Application Note

Zhouyi Compass IR Definition

Version 7.3
Document ID: ACN-61010013-010
Non-Confidential

Release Information

Document History

Issue	Date	Confidentiality	Change
A	18/06/2021	Confidential	First release
B	30/09/2021	Confidential	Second release
C	31/12/2021	Non-Confidential	Third release
D	31/03/2022	Non-Confidential	Fourth release
E	30/06/2022	Non-Confidential	Fifth release
F	30/09/2022	Non-Confidential	Sixth release
G	31/12/2022	Non-Confidential	Seventh release
H	31/03/2023	Non-Confidential	Eighth release
I	30/06/2023	Non-Confidential	Ninth release
J	30/09/2023	Non-Confidential	Tenth release

Contents

1 About this document

This Application Note is intended for developers/programmers/users who use the Arm China *Intermediate Representation* (IR). This Application Note gives you a basic understanding of IR and describes how to use it with the *Neural Network* (NN) compiler of the Zhouyi *Neural Processing Unit* (NPU).

In this document, the *Artificial Intelligence Processing Unit* (AIPU) has the same meaning as the NPU.

1.1 References

Reference	Document number	Title
1	61010011_0303_00	Arm China Zhouyi Compass Software Technical Overview
2	ACN-61010017-010	Arm China Zhouyi Compass Operators Specification Application Note

1.2 Terms and abbreviations

This document uses the following terms and abbreviations.

Term	Meaning
AIPU	Artificial Intelligence Processing Unit
IR	Intermediate Representation
NN	Neural Network

1.3 Conventions and feedback

The following describes the typographical conventions and how to give feedback:

Convention	Meaning
`monospace`	denotes text that can be entered at the keyboard, such as commands, file and program names, and source code.
`monospace`	denotes a permitted abbreviation for a command or option. The underlined text can be entered instead of the full command or option name.
`monospace italic`	denotes arguments to commands and functions where the argument is to be replaced by a specific value.
`monospace bold`	denotes language keywords when used outside example code.
italic	highlights important notes, introduces special terminology, denotes internal cross-references, and citations.
bold	highlights interface elements, such as menu names. Also used for emphasis in descriptive lists, where appropriate, and for Arm China processor signal names.

1.3.1 Feedback on this product

If you have any comments and suggestions about this product, contact your supplier and give:

- Your name and company.
- The serial number of the product.
- Details of the release you are using.
- Details of the platform you are using, such as the hardware platform, operating system type and version.
- A small standalone sample of code that reproduces the problem.
- A clear explanation of what you expected to happen, and what actually happened.
- The commands you used, including any command-line options.
- Sample output illustrating the problem.
- The version string of the tools, including the version number and build numbers.

1.3.2 Feedback on documentation

If you have comments on the documentation, e-mail errata@armchina.com. Give:

- The title.
- The number, [Document ID Value], [Issue].
- If viewing online, the topic names to which your comments apply.
- If viewing a PDF version of a document, the page numbers to which your comments apply.
- A concise explanation of your comments.

Arm China also welcomes general suggestions for additions and improvements.

2 Introduction

An *Intermediate Representation* (IR) is the representation of network in various deep learning frameworks to describe the flow of data from the network input data to inference results.

The Arm China Zhouyi AIPU *Neural Network* (NN) compiler has a standard definition for the IR format.

The following procedure shows how the NN compiler works with an IR:

1. The parser tool in the NN compiler can parse a third-party model and convert it to a standard float (data type) IR.
2. The quantization and optimization tool in the NN compiler performs some optimization operations and quantizes the float IR with the given or specified data set, and then generates the integer IR (int IR).
3. The graph build tool in the NN compiler will:

    a. Optimize the execution sequence.
    b. Lower the integer model to fit AIPU instructions or acceleration operations.
    c. Call the built-in operator library.
    d. Build the model to an executable file for the AIPU or simulator platform.

3 Format

The Zhouyi Compass IR is a text file. A universal network model with the .txt suffix is a recommended file name. Generally, the IR definition is divided into two parts. One is the common information and the other specifies all the necessary layer parameters.

Parameters definition

- Common parameters—Define some basic parameters to describe the NN model, which include the model name, classification (or detection, speech recognition), layer numbers, and precision.
- Layer parameters—Specify the type of the operator or neural network (fusion) layer and define the input tensor (or 'layer_bottom') and output tensor (or 'layer_top'). Then, the detailed attributes will be specified, which are used for operator library computation.

Notes

- The IR definition is a highly abstract description for an operator or a neural network (fusion) layer which gives the required or optional parameters, or a standard syntax paradigm. However, the detailed parameter values, ranges and constrains of an operator or a layer supported in the AIPU are listed in the *Zhouyi Compass Operators Specification Application Note*.
- The IR includes float IR and int IR, which can be significantly different from the precision field in the definition of common parameters. You can also find the detailed difference between them in *4.5 Rules*.
- By default, all the data stacked in a memory device (such as DDR or SRAM) in this document means the NHWC or $ND_0D_1D_iD_nC$ format if there are no special explanations.
- In the definition of common or layer parameters, [**required**] or [**optional**] indicates that the parameter must be defined or is optional in an IR.
- In the definition of layer parameters, **Inputs** means the list of 'layer_bottom', **Outputs** describes the list of 'layer_top', and **Attributes** indicates the detailed parameters description of the layer.
- In the definition of layer parameters, some operators can share the same 'layer_type' parameter but have different 'method' fields, which means that they are a class of operators in the IR definition.
- In the definition of layer parameters, the enumerated parameters will enumerate all the supported values that are separated by colons. For example,

  ```
  method: MIN, MAX, ANY, ALL, SUM, MEAN
  ```

  The reduce method, where MIN computes the minimum value of the input tensor's element along the axis. MAX computes the maximum value of the input tensor's element along the axis. ANY computes the 'Logical OR' value of the input tensor's element along the axis. ALL computes the 'Logical AND' value of the input tensor's element along the axis. SUM computes the sum value of the input tensor's element along the axis. MEAN computes the mean value of the input tensor's element along the axis.
- In the definition of layer parameters, there is a concept of group parameter. All parameters in a group are listed and follow the group name by a colon, and each parameter is separated by a comma. The name of all group parameters is composed of the group name and special strings reflecting some characteristics of the parameter. For example,

  ```
  kernel: kernel_x, kernel_y
  ```

  Where, 'kernel_x' (or 'kernel_y') is the kernel size along the 'width' (or 'height') axis. If not present, it should be inferred from 'weights_ size' and 'weight_shape'.

  ```
  lut: lut_type, lut_shape, lut_offset, lut_size
  ```

  Group parameters in an int IR. That is, the *Look Up Table* (LUT) is an offline table created during quantization. Where, 'lut_type', 'lut_shape', 'lut_offset' and 'lut_size' are the data type, table shape, table offset address and table size.
- In the definition of layer parameters, the parameters suffixed with 'shape' should be defined as a list. For example, 'biases_shape=[64]' and 'weights_shape=[64,3,3,32]'.

- For ‘scale’ and ‘shift’ parameters during quantization, a scalar or tensor type specification will be given, which is related to the quantization algorithm implementation.
- For some parameters, they can have different meanings and definition formats in different operators or (fusion) layers to indicate a special behavior. For example, ‘axis’ is a scalar in the ‘ArgMax’ operator while it is a list in the ‘LayerNormalization’ operator.
- The default value that is specified to an attribute is just a recommended or example value, which does not indicate that the attribute is an optional or required parameter in an IR.
- For a bool-type value, ‘true’ will be replaced by ‘1’ and ‘false’ will be replaced by ‘0’, where ‘1’ and ‘0’ are both uint8 data type.

4 Parameters

This section describes the syntax, common parameters, layer parameters, float IR, int IR and rules.

4.1 Syntax

The following is an IR format for a neural network model. The IR format can be identified by the NN compiler and built into an executable binary to run on the simulator or the AIPU platform.

- `<>` indicates a required parameter.
- `::=` indicates the IR definition.
- `[]` indicates an optional parameter.
- `“”` indicates the raw string.
- `|` indicates an alternative parameter.
- `{}` indicates the repeated parameter definition.
- `,` indicates concatenation.
- `()` indicates a group.

`<model_name>.txt ::=`

```
model_name=<model identifier>
layer_number=<the layer numbers of the model>
precision=<data type>
input_tensors=<the list of input tensor>
output_tensors=<the list of output tensor>
model_bin=<the static binary data in model>

layer_id=<the identifier number of current layer in model>
layer_name=<the name of current layer>
layer_type=<the type of current layer>
layer_bottom=[the input tensor names of current layer]
layer_bottom_shape=[the shape of input tensors of current layer]
layer_bottom_type=[the data type of input tensors of current layer]
layer_top=<the output tensor names of current layer>
layer_top_shape=<the shape of output tensors of current layer>
layer_top_type=<the data type of output tensors of current layer>
[layer_top_scale=<the data stacked in the memory device>]
[layer_top_scale=<the scale coefficient used for de-quantization operation>]
[layer_top_zp=<the zeropoint used for asymmetrical de-quantization operation>]
[layer_top_range=<the minimum and maximum threshold of the activation>]
[weights_range=<the minimum and maximum threshold of the weight>]
[kernel_x=<the kernel size along the width axis>]
[kernel_y=<the kernel size along the height axis>]
[stride_x=<the stride size along the width axis>]
```

```
[stride_y=<the stride size along the height axis>]
[pad_left=<the left padding number along width axis of a cube>]
[pad_right=<the right padding number along width axis of a cube>]
[pad_top=<the top padding number along height axis of a cube>]
[pad_bottom=<the bottom padding number along height axis of a cube>]
[dilation_x=<the dilation factor along the width axis>]
[dilation_y=<the dilation factor along the height axis>]

[weights_type=<the data type of the weight>]
[weights_offset=<the offset address of the weight>]
[weights_size=<the total data size of the weight counted in bytes>]
[weights_shape=<the shape of the weight>]
[biases_type=<the data type of the bias>]
[biases_offset=<the offset address of the bias>]
[biases_size=<the total data size of the bias counted in bytes>]
[biases_shape=<the shape of the bias>]
[with_activation=<the activation added to the output tensor of the current layer>]
[clip_min=<the minimum value of Clip>]
[clip_max=<the maximum value of Clip>]
[num_output=<number of output channels>]
[method=<the operation type of Pooling or Eltwise etc.>]
[group=<the group numbers of the convolution operation>]
[negative_slope_type=<the slope value data type of Prelu>]
[negative_slope_shape=<the slope value shape of Prelu>]
[negative_slope_offset=<the offset address of Prelu slope>]
[negative_slope_size=<the data size of Prelu slope>]
[negative_slope_shift=<the quantization parameter of Prelu>]

[lut_type=<the data type of the lut table during quantization>]
[lut_offset=<the offset address of the lut table during quantization>]
[lut_size=<the data size of the lut table counted in bytes during quantization>]
[lut_shape=<the total shape of the lut table during quantization>]
[scale_type=<the quantization data type of the scale during quantization>]
[scale_offset=<the offset address of the scale during quantization>]
[scale_size=<the data size of the scale counted in bytes during quantization>]
[scale_shape=<the total shape of the scale during quantization>]
[scale_value=<a float value during quantization>]
[shift_type=<the quantization data type of the shift parameter during quantization>]
[shift_offset=<the offset address of the shift parameter during quantization>]
[shift_size=<the data size of the shift parameter counted in bytes during quantization>]
```

```
[shift_shape=<the total shape of the shift parameter during quantization>]
[shift_value=<an integer value during quantization>]

[{other layer parameter definition}]
```

4.2 Common parameters

- model_name [**required**]

  The name of the input network model. It must be a string to identify the model.

- layer_number [**required**]

  The total layer number of the input neural network model.

- precision [**required**]

  The input data type of activation, weight or bias.

  - float

    Means that the IR is a float IR.

  - int

    Means that the IR is an int IR.

  - mixture

    Means that the IR is a mixture, which includes the float and int layer.

- input_tensors [**optional**]

  A list to indicate all the input tensors with the given order.

- output_tensors [**optional**]

  A list to indicate all the output tensors with the given order. Generally, an empty list represents that the tensors are outputted with the default order. For example, 'output_tensors=[]', which means that you do not care about the order.

- model_bin [**optional**]

  An optional parameter to indicate the static binary data in a model including weight and biases. Also, it includes scale, shift, lut and other quantization parameters in an int IR. Generally, the parameter in binary is little-endian in memory and stacked with the specified offset address and size which are separated by suffix '_offset' and '_size' (such as, lut_offset and lut_size) in an IR.

