16
Bias	INPUT	x	float32, float16	TRUE	Input tensor	float16
	INPUT	bias	float32, float16	TRUE	bias	-
	OUTPUT	y	float32, float16	TRUE	Output tensor	-

Operator	Classify	Name	TypeRange	Required	Doc	Restrictions
	ATTR	axis	int	FALSE	The first axis of bottom[0] along which to apply bottom[1] (default = 1). Assume that bottom[0] is 4D with shape 100 x 3 x 40 x 60, the output top[0] will have the same shape, and bottom[1] may have any of the following shapes (for the given value of **axis**): (axis == 0 == -4) 100; 100x3; 100x3x40; 100x3x40x60 (axis == 1 == -3) 3; 3x40; 3x40x60 (axis == 2 == -2) 40; 40x60 (axis == 3 == -1) 60 Furthermore, bottom[1] may have the empty shape (regardless of the value of **axis**) -- a scalar bias.	[-rank(x), rank(x)), complies with the Caffe rules.
	ATTR	num_axes	int	FALSE	The number of axes of the input (bottom[0]) covered by the **bias** parameter. Set to **–1** to cover all axes of bottom[0] starting from **axis**. Set to **0** to add a scalar. (default = 1) Ignored if the input comes from bottom[1] on the fly.	Value range: [–1, rank(x)). If axis_= axis > 0: axis, axis +rank(x), then, num_axes +axis_ <= rank(x).
BNLL	INPUT	x	float16,float32	TRUE	Activation function: if x >0: y = x + log(1 + exp(–x)) else: y = log(1 + exp(x))	float16
	OUTPUT	y	float16,float32	TRUE		float16

Operator	Classify	Name	TypeRange	Required	Doc	Restrictions
Crop	INPUT	x	float16,float32,int8,uint8,int16,uint16,int32,uint32,int64,uint64	TRUE	Crops the input tensor **x** to the shape of **size**. Description: (1) **x**: bottom to be cropped, of size (20, 50, 512, 512). (2) **size**: destination size, of size (20, 10, 256, 256) (3) **y**: output top, cropped from **x**. Has the same shape as **size**. **axis** determines the dimension from which the cropping starts and **offsets** determines the cropping offsets of the three dimensions. The cropped length corresponds to the length of the dimension size of **x2**. Example: (1) axis = 1, offsets = (25, 128, 128) (2) C = A[:,25:25+size.shape[1], 128:128+size.shape[2], 128:128+size.shape[3]]	-

Operator	Classify	Name	TypeRange	Required	Doc	Restrictions
	INPUT	size	float16,float32,int8,uint8,int16,uint16,int32,uint32,int64,uint64	TRUE		-

Operator	Classify	Name	TypeRange	Required	Doc	Restrictions
	OUTPUT	y	float16,float32,int8,uint8,int16,uint16,int32,uint32,int64,uint64	TRUE		-
	ATTR	axis	Anint. Defaults to **2**.	FALSE		-
	ATTR	offset	ListInt	TRUE		-

Operator	Classify	Name	TypeRange	Required	Doc	Restrictions
Power	INPUT	x	float16, float32	TRUE	Power function: y = (shift + scale * x) ^ power	float16
	OUTPUT	y	float16, float32	TRUE		float16
	ATTR	power	float	FALSE	(default = 1.0)	The power has decimal places. The base must be greater than 0. For other restrictions, see the Caffe definition.
	ATTR	scale	float	FALSE	(default = 1.0)	-
	ATTR	shift	float	FALSE	(default = 0.0)	-
TanH	INPUT	x	float16, float32	TRUE	y=tanh(x)	float16

Operator	Classify	Name	TypeRange	Required	Doc	Restrictions
	OUTPUT	y	float16, float32	TRUE		float16
Reverse	INPUT	x	float16,floaat32,int8, uint8,int16, uint16,int32, uint32,int64, uint64	TRUE	Input tensor	-

Operator	Classify	Name	TypeRange	Required	Doc	Restrictions
	OUTPUT	y	float16,float32,int8,uint8,int16,uint16,int32,uint32,int64,uint64	TRUE	Output tensor. Has the data type and shape as the input tensor.	-
	ATTR	axis	int	TRUE	A vector specifying the axis. The value range is [–rank(x), rank(x)).	[-rank(x), rank(x))
Normalize	INPUT	x1	float32,float16	TRUE	Input tensor	float16

Operator	Classify	Name	TypeRange	Required	Doc	Restrictions
	INPUT	x2	float32, float16	TRUE	An ND vector specifying the normalizing scale. If channel_shared==True, then N=1. If channel_shared==False, then N=channel.	float16
	OUTPUT	y	float32, float16	TRUE	Output tensor	float16
	ATTR	across_spatial	bool	FALSE	If True, normalizes CHW. If False, normalizes C only. (default= True)	True or False
	ATTR	channel_shared	bool	FALSE	Controls **x2**. (default = True)	True or False
	ATTR	eps	-	FALSE	Epsilon for not dividing by zero (default = 1e – 10)	-
PSROIPooling	INPUT	x	float16	TRUE	feature Map	float16
	INPUT	rois	float16	TRUE	Has shape [batch, 5-tuple, rois_num], where the 5-tuple is formatted (batchID, x1, y1, x2, y2).	float16
	OUTPUT	y	float16	TRUE	Output obtained from different feature map channels, similar to ROIPooling	float16
	ATTR	spatial_scale	float	TRUE	Multiplicative spatial scale factor to translate ROI coordinates from their input scale to the scale used when pooling	The value must be greater than 0.

Operator	Classify	Name	TypeRange	Required	Doc	Restrictions
	ATTR	output_dim	int	TRUE	Output channels, greater than 0	Supported. Outputs the dimension size of ROI channels.
	ATTR	group_size	int	TRUE	Height and width of the output feature	Outputs the H and W sizes of ROIs. H = W; input_channel==out_channel * group_size^2
Permute	INPUT	x	float16,fp32	TRUE	Permutes the input.	float16
	OUTPUT	y	float16,fp32	TRUE		float16
	ATTR	order	listInt	TRUE	Transpose order	-
PRelu	INPUT	x	int8,uint8,float16,float32	TRUE	Activation function: y = weight min(x, 0) + max(x, 0)	float16

Operator	Classify	Name	TypeRange	Required	Doc	Restrictions
	INPUT	weight	int8,uint8,float16,float32	TRUE	Whether slope parameters are shared across channels is determined based on the dimension count of **weight**. If 1D, **channel_shared==True**; else **channel_shared==False**.	Float16 only, a vector or scalar of the channel dimension
	OUTPUT	y	int8,uint8,float16,float32	TRUE	-	float16

Operator	Classify	Name	TypeRange	Required	Doc	Restrictions
PassThrough	INPUT	x	float16,float32,int8,uint8,int16,uint16,int32,uint32,int64,uint64	TRUE	Rearranges blocks of spatial data into depth, or vise versa.	-

Operator	Classify	Name	TypeRange	Required	Doc	Restrictions
	OUTPUT	y	float16,float32,int8,uint8,int16,uint16,int32,uint32,int64,uint64	TRUE		-
	ATTR	stride	int	FALSE	Stride (default = 2)	float16
	ATTR	reverse	bool	FALSE	If True, DepthToSpace; If False, SpaceToDepth.	float16
scale	INPUT	x	float32,float16	TRUE	Input tensor	-

Operator	Classify	Name	TypeRange	Required	Doc	Restrictions
	INPUT	scale	float32, float16	TRUE	Scaling factor	-
	INPUT	bias	float32, float16	FALSE	bias tensor	-
	OUTPUT	y	float32, float16	TRUE	Output tensor	-
	ATTR	axis	int	FALSE	The start dimension for scaling (default = 1, that is bottom[0]). If the **axis** value is inconsistent, bottom[1] has a different shape. Assume that bottom[0] is 4D with shape 100 x 3 x 40 x 60, the output top[0] will have the same shape, and bottom[1] may have any of the following shapes (for the given value of **axis**): (axis == 0 == -4) 100; 100x3; 100x3x40; 100x3x40x60 (axis == 1 == -3) 3; 3x40; 3x40x60 (axis == 2 == -2) 40; 40x60 (axis == 3 == -1) 60 Furthermore, bottom[1] may have the empty shape (regardless of the value of **axis**) -- a scalar multiplier.	[-rank(x), rank(x))

Operator	Classify	Name	TypeRange	Required	Doc	Restrictions
	ATTR	num_axes	int	FALSE	The number of axes of the input (bottom[0]) covered by the **scale** parameter (default = 1). Ignored unless just one bottom is given. Set to **–1** to cover all axes of bottom[0] starting from **axis**. Set to **0** to multiply with a scalar.	Value range: [–1, rank(x)). If axis_= axis > 0: axis, axis +rank(x), then, num_axes +axis_ <= rank(x).
Convolution (DepthwiseConv, ConvolutionDepthwise)	INPUT	x	float16	TRUE	Convolves the input x.	-

Operator	Classify	Name	TypeRange	Required	Doc	Restrictions

	INPUT	filter	float16	TRUE	Convolution kernel
					Both DepthwiseConv and ConvolutionDepthwise converted into a convolution operator before use, for example, **ConvolutionDepthwise**.
					layer {
					name: "resx1_conv2"
					type: "**ConvolutionDepthwise**"
					bottom: "resx1_conv1"
					top: "resx1_conv2"
					convolution_param {
					num_output: 54
					kernel_size: 3
					stride: 2
					pad: 1
					bias_term: false
					weight_filler {
					type: "msra"
					}
					}
					}
					Modify as follows:
					layer {
					name: "resx1_conv2"
					type: "**Convolution**"
					bottom: "resx1_conv1"
					top: "resx1_conv2"
					convolution_param {
					num_output: 54
					group: 54
					kernel_size: 3
					stride: 2
					pad: 1
					bias_term: false
					weight_filler {
					type: "msra"
					}
					}
					}
					If the **group** attribute is carried in DepthwiseConv or ConvolutionDepthwise, change **type** to **Convolution**. Otherwise, add the **group** attribute: **group==num_output** (Note that **group**, **out_channel**, and **input_channel** must be consistent).
					NOTE
					The op name and op type in the modified model prototxt must be the same as those in the .caffemodel weight file (case sensitive).
						Restrictions:
						1) If **out_w==out_h==1**, then: **feature_w==feature_h==filter_w==filter_h** (value range: 1–11)
						2) Output whose W==1 while H ≠ 1 is not supported.
						3) If **group==Channel**, then: filter_h * filter_w * 32 + feature_w * (31 * stride_h + filter_w) <= 32768, with **filter_w** and **filter_h** within the range 1–255.
						3) **filter_w** and **filter_h** are within the range 1–255 in other situations.