  However, it is valid if the command line does not supply the '-w' option when running the NN compiler, while it is invalid if the '-w' option is given. In addition, it is better to maintain the same relative path between 'model_bin' and IR. Generally, a relative path based on the current work directory is recommended.

- compat_quantized_model [**optional**]

  An optional parameter to indicate whether the input model is a quantization model. If compat_quantized_model = true, the model input compass parse is a quantization model, and the compass quantization weight and bias with scale and shift parameters will be presented by a constant_params in the int IR. If compat_quantized_model = false, the model input compass parse is an un_quantization model.

The following is an example of an int IR with the common necessary parameters.

```
model_name=resnet50
layer_number=77
precision=int
input_tensors=[Placeholder_0]
output_tensors=[resnet_v1_50/predications/Reshape_0]
model_bin=./int_IR/resnet50_int8.bin
```

Example 4-1 An int IR with common necessary parameters

4.3 Layer parameters

4.3.1 Basic operator parameters

- layer_id [**required**]

  The identification sequence number of each layer. Note that it does not mean the real computation sequence.

- layer_name [**required**]

  The name of a layer.

- layer_type [**required**]

  The operator type of a layer.

- layer_bottom [**required**]

  The input tensor names of the current layer. This parameter will be empty when it is the input or constant layer.

- layer_bottom_shape [**required**]

  The input tensor shapes of the current layer in NHWC format. This parameter will be empty when it is the input or constant layer.

- layer_bottom_type [**required**]

  The data type of input tensors of the current layer. This parameter will be empty when it is the input or constant layer.

- layer_top [**required**]

  The output tensor names of the current layer.

- layer_top_shape [**required**]

  The output tensor names of the current layer in NHWC format.

- layer_top_type [**required**]

  The data type of output tensors of the current layer.

- layer_top_scale [**optional**]

  An optional parameter in an int IR. It means the scale coefficient used for de-quantization operation for the output tensors.

- layer_top_zp [**optional**]

An optional parameter in an int IR. It means the zeropoint used for asymmetrical de-quantization operation for the output tensors. Generally, it defaults to 0 during symmetric quantization.

- layer_top_datalayout [optional]

  An optional parameter in an int IR. Currently, it supports:

  - NCHWC32
  - NHWC
  - NCHWC16

  It means the data stacked in the memory device, such as DDR or SRAM. Do not specify this parameter if you are not sure of it. Note that this parameter will support more options depending on the software (such as simulator) or hardware platform in the future.

- layer_top_range [optional]

  A list to indicate the minimum and maximum values of the activation. It will be no use after quantization processing (that is, will be deleted from an int IR). For example, layer_top_range=[-2.0, 3.0] indicates that the activation minimum and maximum values are -2.0 and 3.0, respectively.

- weights_range [optional]

  A list to indicate the minimum and maximum values of the weights. It will be no use after quantization processing (that is, will be deleted from an int IR). For example, weights_range=[-1.2, 5.0] indicates that the weights minimum and maximum values are -1.2 and 5.0, respectively.

Abs

Absolute takes one input data (Tensor) and produces one output data (Tensor) where the absolute is, y = abs(x), is applied to the tensor elementwise.

Inputs

- Input data tensor X.

Outputs

- Output tensor Y with the same shape as X.

Attributes

- layer_type

  The operation type of a layer. Here is ‘layer_type = Abs’.

AccidentalHits

Computes the position ids in sampled_candidates matching true_classes. Where, the input tensor true_classes means the target classes with a shape [batch_size, num_true]. The input tensor sampled_candidates is a vector and means the candidates matched by true_classes with a shape [num_sampled].

Assume that the sampled_candidates is unique. It is called an ‘accidental hit’ when one of the target classes matches one of the sampled candidates. Then, the row number in true_classes and the position in sampled_candidates will be recorded as the result of indices and ids respectively. In addition, the output tensor ‘effective_len’ indicates the actual or effective matching number or length with a tensor shape of [1].

For example, the input tensor is ‘true_classes = [[10, 9, 105, 6, 2], [11, 9, 113, 2, 10]], sampled_candidates = [10, 6, 9]’, then the output tensor will be ‘indices = [0, 0, 0, 1, 1], ids = [0, 2, 1, 2, 0], length = [5]’. Generally, the maximum value of the effective length is defined as **32768**, then the output tensor value of [5] means that only five results are effective in the redundancy output tensor.

Inputs

- Input ‘true_classes’ tensor and ‘sampled_candidates’ tensor.

Outputs

- Output indices, ids and ‘effective_len’ tensor.

Attributes

- layer_type

  The operation type of a layer. Here is ‘layer_type = AccidentalHits’.

Acos

Calculates the arccosine (inverse of cosine) of the given input tensor, element-wise.

Inputs

- Input data tensor X.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is ‘layer_type = Acos’.

- lut: lut_type, lut_shape, lut_offset, lut_size

  The *Look Up Table* (LUT) is an offline table created after quantization. Where, lut_type, lut_shape, lut_offset and lut_size are the data type, table shape, table offset address and table size. The following is a simple equation, lut_size = data_type * lut_shape. Where, data type = 1 (or 2, 4) if lut_type = int8 (or int16, int32).

Acosh

Calculates the hyperbolic arccosine of the given input tensor element-wise.

Inputs

- Input data tensor X.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is ‘layer_type = Acosh’.

- lut: lut_type, lut_shape, lut_offset, lut_size

The *Look Up Table* (LUT) is an offline table created after quantization. Where, lut_type, lut_shape, lut_offset and lut_size are the data type, table shape, table offset address and table size. The following is a simple equation, lut_size = data_type * lut_shape. Where, data type = 1 (or 2, 4) if lut_type = int8 (or int16, int32).

Add

Performs addition of each of the input tensors (with NumPy-style broadcasting support). All inputs and outputs must have the same data type. Besides, this operator supports multidirectional (that is, NumPy-style) broadcasting.

Inputs

- Input data tensor X1, data tensor X2.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = Add'.

- scale: scale_type, scale_value

  The scale is a vector with more than two elements, and its number of elements should be equal to the sum of output and input tensors. Where, scale_type and scale_value are the data type and value of the scale during quantization. Generally, scale_type has int8, uint8, and int16 options. Furthermore, the scales type or value must be in order of 'output_scale, input_scale[i]'.

- shift: shift_type, shift_value

  It can be a scalar, which means an output shift operation during quantization. Where, shift_type and shift_value are the data type of the shift operation.

ArgMax

Returns the index with the largest value of the input tensor X along the specified axis. If 'select_last_index' is 'true', the index of the last appearance of the max is selected if the maximum element appears more than once in the input tensor. Otherwise, the index of the first occurrence is selected when 'select_last_index' is 'false'. Besides, the output tensor defaults to keep the dimension as input tensor X. For example, the input tensor X.shape = [2, 3, 4], axis = 1, then the output tensor Y.shape = [2, 1, 4].

Inputs

- Input data tensor X.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = ArgMinMax'.

- method: MAX

  Method to perform the index of the input tensor.

- axis

A scalar of integer, the axis to reduce across. A negative value means counting dimension from the back. It defaults to -1. The accepted range is [-1, r-1] where r = rank(input data).

- select_last_index: true, false

  An integer to select the last index or the first index if the maximum element appears many times. It defaults to 'false' (that is, to return the first index).

ArgMin

Returns the index with the smallest value of the input tensor X along the specified axis. If 'select_last_index' is 'true', the index of the last appearance of the min is selected if the minimum element appears more than once in the input tensor. Otherwise, the index of the first occurrence is selected when 'select_last_index' is 'false'. Besides, the output tensor defaults to keep the dimension as input tensor X. For example, the input tensor X.shape = [2, 3, 4], axis = 1, then the output tensor Y.shape = [2, 1, 4].

Inputs

- Input data tensor X.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = ArgMinMax'.

- method: MIN

  Method to perform the index of the input tensor.

- axis

  A scalar of integer, the axis to reduce across. A negative value means counting dimension from the back. It defaults to -1. The accepted range is [-1, r-1] where r = rank(input data).

- select_last_index: true, false

  An integer to select the last index or the first index if the minimum element appears many times. It defaults to 'false' (that is, to return the first index).

Asin

Calculates the arcsine (inverse of sine) of the given input tensor, element-wise.

Inputs

- Input data tensor X.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = Asin'.

- lut: lut_type, lut_shape, lut_offset, lut_size

The *Look Up Table* (LUT) is an offline table created after quantization. Where, lut_type, lut_shape, lut_offset and lut_size are the data type, table shape, table offset address and table size. The following is a simple equation, lut_size = data_type * lut_shape. Where, data type = 1 (or 2, 4) if lut_type = int8 (or int16, int32).

Asinh

Calculates the hyperbolic arcsine of the given input tensor element-wise.

Inputs

- Input data tensor X.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = Asinh'.

- lut: lut_type, lut_shape, lut_offset, lut_size

  The *Look Up Table* (LUT) is an offline table created after quantization. Where, lut_type, lut_shape, lut_offset and lut_size are the data type, table shape, table offset address and table size. The following is a simple equation, lut_size = data_type * lut_shape. Where, data type = 1 (or 2, 4) if lut_type = int8 (or int16, int32).

AveragePooling2D

AveragePooling-2D takes an input tensor X and applies average pooling across the tensor by the kernel size, stride size, and padding. AveragePooling-2D consists of computing the average on all values of a subset of the input tensor according to the kernel size and down sampling the data into the output tensor Y for further processing. If no padding is present, the 'pad' group parameters default to keep the shape the same as the input tensor during computing.

Inputs

- Input data tensor X.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = Pooling'.

- kernel: kernel_x, kernel_y

  'kernel_x' ('kernel_y') is the kernel size along the 'width' ('height') axis.

- stride: stride_x, stride_y

  Stride along 'batch', 'height', 'width' and 'channel' axis. Generally, the 'stride_x' ('stride_y') stride along the width (height) axis. It defaults to 1 if not present.

- pad: pad_bottom, pad_top, pad_left, pad_right

  Padding for the beginning and ending along spatial axis, it can take any value greater than or equal to 0. The value represents the number of pixels added to the beginning and end part of the corresponding axis. The 'pad' parameter format should be as follows (‘pad_bottom’, ‘pad_top’, ‘pad_left’, ‘pad_right’), where it means the pad pixels of the data cube (that is, bottom and top along the ‘height’ axis; left and right along the ‘width’ axis). If not present, compute and output the same shape as the input tensor.

- dilation: dilation_x, dilation_y

  The dilation value along the spatial axis of the filter. If not present, the dilation default value is 1 along each spatial axis. Where the dilation_x (dilation_y) means the height (width) axis of the filter in the NHWC data format. Generally, it defaults to 1 if not present.

- method: AVG

  The downsampling method is average during computing pooling.

- count_include_pad: true, false

  Indicates whether to include pad pixels when calculation values are for the edges. Where ‘false’ means not counting pad pixels, while ‘true’ means counting the paddings.

- ceil_mode: true, false

  Optional parameter. Where ‘true’ means using **Ceil** instead of the **Floor** function to compute the output shape. Generally, the default value is ‘false’.

AveragePooling3D

AveragePooling-3D takes an input tensor X and applies average pooling across the tensor by the kernel size, stride size, and padding. AveragePooling-3D consists of computing the average on all values of a subset of the input tensor according to the kernel size and downsampling the data into the output tensor Y for further processing. If no padding is present, the ‘pad’ group parameters keep the shape as the input tensor by default during computing. Generally, the output shape can be calculated as:

output_shape[i] = round(input_shape + pad_begin[i] + pad_end[i] - ((kernel_size[i] -1) * dilation[i]+1) / stride[i] + 1)

Where ‘round’ represents the **floor** or **ceil** function during the round operation.

Inputs

- Input data tensor X.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is ‘layer_type = Pooling3D’.

- kernel: kernel_x, kernel_y, kernel_z

  ‘kernel_x’ (‘kernel_y’, ‘kernel_z’) is the kernel size along the ‘width’ (‘height’, ‘depth’) axis.

- stride: stride_x, stride_y, stride_z

  The stride of the sliding window for each dimension of the input tensor. Where the ‘stride_x’ (‘stride_y’, ‘stride_z’) stride is along the ‘width’ (‘height’, ‘depth’) axis under the NDHWC data format.

- pad: pad_x_begin, pad_x_end, pad_y_begin, pad_y_end, pad_z_begin, pad_z_end

  Padding for the beginning and ending along the spatial axis, it can take any value greater than or equal to 0. The value represents the number of pixels that are added to the beginning and end part of the corresponding axis. If not present, it is provided by the shape of the output.

- dilation: dilation_x, dilation_y, dilation_z

  The dilation value along the spatial axis of the filter. If not present, the dilation default value is 1 along each spatial axis. Where, 'dilation_x' ('dilation_y' or 'dilation_z') means the Height (Width or Depth) dimension of the filter under the NDHWC data format.

- method: AVG

  The downsampling method is 'Average' during computing pooling.

- count_include_pad: true, false

  Means whether to include pad pixels when calculation values are for the edges. Where 'false' means not counting pad pixels, while 'true' means counting the paddings.

- ceil_mode: true, false

  Optional parameter. Where 'true' means using **Ceil** instead of the **Floor** function to compute the output shape. Generally, the default value is 'false'.

BNLL

Takes one input tensor and produces one output tensor, where the sigmoid function 'y = x + log(1 + exp(-x)) if x > 0, otherwise y = log(1 + exp(x))', is applied to the tensor elementwise.

Inputs

- Input data tensor X.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = BNLL'.

- lut: lut_type, lut_shape, lut_offset, lut_size

  The *Look Up Table* (LUT) is an offline table created during quantization. Where, lut_type, lut_shape, lut_offset and lut_size are the data type, table shape, table offset address and table size. The following is an example equation for data type int8, 'lut_size = data_type * lut_shape'. Where, data_type = 1.

BasicLSTM

Computes a one-layer LSTM.

Notations:

x: Input tensor

i: Input gate

o: Output gate

f: Forget gate

c: Cell gate

t: Time step (t-1 means the previous time step)

W[i, o, f, c]: Parameter weight matrix for input, output, forget and cell gates

R[i, o, f, c]: Recurrence weight matrix for input, output, forget and cell gates

b[i, o, f, c]: Bias vectors for input, output, forget and cell gates

H: Hidden state

f: Activation function

g: Activation function

h: Activation function

Activation functions:

Relu: max(0, x)

Tanh: (1 - e^{-2x}) / (1 + e^{-2x})

Sigmoid: 1/(1 + e^{-x})

LeakyRelu: x if x >= 0 else alpha * x

ThresholdedRelu: x if x >= alpha else 0

HardSighmoid: min(max (alpha * x + beta, 0), 1)

Elu: x if x >= 0 else alpha * (e^x -1)

Softsign: x / (1 + |x|)

Softplus: log(1 + e^x)

Equations (Default: f = SIGMOID, g = TANH, h = TANH):

$i_t = f(W_i\, x_t + R_i\, H_{t-1} + b_i)$

$f_t = f(W_f\, x_t + R_f\, H_{t-1} + b_f)$

$c_t = g(W_c\, x_t + R_c\, H_{t-1} + b_c)$

$C_t = f_t \otimes C_{t-1} + i_t \otimes c_t$

$o_t = f(W_o\, x_t + R_o\, H_{t-1} + b_o)$

$H_t = o_t \otimes h(C_t)$

Where, ⊗ means element-wise multiplication (or Hadamard product). During computation, this operator has some optional inputs. An empty string means using the default value or unspecified arguments.

Note that, in the following Inputs or Outputs sections:

- X means the input with the shape as [batch size, time step, input_size].
- H_0 means the initial hidden state for each element in the batch with the shape as [batch size, hidden size].
- C_0 means the initial cell gate with the shape as [batch_size, hidden_size].
- Y means a tensor that concats the intermediate output values of the hidden state and it has the shape '[batch_size, seq_length, hidden_size]'.
- H means the last output value of the hidden state and it has the shape '[batch_size, hidden_size]'.
- C means the last output value of the LSTM cell and it has the shape '[batch_size, hidden_size]'.

Inputs

- Input data tensor X, H_0, C_0.

Outputs

- The output tensor is a certain combination of Y, C and H tensors which are described as 'out_sequence' parameters.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = BasicLSTM'.

- out_sequence

  The output sequence is one of enumerated lists and in a fixed order. Here, the enumerated list includes (Y), (H), (C), (Y, H), (Y, C), (H, C) or (Y, H, C). Generally, Y means a tensor that concats the intermediate output values of the hidden state and it has the shape '[batch_size, seq_length, hidden_size]'. H means the last output value of the hidden state and it has the shape '[batch_size, hidden_size]'. C means the last output value of the LSTM cell and it has the shape '[batch_size, hidden_size]'.

- activations

  A list of activation functions for input (or output), cell and output gates in order of (f, g, h) in BasicLSTM. The activation must be one of the activation functions defined in the preceding section.

- activation_alpha

  Optional scaling parameters in some activation functions. The values compose of an order in activation functions, such as (f, g, h) in BasicLSTM. Generally, the values are the same as the values of corresponding operators. For example, the alpha value in LeakyRelu is 0.01.

- activation_beta

  Optional scaling parameters in some activation functions. The values compose of an order in activation functions, such as (f, g, h) in BasicLSTM. Generally, the values are the same as the values of corresponding operators.

- threshold

  Optional parameter in a float IR, which is applied to the input of activations (or input tensor X). Clipping bounds the elements of a tensor in the range of [- threshold, + threshold]. There is no clip if not specified.

- clip: clip_min, clip_max

  Optional parameter in a float IR, which is applied to the CLIP activation functions. Where, clip_min (or clip_max) means the minimum (or maximum) saturation threshold that is applied to the activation functions.

- direction: Forward, Reverse

Indicates that the RNN is forward or reverse and must be either of the directions.

- time_steps

  Time step.

- input_size

  Size of input tensor x.

- cell_size

  Size of hidden state h.

- weights: weights_type, weights_offset, weights_size, weights_shape

  The weight tensor that will be used in the computation and has weights_type, weights_offset, weights_size and weights_shape which mean the weights data format, weights address offset, total weights size (counted in bytes) and weights shape of the current layer. The corresponding equation is, 'weights_size = data_type * h * (x + h)' and 'weights_shape = 4 * h, x + h'. Where, data type = 1 (or 2, 4) if weight_type = int8 (or int16, int32).

  Note that 'wight_shape' is stacked in order of [[W_{i0} , R_{i0}], [W_{c0} , R_{c0}], [W_{f0} , R_{f0}], [W_{o0} , R_{o0}]], ..., [[$W_{i(h-1)}$, $R_{i(h-1)}$], [$W_{c(h-1)}$, $R_{c(h-1)}$], [$W_{f(h-1)}$, $R_{f(h-1)}$], [$W_{o(h-1)}$, $R_{o(h-1)}$]] (that is, weights_shape = [4*h, x+h]). Where, 'x' and 'h' mean the input size and hidden size as described in the preceding section.

- biases: biases_type, biases_offset, biases_size, biases_shape

  To be added to the computation and has biases_type, biases_offset, biases_size and biases_shape which mean the biases data format, biases address offset, total biases size (counted in bytes) and biases shape of the current layer. The corresponding equation is, 'biases_size = data_type * 4 * h' and 'biases_shape = 4 * h'. Where, data type = 1 (or 2, 4) if weight_type = int8 (or int16, int32).

  Note that 'biases_shape' is stacked in order of [b_{i0}, ..., $b_{i(h-1)}$, b_{c0}, ..., $b_{c(h-1)}$, b_{f0}, ..., $b_{f(h-1)}$, b_{o0}, ..., $b_{o(h-1)}$] (that is, biases_shape = 4*h). Where, 'h' means the hidden size as described in the preceding section.

- scale: scale_type, scale_shape, scale_offset, scale_size

  It can be a scalar, which means a per-tensor/layer or per output channel quantization. Where, scale_type, scale_shape, scale_offset and scale_size are the data type and numbers, offset address and data size of the scale and coefficient during quantization. The corresponding equation is, 'scales_size = (5 + 2 * t) * data_type' and 'scales_shape = (5 + 2 * t)'. Where, data type = 1 (or 2, 4) if weight_type = int8 (or int16, int32).

  In addition, the scale_type has int8, uint8, int16, uint16 and int32, where the relationship is, 'scale_size=data_type * scale_shape'.

- shift: shift_type, shift_shape, shift_offset, shift_size

  It can be a scalar or a 1-D tensor, which means a per-tensor/layer or per output channel quantization. Where, shift_type, shift_shape, shift_offset and shift_size are the data type, shape, offset address and data size of the shift coefficient during quantization. The equation is, 'shift_size = data_type * (5 + 2 * t)' and 'shift_shape = 5 + 2 * t'. Where, data type = 1 (or 2, 4) if weight_type = int8 (or int16, int32).

- lut_it: lut_it_type, lut_it_shape, lut_it_offset, lut_it_size

  The *Look Up Table* (LUT) is an offline table created during activation quantization in 'i_t' calculation. Where, lut_it_type, lut_it_shape, lut_it_offset and lut_it_size are the data type, table shape, table offset address and table size. The following is a simple equation, 'lut_it_size = data_type * lut_it_shape'. Where, data type = 1 (or 2, 4) if lut_it_type = int8 (or int16, int32).

- lut_ft: lut_ft_type, lut_ft_shape, lut_ft_offset, lut_ft_size

The *Look Up Table* (LUT) is an offline table created during activation quantization in 'f_t' calculation. Where, lut_ft_type, lut_ft_shape, lut_ft_offset and lut_ft_size are the data type, table shape, table offset address and table size. The following is a simple equation, 'lut_ft_size = data_type * lut_ft_shape'. Where, data type = 1 (or 2, 4) if lut_ft_type = int8 (or int16, int32).

- lut_ct: lut_ct_type, lut_ct_shape, lut_ct_offset, lut_ct_size

  The *Look Up Table* (LUT) is an offline table created during quantization activation in 'c_t' calculation. Where, lut_ct_type, lut_ct_shape, lut_ct_offset and lut_ct_size are the data type, table shape, table offset address and table size. The following is a simple equation, 'lut_ct_size = data_type * lut_ct_shape'. Where, data type = 1 (or 2, 4) if lut_ct_type = int8 (or int16, int32).

- lut_ot: lut_ot_type, lut_ot_shape, lut_ot_offset, lut_ot_size

  The *Look Up Table* (LUT) is an offline table created during activation quantization in 'o_t' calculation. Where, lut_ot_type, lut_ot_shape, lut_ot_offset and lut_ot_size are the data type, table shape, table offset address and table size. The following is a simple equation, 'lut_ot_size = data_type * lut_ot_shape'. Where, data type = 1 (or 2, 4) if lut_ot_type = int8 (or int16, int32).

- lut_h: lut_h_type, lut_h_shape, lut_h_offset, lut_h_size

  The *Look Up Table* (LUT) is an offline table created during activation quantization in 'H_t' calculation. Where, lut_h_type, lut_h_shape, lut_h_offset and lut_h_size are the data type, table shape, table offset address and table size. The following is a simple equation, 'lut_h_size = data_type * lut_h_shape'. Where, data type = 1 (or 2, 4) if lut_h_type = int8 (or int16, int32).

- lut_shift: lut_shift_value,lut_shuft_type

  It can be a scalar during quantization, where lut_shift_type and lut_ shift_value are the data type and value of the shift during quantization. The lut_shift_value is used to prevent index number from exceeding the lut size.

BatchNormalization

Computes the batch normalization according to the model's running. Generally, there are five required input tensors including data tensor X, scale tensor 'Gamma', bias tensor 'Beta', 'input_mean' tensor, and 'input_var' tensor. Note that the 'input_mean' and 'input_var' tensors represent the mean and variance, which are expected to be the estimated statistics in inference mode and running statistics in training mode.

Inputs

- Input data tensor X.

Outputs

- Output tensor Y with the same shape as X.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = BatchNorm'.

- weights: weights_type, weights_offset, weights_size, weights_shape

  Where, weight = (gamma / sqrt(var + epsilon)) and 'weights_type' (or weights_offset, weights_size, weights_shape) represents the weights data format (or weights address offset, total weights size which counted in bytes, and weights shape of the current layer) after 'Gamma' and 'input_var' parameter conversion. Generally, the 'weights_shape' is the shape mutilation of the dimension that 'axis' specifies. In addition, the 'epsilon' is a parameter during float computation.

- biases: biases_type, biases_offset, biases_size, biases_shape

  Where, bias = beta - mean * (gamma / sqrt(var + epsilon)) and biases_type (or biases_offset, biases_size, biases_shape) represents the biases data format (or biases address offset, total biases size which counted in bytes, and biases shape of the current layer) after 'Beta', 'input_mean', 'Gamma' and 'input_var' parameter conversion. Generally, the 'biases_shape' is the shape mutilation of the dimension that 'axis' specifies. In addition, the 'epsilon' is a parameter during float computation.