Operator	Classify	Name	TypeRange	Required	Doc	Restrictions
	INPUT	bias	float16	FALSE	The bias. If **None**, applies no bias.	-
	OUTPUT	y	float16	TRUE	-	-
	ATTR	pad	ListInt	FALSE	Padding size. **pad**, and **pad_h/pad_w** are mutually exclusive. Defaults to **0**.	pad<filter.size
	ATTR	pad_h	int	FALSE		
	ATTR	pad_w	int	FALSE		
	ATTR	stride	ListInt	FALSE	Stride. **stride**, and **stride_h/stride_w** are mutually exclusive. Defaults to **1**.	1<=stride<=63
	ATTR	stride_h	int	FALSE		
	ATTR	stride_w	int	FALSE		
	ATTR	dilation	ListInt	FALSE	Factor used to dilate the kernel. A list of length up to 2 (default = 1).	Value range: 1–255. 1)*dilation +1<256;
	ATTR	group	int	FALSE	-	(kernel – group) is divisible by **channel**.
Deconvolution	INPUT	x	float16	TRUE	Input tensor	float16

Operator	Classify	Name	TypeRange	Required	Doc	Restrictions
	INPUT	filter	float16	TRUE	Convolution kernel. Has shape [H, W, filter_channel, filter_number], where, **filter_channel** must be the same as that of **x**.	Restrictions: 1) If **out_w==out_h==1**, then: **feature_w==feature_h==filter_w==filter_h** (value range: 1–11) 2) Output whose W==1 while H ≠ 1 is not supported. 3) If **group==Channel**, then: filter_h * filter_w * 32 + feature_w * (31 * stride_h + filter_w) <= 32768, with **filter_w** and **filter_h** within the range 1–255. 3) **filter_w** and **filter_h** are within the range 1–255 in other situations.

Operator	Classify	Name	TypeRange	Required	Doc	Restrictions
	INPUT	bias	float16	FALSE	-	float16
	OUTPUT	y	float16	TRUE	-	-
	ATTR	pad	ListInt	FALSE	Padding size (default = 0). **pad** and **pad_h/pad_w** are mutually exclusive. **pad** is a list of up to 2 elements.	pad < filter.size
	ATTR	pad_h	int	FALSE		
	ATTR	pad_w	int	FALSE		
	ATTR	stride	ListInt	FALSE	Stride (default = 1). **stride** and **stride_h/stride_w** are mutually exclusive. **stride** is a list of up to 2 elements.	0<=stride<=63
	ATTR	stride_h	int	FALSE		
	ATTR	stride_w	int	FALSE		
	ATTR	dilation	ListInt	FALSE	Factor used to dilate the kernel. A list of length up to 2 (default = 1).	Value range: 1–255. 1)*dilation +1<256;
	ATTR	group	int	FALSE	-	(kernel – group) is divisible by **channel**.

Operator	Classify	Name	TypeRange	Required	Doc	Restrictions
ELU	INPUT	x	float16,float32	TRUE	Activation function, which integrates sigmoid and ReLU: if x > 0: y = x else: y = alpha*(exp(x)-1)	float16
	OUTPUT	y	float16,float32	TRUE		float16
	ATTR	alpha	default:1	FALSE		float16

Operator	Classify	Name	TypeRange	Required	Doc	Restrictions
Slice	INPUT	x	float16,float32,int8,uint8,int16,uint16,int32,uint32,int64,uint64	TRUE	default:1	-

Operator	Classify	Name	TypeRange	Required	Doc	Restrictions
	OUTPUT	y	float16,float32,int8,uint8,int16,uint16,int32,uint32,int64,uint64	TRUE		-
	ATTR	slice_dim	uint	-	Alias for **axis** (default = 1)	Value range: [1, rank(x)), Caffe compatible
	ATTR	slice_point	uint	-	Number of sliced blobs minus 1. If not defined, the input blob is evenly sliced.	< 32
	ATTR	axis	int	-	Begin axis for slicing (default = 1)	Value range: [–rank(x), rank(x)), Caffe compatible

Operator	Classify	Name	TypeRange	Required	Doc	Restrictions
Exp	INPUT	x	float16,float32	TRUE	Computes exponential of **x**. Description: if base>0: y = base ^ (shift + scale * x) if base == -1: y = exp(shift + scale * x)	float16
	OUTPUT	y	float16,float32	TRUE		float16
	ATTR	base	float	FALSE	The base (default = –1.0)	> 0 or = –1
	ATTR	scale	float	FALSE	(default = 1.0)	-
	ATTR	shift	float	FALSE	(default = 0)	-

Operator	Classify	Name	TypeRange	Required	Doc	Restrictions
Flatten	INPUT	x	float16,float32,int8,uint8,int16,uint16,int32,uint32,int64,uint64	TRUE	Input tensor	-

Operator	Classify	Name	TypeRange	Required	Doc	Restrictions
	OUTPUT	y	float16,float32,int8,uint8,int16,uint16,int32,int64,uint64	TRUE	Output tensor	-
	ATTR	axis	int	FALSE	The first axis to flatten (default = 1). May be negative to index from the end (for example, **–1** for the last axis).	[-rank(x), rank(x))
	ATTR	end_axis	int	FALSE	The last axis to flatten (default = 1). May be negative to index from the end (for example, **–1** for the last axis).	[0, rank(x))
INPUT	INPUT	images	uint8	TRUE	Input tensor	-

Operator	Classify	Name	TypeRange	Required	Doc	Restrictions

	INPUT	params	uint8	FALSE	Data flow of the dynamic AIPP structure: kAippDynamicPara + (batchNum – 1) * kAippDynamicBatchPara
					/**
					* @ingroup dnn
					* @brief struct define of dynamic aipp batch parameter.
					*/
					typedef struct tagAippDynamicBatchPara
					{
					int8_t cropSwitch; //crop switch
					int8_t scfSwitch; //resize switch
					int8_t paddingSwitch; //0: unable padding, 1: padding config value,sfr_filling_hblank_ch0 ~ sfr_filling_hblank_ch2
					//2: padding source picture data, single row/ collumn copy
					//3: padding source picture data, block copy
					//4: padding source picture data, mirror copy
					int8_t rotateSwitch; //rotate switch, 0: no rotation, 1: rotate 90° clockwise, 2: rotate 180° clockwise, 3: rotate 270° clockwise
					int8_t reserve[4];
					int32_t cropStartPosW; //the start horizontal position of cropping
					int32_t cropStartPosH; //the start vertical position of cropping
					int32_t cropSizeW; //crop width
					int32_t cropSizeH; //crop height
					int32_t scfInputSizeW; //input width of scf
					int32_t scfInputSizeH; //input height of scf
					int32_t scfOutputSizeW; //output width of scf
					int32_t scfOutputSizeH; //output height of scf
					int32_t paddingSizeTop; //top padding size
					int32_t paddingSizeBottom; //bottom padding size
					int32_t paddingSizeLeft; //left padding size
					int32_t paddingSizeRight; //right padding size
					int16_t dtcPixelMeanChn0; //mean value of channel 0
					int16_t dtcPixelMeanChn1; //mean value of channel 1
					int16_t dtcPixelMeanChn2; //mean value of channel 2
					int16_t dtcPixelMeanChn3; //mean value of channel 3
					uint16_t dtcPixelMinChn0; //min value of channel 0
					uint16_t dtcPixelMinChn1; //min value of channel 1
						-

Operator	Classify	Name	TypeRange	Required	Doc	Restrictions

					uint16_t dtcPixelMinChn2; //min value of channel 2
					uint16_t dtcPixelMinChn3; //min value of channel 3
					uint16_t dtcPixelVarReciChn0; // sfr_dtc_pixel_variance_reci_ch0
					uint16_t dtcPixelVarReciChn1; // sfr_dtc_pixel_variance_reci_ch1
					uint16_t dtcPixelVarReciChn2; // sfr_dtc_pixel_variance_reci_ch2
					uint16_t dtcPixelVarReciChn3; // sfr_dtc_pixel_variance_reci_ch3
					int8_t reserve1[16]; //32B assign, for ub copy
					}kAippDynamicBatchPara;
					
					/**
					* @ingroup dnn
					* @brief struct define of dynamic aipp parameter. lite:64+96*batchNum byte ; tiny: 64+64*batchNum byte
					*/
					typedef struct tagAippDynamicPara
					{
					uint8_t inputFormat; //input format: YUV420SP_U8, XRGB8888_U8, RGB888_U8
					int8_t cscSwitch; //csc switch
					int8_t rbuvSwapSwitch; //rb/ub swap switch
					int8_t axSwapSwitch; //RGBA->ARGB, YUVA->AYUV swap switch
					int8_t batchNum; //batch parameter number
					int8_t reserve1[3];
					int32_t srcImageSizeW; //source image width
					int32_t srcImageSizeH; //source image height
					int16_t cscMatrixR0C0; // csc_matrix_r0_c0
					int16_t cscMatrixR0C1; // csc_matrix_r0_c1
					int16_t cscMatrixR0C2; // csc_matrix_r0_c2
					int16_t cscMatrixR1C0; // csc_matrix_r1_c0
					int16_t cscMatrixR1C1; // csc_matrix_r1_c1
					int16_t cscMatrixR1C2; // csc_matrix_r1_c2
					int16_t cscMatrixR2C0; // csc_matrix_r2_c0
					int16_t cscMatrixR2C1; // csc_matrix_r2_c1
					int16_t cscMatrixR2C2; // csc_matrix_r2_c2
					int16_t reserve2[3];
					uint8_t cscOutputBiasR0; //output Bias for RGB to YUV, element of row 0, unsigned number
					uint8_t cscOutputBiasR1; //output Bias	

Operator	Classify	Name	TypeRange	Required	Doc	Restrictions

					for RGB to YUV, element of row 1, unsigned number
					uint8_t cscOutputBiasR2; //output Bias for RGB to YUV, element of row 2, unsigned number
					uint8_t cscInputBiasR0; //input Bias for YUV to RGB, element of row 0, unsigned number
					uint8_t cscInputBiasR1; //input Bias for YUV to RGB, element of row 1, unsigned number
					uint8_t cscInputBiasR2; //input Bias for YUV to RGB, element of row 2, unsigned number
					uint8_t reserve3[2];
					int8_t reserve4[16]; //32B assign, for ub copy
					
					kAippDynamicBatchPara aippBatchPara; // allow transfer several batch para.
					} kAippDynamicPara;	

	OUTPUT	features	uint8,float16	TRUE	Output tensor	-
	ATTR	aipp_config_path	-	TRUE	Path of the AIPP configuration file (default = './aipp.cfg')	-
ROIAlign	INPUT	features	float16,float32	TRUE	Input tensor	-
	INPUT	rois	float16,float32	TRUE	ROI information with shape [N, 5-tuple] or [N, 8-tuple]. The 5-tuple is formatted (batchID, x1, y1, x2, y2). For the 8-tuple, only the first 5 tuples are valid.	-
	INPUT	rois_n	int32	FALSE	An optional tensor formatted [batch, 8-tuple]. The actual number of ROIs output per batch.	-

Operator	Classify	Name	TypeRange	Required	Doc	Restrictions
	OUTPUT	output	float16,float32	TRUE	-	-
	ATTR	spatial_scale	float	TRUE	Multiplicative spatial scale factor to translate ROI coordinates from their input scale to the scale used when pooling	-
	ATTR	pooled_h	int	TRUE	Height of output **y**	-
	ATTR	pooled_w	int	TRUE	Width of output **y**	-
	ATTR	sampling_ratio	int	FALSE	Used to determine the divided bins	2
SSDDetectionOutput	INPUT	bbox_delta	Float16,float32	TRUE	Box offset with shape [batch, N, 4-tuple]	float16
	INPUT	score	Float16,float32	TRUE	Confidence score	Float16
	INPUT	anchors	Float16,float32	TRUE	Anchor information	Float16