- scale: scale_type, scale_value

  It can be a scalar or a 1-D tensor, which means a per-tensor/layer (by default) or per output channel (need extended parameters in the future) quantization. If it is a 1-D tensor, its number of elements should be equal to the number of output channels, where scale_value, scale_type, scale_shape, scale_offset, and scale_size are the value, data type and numbers, offset address and data size of the 'scale' coefficient during quantization. If it is a scalar, scale_value and scale_type mean the value and data type of the scale parameter during per-tensor/layer quantization.

  Besides, scale_type has the int8, uint8, and int16 options. Where, the relationship is scale_size= data_type * scale_shape. Here, data type = 1 (or 2, 4) if shift_type = int8 (or int16, int32).

- shift: shift_type, shift_value

  It can be a scalar or a 1-D tensor, which means a per-tensor/layer (by default) or per output channel (need extended parameters in the future) quantization. If it is a 1-D tensor, its number of elements should be equal to the number of output channels, where shift_value, shift_type, shift_shape, shift_offset, and shift_size are the value, data type and numbers, offset address and data size of the 'shift' coefficient during quantization. If it is a scalar, shift_value and shift_type mean the value and data type of the shift parameter during per-tensor/layer quantization.

  Besides, shift_type is an int8 format. Where, the relationship is shift_size= data_type * shift_shape. Here, data type = 1 (or 2, 4) if shift_type = int8 (or int16, int32).

- axis

  A scalar of integer indicates that the statistics are continuously computed for the specified dimension over N in [N, H, W, C] or [N, D1, D2, Di, …, C] data format. For example, axis = 3 means that statistics are computed for every channel of C over N, H and W dimensions in NHWC format. That is, all 'N', 'H', 'W' dimensions share the same mean and variance during batch normalization when axis = 3.

- epsilon

  A float IR parameter. The 'epsilon' value is used to avoid division by zero. Generally, the default value is less than 10^{-5}.

BatchToSpace

BatchToSpace rearranges (permutes) data from the batch into blocks of spatial data. This is the reverse transformation of SpaceToBatch. More specifically, this operator outputs a copy of the input tensor where values from the batch dimension are moved in spatial blocks to the 'Height' and 'Width' dimensions.

Inputs

- Input data tensor X.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = BatchToSpace'.

- block_size: block_size_x, block_size_y

  It means that blocks of [block_size_x, block_size_y] are moved, where block_size_x (or block_size_y) is the block size along with the Width (or Heights) dimension in NHWC data format. Generally, the output tensor will be [N/(block_size_x * block_size_y), H * block_size_y - crop_top - crop_bottom, W * block_size_x - crop_left - crop_right, C].

- crop: crop_left, crop_right, crop_top, crop_bottom

Outputs tensor after cropping the Width and Heights dimension with the size of crop_left, crop_right, crop_top, and crop_bottom.

BiasAdd

Performs bias addition to the input data tensor X. It is a special case of the ElementwiseAdd operation, where bias is restricted to 1-D tensor with the size matching the channel dimension of the input data tensor. Broadcasting is supported so that the value tensor can have different dimensions.

Inputs

- Input data tensor X.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = BiasAdd'.

- biases: biases_type, biases_offset, biases_size, biases_shape

  The bias to be added to the input data tensor. Where, biases_type (or biases_offset, biases_size, biases_shape) means the biases data format (or biases address offset, total biases size which counted in bytes, and biases shape of the current layer). The bias has the same size as the channel dimension of the input data tensor in NHWC format.

- scale: scale_type, scale_value

  It can be a scalar during quantization, where scale_type (or scale_value) is the data type (or value) of the scale during per tensor or layer quantization.

- shift: shift_type, shift_value

  It can be a scalar during quantization, where shift_type (or shift_value) is the data type (or value) of the shift during per tensor or layer quantization.

BitwiseAnd

Returns the tensor resulting from performing the bitwise and operation elementwise on the input tensors A and B (with NumPy-style broadcasting support).

This operator supports multidirectional (that is, NumPy-style) broadcasting.

Inputs

- First input data tensor X1, second input data tensor X2.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = Bitwise'.

- method: AND

Bitwise operation method. AND returns the element-wise truth value of ‘x AND y’.

BitwiseNot

Returns the tensor resulting from performing the bitwise and operation elementwise on the input tensors not A (with NumPy-style broadcasting support).

This operator supports multidirectional (that is, NumPy-style) broadcasting.

Inputs

- Input data tensor X1.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is ‘layer_type = Bitwise’.

- method: NOT

  Bitwise operation method. NOT returns the element-wise value of ‘NOT x ‘.

BitwiseOr

Returns the tensor resulting from performing the bitwise and operation elementwise on the input tensor A or B (with NumPy-style broadcasting support).

This operator supports multidirectional (that is, NumPy-style) broadcasting.

Inputs

- First input data tensor X1, second input data tensor X2.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is ‘layer_type = Bitwise’.

- method: OR

  Bitwise operation method. OR returns the element-wise truth value of ‘x OR y’.

BitwiseXor

Returns the tensor resulting from performing the bitwise and operation elementwise on the input tensor A xor B (with NumPy-style broadcasting support).

This operator supports multidirectional (that is, NumPy-style) broadcasting.

Inputs

- First input data tensor X1, second input data tensor X2.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = Bitwise'.

- method: XOR

  Bitwise operation method. XOR returns the element-wise truth value of 'x XOR y'.

BoundingBox

Boundingbox regression.

Note that, in the following Inputs or Outputs sections:

- X1 means the input boxes.
- X2 means the delta of box.
- Y1 means the output box.

Inputs

- Input tensor X1, X2.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = BoundingBox'.

- th_lut: th_lut_type, th_lut_shape, th_lut_offset, th_lut_size

  The look up table (th_lut) is an offline table created after quantization for the gaussian method. Where, th_lut_type, th_lut_shape, th_lut_offset, and th_lut_size are the data type, table shape, table offset address, and table size. The following is a simple equation, lut_size = data_type * lut_shape. Where, data type = 1 (or 2, 4) if lut_type = int8 (or int16, int32).

- tw_lut: tw_lut_type, tw_lut_shape, tw_lut_offset, tw_lut_size

  The look up table (tw_lut) is an offline table created after quantization for the gaussian method. Where, th_lut_type, th_lut_shape, th_lut_offset, and th_lut_size are the data type, table shape, table offset address, and table size. The following is a simple equation, lut_size = data_type * lut_shape. Where, data type = 1 (or 2, 4) if lut_type = int8 (or int16, int32).

- box_scale : box_scale_type, box_scale_value

  The scale is a vector with five elements in order of [y, x, h, w, box]. Where, scale_type and scale_value are the data type and value of the scale during quantization. Generally, scale_type has the int8, uint8 and int16 options.

- box_shift: box_shift_type, box_shift_value

  The shift is a vector with five elements in order of [y, x, h, w, box]. Where, shift_type and shift_value are the data type and value of the shift during quantization.

- delta_shift

  It is a scalar value during quantization for box regression to avoid the result exceeding 32 bits.

CRelu

Performs a Relu operation on the input tensor (X) which selects the positive part of the activation and negative part of the activation respectively, then concatenates them as an output tensor (Y) along with the specified axis. That is, Crelu(x) = Concat [Relu(x), Relu(-x)].

Inputs

- Input data tensor X.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = Activation'.

- method: CRELU

  Method to perform the input tensor.

- axis

  The axis that the output values are concatenated along. The default value is '-1' under NHWC format. The accepted range is [-1, r-1] where r = rank(input data).

- scale: scale_type, scale_value

  It can be a scalar during quantization, where scale_type and scale_value are the data type and value of the scale during per tensor or layer quantization.

- shift: shift_type, shift_value

  It can be a scalar during quantization, where shift_type and shift_value are the data type and value of the shift during per tensor or layer quantization.

CTCGreedyDecoder

Performs greedy decoding on the logits given in the input tensor, including the score tensor and sequence length tensor. Where the sequence length's shape equals the batch size of input tensor X. Generally, 'mege_repeated' is set to 'true', and the operator merges repeated classes in output results. This means that if the maximum indices of consecutive logits are the same, only the first of these is emitted.

Inputs

- Input data tensor X, seq_len (the same shape as the batch size of input tensor X).

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = CTCGreedyDecoder'.

- merge_repeated: true, false

It defaults to ‘true’ if you are not sure of it, which means merging repeated classes in output results. That is, if consecutive logits’ maximum indices are the same, only the first one is emitted. For example, giving an input sequence X = [A, B, B, C, B, D, B], then returns Y = [A, B, C, B, D, B] when ‘merge_repeated’ is ‘true’, or Y = [A, B, B, C, B, D, B] when ‘merge_repeated’ is ‘false’.

Cast

Casts an input tensor (X) with a given data type and returns a new output tensor (Y). Currently, the supported data types are int8, uint8, int16, uint16, int32, float32, float16 and bfloat16.

Inputs

- Input data tensor X.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is ‘layer_type = Cast’.

- to_dtype

  The destination data type.

- scale: scale_type, scale_value

  It can be a scalar during quantization, where scale_type and scale_value are the data type and value of the scale during per tensor or layer quantization.

- shift: shift_type, shift_value

  It can be a scalar during quantization, where shift_type and shift_value are the data type and value of the shift during per tensor or layer quantization.

- clip_mode: SATURETION / TRUNCATION

  This parameter defines how the conversion behaves if an input value is out of range of the destination type. When clip_mode=SATUETION, the input data will saturate into the output type. When clip_mode=TRUNCATION, the input data will be truncated into the output bit width.

- ignore_scale_zp: true / false

  This parameter defines whether the paramters of scale and shift are required in int IR. When ignore_scale_zp=true, the scale and shift parameter will not be invoked in the calculation procedure.

Ceil

The ceil operation takes one input tensor (X) and produces one output tensor (Y). Where the ceil function ‘y = ceil(x)’ is applied to the tensor elementwise.

Inputs

- Input data tensor X.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = Ceil'.

- lut: lut_type, lut_shape, lut_offset, lut_size

  The *Look Up Table* (LUT) is an offline table created after quantization. Where, lut_type, lut_shape, lut_offset and lut_size are the data type, table shape, table offset address and table size. The following is a simple equation, lut_size = data_type * lut_shape. Where, data type = 1 (or 2, 4) if lut_type = int8 (or int16, int32).

Celu

Continuously Differentiable Exponential Linear Units: Perform the linear unit element-wise on the input tensor X using formula:

$$\max(0, x) + \min\left(0, alpha * \exp\left(\frac{x}{alpha}\right) - 1\right)$$

Inputs

- Input data tensor X.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = Activation'.

- method: CELU

  Method to perform the input tensor.

- alpha: float(default is 1.0)

  The alpha value in the celu formula which controls the shape of the unit. Only needed in float IR. The default value is 1.0.

- lut: lut_type, lut_shape, lut_offset, lut_size

  The *Look Up Table* (LUT) is an offline table created after quantization. Where, lut_type, lut_shape, lut_offset and lut_size are the data type, table shape, table offset address and table size. The following is a simple equation, lut_size = data_type * lut_shape. Where, data type = 1 (or 1, 2, 2, 4, 4) if lut_type = int8 (or uint8, int16, uint16, int32, uint32).

ChannelShuffle

Shuffles the input tensor along its channel dimension according to the group parameter. That is, divide the channels into 'group' groups and rearrange them, then output the results.

Inputs

- Input data tensor X.

Outputs

- Output tensor Y[i].

Attributes

- layer_type

The operation type of a layer. Here is ‘layer_type = ChannelShuffle’.

- group

  Means the group number of the input tensor. Along the input channel dimension, the output is calculated using this formula:

  output_channel[k * group + g] = input_channel[g * group_size + k]

  Where, group_size = input_channels / group.

- splits

  Means the number of output tensors, that is, how many numbers of output tensors will be split from the inputs after channel shuffle. For example, ‘splits = 1’ means one output tenor.

Clip

Limits the given input tensor within an interval. The interval is specified by the inputs ‘clip_min’ and ‘clip_max’.

Inputs

- Input data tensor X.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is ‘layer_type = Activation’.

- method: CLIP

  Method to perform the input tensor.

- clip_min

  Minimum value, under which the element is replaced by min. It must be a scalar.

- clip_max

  Maximum value, above which the element is replaced by max. It must be a scalar.

- scale: scale_type, scale_value

  It can be a scalar during quantization, where scale_type (or scale_value) is the data type (or value) of the scale during per tensor or layer quantization.

- shift: shift_type, shift_value

  It can be a scalar during quantization, where shift_type (or shift_value) is the data type (or value) of the shift during per tensor or layer quantization.

Compress

Selects slices from an input tensor along a given axis where the condition evaluates to True for each axis index. If the axis is not provided, the input is flattened before elements are selected. The compress operator behaves like numpy.compress.