Operator	Classify	Name	TypeRange	Required	Doc	Restrictions
	OUTPUT	out_boxnum	int32	TRUE	Number of output boxes	-
	OUTPUT	y	Float16,float32	TRUE	Output box information with shape [batch, keep_top_k, 8-tuple], where the 8-tuple is formatted (batchID, label (classID), score (category probability), xmin, ymin, xmax, yman, null).	float16
	ATTR	num_classes	int	TRUE	Number of classes to be predicted, greater than 1	Up to 1024
	ATTR	share_location	bool	FALSE	If **True**, classifies all classes as a category for position prediction. (default = True)	-
	ATTR	background_label_id	int	FALSE	Background label ID. Must be 0.	0 only
	ATTR	iou_threshold	float	FALSE	Intersection over Union threshold (default = 0.3). The value range is (0, 1].	(0,1]
	ATTR	top_k	int	FALSE	Number of BBoxes before the NMS operation (default = 200). The value range is (0, 1024].	(0,1024]
	ATTR	eta	float	FALSE	NMS parameter (default = 1). Must be **1**.	1 only
	ATTR	variance_encoded_in_target	bool	FALSE	If **False**, the result without using variance is brought into the position prediction.	-
	ATTR	code_type	int	FALSE	BBox coding type (default = 1). **1**: corner; **2**: center_size; **3**: corner_size	2 only

Operator	Classify	Name	TypeRange	Required	Doc	Restrictions
	ATTR	keep_top_k	int	FALSE	Number of BBoxes after the NMS operation (default = 200). The value range is (0, 1024]. If –1, keeps all BBoxes after the NMS operation.	(0,1024]
	ATTR	confidence_threshold	float	FALSE	Confidence threshold The value range is [0, 1].	[0,1]
PriorBox	INPUT	x	float16, float32	TRUE	Input tensor. Only the height and width dimensions in the shape are used.	float16
	INPUT	img	float16, float32	FALSE	Image information. Only the height and width dimensions in the shape are used. **img_size** nor **img_h/img_w** in attr does not need to be specified.	float16
	OUTPUT	y	float16, float32	TRUE	Output tensor	float16
	ATTR	min_size	ListFloat	TRUE	Minimum box size (in pixels)	-
	ATTR	max_size	ListFloat	TRUE	Maximum box size (in pixels)	-

Operator	Classify	Name	TypeRange	Required	Doc	Restrictions
	ATTR	aspect_ratio	ListFloat	TRUE	aspect ratios: ratio of the width and height of the generated box to the input box. Duplicate values are deleted.	A list of floats, greater than 0
	ATTR	flip	bool	FALSE	Whether to flip each **aspect_ratio** (default = True)	-
	ATTR	clip	bool	FALSE	Whether to clip the generated boxes (default = False)	-
	ATTR	variance	ListFloat	FALSE	Variance of the generated boxes	-
	ATTR	step_h	float	FALSE	Stride height	-
	ATTR	step_w	float	FALSE	Stride width	-
	ATTR	offset	float	FALSE	Offset of each small grid relative to the upper left corner	-
	ATTR	img_h	int	FALSE	Image height	-
	ATTR	img_w	int	FALSE	Image width	-
FSRDetectionOutput	INPUT	rois	Float16,float32	TRUE	A tensor for the proposal output information, with shape [batch, 5-tuple, max_rois_num], where **max_rois_num** is the maximum ROI count per batch rounded up to the nearest multiple of 16, the 5-tuple is (batchID, x1, y1, x2, y2), and output per batch is based on **actual_rois_num**	float16

Operator	Classify	Name	TypeRange	Required	Doc	Restrictions
	INPUT	bbox_delta	Float16,float32	TRUE	A tensor for **BBox_delta**, with shape [batch, num_classes * 4-tuple, height, width, Ci0], where, the 4-tuple is formatted (delta_x, delta_y, delta_w, delta_h), and **batch** indicates the number of valid ROIs.	Float16
	INPUT	score	Float16,float32	TRUE	Probability of each class, with shape [batch, num_classes, height, width, Ci0], where, class 0 indicates the background class and **batch** indicates the number of valid ROIs.	float16
	INPUT	im_info	Float16,float32	FALSE	Height and width of the input image.	-
	INPUT	actual_rois_num	int	FALSE	Actual number of ROIs output per batch, with shape [batch_rois, 8-tuple]. The first one is valid.	-
	OUTPUT	actual_bbox_num	int32	TRUE	Actual number of output BBoxes, with shape [batch, num_classes], fo type int32	-
	OUTPUT	box	Float16,float32	TRUE	Output proposal information, with shape [batch, numBoxes, 8-tuple], where, the 8-tuple is [x1, y1, x2, y2, score, label, batchID, NULL], and **numBoxes** is up to 1024.	float16
	ATTR	batch_rois	int	FALSE	Number of batches (default = 1). (This parameter is reserved and the value is obtained from the shape.)	-
	ATTR	num_classes	int	True	Number of classes, including the background	-

Operator	Classify	Name	TypeRange	Required	Doc	Restrictions
	ATTR	score_threshold	float	True	Score threshold	-
	ATTR	iou_threshold	float	True	Intersection over Union threshold	-
Reshape	INPUT	x	float16,float32,int8,uint8,int16,uint16,int32,uint32,int64,uint64	TRUE	Input tensor	-

Operator	Classify	Name	TypeRange	Required	Doc	Restrictions
	INPUT	shape	int32,int64	FALSE	This parameter defines the output dimensions. The value **0** indicates that the corresponding dimension from the bottom layer is used (unchanged). The value **–1** indicates that the size of the output dimensions is inferred from the count of the bottom BLOB and the remaining dimensions.	-
	OUTPUT	y	float16,float32,int8,uint8,int16,uint16,int32,uint32,int64,uint64	TRUE	Output tensor	-
	ATTR	axis	int	TRUE	Start axis of the bottom blob's shape to be replaced by the reshape (default = 0)	[-rank(x), rank(x))

Operator	Classify	Name	TypeRange	Required	Doc	Restrictions
	ATTR	num_axes	int	FALSE	Number of dimensions of the output shape. • If **num_axes ==–1,** shape.size() = shape.size() + axis; • otherwise, shape.size() = shape.size() + tensor.size() – num_axes	Value range: [–1, rank(x)) Defaults to –1, indicating that the entire bottom blob shape is included in the reshape.
Sigmoid	INPUT	x	float16, float32	TRUE	Sigmoid activation function: y = (1 + exp(-x))^{-1}	Float16
	OUTPUT	y	float16, float32	TRUE		float16

Operator	Classify	Name	TypeRange	Required	Doc	Restrictions
Concat	INPUT	x	float16,fp32,int8,int16,int32,int64,uint8,uint16,uint32,uint64	TRUE	Concatenates its multiple input blobs to one single output blob. **x** is a variable-length list of tensors.	bottom num <= 16

Operator	Classify	Name	TypeRange	Required	Doc	Restrictions
	OUTPUT	y	float16,fp32,int8,int16,int32,int64,uint8,uint16,uint32,uint64	TRUE		-
	ATTR	axis	int	FALSE	The axis along which to concatenate (default = 1). May be negative to index from the end.	Within the rank range.
	ATTR	concat_dim	int	FALSE	Alias for **axis** (default = 1)	Does not support negative indexing.
Upsample(darknet)	INPUT	x	Float16,float32	TRUE	y[N,C,i,j] = scale * x[N,C,i/stride_h,j/stride_w], where, 0<=i<=stride_h * x.h – 1, 0 <= j <= stride_w * x.w – 1	float16

Operator	Classify	Name	TypeRange	Required	Doc	Restrictions
	OUTPUT	y	Float16,float32	TRUE		float16
	ATTR	stride	int	FALSE	If **stride** and **stride_h/stride_w** are both specified, **stride** applies.	-
	ATTR	stride_h	int	FALSE	Stride height (default = 2)	> 1
	ATTR	stride_w	int	FALSE	Stride width (default = 2)	> 1
	ATTR	scale	float	FALSE	Scale factor (default = 1)	-
Yolo	INPUT	x	float16,float32	TRUE	[batch, boxes x (coords + 1 + classes), height, width]	float16
	OUTPUT	coord_data	float16,float32	TRUE	Predicted coordinates, with shape [batch, boxes * coords, ceil(height*width*2+32, 32)/2]	float16
	OUTPUT	obj_data	float16,float32	TRUE	[batch, ceil(boxes * height * width *2+32, 32)/2], where each anchor has only one obj value	float16

Operator	Classify	Name	TypeRange	Required	Doc	Restrictions
	OUTPUT	classes_data	float16,float32	TRUE	[batch, classes, ceil(boxes * height * width *2+32, 32)/2]. The score of each anchor is rounded up to the nearest multiple of 16 for calculation convenience.	float16
	ATTR	boxes	int	FALSE	Number of anchor boxes in each grid (default = 3)	-
	ATTR	coords	int	FALSE	Number of anchor boxes in each grid (default = 3)	4
	ATTR	classes	int	FALSE	Number of classes (default = 80), up to 1024	Up to 1024
	ATTR	yolo_version	string	FALSE	Yolo version: V2 or V3, corresponding to detection_layer and Yolo_layer (default = V3)	V2 or V3
	ATTR	softmax	bool	FALSE	Softmax enable (default = False)	True or False
	ATTR	softmaxtree	bool	FALSE	Softmaxtree enable (default = False). This attribute is reserved (see the definition in Darknet).	False
	ATTR	background	bool	FALSE	Whether to work with softmax to determine the operation type for **b** and **classes** (default = False)	True or False
YoloV2DetectionOutput	INPUT	coord_data	float16,float32	TRUE	Predicted coordinates, with shape [batch, coords *boxes, height, width]	float16, height x width x Dtype_Size >= 32 bytes

Operator	Classify	Name	TypeRange	Required	Doc	Restrictions
	INPUT	obj_prob	float16,float32	TRUE	[batch, boxes, height, width], where each anchor has only one obj value	float16
	INPUT	classes_prob	float16,float32	TRUE	[batch, boxes * classes, height, width]. The score of each anchor is rounded up to the nearest multiple of 16 for AI Core calculation convenience.	float16, height x width x Dtype_Size >= 32 bytes
	INPUT	img_info	float16,float32	TRUE	Original image information, with shape [batch, 4-tuple], where the 4-tuple is formatted [netH, netW, scaleH, scaleW]. **netH** and **netW** are H and W of the network model input, and **scaleH** and **scaleW** are H and W of the original image.	float16
	OUTPUT	box_out	float16,float32	TRUE	[batch, 6-tuple, post_nms_topn], where the 6-tuple is [x1, y1, x2, y2, score, label(class)]. **box_out_num** applies.	float16
	OUTPUT	box_out_num	int32	TRUE	Maximum number of valid boxes per batch, up to 1024. Has shape [batch, 8-tuple, 1, 1], of type int32. Only the first tuple of the 8-tuple is valid.	-
	ATTR	biases	ListFloat	TRUE	[boxes, 2-tuple], where the 2-tuple is x (w) and y (h).	-
	ATTR	boxes	int	FALSE	Number of anchor boxes in each grid (default = 5)	-

Operator	Classify	Name	TypeRange	Required	Doc	Restrictions
	ATTR	coords	int	FALSE	Number of coordinates. Fixed at 4, indicating x, y, h, and w.	Fixed at 4
	ATTR	classes	int	FALSE	Number of classes (default = 20)	Up to 1024
	ATTR	relative	bool	FALSE	Whether the values in correct_region_boxes are relative values (default = True)	True or False
	ATTR	obj_threshold	float	FALSE	Threshold of the probability that an object exists, corresponding to the threshold in **clsProb** (default = 0.5)	[0,1]
	ATTR	pre_nms_topn	int	FALSE	Number of BBoxes before NMS (default = 512), corresponding to multiClassNMS. HiSilicon SoC supports up to 512. Mini and Cloud supports up to 1024.	Up to 1024
	ATTR	post_nms_topn	int	FALSE	Number of BBoxes after NMS, up to 1024 (default = 512).	Up to 1024
	ATTR	score_threshold	float	FALSE	Score threshold of each class (default = 0.5).	[0,1]
	ATTR	iou_threshold	float	FALSE	Intersection over Union threshold (default = 0.45)	[0,1]
YoloV3DetectionOutput	INPUT	coord_data_low	float16,float32	TRUE	Predicted coordinates, with shape [batch, coords *boxes, ceil(height_low*width_low,16) + 16]	float16, height_low x width_low x Dtype_Size >= 32 bytes

Operator	Classify	Name	TypeRange	Required	Doc	Restrictions
	INPUT	coord_data_mid	float16,float32	TRUE	Predicted coordinates, with shape [batch, coords * boxes, ceil(height_mid * width_mid,16) + 16]	float16, height_mid x width_mid x Dtype_Size >= 32 bytes
	INPUT	coord_data_high	float16,float32	TRUE	Predicted coordinates, with shape [batch, coords * boxes, ceil(height_high * width_high,16) + 16]	float16, height_high x width_high x Dtype_Size >= 32 bytes
	INPUT	obj_prob_low	float16,float32	TRUE	[batch, ceil(boxes * height_low * width_low,16) + 16], where each anchor has only one obj value.	float16
	INPUT	obj_prob_mid	float16,float32	TRUE	[batch, ceil(boxes * height_mid * width_mid,16) + 16], where each anchor has only one obj value.	float16
	INPUT	obj_prob_high	float16,float32	TRUE	[batch, ceil(boxes * height_high * width_high, 16) + 16], where each anchor has only one obj value.	float16
	INPUT	classes_prob_low	float16,float32	TRUE	[batch, classes, ceil(boxes * height_low * width_low, 16) + 16]. The score of each anchor is rounded up to the nearest multiple of 16 for calculation convenience.	float16, height_low x width_low x Dtype_Size >= 32 bytes

Operator	Classify	Name	TypeRange	Required	Doc	Restrictions
	INPUT	classes_prob_mid	float16,float32	TRUE	[batch, classes, ceil(boxes * height_mid * width_mid,16) + 16] The score of each anchor is rounded up to the nearest multiple of 16 for calculation convenience.	float16, height_low x width_low x Dtype_Size >= 32 bytes
	INPUT	classes_prob_high	float16,float32	TRUE	[batch, classes,ceil(boxes * height_high * width_high, 16) + 16]. The score of each anchor is rounded up to the nearest multiple of 16 for calculation convenience.	float16, height_low x width_low x Dtype_Size >= 32 bytes
	INPUT	img_info	float16,float32	TRUE	Original image information, with shape [batch, 4-tuple], where the 4-tuple is formatted [netH, netW, scaleH, scaleW]. **netH** and **netW** are H and W of the network model input, and **scaleH** and **scaleW** are H and W of the original image.	float16
	OUTPUT	box_out	float16,float32	TRUE	[batch, 6-tuple, post_nms_topn], where the 6-tuple is [x1, y1, x2, y2, score, label(class)]. **box_out_num** applies.	float16
	OUTPUT	box_out_num	int32	TRUE	Maximum number of valid boxes per batch, up to 1024. Has shape [batch, 8-tuple, 1, 1], of type int32. Only the first tuple of the 8-tuple is valid.	float16
	ATTR	biases_low	ListFloat	TRUE	[boxes * 2-tuple], where the 2-tuple is x (w) and y (h), corresponding to box 1	-
	ATTR	biases_mid	ListFloat	TRUE	[boxes * 2-tuple], where the 2-tuple is x (w) and y (h), corresponding to box 2	-

Operator	Classify	Name	TypeRange	Required	Doc	Restrictions
	ATTR	biases_high	ListFloat	TRUE	[boxes * 2-tuple], where the 2-tuple is x (w) and y (h), corresponding to box 3	-
	ATTR	boxes	int	FALSE	Number of anchor boxes in each grid (default = 3)	-
	ATTR	coords	int	FALSE	Number of coordinates. Fixed at 4, indicating x, y, h, and w.	Fixed at 4
	ATTR	classes	int	FALSE	Number of classes (default = 80)	Up to 1024
	ATTR	relative	bool	FALSE	Whether the values in correct_region_boxes are relative values (default = True)	True or False
	ATTR	obj_threshold	float	FALSE	Threshold of the probability that an object exists, corresponding to the threshold in **clsProb** (default = 0.5)	[0,1]
	ATTR	pre_nms_topn	int	FALSE	Number of BBoxes before NMS, corresponding to multiClassNMS, up to 1024	Up to 1024
	ATTR	post_nms_topn	int	FALSE	Number of BBoxes after NMS, up to 1024 (default = 1024)	Up to 1024
	ATTR	score_threshold	float	FALSE	Score threshold of each class (default = 0.5).	[0,1]
	ATTR	iou_threshold	float	FALSE	Intersection over Union threshold (default = 0.45)	[0,1]
Log	INPUT	x	float16,float32	TRUE	Computes natural logarithm of **x**. Description: if base>0: y = log_base(shift + scale * x) if base == -1: y = log_e(shift + scale * x)	float16

Operator	Classify	Name	TypeRange	Required	Doc	Restrictions
	OUTPUT	y	float16,float32	TRUE	Operator prototype: The TensorFlow and Caffe prototypes are different. Therefore, the prototype needs to be redefined.	float16
	ATTR	base	float	FALSE	(default = 1.0)	base > 0 or = –1
	ATTR	scale	float	FALSE	(default = 1.0)	-
	ATTR	shift	float	FALSE	(default = 0.0)	-
LRN	INPUT	x	float16,float32	TRUE	Performs a kind of lateral inhibition by normalizing over local input regions. power(k + (alpha/n) * sum(square(x)),beta)	float16
	OUTPUT	y	float16,float32	TRUE		float16
	ATTR	local_size	uint	FALSE	Must be an odd value up to 255 (default = 5)	Odd value up to 255
	ATTR	alpha	float	FALSE	(default = 1.0)	-
	ATTR	beta	float	FALSE	(default = 0.75)	-

Operator	Classify	Name	TypeRange	Required	Doc	Restrictions
	ATTR	norm_region	int	FALSE	0: ACROSS_CHANNELS 1: WITHIN_CHANNEL Only ACROSS_CHANNELS is supported (default = 0).	0 only
	ATTR	k	float	FALSE	(default = 1.0)	-
MVN	INPUT	x	float16, float32	TRUE	Input tensor	float16
	OUTPUT	y	float16, float32	TRUE	y=(x-mean(x))/std(x)	float16
	ATTR	normalize_variance	bool	FALSE	If **False**, normalizes mean only (default = True).	True or False
	ATTR	across_channels	bool	FALSE	If set to **True**, CHW is considered as a vector. Defaults to **False**.	True or False
	ATTR	eps	float	FALSE	Epsilon for not dividing by zero (default = 1e – 9)	Configurable
Reduction	INPUT	x	float16, float32	TRUE	-	float16

Operator	Classify	Name	TypeRange	Required	Doc	Restrictions
	OUTPUT	y	float16, float32	TRUE	-	float16
	ATTR	operation	int	FALSE	(default = 1) 1: SUM 2: ASUM 3: SUMSQ 4: MEAN	Value range: 1–4
	ATTR	axis	int	FALSE	(default = 0) The first axis to reduce	[-rank(x), rank(x))
	ATTR	coeff	float	FALSE	Coefficient for output (default = 1.0)	Configurable

Operator	Classify	Name	TypeRange	Required	Doc	Restrictions
Split	INPUT	x	float16,float32,int8,uint8,int16,uint16,int32,uint32,int64,uint64	TRUE	Splits an input blob to multiple output blobs	-

Operator	Classify	Name	TypeRange	Required	Doc	Restrictions
	OUTPUT	y	float16,float32,int8,uint8,int16,uint16,int32,uint32,int64,uint64	TRUE		-
SPP	INPUT	x	float16,float32	TRUE	Pools the features and generates fixed-length outputs (1 x 1, 2 x 2, 4 x 4, and so on).	float16 only; H < 510, W < 510
	OUTPUT	y	float16,float32	TRUE		float16

Operator	Classify	Name	TypeRange	Required	Doc	Restrictions
	ATTR	pyramid_height	int	TRUE	Pyramid height. The value range is [1, 7). The calculated **kernel_size** and **Pad_size** must meet the Caffe pooling restriction: Pad_size < kernel_size	[1,7)
	ATTR	pool_method	int	FALSE	0: max pooling 1: avg pooling (default = MAX)	0,1
Threshold	INPUT	x	float16, float32	TRUE	The input is compared with a threshold value. The value **1** is output when the input is above the threshold. The value **0** is output when the input is below or equal to the threshold.	float16
	OUTPUT	y	float16, float32	TRUE		float16
	ATTR	threshold	float	FALSE	Result: **0** (default) or **1**	-

Operator	Classify	Name	TypeRange	Required	Doc	Restrictions
Tile	INPUT	x	float16,float32,int8,uint8,int16,uint16,int32,uint32,int64,uint64	TRUE	Input tensor	-

Operator	Classify	Name	TypeRange	Required	Doc	Restrictions
	OUTPUT	y	float16,float32,int8,uint8,int16,uint16,int32,uint32,int64,uint64	TRUE	Output tensor	-
	ATTR	axis	int	FALSE	The axis (default = 1)	-
	ATTR	tiles	int	TRUE	Tiling multiple. For example, if **x** has shape [n,c,h,w], axis=2, and tiles=3, **y** has shape [n, c, 3 * h, w].	-

Operator	Classify	Name	TypeRange	Required	Doc	Restrictions
ShuffleChannel	INPUT	x	float16,floaat32,int8,uint8,int16,uint16,int32,uint32,int64,uint64	TRUE	-	-