Inputs

- Input data tensor X1, X2.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is ‘layer_type = Compress’.

- axis

  A scalar indicates the axis along which to take slices. If not specified, the input is flattened before elements are selected. A negative value means counting dimensions from the back. The accepted range is [-r, r-1] where r = rank(input).

Concat

This operator concatenates a list of tensors into a single tensor. All input tensors must have the same shape, except for the dimension size of the axis to concatenate on. The input tensors are more than one.

Inputs

- Input data tensor X1, data tensor X2, …

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is ‘layer_type = Concat’.

- axis

  A scalar means which axis to concatenate on. A negative value means counting dimensions from the back. The accepted range is [-r, r-1], where r = rank(input data tensor).

- scale: scale_type, scale_value

  Optional quantization parameter. The scale is a 1-D tensor and in order of input tensors. Where, scale_type and scale_value are the data type and value of the corresponding input tensor scale during quantization. That is, the scale_type and scale_value will be in order of [input0, input1, input2, …]. Generally, the scale_type has the int8, uint8, and int16 options.

- shift: shift_type, shift_value

  Optional quantization parameter. Where, shift_type (or shift_value) is the data type (or value) of the shift. The shift_type and shift_value are in order of [input0, input1, input2, …]. Generally, the shift_type has the int8 (or uint8, int16) option.

Constant

Creates a constant tensor as a given format. Generally, the data type is inferred from the type of value.

Inputs

- Empty (that is, no input initial data tensor X).

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = Constant'.

- weights: weights_type, weights_shape, weights_offset, weights_size

  Constant tensor description parameters, where weights_type, weights_offset, weights_size, and weights_shape mean the constant data format, address offset, total data size (counted in bytes) and output shape of the constant layer, respectively.

ConvTranspose2D

ConvTranspose2D is really the transpose of convolution2D. Performs a filter on the input tensor (X) and produces the output tensor (Y).

Inputs

- Input data tensor X.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = ConvTranspose'.

- kernel: kernel_x, kernel_y

  'kernel_x' ('kernel_y') is the kernel size along the width (height) axis. If not present, it should be inferred from inputs 'weights_ size' and 'weight_shape'.

- stride: stride_x, stride_y

  Stride along 'batch', 'height', 'width' and 'channel' axis. Where, 'stride_x' ('stride_y') stride along the width (height) axis is in the NHWC data format.

- pad: pad_bottom, pad_left, pad_right, pad_top

  Padding for the beginning and ending along spatial axis, it can take any value greater than or equal to 0. The value represents the number of pixels added to the beginning and end part of the corresponding axis. The 'pad' format should be as follows ('pad_bottom', 'pad_left', 'pad_right', 'pad_top'), where it means the pad pixels of the data cube (that is, bottom and top along the 'height' axis, left and right along the 'width' axis). If not sure, use a simple way to calculate paddings through the following equation:

  First, (output_shape, input_shape) = (input_shape, output_shape) if transpose convolution

  Then, pad = (output_shape - 1) * stride + ((kernel_size -1) * dilation +1] - input_shape

  pad_x = padding //2, pad_y = padding - pad_x. Where, pad_x and pad_y are along with the different framework and explicitly specified strategy.

- dilation: dilation_x, dilation_y

  The dilation value along the spatial axis of the filter. If not present, the dilation default value is 1 along each spatial axis. Where, dilation_x (dilation_y) means the width (height) axis of the filter in the NHWC data format.

- weights: weights_type, weights_offset, weights_size, weights_shape

  The weight tensor that will be used in the convolutions. Where, weights_type (or weights_offset, weights_size, weights_shape) means the weights data format (or weights address offset, total weights size which counted in bytes, and weights shape of the current layer). Generally, the 'weights' are stacked in order of [out_channel, kernel_y, kernel_x, input_channel] (that is, weights_shape = [out_channel, kernel_y, kernel_x, input_channel]).

- biases: biases_type, biases_offset, biases_size, biases_shape

  Bias to be added to the convolution. Where, biases_type, biases_offset, biases_size and biases_shape mean the biases data format, biases address offset, total biases size (counted in bytes) and biases shape of the current layer. Generally, biases_shape equals the number of output channels.

- with_activation: NONE, CLIP, RELU, RELU6, LEAKYRELU, PRELU

  Means whether to append an activation fusion operation to the current layer (such as Convolution, Fully Connected, Elementwise). 'None' means no this fusion.

  - Clip: clip_min, clip_max
    Where, 'clip_min' (or clip_max) means the minimum (or maximum) saturation threshold during the clip operation to produce output tensor after convolution.
  - LeakyRelu: negative_slope_type, negative_slope_value, negative_slope_scale, negative_slope_shift (a scalar)
    LeakyRelu takes input data (Tensor) and an argument alpha, then produces one output data (Tensor) where the function f(x) = alpha * x for x <0, f(x) = x for x >=0, is applied to the data tensor elementwise, where 'alpha' is the slope parameter. That is, 'negative_slope_type' and 'negative_slope_value' mean the negative slope coefficient data type (such as a uint8 data type) and value, which are all a scalar.
    Besides, 'negative_slope_scale' and 'negative_slope_shift' mean the parameter during quantization, which are all a scalar as well.
  - PRelu: negative_slope_type, negative_slope_shape, negative_slope_offset, negative_slope_size, negative_slope_scale, negative_slope_shift
    PRelu takes input data (Tensor) and slope tensor as input, then produces one output data (Tensor) where the function f(x) = slope * x for x < 0, f(x) = x for x >= 0, is applied to the data tensor elementwise. This operator supports unidirectional broadcasting (tensor slope should be unidirectional broadcastable to the input tensor). Where, negative_slope_type (or negative_slope_shape, negative_slope_offset, negative_slope_size) represents the data type and shape, offset address in memory, and total data size of negative coefficient. Similarly, negative_slope_scale and negative_slope_shift mean the parameter during quantization as well. Note that the shape of this 'negative_slope_shape' can be smaller than input tensor (X), and if so, its shape must be unidirectional broadcastable to input tensor (X). Generally, negative_slop_shape = [out_channel].

- num_output

  Channels of the output tensor.

- group

  Number of groups that input channels and output channels are divided into.

- scales: scale_type, scale_value

  It can be a scalar or a 1-D tensor, which means a per-tensor/layer (by default) or per output channel (need extended parameters in the future) quantization. If it is a 1-D tensor, its number of elements should be equal to the number of output channels, where scale_value, scale_type, scale_shape, scale_offset, and scale_size are the value, data type and numbers, offset address and data size of the scale, coefficient during quantization. If it is a scalar, scale_value and scale_type mean the value and data type of scale parameter during per-tensor/layer quantization.

  Besides, scale_type has the int8, uint8 and int16 options. Where, the relationship is scale_size= data_type *scale_shape. Here, data type = 1 (or 2, 4) if shift_type = int8 (or int16, int32).

- shifts: shift_type, shift_value

  It can be a scalar or a 1-D tensor, which means a per-tensor/layer (by default) or per output channel (need extended parameters in the future) quantization. If it is a 1-D tensor, its number of elements should be equal to the number of output channels, where shift_value, shift_type, shift_shape, shift_offset, and shift_size are the value, data type and numbers, offset address and data size of the shift, coefficient during quantization. If it is a scalar, shift_value and shift_type mean the value and data type of the shift parameter during per-tensor/layer quantization.

Besides, shift_type is an int8 format. Where, the relationship is shift_size= data_type * shift_shape. Here, data type = 1 (or 2, 4) if shift_type = int8 (or int16, int32).

ConvTranspose3D

ConvTranspose3D applies a 3D transposed convolution operator over an input image composed of several input planes. That is, it performs a 3D-filter on the input tensor (X) and produces the output tensor (Y). Generally, the output shape is calculated through the following equation:

output_shape[i] = stride[i] * (input_size[i] - 1) + output_padding[i] + ((kernel_size[i] -1) * dilation[i] + 1) - pad_i_start - pad_i_end

Where output_padding is the padding added to the output.

Inputs

- Input data tensor X.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is ‘layer_type = ConvTranspose 3D’.

- kernel: kernel_x, kernel_y, kernel_z

  ‘kernel_x’ (‘kernel_y’, ‘kernel_z’) is the kernel size along the width (height, depth) dimension. If not present, it should be inferred from inputs ‘weights_ size’ and ‘weight_shape’.

- stride: stride_x, stride_y, kernel_z

  ‘stride_x’ (‘stride_y’, ‘stride_z’) is along the width (height, depth) dimension under the NDHWC data format.

- pad: pad_x_begin, pad_x_end, pad_y_begin, pad_y_end, pad_z_begin, pad_z_end

  Padding for the beginning and ending along spatial axis, it can take any value greater than or equal to 0. The value represents the number of pixels added to the beginning and end part of the corresponding axis. If not sure, use a simple way to calculate the paddings through the following equation:

  First, (output_shape, input_shape) = (input_shape, output_shape) if transpose convolution

  Then, pad[i] = (output_shape[i] - 1) * stride[i] + ((kernel_size[i] -1) * dilation[i] +1] - input_shape[i]

  pad_x = pad //2, pad_y = padding - pad_x. Where, pad_x and pad_y are along with the different framework and explicitly specified strategy.

- dilation: dilation_x, dilation_y, dilation_z

  The dilation value along the spatial axis of the filter. If not present, the dilation default value is 1 along each spatial axis. Where, ‘dilation_x’ (‘dilation_y’ or ‘dilation_z’) means the ‘width’ (‘height’, ‘depth’) axis of the filter in the NDHWC data format.

- weights: weights_type, weights_offset, weights_size, weights_shape

  The weight tensor that will be used in the convolutions. Where, weights_type (or weights_offset, weights_size, weights_shape) means the weights data format (or weights address offset, total weights size which counted in bytes, and weights shape of the current layer). Generally, the 'weights' are stacked in order of [out_channel, kernel_y, kernel_x, kernel_z, input_channel] (that is, weights_shape = [out_channel, kernel_y, kernel_x, kernel_z, input_channel]).

- biases: biases_type, biases_offset, biases_size, biases_shape

  Bias to be added to the convolution. Where, biases_type, biases_offset, biases_size and biases_shape mean the biases data format, biases address offset, total biases size (counted in bytes) and biases shape of the current layer. Generally, biases_shape equals the number of output channels.

- with_activation: NONE, CLIP, RELU, RELU6, LEAKYRELU, PRELU

  Means whether to append an activation fusion operation to the current layer (such as Convolution, FullyConnected, Elementwise). 'None' means no this fusion.

  - Clip: clip_min, clip_max
    Where, 'clip_min' (or clip_max) means the minimum (or maximum) saturation threshold during the clip operation to produce output tensor after convolution.
  - LeakyRelu: negative_slope_type, negative_slope_value, negative_slope_scale, negative_slope_shift (a scalar)
    LeakyRelu takes input data (Tensor) and an argument alpha, then produces one output data (Tensor) where the function f(x) = alpha * x for x <0, f(x) = x for x >=0, is applied to the data tensor elementwise, where 'alpha' is the slope parameter. That is, 'negative_slope_type' and 'negative_slope_value' mean the negative slope coefficient data type (such as a uint8 data type) and value, which are all a scalar.
    Besides, 'negative_slope_scale' and 'negative_slope_shift' mean the parameter during quantization, which are all a scalar as well.
  - PRelu: negative_slope_type, negative_slope_shape, negative_slope_offset, negative_slope_size, negative_slope_scale, negative_slope_shift
    PRelu takes input data (Tensor) and slope tensor as input, then produces one output data (Tensor) where the function f(x) = slope * x for x < 0, f(x) = x for x >= 0, is applied to the data tensor elementwise. This operator supports unidirectional broadcasting (tensor slope should be unidirectional broadcastable to the input tensor). Where, negative_slope_type (or negative_slope_shape, negative_slope_offset, negative_slope_size) represents the data type and shape, offset address in memory, and total data size of negative coefficient. Similarly, negative_slope_scale and negative_slope_shift mean the parameter during quantization as well. Note that the shape of this 'negative_slope_shape' can be smaller than input tensor (X), and if so, its shape must be unidirectional broadcastable to input tensor (X). Generally, negative_slop_shape = [out_channel].

- num_output

  Channels of the output tensor.

- group

  Number of groups that input channels and output channels are divided into.

- scales: scale_type, scale_value

  It can be a scalar or a 1-D tensor, which means a per-tensor/layer (by default) or per output channel (need extended parameters in the future) quantization. If it is a 1-D tensor, its number of elements should be equal to the number of output channels, where scale_value, scale_type, scale_shape, scale_offset, and scale_size are the value, data type and numbers, offset address and data size of the scale, coefficient during quantization. If it is a scalar, scale_value and scale_type mean the value and data type of scale parameter during per-tensor/layer quantization.