Operator	Classify	Name	TypeRange	Required	Doc	Restrictions
	OUTPUT	y	float16,float32,int8,uint8,int16,uint16,int32,uint32,int64,uint64	TRUE	-	-
	ATTR	group	uint	FALSE	Greater than 0 (default = 1). The channel size must be divisible by **group**.	Greater than 0. The channel size must be divisible by **group**.
BatchedMatMul	INPUT	x1	fp16	TRUE	A tensor of 2–8 dimensions. If **adj_x1 = False**, **x1** has shape [batch, ..., M, K]. If **adj_x1 = True**, **x1** has shape [batch, ..., K, M]. **batch, ...** must be the same as those in **x2**.	-

Operator	Classify	Name	TypeRange	Required	Doc	Restrictions
	INPUT	x2	fp16	TRUE	A tensor of 2–8 dimensions. If **adj_x2 = False**, **x2** has shape [batch, ..., M, K]. If **adj_x2 = True**, **x2** has shape [batch, ..., K, M]. **batch, ...** must be the same as those in **x1**.	-
	OUTPUT	y	fp16	TRUE	Output y (x1 * x2), with shape [batch, ..., M, N]	-
	ATTR	adj_x1	bool	FALSE	Indicates whether to transpose **x1**. Defaults to **False**.	-
	ATTR	adj_x1	bool	FALSE	Indicates whether to transpose **x2**. Defaults to **False**.	-
LSTM	INPUT	x	float16	TRUE	Time change input x, with shape [T, N, input_size]	T <= 256
	INPUT	cont	float16,float32	TRUE	Sequence continuity flag, with shape [T, N]	T <= 256
	INPUT	w_x	float16	TRUE	**x**'s weight, with shape [input_size, 4 * num_output]	-
	INPUT	bias	float16,float32	TRUE	Offset, with shape [4 * num_output]	-
	INPUT	w_h	float16	TRUE	H weight, with shape [4 * num_output, num_output]	-

Operator	Classify	Name	TypeRange	Required	Doc	Restrictions
	INPUT	x_static	float16	FALSE	Static data with time unchanged, with shape [N, input_size]. The parameter is reserved.	-
	INPUT	h_0	float16,float32	FALSE	The initial hidden status, with shape [1,N,num_output]. Required when **expose_hidden = True**.	-
	INPUT	c_0	float16,float32	FALSE	The initial cell status, with shape [1,N,num_output]. Required when **expose_hidden = True**.	-
	INPUT	w_x_static	float16	FALSE	**x_static** weight, with shape [4 * num_output, input_size]. The parameter is reserved.	-
	OUTPUT	h	float16,float32	TRUE	Output tensor (T * N * num_output)	-
	OUTPUT	h_t	float16,float32	FALSE	Hidden status of the last time step (1 * N * num_output)	-
	OUTPUT	c_t	float16,float32	FALSE	Cell status of the last time step (1 * N * num_output)	-

Operator	Classify	Name	TypeRange	Required	Doc	Restrictions
	ATTR	num_output	uint	TRUE	Number of output features. Must be greater than 0.	Must be greater than 0.
	ATTR	expose_hidden	bool	FALSE	Indicates whether the initial hidden status and initial cell status exist. Defaults to False.	-
ArgMax	INPUT	x	fp32,fp16	TRUE	Input tensor	-
	OUTPUT	indices	int32	FALSE	Index of the maximum output value	-
	OUTPUT	values	fp32,fp16	FALSE	Output tensor, including the maximum value index or maximum value	-
	ATTR	axis	int	FALSE	Axis to be reduced in the input tensor. If this parameter is not provided, **topk** is calculated per batch.	-

Operator	Classify	Name	TypeRange	Required	Doc	Restrictions

	ATTR	out_max_val	bool	FALSE	Indicates whether to output the maximum value.
					• If True and there are two TOPs, the maximum value and index are displayed.
					layer {
					name: "argmax"
					type: "ArgMax"
					bottom: "data"
					top: "indices"
					top: "values"
					argmax_param {
					out_max_val: True
					top_k: 1
					}
					• If True and there is one TOP, the maximum value is displayed.
					layer {
					name: "argmax"
					type: "ArgMax"
					bottom: "data"
					top: "values"
					argmax_param {
					out_max_val: True
					top_k: 1
					axis: 1
					}
					}
					• If False, the following index is displayed:
					layer {
					name: "argmax"
					type: "ArgMax"
					bottom: "data"
					top: "indices"
					argmax_param {
					out_max_val: False
					top_k: 1
					axis: 1
					}
					}
						-

	ATTR	topk	int	FALSE	Defaults to **1**, indicating the top k in each axis. The value range is [1, x.shape(axis)], corresponding to **top_k** in Caffe.	1 only
RNN	INPUT	x	fp16	TRUE	Time change input, (T * N * ...)	T <= 256
	INPUT	cont	fp16	TRUE	Sequence continuity flag, (T * N)	T <= 256

Operator	Classify	Name	TypeRange	Required	Doc	Restrictions
	OPTIONAL_INPUT	x_static	fp16	FALSE	Static data with time unchanged (N * ...)	-
	OPTIONAL_INPUT	h_0	fp32,fp16	FALSE	Initial hidden state (1 * N * num_output)	-
	INPUT	w_xh	fp16	TRUE	xH weight, with shape [input_size, num_output]	-
	INPUT	bias_h	fp32,fp16	TRUE	Offset, with shape [num_output,].	-
	INPUT	w_sh	fp16	FALSE	SH weight, with shape [input_size, num_output]	-
	INPUT	w_hh	fp16	TRUE	HH weight, with shape [num_output, num_output]	-
	INPUT	w_ho	fp16	TRUE	Ho weight, with shape [num_output, num_output].	-
	INPUT	bias_o	fp32,fp16	TRUE	Offset, with shape [num_output,].	-
	OUTPUT	o	fp32,fp16	TRUE	Output tensor (T * N * num_output)	-
	OUTPUT	h_t	fp32,fp16	TRUE	Hidden status of the last time step (1 * N * num_output)	-

Operator	Classify	Name	TypeRange	Required	Doc	Restrictions
	ATTR	expose_hidden	bool	FALSE	Indicates whether to expose the hidden status. Defaults to false.	-
	ATTR	num_output	int	FALSE	Number of output features. If the value is 0, the value is obtained from the shape.	-

7.1.1.5 Sample Reference

This section provides instructions for modifying frequently-used networks.

7.1.1.5.1 Modifying Faster R-CNN Prototxt

NOTE

The provided code samples must be modified before using them on the network. You need to modify the arguments, for example, the **bottom** and **top** arguments, to match the network in use.

The following uses the Faster R-CNN ResNet-34 model as an example.

1. Modify the Proposal operator.

   According to **7.1.1.4 Caffe Operator Specifications**, the operator has three inputs and two outputs. Modify the original prototxt accordingly. Change **type** to that defined in the **caffe.proto** file, and add the actual_rois_num output node. Add attribute description by referring to the attribute definition in the **caffe.proto** file. **Figure 7-4** shows the modification. The original prototxt is on the left, and the prototxt adapted to the Ascend AI Processor is on the right.

   Figure 7-4 Prototxt file before and after modification (1)

   A code example is as follows:

```
layer {
name: "faster_rcnn_proposal"
type: "Proposal"                //Operator type
```



```
bottom: "rpn_cls_prob_reshape"
bottom: "rpn_bbox_pred"
bottom: "im_info"
top: "rois"
top: "actual_rois_num"       // Added operator output
  proposal_param {

  feat_stride: 16
  base_size: 16
  min_size: 16
  pre_nms_topn: 3000
  post_nms_topn: 304
  iou_threshold: 0.7
  output_actual_rois_num: true
  }
}
```

For details about parameter descriptions, see **7.1.1.4 Caffe Operator Specifications**.

2. Add a FSRDetectionOutput operator to the last layer to output the final detection result.

   On the Faster R-CNN network, add a post-processing layer FSRDetectionOutput at the end of the original .prototxt file by referring to **7.1.1.2 Extended Operator List**. The FSRDetectionOutput operator has five inputs and two outputs as described in **7.1.1.4 Caffe Operator Specifications**. Define the data types and the attributes of the operator accordingly.

   A code example is as follows:

```
layer {
  name: "FSRDetectionOutput_1"
  type: "FSRDetectionOutput"

  bottom: "rois"
  bottom: "bbox_pred"
  bottom: "cls_prob"
  bottom: "im_info"
  bottom: "actual_rois_num"
  top: "actual_bbox_num1"
  top: "box1"

  fsrdetectionoutput_param {

    num_classes:3
    score_threshold:0.0
    iou_threshold:0.7
    batch_rois:1
  }
}
```

   For details about the parameter description, see **7.1.1.4 Caffe Operator Specifications**.

7.1.1.5.2 Modifying YOLOv3 Prototxt

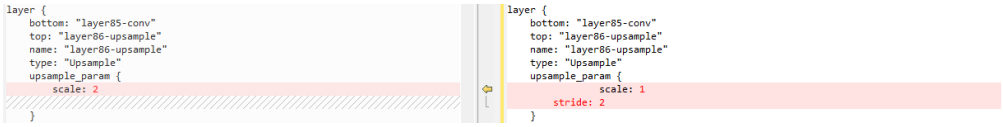
> **NOTE**
>
> The provided code samples must be modified before using them on the network. You need to modify the arguments, for example, the **bottom** and **top** arguments, to match the network in use.

1. Modify the **upsample_param** attribute of the Upsample operator.

   Change **scale:2** in the .prototxt file of the original operator to **scale:1 stride:2** by referring to **7.1.1.4 Caffe Operator Specifications**.

Figure 7-5 shows the comparison. The left part is the native operator prototxt, and the right part is the prototxt adapted to the Ascend AI Processor.

Figure 7-5 Prototxt file before and after modification (4)



For details about parameter descriptions, see **7.1.1.4 Caffe Operator Specifications**.

2. Add three Yolo operators.

   The Yolo and DetectionOutput operators complete the post-processing logic of the feature detection network. According to the original operator .prototxt file, three Yolo operators should be added before adding the YoloV3DetectionOutput operator.

   According to **7.1.1.4 Caffe Operator Specifications**, a Yolo operator has one input and three outputs. The code examples of the Yolo operators are provided.

   - Code example of operator 1

     ```
     layer {
         bottom: "layer82-conv"
         top: "yolo1_coords"
         top: "yolo1_obj"
         top: "yolo1_classes"
         name: "yolo1"
         type: "Yolo"
         yolo_param {
             boxes: 3
             coords: 4
             classes: 80
             yolo_version: "V3"
             softmax: true
             background: false
         }
     }
     ```

   - Code example of operator 2

     ```
     layer {
         bottom: "layer94-conv"
         top: "yolo2_coords"
         top: "yolo2_obj"
         top: "yolo2_classes"
         name: "yolo2"
         type: "Yolo"
         yolo_param {
             boxes: 3
             coords: 4
             classes: 80
             yolo_version: "V3"
             softmax: true
             background: false
         }
     }
     ```

   - Code example of operator 3

     ```
     layer {
         bottom: "layer106-conv"
         top: "yolo3_coords"
         top: "yolo3_obj"
     ```



```
    top: "yolo3_classes"
    name: "yolo3"
    type: "Yolo"
    yolo_param {
        boxes: 3
        coords: 4
        classes: 80
        yolo_version: "V3"
        softmax: true
        background: false
    }
}
```

For details about parameter descriptions, see **7.1.1.4 Caffe Operator Specifications**.

3. Add a YoloV3DetectionOutput operator to the last layer.

   On the YOLOv3 network, add a post-processing layer YoloV3DetectionOutput to the end of the original .prototxt file by referring to **7.1.1.2 Extended Operator List**. The YoloV3DetectionOutput operator has 10 inputs and two outputs as described in **7.1.1.4 Caffe Operator Specifications**.

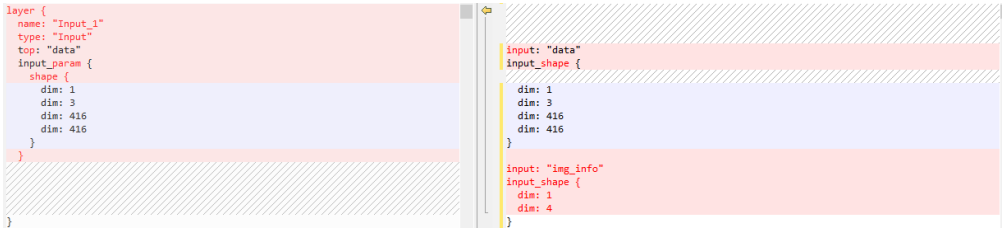
```
layer {
    name: "detection_out3"
    type: "YoloV3DetectionOutput"
    bottom: "yolo1_coords"
    bottom: "yolo2_coords"
    bottom: "yolo3_coords"
    bottom: "yolo1_obj"
    bottom: "yolo2_obj"
    bottom: "yolo3_obj"
    bottom: "yolo1_classes"
    bottom: "yolo2_classes"
    bottom: "yolo3_classes"
    bottom: "img_info"
    top: "box_out"
    top: "box_out_num"
    yolov3_detection_output_param {
                boxes: 3
                classes: 80
                relative: true
                obj_threshold: 0.5
                score_threshold: 0.5
                iou_threshold: 0.45
                pre_nms_topn: 512
                post_nms_topn: 1024
                biases_high: 10
                biases_high: 13
                biases_high: 16
                biases_high: 30
                biases_high: 33
                biases_high: 23
                biases_mid: 30
                biases_mid: 61
                biases_mid: 62
                biases_mid: 45
                biases_mid: 59
                biases_mid: 119
                biases_low: 116
                biases_low: 90
                biases_low: 156
                biases_low: 198
                biases_low: 373
                biases_low: 326
    }
}
```

For details about parameter descriptions, see **7.1.1.4 Caffe Operator Specifications**.

4. Added the inputs.

   The YoloV3DetectionOutput operator has the **img_info** input. Add **img_info** to model inputs. **Figure 7-6** shows the comparison. The left part is the native operator prototxt, and the right part is the prototxt adapted to the Ascend AI Processor.

   Figure 7-6 Prototxt file before and after modification (5)



   The following is a code example. **img_info** has shape [batch, 4-tuple], where the 4-tuple is formatted [netH, netW, scaleH, scaleW]. **netH** and **netW** are H and W of the network model input, and **scaleH** and **scaleW** are H and W of the original image.

```
input: "img_info"
input_shape {
  dim: 1
  dim: 4
}
```

7.1.1.5.3 Modifying YOLOv2 Prototxt

NOTE

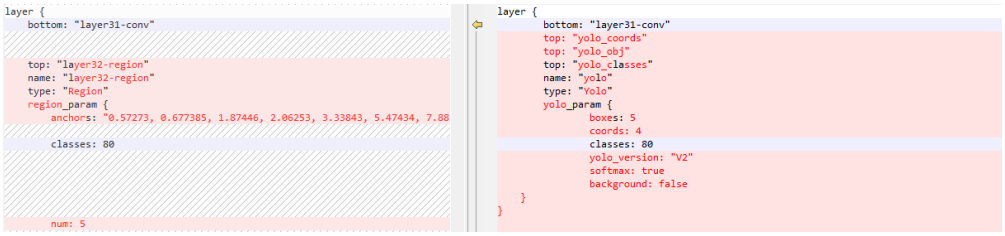
The provided code samples must be modified before using them on the network. You need to modify the arguments, for example, the **bottom** and **top** arguments, to match the network in use.

1. Modify the Region operator.

   The Yolo and DetectionOutput operators complete the post-processing logic of the feature detection network. Before adding the YoloV2DetectionOutput operator, replace the Region operator with a Yolo operator.

   A Yolo operator has one input and three outputs according to **7.1.1.4 Caffe Operator Specifications**. **Figure 7-7** shows the .prototxt file before and after modification for adapting to Ascend AI Processor.

   Figure 7-7 Prototxt file before and after modification (2)



   A code example is as follows:

```
layer {
  bottom: "layer31-conv"
```

```
  top: "yolo_coords"
  top: "yolo_obj"
  top: "yolo_classes"
  name: "yolo"
  type: "Yolo"
  yolo_param {
      boxes: 5
      coords: 4
      classes: 80
      yolo_version: "V2"
      softmax: true
      background: false
  }
}
```

For details about parameter descriptions, see **7.1.1.4 Caffe Operator Specifications**.

2. Add a YoloV2DetectionOutput operator to the last layer.

   On the YOLOv2 network, add a post-processing layer YoloV2DetectionOutput to the end of the original .prototxt file by referring to **7.1.1.2 Extended Operator List**. The YoloV2DetectionOutput operator has four inputs and two outputs as described in **7.1.1.4 Caffe Operator Specifications**.

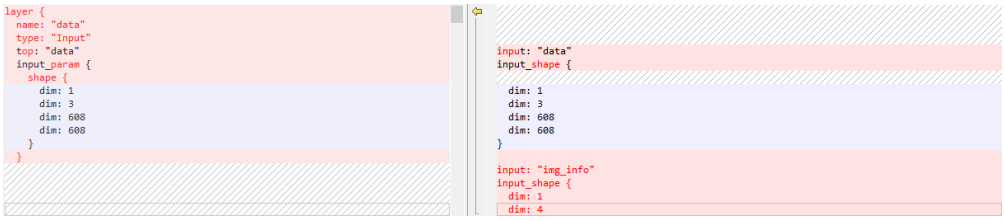
```
layer {
name: "detection_out2"
type: "YoloV2DetectionOutput"
bottom: "yolo_coords"
bottom: "yolo_obj"
bottom: "yolo_classes"
bottom: "img_info"
top: "box_out"
top: "box_out_num"
yolov2_detection_output_param {
  boxes: 5
  classes: 80
  relative: true
  obj_threshold: 0.5
  score_threshold: 0.5
  iou_threshold: 0.45
  pre_nms_topn: 512
  post_nms_topn: 1024
  biases: 0.572730
  biases: 0.677385
  biases: 1.874460
  biases: 2.062530
  biases: 3.338430
  biases: 5.474340
  biases: 7.882820
  biases: 3.527780
  biases: 9.770520
  biases: 9.168280
  }
}
```

   For details about parameter descriptions, see **7.1.1.4 Caffe Operator Specifications**.

3. Added the inputs.

   The YoloV2DetectionOutput operator has the **img_info** input. Add **img_info** to model inputs. **Figure 7-8** shows the comparison. The left part is the native operator prototxt, and the right part is the prototxt adapted to the Ascend AI Processor.

Figure 7-8 Prototxt file before and after modification (1)



The following is a code example. **img_info** has shape [batch, 4-tuple], where the 4-tuple is formatted [netH, netW, scaleH, scaleW]. **netH** and **netW** are H and W of the network model input, and **scaleH** and **scaleW** are H and W of the original image.

```
input: "img_info"
input_shape {
  dim: 1
  dim: 4
}
```

7.1.1.5.4 Modifying SSD Prototxt

NOTE

The provided code samples must be modified before using them on the network. You need to modify the arguments, for example, the **bottom** and **top** arguments, to match the network in use.

On the SSD network, add a post-processing layer SSDDetectionOutput at the end of the original .prototxt file by referring to **7.1.1.2 Extended Operator List**.

Add the declaration of the custom layer to the **LayerParameter** message by referring to the **caffe.proto** file.

```
message LayerParameter {
...
  optional SSDDetectionOutputParameter ssddetectionoutput_param = 232;
...
}
```

For details, see the **caffe.proto** file. The data types and attribute of this operator are defined as follows:

```
message SSDDetectionOutputParameter {
    optional int32 num_classes= 1 [default = 2];
    optional bool share_location = 2 [default = true];
    optional int32 background_label_id = 3 [default = 0];
    optional float iou_threshold = 4 [default = 0.45];
    optional int32 top_k = 5 [default = 400];
    optional float eta = 6 [default = 1.0];
    optional bool variance_encoded_in_target = 7 [default = false];
    optional int32 code_type = 8 [default = 2];
    optional int32 keep_top_k = 9 [default = 200];
    optional float confidence_threshold = 10 [default = 0.01];
}
```

As described in **7.1.1.4 Caffe Operator Specifications**, the SSDDetectionOutput operator has three inputs and two outputs. A code example is provided as follows:

```
layer {
  name: "detection_out"
  type: "SSDDetectionOutput"
  bottom: "bbox_delta"
```



```
  bottom: "score"
  bottom: "anchors"
  top: "out_boxnum"
  top: "y"
  ssddetectionoutput_param {
    num_classes: 2
    share_location: true
    background_label_id: 0
    iou_threshold: 0.45
    top_k: 400
    eta: 1.0
    variance_encoded_in_target: false
    code_type: 2
    keep_top_k: 200
    confidence_threshold: 0.01
  }
}
```

- **bbox_delta** corresponds to **mbox_loc** on the original Caffe network, **score** corresponds to **mbox_conf_flatten** on the original Caffe network, and **anchors** corresponds to **mbox_priorbox** on the original Caffe network. The value of **num_classes** must be the same as that in the original Caffe network.
- In the scenario where the top output has a batch size greater than 1:
  - The output shape of **out_boxnum** is (batchnum, 8). The first element of **batchnum** is the number of actual boxes.
  - The output shape of **y** is (batchnum,len,8), where **len** is the value of **keep_top_k** after 128-byte alignment. For example, if **batch = 2** and **keep_top_k = 200**, the output shape is (2,256,8), the first 256 x 8 data elements is the result of the first batch.

For details about parameter descriptions, see **7.1.1.4 Caffe Operator Specifications**.

7.1.1.5.5 Modifying BatchedMatMul Prototxt

NOTE

The provided code samples must be modified before using them on the network. You need to modify the arguments, for example, the **bottom** and **top** arguments, to match the network in use.

The BatchedMatMul operator multiply the two tensors: **y** = **x1** x **x2**. (The number of **x1** and **x2** dimensions must be greater than 2 and less than or equal to 8.) To use this operator in a network model, you need to modify its .prototxt file by referring to this section and then convert the model.