  Besides, scale_type has the int8, uint8 and int16 options. Where, the relationship is scale_size= data_type *scale_shape. Here, data type = 1 (or 2, 4) if shift_type = int8 (or int16, int32).

- shifts: shift_type, shift_value

  It can be a scalar or a 1-D tensor, which means a per-tensor/layer (by default) or per output channel (need extended parameters in the future) quantization. If it is a 1-D tensor, its number of elements should be equal to the number of output channels, where shift_value, shift_type, shift_shape, shift_offset, and shift_size are the value, data type and numbers, offset address and data size of the shift, coefficient during quantization. If it is a scalar, shift_value and shift_type mean the value and data type of the shift parameter during per-tensor/layer quantization.

Besides, shift_type is an int8 format. Where, the relationship is shift_size= data_type * shift_shape. Here, data type = 1 (or 2, 4) if shift_type = int8 (or int16, int32).

Convolution2D

- General convolution

  Performs a filter on the input tensor (X) and produces the output tensor (Y). If bias is necessary, a bias vector will be added to the output tensor. Similarly, if the activation function is necessary, it is applied to the output tensor as well.

- Fast algorithm with Winograd

  See *Andrew Lavin, Scott Gray. Fast Algorithm for Convolutional Neural Networks. arXiv:1509.090308v2 [cs.NE] 10 Nov 2015*.

  Winograd algorithm is a minimal filtering algorithm for computing 'm' outputs with a r-tap FIR filter which is called *F(m, r)* and requires '*μ(F(m, r)) = m + r - 1*' multiplications. Also, nesting minimal 1-D algorithms *F(m, r)* and *F(n,s)* is to form minimal 2-D algorithms for computing 'm x n' outputs with 'r x s' filter, which is called *F(m x n, r x s)* and requires 'μ(F(m x n, r x s)) = μ(F(m, r)) μ(F(n, s)) = (m + r - 1)(n + s -1)' multiplications. Then, fast filtering algorithms can be written in matrix form as:

  $$Y = A^T[(Gg) \otimes (B^Td)]$$

  Where, $\otimes$ indicates the element-wise multiplication, and G, A^T and B^T are filter, data and inverse transforms. For F(2, 3), the matrices are:

  B^T = [[1, 0, -1, 0], [0, 1, 1, 0], [0, -1, 1, 0], [0, 1, 0, -1]]

  G = [[1, 0, 0], [½, ½, ½], [½, -½, ½], [0, 0, 1]]

  A^T = [[1, 1, 1, 0], [0, 1, -1, -1]]

  g = $[g0, g1, g2]^T$

  d = $[d0, d1, d2, d3]^T$

  Similarly, a minimal 1-D algorithm F(m, r) is nested with itself to obtain a minimal 2-D algorithm, and F(m x m, r x r) is in the similar way:

  $$Y = A^T[(Gg\,G^T) \otimes (B^TdB)]\,A$$

  Where, $\otimes$ indicates the element-wise multiplication. 'g' is a 'r x r' filter and 'd' is a (m + r -1) x (m + r -1) image tile. Also, the nesting technique can be generalized for non-square filters and outputs, such as F(m x n, r x s), by nesting an algorithm for F(m, r) with an algorithm for F(n, s).

  Generally, 'P = [H / m] [W / m]' tiles per channel under F(m x m, r x r). The input tile size is 'tile_size = m + r -1', and neighboring tiles overlap by 'r - 1', then the padding during the Winograd algorithm can be given by:

  winograd_pad = (output_shape - 1) * m + (m + r -1) - input_shape - input_padding

  Where, 'winograd_pad' means the padding during transforming to the Winograd algorithm. 'input_padding' means the input beginning or end padding to the input tensor.

  The magnitude of the transform matrix elements also increases with the increasing tile size and might sacrifice some numeric accuracy dropping during the filter computation. Therefore, whether to apply the convolutional neural networks by using the Winograd algorithm depends on the performance improvement (including the hardware resource). That is, a smaller feature map and bigger channel with small filter size are potential for transforming.

Inputs

- Input data tensor X.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = Convolution'.

- with_winograd: true, false

  An algorithm optimization parameter in an int IR. 'true' means computing convolution with Winograd algorithm mode. 'false' means computing the convolution as general matrix multiplication.

- Fmnrs

  A list of integers in an int IR to indicate a 1-D or 2-D algorithm for F(m, r) or F(m x n, r x s). For example, [2, 3] means the 1-D algorithm F(2, 3). [2, 2, 3, 3] means the 2-D algorithm F(2x2, 3x3).

- kernel: kernel_x, kernel_y

  'kernel_x' ('kernel_y') is the kernel size along the 'width' ('height') axis. If not present, it should be inferred from inputs 'weights_ size' and 'weight_shape'. If with_winograd = 'true' in an int IR, the transformation is 'kernel_x = m + r -1' and 'kernel_y = n +s -1' (only kernel_x is valid under 1-D algorithm).

- stride: stride_x, stride_y

  Stride along 'batch', 'height', 'width' and 'channel' axis. Generally, the 'stride_x' ('stride_y') stride is along the 'width' ('height') axis. If with_winograd = 'true' in an int IR, the transformation is 'stride_x = m' and 'stride_y = n' (only stride_x is valid under 1-D algorithm).

- pad: pad_bottom, pad_top, pad_left, pad_right

  Padding for the beginning and ending along spatial axis, it can take any value greater than or equal to 0. The value represents the number of pixels added to the beginning and end part of the corresponding axis. The 'pad' parameter format should be as follows ('pad_bottom', 'pad_top', 'pad_left', 'pad_right'), where it means the pad pixels of the data cube (that is, bottom and top along the 'height' axis; left and right along the 'width' axis). If not present, compute and output the same shape as the input tensor.

- dilation: dilation_x, dilation_y

  The dilation value along the spatial axis of the filter. If not present, the dilation default value is 1 along each spatial axis. Where dilation_x (dilation_y) means the width (height) axis of the filter in the NHWC data format.

- weights: weights_type, weights_offset, weights_size, weights_shape

  The weight tensor that will be used in the convolutions. Where, weights_type (or weights_offset, weights_size, weights_shape) means the weights data format (or weights address offset, total weights size which counted in bytes, and weights shape of the current layer). Generally, the 'weights' are stacked in order of [out_channel, kernel_y, kernel_x, input_channel] (that is, the weights_shape = [out_channel, kernel_y, kernel_x, input_channel]). Especially, if with_winograd = 'true' in an int IR, then weights_shape = [output_channel] + $((Gg)^T)$.shape + [input_channel] under 1-D algorithm, and weights_shape = [output_channel] + $((GgG^T)^T)$.shape + [input_channel] under 2-D algorithm.

- biases: biases_type, biases_offset, biases_size, biases_shape

  Bias to be added to the convolution. Where, biases_type, biases_offset, biases_size and biases_shape mean the biases data format, biases address offset, total biases size (counted in bytes) and biases shape of the current layer. Generally, biases_shape equals the number of output channels.

- with_activation: NONE, CLIP, RELU, RELU6, LEAKYRELU, PRELU

  Means whether to append an activation fusion operation to the current layer (such as Convolution, Fully Connected, Elementwise). 'None' means no this fusion.

  - Clip: clip_min, clip_max
    Where 'clip_min' (or clip_max) means the minimum (or maximum) saturation threshold during the clip operation to produce the output tensor after convolution.
  - LeakyRelu: negative_slope_type, negative_slope_value, negative_slope_scale, negative_slope_shift
    LeakyRelu takes input data (Tensor) and an argument alpha, then produces one output data (Tensor) where the function f(x) = alpha * x for x <0, f(x) = x for x >=0, is applied to the data tensor elementwise, where 'alpha' is the slope parameter. That is, 'negative_slope_type' and 'negative_slope_value' mean the negative slope coefficient data type (such as a uint8 data type) and value, which are all a scalar.
    Besides, 'negative_slope_scale' and 'negative_slope_shift' mean the parameter during quantization, which are all a scalar as well.
  - PRelu: negative_slope_type, negative_slope_shape, negative_slope_offset, negative_slope_size, negative_slope_scale, negative_slope_shift
    PRelu takes input data (Tensor) and slope tensor as input, then produces one output data (Tensor) where the function f(x) = slope * x for x < 0, f(x) = x for x >= 0, is applied to the data tensor elementwise. This operator supports unidirectional broadcasting (tensor slope should be unidirectional broadcastable to the input tensor). Where negative_slope_type (or negative_slope_shape, negative_slope_offset, negative_slope_size) represents the data type and shape, offset address in memory, and total data size of negative coefficient. Similarly, negative_slope_scale and negative_slope_shift mean the parameter during quantization as well. Note that the shape of this 'negative_slope_shape' can be smaller than input tensor (X), and if so, its shape must be unidirectional broadcastable to input tensor (X). Generally, negative_slop_shape = [out_channel].
- num_output

  Channels of the output tensor.

- group

  Number of groups that input channels and output channels are divided into.

- scale: scale_type, scale_value

  It can be a scalar or a 1-D tensor, which means a per-tensor/layer (by default) or per output channel (need extended parameters in the future) quantization. If it is a 1-D tensor, its number of elements should be equal to the number of output channels, where scale_value, scale_type, scale_shape, scale_offset, and scale_size are the value, data type and numbers, offset address and data size of the scale, coefficient during quantization. If it is a scalar, scale_value and scale_type mean the value and data type of the scale parameter during per-tensor/layer quantization.

  Besides, scale_type has the int8, uint8, and int16 options. Where, the relationship is scale_size= data_type * scale_shape. Here, data type = 1 (or 2, 4) if shift_type = int8 (or int16, int32).

- shift: shift_type, shift_value

  It can be a scalar or a 1-D tensor, which means a per-tensor/layer (by default) or per output channel (need extended parameters in the future) quantization. If it is a 1-D tensor, its number of elements should be equal to the number of output channels, where shift_value, shift_type, shift_shape, shift_offset, and shift_size are the value, data type and numbers, offset address and data size of the shift, coefficient during quantization. If it is a scalar, shift_value and shift_type mean the value and data type of the shift parameter during per-tensor/layer quantization.

  Besides, shift_type is an int8 format. Where, the relationship is shift_size= data_type * shift_shape. Here, data type = 1 (or 2, 4) if shift_type = int8 (or int16, int32).

Convolution3D

Performs a filter on the input tensor (X) and produces the output tensor (Y). If bias is necessary, a bias vector will be added to the output tensor. Similarly, if the activation function is necessary, it is applied to the output tensor as well. By default, the input or output data format is NDHWC.

Inputs

- Input data tensor X.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = Convolution3D'.

- kernel: kernel_x, kernel_y, kernel_z

  'kernel_x' ('kernel_y', 'kernel_z') is the kernel size along the 'width' ('height', 'depth') dimension. If not present, it should be inferred from inputs 'weights_ size' and 'weight_shape'.

- stride: stride_x, stride_y, stride_z

  Stride along the 'batch', 'depth', 'height', 'width' and 'channel' axis. Generally, the 'stride_x' ('stride_y', 'stride_z') stride is along the 'width' ('height', 'depth') dimension.

- pad: pad_x_begin, pad_x_end, pad_y_begin, pad_y_end, pad_z_begin, pad_z_end

  Padding for the beginning and ending along the spatial axis, it can take any value greater than or equal to 0. The value represents the number of pixels added to the beginning and end part of the corresponding axis. The 'pad' parameter format should be as follows:

  ('pad_x_begin', 'pad_x_end', 'pad_y_begin', 'pad_y_end', 'pad_z_begin', 'pad_z_end')

  Where it means the pad pixels of the data cube (that is, 'x', 'y' or 'z' along the 'width', 'height' or 'depth' dimension respectively). If not present, compute and output the same shape as the input tensor.

- dilation: dilation_x, dilation_y, dialation_z

  The dilation value along the spatial axis of the filter. If not present, the dilation default value is 1 along each spatial axis. Where dilation_x (dilation_y, dilation_z) means the width (height, depth) axis of the filter under the default NDHWC data format.

- weights: weights_type, weights_offset, weights_size, weights_shape

  The weight tensor that will be used in the convolutions. Where, weights_type (or weights_offset, weights_size, and weights_shape) means the weights data format (or weights address offset, total weights size which counted in bytes, and weights shape of the current layer). Generally, the 'weights' are stacked in order of [out_channel, kernel_y, kernel_x, kernel_z, input_channel] (that is, weights_shape = [out_channel, kernel_y, kernel_x, kernel_z, input_channel]).

- biases: biases_type, biases_offset, biases_size, biases_shape

  Bias to be added to the convolution. Where, biases_type, biases_offset, biases_size and biases_shape mean the biases data format, biases address offset, total biases size (counted in bytes) and biases shape of the current layer. Generally, biases_shape equals the number of the output channel.

- with_activation: NONE, CLIP, RELU, RELU6, LEAKYRELU

  Means whether to append an activation fusion operation to the current layer (such as Convolution, FullyConnected, Elementwise). 'None' means no this fusion.

  - Clip: clip_min, clip_max
    Where 'clip_min' (or clip_max) means the minimum (or maximum) saturation threshold during the clip operation to produce the output tensor after convolution.
  - LeakyRelu: negative_slope_type, negative_slope_value, negative_slope_scale, negative_slope_shift
    LeakyRelu takes input data (Tensor) and an argument alpha, then produces one output data (Tensor) where the function f(x) = alpha * x for x <0, f(x) = x for x >=0, is applied to the data tensor elementwise, where 'alpha' is the slope parameter. That is, 'negative_slope_type' and 'negative_slope_value' mean the negative slope coefficient data type (such as a uint8 data type) and value, which are all a scalar.
    Besides, 'negative_slope_scale' and 'negative_slope_shift' mean the parameter during quantization, which are all a scalar as well.

- num_output

  Channels of the output tensor.

- group

  Number of groups that input channels and output channels are divided into.

- scale: scale_type, scale_value

  It can be a scalar or a 1-D tensor, which means a per-tensor/layer (by default) or per output channel (need extended parameters in the future) quantization. If it is a 1-D tensor, its number of elements should be equal to the number of output channels, where scale_value, scale_type, scale_shape, scale_offset, and scale_size are the value, data type and numbers, offset address and data size of the scale, coefficient during quantization. If it is a scalar, scale_value and scale_type mean the value and data type of the scale parameter during per-tensor/layer quantization.

  Besides, scale_type has the int8, uint8, and int16 options. Where, the relationship is scale_size= data_type * scale_shape. Here, data type = 1 (or 2, 4) if shift_type = int8 (or int16, int32).

- shift: shift_type, shift_value

  It can be a scalar or a 1-D tensor, which means a per-tensor/layer (by default) or per output channel (need extended parameters in the future) quantization. If it is a 1-D tensor, its number of elements should be equal to the number of output channels, where shift_value, shift_type, shift_shape, shift_offset, and shift_size are the value, data type and numbers, offset address and data size of the shift, coefficient during quantization. If it is a scalar, shift_value and shift_type mean the value and data type of the shift parameter during per-tensor/layer quantization.

  Besides, shift_type is an int8 format. Where, the relationship is shift_size= data_type * shift_shape. Here, data type = 1 (or 2, 4) if shift_type = int8 (or int16, int32).

Cosh

Calculates the hyperbolic cosine of the given input tensor element-wise.

Inputs

- Input data tensor X.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = Cosh'.

- lut: lut_type, lut_shape, lut_offset, lut_size

The *Look Up Table* (LUT) is an offline table created after quantization. Where, lut_type, lut_shape, lut_offset and lut_size are the data type, table shape, table offset address and table size. The following is a simple equation, lut_size = data_type * lut_shape. Where, data type = 1 (or 2, 4) if lut_type = int8 (or int16, int32).

Cosine

Calculates the cosine of the given input tensor by element-wise.

Inputs

- Input data tensor X.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = Cosine'.

- lut: lut_type, lut_shape, lut_offset, lut_size

  The *Look Up Table* (LUT) is an offline table created after quantization. Where, lut_type, lut_shape, lut_offset and lut_size are the data type, table shape, table offset address and table size. The following is a simple equation:

  'lut_size = data_type * lut_shape'. where, data type = 1 (or 2, 4) if lut_type = int8 (or int16, int32).

Count

Applies to input numeric tensor X and returns a 2-D histogram to count the number of entries in the numeric tensor that fell into every bin. Generally, the bins are equal width and determined by the parameters 'min', 'max' and 'nbins'. Note that elements in input numeric tensor (X) lower than 'min' and higher than 'max' are ignored. By default, the numeric data is counted over batch dimension. For example, numeric_X = [[-1, 1, 0, 1, 2, 3, 12]], min = 0, max = 4, nbins = 5 and 'discrete = true', then returns a 2-D tensor Y =[[1, 2, 1, 1, 0]].

Inputs

- Input numeric tensor X.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = Count'.

- min

  Lower end of the range (inclusive).

- max

  Upper end of the range (inclusive).

- nbins

  A scalar of integer to indicate the number of histogram bins. Generally, every bin is a left closed and right open interval.

- discrete: true, false

A statistic mode. ‘true’ means treating the bins as discrete values from minimum to maximum when ‘min’ and ‘max’ are integers. ‘false’ means treating the bins as an interval.

Crop

Crops the input tensor into the output tensor with the shape computing by the given beginning and end index (exclusive the ending indices) and the shape of the output tensor. For example, considering an input tensor X. shape = [1, 4, 5, 3], crops = [[0, 1], [1, 4], [2, 4], [0, -1]], and the result will return a tensor Y with shape [1, 3, 2, 2].

Inputs

- Input data tensor X.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is ‘layer_type = Crop’.

- crops

  A nested list means cropping from the input tensor (X), such as crops = [[crop0_begin, crop0_end], [crop1_begin, crop1_end], ..., [cropn_begin, cropn_end]], where [D, 0] means the effective beginning indices and [D, 1] means the effective ending indices (exclusive the ending indices) in the D dimension. Generally, a negative value means cropping backward.

CropAndResize

Extracts crops from the input image or feature map and resizes it by **bilinear** sampling or **nearest neighbor** sampling to a common output size specified by the ‘crop_size’ parameter.

Note that, in the following Inputs or Outputs sections:

- **X** means the input image or feature map with a shape of [batch, height, width, depth].
- **Boxes** with a shape of [num_boxes, 4]. The **i**-th row of this tensor specifies the coordinates of a box in the **Box_Indices**[i], which is specified in normalized coordinates [y1, x1, y2, x2]. A normalized coordinates value of ‘y’ is mapped to the image or feature map coordinate at ‘y * (height-1)’. That is, the coordinate in [0, 1] interval of normalized height is mapped to [0, height -1] in the input image or feature map coordinates.
- **Box_Indices** with a shape of [num_boxes] with int32 values in [0, batch). That is, the value of **Box_Indices**[i] specifies the input image or feature map that the **i**-th box refers to.
- **Y** means the output tensor with a shape [num_boxes, crop_height, crop_width, depth].

Inputs

- Input data tensor X, Boxes tensor, Box_Indices tensor.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is ‘layer_type = CropAndResize’.

- crop_size

A vector of two elements in order of [crop_height, crop_width]. Generally, the cropped image or feature map patches are resized to this specified size and the aspect ratio is not preserved as well. Besides, both 'crop_height' and 'crop_width' should be positive.

- method: BILINEAR, NEAREST

  The method to resize the input tensor.

- extrapolation_value

  An optional parameter, which means the value used for extrapolation. Generally, the value is 0.

- scale: scale_type, scale_value

  It can be a scalar during quantization, where scale_type and scale_value are the data type and value of the scale during per tensor or layer quantization.

- shift: shift_type, shift_value

  It can be a scalar during quantization, where shift_type and shift_value are the data type and value of the shift during per tensor or layer quantization.

- is_perf_mode: (default: true)

  It is a Boolean value to indicate whether the operator is running with high performance mode. When the value is true, the operator will run in high performance mode. When it is false, the performance will not be the highest. (Note: The high performance strongly depends on the hardware architecture, and some runtime data cannot release the mode. If the operator cannot run correctly, change is_perf_mode to false mode).

CumProd

Computes the cumulative product of tensor x along the axis. By default, the output tensor will keep the dimension as input tensor X.

Inputs

- Input data tensor X.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = Cumulate'.

- method: PROD

  The cumulate method.

- axis

  It indicates along which axis to reduce. The value cannot be null. For example, axis = [1], or [2], ... The accepted range is [-1, r-1] where r = rank(input data).

- exclusive: true / false

  By default, this operator performs an inclusive cumprod, which means that the first element of the input is identical to the first element of the output. By setting the "exclusive" to "true", an exclusive cumprod is performed, which means that the first element of the input is not identical to the first element of the output.

- reverse: true / false

By setting the “reverse” to “true”, the cumprod is performed in the opposite direction.

- scale: scale_type, scale_shape, scale_offset, scale_size

  The *Look Up Table* (LUT) is an offline table created after quantization. Where, scale_type, scale_shape, scale_offset and scale_size are the data type, table shape, table offset address and table size. The following is a simple equation, scale_size = data_type * scale_shape. Where, data type = 1 (or 1, 2, 2, 4, 4) if scale_type = int8 (or uint8, int16, uint16, int32, uint32).

- shift: shift_type, shift_shape, shift_offset, shift_size

  The *Look Up Table* (LUT) is an offline table created after quantization. Where, shift_type, shift_shape, shift_offset and shift_size are the data type, table shape, table offset address and table size. The following is a simple equation, shift_size = data_type * shift_shape. Where, data type = 4 and shift_type = int32.

CumSum

Computes the cumulative sum of tensor x along the axis. By default, the output tensor will keep the dimension as input tensor X.

Inputs

- Input data tensor X.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is ‘layer_type = Cumulate’.

- method: SUM

  The cumulate method.

- axis

  It indicates along which axis to reduce. The value cannot be null. For example, axis = [1], or [2], ... The accepted range is [-1, r-1] where r = rank(input data).

- exclusive: true / false

  By default, this operator performs an inclusive cumsum, which means that the first element of the input is identical to the first element of the output. By setting the “exclusive” to “true”, an exclusive cumsum is performed, which means that the first element of the input is not identical to the first element of the output.

- reverse: true / false

  By setting the “reverse” to “true”, the cumsum is performed in the opposite direction.

- scale: scale_type, scale_value

  It can be a vector during quantization, where scale_type and scale_value are the data type and value of the scale during per tensor or layer quantization.

- shift: shift_type, shift_value

  It can be a scalar during quantization, where shift_type and shift_value are the data type and value of the shift during per tensor or layer quantization.

DataStride

Performs an input data tensor (X) to produce an output tensor (Y) with a stride combination, which means taking kernel size numbers of data after every stride operation and gathering all these data along each axis respectively. For example, given an input tensor with the shape as [1, 8, 8, 3] and kernel_x (kernel_y) =3, stride_x (stride_y) = 3, then it returns an output tensor Y with the shape as [1, 6, 6, 3].

Inputs

- Input data tensor X.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = DataStride'.

- kernel: kernel_x, kernel_y

  'kernel_x' ('kernel_y') is the kernel size along the 'width' ('height') axis.

- stride: stride_x, stride_y

  The 'stride_x' ('stride_y') stride along the 'width' ('height') axis.

DecodeBox

Boxes with scores less than Score_Threshold are dropped or discarded in the last output. The box needs to be translated into bounding box.

Note that, in the following Inputs or Outputs sections:

- X1 means the scores of all the boxes per batch and per class, of which the shape is [batch_size, num_boxes, class_num + 1].
- X2 means the coordinates of the maximum output boxes per batch, of which the shape is [batch_size, num_boxes, 4].
- Y1 means the coordinates of the selected boxes per batch, of which the shape is [batch_size, max_output_size, 4].
- Y2 means the number of the selected boxes that belong to cluster class_id per batch, of which the shape is [batch_size, max_output_size].
- Y3 means the number of all the selected boxes that belong to the class per batch, of which the shape is [batch_size, 1].
- Y4 means the scores of all the selected boxes per batch, of which the shape is the same as Y2.
- Y5 means the class_ids of the selected boxes per batch in input tensor X4, of which the shape is the same as Y2 as well.

Inputs

- Input tensor X1, X2.

Outputs

- Output tensor Y1, Y2, Y3, Y4, Y5.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = DecodeBox'.

- class_num

Integer representing the number of classes for the input. It is a scalar.

- width

  The float IR parameters during normalized coordinate quantization, which indicate the width of the input feature map.

- height

  The float IR parameters during normalized coordinate quantization, which indicate the height of the input feature map.

- score_threshold_uint8

  The threshold for deciding when to remove boxes based on the score. It is a scalar.

- box_shift

  It can be a scalar during quantization for bounding box.

- weight: weight_type, weight_shape, weight_offset, weight_size

  The look up table (weight) is an offline table created after quantization for bounding box method. Where, weight_type, weight_shape, weight_offset, and weight_size are the data type, table shape, table offset address, and table size. The following is a simple equation, weight_size = data_type * weight_shape. Where, data type = 1 (or 2, 4) if weight_type = int8 (or int16, int32).