Add the declaration of the custom layer parameters to the **LayerParameter** message by referring to the **caffe.proto** file.

```
message LayerParameter {
...
  optional BatchMatMulParameter batch_matmul_param = 235;
...
}
```

According to the **caffe.proto** file, the operator type and attributes are defined as follows:

```
message BatchMatMulParameter{
    optional bool adj_x1 = 1 [default = false];
    optional bool adj_x2 = 2 [default = false];
}
```


According to **7.1.1.4 Caffe Operator Specifications**, the BatchedMatMul operator has two inputs and one output. An example of the constructed operator code is as follows:

```
layer {
  name: "batchmatmul"
  type: "BatchedMatMul"
  bottom: "matmul_data_1"
  bottom: "matmul_data_2"
  top: "batchmatmul_1"
  batch_matmul_param {
  adj_x1:false
  adj_x2:true
 }
```

For details about the parameter description, see **7.1.1.4 Caffe Operator Specifications**.

7.2 TensorFlow Network Model

7.2.1 TensorFlow Operator Specifications

Supported TF Operator	Category	Description
Abs	math_ops	Computes the absolute value of a tensor.
AccumulateNV2	math_ops	Returns the element-wise sum of a list of tensors.
Acos	math_ops	Computes acos of x element-wise.
Acosh	math_ops	Computes inverse hyperbolic cosine of x element-wise.
Add	math_ops	Returns x + y element-wise.
AddN	math_ops	Add all input tensors element wise.
AddV2	math_ops	Returns x + y element-wise.
All	math_ops	Computes the "logical and" of elements across dimensions of a tensor.
Any	math_ops	Computes the "logical or" of elements across dimensions of a tensor.
ApproximateEqual	math_ops	Returns the truth value of abs(x-y) < tolerance element-wise.
ArgMax	math_ops	Returns the index with the largest value across dimensions of a tensor.

Supported TF Operator	Category	Description
ArgMin	math_ops	Returns the index with the smallest value across dimensions of a tensor.
Asin	math_ops	Computes asin of x element-wise.
Asinh	math_ops	Computes inverse hyperbolic sine of x element-wise.
Atan	math_ops	Computes atan of x element-wise.
Atan2	math_ops	Computes arctangent of y/x element-wise, respecting signs of the arguments.
Atanh	math_ops	Computes inverse hyperbolic tangent of x element-wise.
AvgPool	nn_ops	Performs average pooling on the input.
Batch	batch_ops	-
BatchMatMul	math_ops	Multiplies slices of two tensors in batches.
BatchToSpace	array_ops	BatchToSpace for 4-D tensors of type T.
BatchToSpaceND	array_ops	BatchToSpace for N-D tensors of type T.
BesselI0e	math_ops	Computes the Bessel i0e function of x element-wise.
BesselI1e	math_ops	Computes the Bessel i1e function of x element-wise.
Betainc	math_ops	Compute the regularized incomplete beta integral \(I_x(a, b) \).
BiasAdd	nn_ops	Adds bias to value.
Bincount	math_ops	Counts the number of occurrences of each value in an integer array.
BitwiseAnd	bitwise_ops	-
BitwiseOr	bitwise_ops	-
BitwiseXor	bitwise_ops	-
BroadcastTo	array_ops	Broadcast an array for a compatible shape.

Supported TF Operator	Category	Description
Bucketize	math_ops	Bucketizes 'input' based on 'boundaries'.
Cast	math_ops	Cast x of type SrcT to y of DstT.
Ceil	math_ops	Returns element-wise smallest integer not less than x.
CheckNumerics	array_ops	Checks a tensor for NaN and Inf values.
Cholesky	linalg_ops	-
CholeskyGrad	linalg_ops	-
ClipByValue	math_ops	Clips tensor values to a specified min and max.
CompareAndBitpack	math_ops	Compare values of input to threshold and pack resulting bits into a uint8.
Concat	array_ops	Concatenates tensors along one dimension.
ConcatV2	array_ops	-
Const	array_ops	-
ControlTrigger	control_flow_ops	Does nothing.
Conv2D	nn_ops	Computes a 2-D convolution given 4-D input and filter tensors.
Conv2DBackpropFilter	nn_ops	Computes the gradients of convolution with respect to the filter.
Conv2DBackpropInput	nn_ops	Computes the gradients of convolution with respect to the input.
Cos	math_ops	Computes cos of x element-wise.
Cosh	math_ops	Computes hyperbolic cosine of x element-wise.
Cumprod	math_ops	Compute the cumulative product of the tensor x along axis.
Cumsum	math_ops	Compute the cumulative sum of the tensor x along axis.

Supported TF Operator	Category	Description
DataFormatDimMap	nn_ops	Returns the dimension index in the destination data format given the one in.
DataFormatVecPermute	nn_ops	Returns the permuted vector/tensor in the destination data format given the.
DepthToSpace	array_ops	DepthToSpace for tensors of type T.
DepthwiseConv2dNative	nn_ops	Computes a 2-D depthwise convolution given 4-D input and filtertensors.
DepthwiseConv2dNativeBackpropFilter	nn_ops	Computes the gradients of depthwise convolution with respect to the filter.
DepthwiseConv2dNativeBackpropInput	nn_ops	Computes the gradients of depthwise convolution with respect to the input.
Dequantize	array_ops	Dequantize the 'input' tensor into a float Tensor.
Diag	array_ops	Returns a diagonal tensor with a given diagonal values.
DiagPart	array_ops	Returns the diagonal part of the tensor.
Div	math_ops	Returns x / y element-wise.
DivNoNan	math_ops	Returns 0 if the denominator is zero.
Elu	nn_ops	Computes exponential linear: exp(features) - 1 if < 0, features otherwise.
Empty	array_ops	Creates a tensor with the given shape.
Enter	control_flow_ops	-
Equal	math_ops	Returns the truth value of (x == y) element-wise.
Erf	math_ops	Computes the Gauss error function of x element-wise.
Erfc	math_ops	Computes the complementary error function of x element-wise.

Supported TF Operator	Category	Description
Exit	control_flow_ops	-
Exp	math_ops	Computes exponential of x element-wise.
ExpandDims	array_ops	Inserts a dimension of 1 into a tensor's shape.
Expm1	math_ops	Computes exponential of x - 1 element-wise.
ExtractImagePatches	array_ops	Extract patches from images and put them in the "depth" output dimension.
FakeQuantWithMinMaxArgs	array_ops	Fake-quantize the 'inputs' tensor, type float to 'outputs' tensor of same type.
FakeQuantWithMinMaxVars	array_ops	Fake-quantize the 'inputs' tensor of type float via global float scalars min
FakeQuantWithMinMaxVarsPerChannel	array_ops	Fake-quantize the 'inputs' tensor of type float and one of the shapes: [d],.
Fill	array_ops	Creates a tensor filled with a scalar value.
Floor	math_ops	Returns element-wise largest integer not greater than x.
FloorDiv	math_ops	Returns x // y element-wise.
FloorMod	math_ops	Returns element-wise remainder of division.
FractionalAvgPool	nn_ops	Performs fractional average pooling on the input.
FractionalAvgPoolGrad	nn_ops	-
FractionalMaxPool	nn_ops	Performs fractional max pooling on the input.
FractionalMaxPoolGrad	nn_ops	-
FusedBatchNorm	nn_ops	Batch normalization.
FusedBatchNormV2	nn_ops	Batch normalization.

Supported TF Operator	Category	Description
Gather	array_ops	Gather slices from params according to indices.
GatherNd	array_ops	Gather slices from params into a Tensor with shape specified by indices.
GatherV2	array_ops	Gather slices from params axis axis according to indices.
Greater	math_ops	Returns the truth value of (x > y) element-wise.
GreaterEqual	math_ops	Returns the truth value of (x >= y) element-wise.
GuaranteeConst	array_ops	Gives a guarantee to the TF runtime that the input tensor is a constant.
HistogramFixedWidth	math_ops	Return histogram of values.
Identity	array_ops	Return a tensor with the same shape and contents as the input tensor or value.
IdentityN	array_ops	Returns a list of tensors with the same shapes and contents as the input.
Igamma	math_ops	Compute the lower regularized incomplete Gamma function P(a, x).
Igammac	math_ops	Compute the upper regularized incomplete Gamma function Q(a, x).
IgammaGradA	math_ops	-
InplaceAdd	array_ops	Adds v into specified rows of x.
InplaceSub	array_ops	Subtracts v into specified rows of x.
InplaceUpdate	array_ops	Updates specified rows with values in v.
InTopK	nn_ops	Says whether the targets are in the top K predictions.
InTopKV2	nn_ops	Says whether the targets are in the top K predictions.

Supported TF Operator	Category	Description
Inv	math_ops	Computes the reciprocal of x element-wise.
Invert	bitwise_ops	-
InvertPermutation	array_ops	Computes the inverse permutation of a tensor.
IsVariableInitialized	state_ops	Checks whether a tensor has been initialized.
L2Loss	nn_ops	L2 Loss.
Less	math_ops	Returns the truth value of (x < y) element-wise.
LessEqual	math_ops	Returns the truth value of (x <= y) element-wise.
LinSpace	math_ops	Generates values in an interval.
ListDiff	array_ops	-
Log	math_ops	Computes natural logarithm of x element-wise.
Log1p	math_ops	Computes natural logarithm of (1 + x) element-wise.
LogicalAnd	math_ops	Returns the truth value of x AND y element-wise.
LogicalNot	math_ops	Returns the truth value of NOT x element-wise.
LogicalOr	math_ops	Returns the truth value of x OR y element-wise.
LogMatrixDeterminant	linalg_ops	-
LogSoftmax	nn_ops	Computes log softmax activations.
LoopCond	control_flow_ops	Forwards the input to the output.
LowerBound	array_ops	-
LRN	nn_ops	Local Response Normalization.
MatMul	math_ops	Multiply the matrix "a" by the matrix "b".
MatrixBandPart	array_ops	Copy a tensor setting everything outside a central band in each innermost matrix.

Supported TF Operator	Category	Description
MatrixDeterminant	linalg_ops	-
MatrixDiag	array_ops	Returns a batched diagonal tensor with a given batched diagonal values.
MatrixDiagPart	array_ops	Returns the batched diagonal part of a batched tensor.
MatrixInverse	linalg_ops	-
MatrixSetDiag	array_ops	Returns a batched matrix tensor with new batched diagonal values.
MatrixSolve	linalg_ops	-
MatrixSolveLs	linalg_ops	-
MatrixTriangular-Solve	linalg_ops	-
Max	math_ops	Computes the maximum of elements across dimensions of a tensor.
Maximum	math_ops	Returns the max of x and y (i.e.
MaxPool	nn_ops	Performs max pooling on the input.
MaxPoolV2	nn_ops	Performs max pooling on the input.
MaxPoolWithArgmax	nn_ops	Performs max pooling on the input and outputs both max values and indices.
Mean	math_ops	Computes the mean of elements across dimensions of a tensor.
Merge	control_flow_ops	Forwards the value of an available tensor from inputs to output.
Min	math_ops	Computes the minimum of elements across dimensions of a tensor.
Minimum	math_ops	Returns the min of x and y (i.e.
MirrorPad	array_ops	Pads a tensor with mirrored values.
MirrorPadGrad	array_ops	-
Mod	math_ops	Returns element-wise remainder of division.

Supported TF Operator	Category	Description
Mul	math_ops	-
Multinomial	random_ops	Draws samples from a multinomial distribution.
Neg	math_ops	-
NextIteration	control_flow_ops	Makes its input available to the next iteration.
NoOp	no_op	Does nothing.
NotEqual	math_ops	Returns the truth value of (x != y) element-wise.
NthElement	nn_ops	Finds values of the n-th order statistic for the last dimension.
OneHot	array_ops	Returns a one-hot tensor.
OnesLike	array_ops	Returns a tensor of ones with the same shape and type as x.
Pack	array_ops	-
Pad	array_ops	-
ParallelConcat	array_ops	-
ParameterizedTruncatedNormal	random_ops	Outputs random values from a normal distribution.
Placeholder	array_ops	-
PlaceholderWithDefault	array_ops	-
PopulationCount	bitwise_ops	-
Pow	math_ops	Computes the power of one value to another.
PreventGradient	array_ops	-
Prod	math_ops	Computes the product of elements across dimensions of a tensor.
Qr	linalg_ops	-
RandomGamma	random_ops	Outputs random values from the Gamma distribution(s) described by alpha.
RandomGammaGrad	random_ops	

Supported TF Operator	Category	Description
RandomShuffle	random_ops	Randomly shuffles a tensor along its first dimension.
RandomStandardNormal	random_ops	-
RandomUniform	random_ops	Outputs random values from a uniform distribution.
Range	math_ops	Creates a sequence of numbers.
RandomUniformInt	random_ops	Outputs random integers from a uniform distribution.
Rank	array_ops	Returns the rank of a tensor.
ReadVariableOp	resource_variable_ops	-
RealDiv	math_ops	Returns x / y element-wise for real types.
Reciprocal	math_ops	Computes the reciprocal of x element-wise.
RefEnter	control_flow_ops	-
RefExit	control_flow_ops	-
RefMerge	control_flow_ops	-
RefNextIteration	control_flow_ops	Makes its input available to the next iteration.
RefSwitch	control_flow_ops	Forwards the ref tensor data to the output port determined by pred.
Relu	nn_ops	Computes rectified linear: max(features, 0).
Relu6	nn_ops	Computes rectified linear 6: min(max(features, 0), 6).
Reshape	array_ops	Reshapes a tensor.
ReverseSequence	array_ops	Reverses variable length slices.
ReverseV2	array_ops	-
RightShift	bitwise_ops	-
Rint	math_ops	Returns element-wise integer closest to x.
Round	math_ops	Rounds the values of a tensor to the nearest integer, element-wise.

Supported TF Operator	Category	Description
Rsqrt	math_ops	Computes reciprocal of square root of x element-wise.
SegmentMax	math_ops	Computes the maximum along segments of a tensor.
Select	math_ops	-
Selu	nn_ops	Computes scaled exponential linear: scale * alpha * (exp(features) - 1)
Shape	array_ops	Returns the shape of a tensor.
ShapeN	array_ops	Returns shape of tensors.
Sigmoid	math_ops	Computes sigmoid of x element-wise.
Sign	math_ops	Returns an element-wise indication of the sign of a number.
Sin	math_ops	Computes sin of x element-wise.
Sinh	math_ops	Computes hyperbolic sine of x element-wise.
Size	array_ops	Returns the size of a tensor.
Slice	array_ops	Return a slice from 'input'.
Snapshot	array_ops	Returns a copy of the input tensor.
Softmax	nn_ops	Computes softmax activations.
Softplus	nn_ops	Computes softplus: log(exp(features) + 1).
Softsign	nn_ops	Computes softsign: features / (abs(features) + 1).
SpaceToBatch	array_ops	SpaceToBatch for 4-D tensors of type T.
SpaceToBatchND	array_ops	SpaceToBatch for N-D tensors of type T.
SpaceToDepth	array_ops	SpaceToDepth for tensors of type T.
Split	array_ops	Splits a tensor into num_split tensors along one dimension.
SplitV	array_ops	Splits a tensor into num_split tensors along one dimension.

Supported TF Operator	Category	Description
Sqrt	math_ops	Computes square root of x element-wise.
Square	math_ops	Computes square of x element-wise.
SquaredDifference	math_ops	Returns (x - y)(x - y) element-wise.
Squeeze	array_ops	Removes dimensions of size 1 from the shape of a tensor.
StatelessMultinomial	stateless_random_ops	-
StopGradient	array_ops	Stops gradient computation.
StridedSlice	array_ops	Return a strided slice from input.
Sub	math_ops	-
Sum	math_ops	Computes the sum of elements across dimensions of a tensor.
Svd	linalg_ops	-
Switch	control_flow_ops	Forwards data to the output port determined by pred.
Tan	math_ops	Computes tan of x element-wise.
Tanh	math_ops	Computes hyperbolic tangent of x element-wise.
Tile	array_ops	Constructs a tensor by tiling a given tensor.
TopK	nn_ops	Finds values and indices of the k largest elements for the last dimension.
TopKV2	nn_ops	-
Transpose	array_ops	Shuffle dimensions of x according to a permutation.
TruncateDiv	math_ops	Returns x / y element-wise for integer types.
TruncatedNormal	random_ops	Outputs random values from a truncated normal distribution.
TruncateMod	math_ops	Returns element-wise remainder of division.
Unbatch	batch_ops	-

Supported TF Operator	Category	Description
UnbatchGrad	batch_ops	-
Unique	array_ops	Finds unique elements in a 1-D tensor.
UniqueWithCounts	array_ops	Finds unique elements in a 1-D tensor.
Unpack	array_ops	-
UnravelIndex	array_ops	Converts a flat index or array of flat indices into a tuple of.
UnsortedSegment-Min	math_ops	Computes the minimum along segments of a tensor.
UnsortedSegment-Prod	math_ops	Computes the product along segments of a tensor.
UnsortedSegment-Sum	math_ops	Computes the sum along segments of a tensor.
UpperBound	array_ops	-
Variable	state_ops	Holds state in the form of a tensor that persists across steps.
Where	array_ops	Returns locations of nonzero / true values in a tensor.
Xdivy	math_ops	Returns 0 if x == 0, and x / y otherwise, elementwise.
Xlogy	math_ops	Returns 0 if x == 0, and x * log(y) otherwise, elementwise.
ZerosLike	array_ops	Returns a tensor of zeros with the same shape and type as x.
Zeta	math_ops	Compute the Hurwitz zeta function \((x, q)\).
_Retval	function_ops	-
LeakyRelu	nn_ops	-
FusedBatchNormV3	nn_ops/mkl_nn_ops	-

8 FAQs

8.1 What Do I Do If Model Conversion Takes Too Long When the OS and Architecture Configuration of the Development Environment Is Arm (AArch64)?

For an Arm (AArch64) development environment, to accelerate model conversion, you can use the numactl tool to specify CPU cores for model conversion as follows:

1. Log in to the development environment as the ATC installation user and run the **su root** command to switch to the **root** user.
2. Ensure that the development environment is connected to the network, and run the following command to install the numactl tool:
   ```
   yum -y install numactl
   ```
3. Switch to the ATC installation user and run the **numactl -C** command to specify CPU cores 16–31 to process model conversion:

   CPU cores 16–31 are recommended for better processing performance. You can also change the CPU cores as required.

   ```
   numactl -C 16-31 --localalloc <args>
   ```

   Replace *<args>* with the actual ATC model conversion command.

8.2 How Do I Determine the Video Stream Format Standard When I Perform CSC on a Model Using AIPP?

Q: How do I determine the video stream format standard when I perform CSC on a model using AIPP?

A: The third-party tool **ffprobe** is used as an example. You can use other third-party tools.

1. Download the tool and related documents from **https://www.ffmpeg.org/ffprobe-all.html#Description**.
2. Obtain the video information by using the **ffprobe -show_frames *filename*** parameter.

   Parameter function: Show information about each frame and subtitle contained in the input multimedia stream. The information for each single frame is printed within a dedicated section with name "FRAME" or "SUBTITLE".
3. Determine the video standard based on the result:

   color_range: **tv** or **pc**

   color_space: **bt709** or **bt601**

   tv indicates "limited", that is, narrow range. **pc** indicates "full", that is, wide range.

   For example, **color_range=tv and color_space=bt709**. The video stream format standard is NARROW, BT-709.

   NOTE

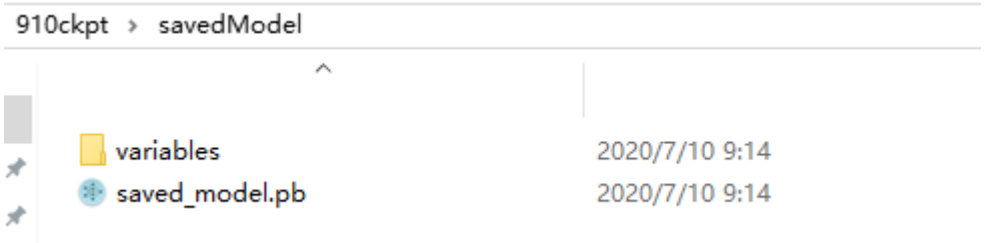
   If the command is different from the example, refer to the official description of the tool.

8.3 What Do I Do If a Network Model Containing an NPU-based Custom Operator Failed to Be Frozen?

During model training on the NPU for a model containing an NPU-based custom operator, the **freeze** method in TensorFlow fails to be called to freeze the trained model. This problem can be solved by importing the NPU-based custom operator library to the **freeze** method of TensorFlow. For example:

Figure 8-1 shows the model file exported by **estimator.export_model** after the bert model is trained on the NPU. The model file contains an NPU-based custom operator.

Figure 8-1 Trained model file



Run the following command to call the **freeze** method in TensorFlow to freeze the model:

```
python3 freeze_npu_savedModel.py --input_saved_model_dir=savedModel --output_node_names=loss/Softmax --output_graph=bert_910.pb
```


The following error information is displayed.

The model file contains an NPU-based custom operator, which is stored in the **npu_ops** package. Therefore, the **freeze** method cannot find the operator in the TensorFlow operator library.

Perform the following steps to solve the problem:

Step 1 Find the Python file of the **freeze** method of TensorFlow, add **npu_ops** to the modules to import in this file by adding the following content.

```
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

import argparse
import re
import sys

from google.protobuf import text_format

from npu_bridge.estimator import npu_ops

from tensorflow.core.framework import graph_pb2
from tensorflow.core.protobuf import saver_pb2
from tensorflow.core.protobuf.meta_graph_pb2 import MetaGraphDef
from tensorflow.python import pywrap_tensorflow
```

Step 2 Run the following command to call the **freeze** method in TensorFlow to freeze the model:

```
python3 freeze_npu_savedModel.py --input_saved_model_dir=savedModel --output_node_names=loss/Softmax --output_graph=bert_910.pb
```

The model is successfully frozen by calling the **freeze** method in TensorFlow, and a normal **bert_910.pb** file is generated.

----End

NOTE

The **freeze** method in TensorFlow can be independently called. You are advised to copy the Python file of the **freeze** method to the user directory and add **npu_ops** to the modules to import. In this way, the **freeze** method can be repeatedly used for models trained on the NPU. Ensure that the environment variables are correctly configured.

A Appendix

A.1 Change History

Date	Change Description
2020-07-25	This is the first official release.