DepthToSpace

DepthToSpace permutes data from depth into blocks of spatial data. This is the reverse transformation of SpaceToDepth. More specially, this operator outputs a copy of the input tensor where values from the depth dimension are moved in spatial blocks to the 'height' and 'width' dimensions. In the DCR mode, elements along the depth dimension from the input tensor are rearranged in the following order: 'height', 'width' and 'channel'. The input will be reshaped into [N, H , W , block_size_y, block_size_x, C/(block_size_y * block_size_x)], and transposed to [N, H, block_size_y, W, block_size_x, C/(block_size_y * block_size_x)]. The output shape will be reshaped to [N, H * block_size_y, W * block_size_x, C/(block_size_y * block_size_x)]. In the CRD mode, elements along the depth dimension from the input tensor are rearranged in the following order: 'height', 'width' and 'channel'. The input will be reshaped into [N, H , W , C/(block_size_y * block_size_x), block_size_y, block_size_x], and transposed to [N, H, block_size_y, W, block_size_x, C/(block_size_y * block_size_x)]. The output shape will be reshaped to [N, H * block_size_y, W * block_size_x, C/(block_size_y * block_size_x)]. The output shape will be [N, H * block_size_y, W * block_size_x, C/(block_size_y * block_size_x)].

Inputs

- Input data tensor X.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = DepthToSpace'.

- block_size: block_size_x, block_size_y

  Blocks of [block_size_y, block_size_x] are moved from the channel dimension of the input tensor.

- mode: CRD/DCR

  Indicates in which mode the elements along the depth dimension from the input tensor are rearranged. DCR is the default mode.

DepthwiseConvolution

Performs a different filter to each channel of input tensor (X) and then concatenates the results of convolution together. Generally, the channel number of the output tensor will be out_channel = in_channels * multiplier.

Inputs

- Input data tensor X.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is ‘layer_type = DepthwiseConv’.

- kernel: kernel_x, kernel_y

  ‘kernel_x’ (‘kernel_y’) is the kernel size along the width (height) axis. If not present, it should be inferred from input ‘weights_ size’ and ‘weight_shape’.

- stride: stride_x, stride_y

  Stride along ‘batch’, ‘height’, ‘width’ and ‘channel’ axis. Generally, the ‘stride_x’ (‘stride_y’) stride is along the ‘width’ (‘height’) axis.

- pad: pad_bottom, pad_left, pad_right, pad_top

  Padding for the beginning and ending along the spatial axis, it can take any value greater than or equal to 0. The value represents the number of pixels added to the beginning and end part of the corresponding axis. The ‘pad’ parameter format should be as follows (‘pad_bottom’, ‘pad_top’, ‘pad_left’, ‘pad_right’), where it means the pad pixels of the data cube (that is, bottom and top along the ‘height’ axis, left and right along the ‘width’ axis). If not present, compute and output the same shape as the input tensor.

- dilation: dilation_x, dilation_y

  The dilation value along the spatial axis of the filter. If not present, the dilation default value is 1 along each spatial axis. Where dilation_x (dilation_y) means the ‘height’ (‘width’) axis of the filter in the NHWC data format.

- weights: weights_type, weights_offset, weights_size, weights_shape

  The weight tensor that will be used in the convolutions. Where, weights_type (or weights_offset, weights_size, weights_shape) means the weights data format (or weights address offset, total weights size which counted in bytes, and weights shape of the current layer). Generally, the ‘weights’ are stacked in order of [out_channel, kernel_y, kernel_x]. That is, weights_shape = [out_channel, kernel_y, kernel_x, 1], where ‘out_channel’ will be composed in order of [in_channel, multiplier].

- biases: biases_type, biases_offset, biases_size, biases_shape

  Bias to be added to the convolution. Where, biases_type, biases_offset, biases_size and biases_shape) mean the biases data format, biases address offset, total biases size (counted in bytes) and biases shape of the current layer. Generally, biases_shape equals the number of output channels.

- with_activation: NONE, CLIP, RELU, RELU6, LEAKYRELU, PRELU

  Means whether to append an activation fusion operation to the current layer (such as Convolution, Fully Connected, Elementwise). ‘None’ means no this fusion.

  - Clip: clip_min, clip_max

Where, clip_min (or clip_max) means the minimum (or maximum) saturation threshold during the clip operation to produce output tensor after convolution.

- LeakyRelu: negative_slope_type, negative_slope_value, negative_slope_scale, negative_slope_shift (a scalar)
  LeakyRelu takes input data (Tensor) and an argument alpha, then produces one output data (Tensor) where the function f(x) = alpha * x for x <0, f(x) = x for x >=0, is applied to the data tensor elementwise, where 'alpha' is the slope parameter. That is, 'negative_slope_type' and 'negative_slope_value' mean the negative slope coefficient data type (such as a uint8 data type) and value, which are all a scalar.
  Besides, 'negative_slope_scale' and 'negative_slope_shift' mean the parameter during quantization, which are all a scalar as well.
- PRelu: negative_slope_type, negative_slope_shape, negative_slope_offset, negative_slope_size, negative_slope_scale, negative_slope_shift
  PRelu takes input data (Tensor) and slope tensor as input, then produces one output data (Tensor) where the function f(x) = slope * x for x < 0, f(x) = x for x >= 0, is applied to the data tensor elementwise. This operator supports unidirectional broadcasting (tensor slope should be unidirectional broadcastable to the input tensor). Where negative_slope_type (or negative_slope_shape, negative_slope_offset, negative_slope_size) represents the data type and shape, offset address in memory and total data size of negative coefficient. Similarly, negative_slope_scale and negative_slope_shift mean the parameter during quantization as well. Note that the shape of this 'negative_slope_shape' can be smaller than input tensor (X), and if so, its shape must be unidirectional broadcastable to input tensor (X). Generally, negative_slop_shape = [out_channel].

- num_output

  Channels of the output tensor.

- group

  Number of groups that input channels and output channels are divided into.

- scales: scale_type, scale_shape, scale_offset, scale_size

  It can be a 1-D tensor, which means a per-tensor/layer or per output channel quantization. Its number of elements should be equal to the number of output channels, where scale_value, scale_type, scale_shape, scale_offset and scale_size are the value, data type and numbers, offset address and data size of the scale, coefficient during quantization.

  Besides, scale_type has the int8, uint8 and int16 options. Where the relationship is scale_size= data_type *scale_shape. Here, data type = 1 (or 2, 4) if shift_type = int8 (or int16, int32).

- shifts: shift_type, shift_shape, shift_offset, shift_size

  It can be a 1-D tensor, which means a per-tensor/layer or per output channel quantization. Its number of elements should be equal to the number of output channels, where shift_value, shift_type, shift_shape, shift_offset and shift_size are the value, data type and numbers, offset address and data size of the shift, coefficient during quantization.

  Besides, shift_type is an int8 format. Where, the relationship is shift_size= data_type * shift_shape. Here, data type = 1 (or 2, 4) if shift_type = int8 (or int16, int32).

- multiplier

  Depthwise convolution applies a different filter to each input channel (expanding from 1 channel to multiplier channels for each), then concatenates the results together. The output has 'input_channels * multiplier' channels.

Dilation2D

Computes the grayscale dilation of 4-D input and 3-D filters tensors. The input tensor has shape [batch, in_height, in_width, depth] and the filters tensor has shape [filter_height, filter_width, depth], that is, each input channel is processed independently of the others with its own structuring function. The output tensor has shape [batch, out_height, out_width, depth]. The spatial dimensions of the output tensor depend on the padding algorithm. Currently only the default "NHWC" data_format is supported.

In detail, the grayscale morphological 2-D dilation is the max-sum correlation (for consistency with conv2d, unmirrored filters are used).

output[b, y, x, c] = max_{dy, dx} input[b, strides_y * y + dilation_y * dy, strides_x * x + dilation_x * dx,c] + weight[dy, dx, c]

Inputs

- Input data tensor X.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = Dilation'.

- kernel: kernel_x, kernel_y

  'kernel_x' ('kernel_y') is the kernel size along the 'width' ('height') axis. If not present, it should be inferred from inputs 'weights_size' and 'weight_shape'.

- stride: stride_x, stride_y

  Stride along 'batch', 'height', 'width' and 'channel' axis. Generally, the 'stride_x' ('stride_y') stride is along the 'width' ('height') axis.

- pad: pad_bottom, pad_top, pad_left, pad_right

  Padding for the beginning and ending along spatial axis, it can take any value greater than or equal to 0. The value represents the number of pixels added to the beginning and end part of the corresponding axis. The 'pad' parameter format should be as follows ('pad_bottom', 'pad_top', 'pad_left', 'pad_right'), where it means the pad pixels of the data cube (that is, bottom and top along the 'height' axis, left and right along the 'width' axis). If not present, compute and output the same shape as the input tensor.

- dilation: dilation_x, dilation_y

  The dilation value along the spatial axis of the filter.

- weights: weights_type, weights_offset, weights_size, weights_shape

  The weight tensor that will be used in the filter. Where, weights_type (or weights_offset, weights_size, weights_shape) means the weights data format (or weights address offset, total weights size which counted in bytes, and weights shape of the current layer). Generally, the 'weights' are stacked in order of [input_channel, kernel_y, kernel_x, 1].

- scale: scale_type, scale_value

  It is a 1-D tensor in order of [output_scale, input_scale, weight_scale]. Generally, the scale_type has the int8, uint8 and int16 options.

- shift: shift_type, shift_value

  It can be a scalar, which means an output shift operation during quantization. Where shift_type and shift_value are the data type of the shift operation.

Div

Performs element-wise division (with NumPy-style broadcasting support). This operator also supports multidirectional (that is, NumPy-style) broadcasting.

Inputs

- Input data tensor X1, data tensor X2.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = Div'.

- scale: scale_type, scale_value

  The scale is a vector with more than three elements, and its number of elements should be equal to the sum of output and input tensors. Where, scale_type and scale_value are the data type and value of the scale during quantization. Generally, scale_type has the int8, uint8 and int16 options. Furthermore, the scales type or value must be in order of 'output_scale, input_scale[i]'.

- shift: shift_type, shift_value

  It can be a scalar, which means an output shift operation during quantization. Where, shift_type and shift_value are the data type and value of the shift operation.

- lut: lut_type, lut_shape, lut_offset, lut_size

  Optional parameter during quantization. The *Look Up Table* (LUT) is an offline table created after quantization. Where, lut_type, lut_shape, lut_offset and lut_size are the data type, table shape, table offset address and table size. The following is a simple equation, lut_size = data_type * lut_shape. Where, data type = 1 (or 2, 4) if lut_type = int8 (or int16, int32).

ElementwiseAdd

Performs elementwise addition of each of the input tensors (with NumPy-style broadcasting support). All inputs and outputs must have the same data type. Besides, this operator supports multidirectional (that is, NumPy-style) broadcasting and sums all the input tensors into the output tensor under the OPERATOR LIBRARY support.

Inputs

- Input data tensor X1, data tensor X2, …

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = Eltwise'.

- method: ADD

  The elementwise sum for computing input tensors.

- with_activation: NONE, CLIP, RELU, RELU6, LEAKYRELU, PRELU

  Means whether to append an activation fusion operation to the current layer (such as Convolution, Fully Connected, Elementwise). 'None' means no this fusion.

  - Clip: clip_min, clip_max
    Where, clip_min (or clip_max) means the minimum (or maximum) saturation threshold during the clip operation to produce the output tensor after convolution.

  - LeakyRelu: negative_slope_type, negative_slope_value, negative_slope_scale, negative_slope_shift
    LeakyRelu takes input data (Tensor) and an argument alpha, then produces one output data (Tensor) where the function f(x) = alpha * x for x <0, f(x) = x for x >=0, is applied to the data tensor elementwise, where 'alpha' is the slope parameter. That is, 'negative_slope_type' and 'negative_slope_value' mean the negative slope coefficient data type (such as a uint8 data type) and value, which are all a scalar.
    Besides, 'negative_slope_scale' and 'negative_slope_shift' mean the parameter during quantization, which are all a scalar as well.
  - PRelu: negative_slope_type, negative_slope_shape, negative_slope_offset, negative_slope_size, negative_slope_scale, negative_slope_shift
    PRelu takes input data (Tensor) and slope tensor as input, then produces one output data (Tensor) where the function, f(x) = slope * x for x < 0, f(x) = x for x >= 0, is applied to the data tensor elementwise. This operator supports unidirectional broadcasting (tensor slope should be unidirectional broadcastable to the input tensor). Where negative_slope_type (or negative_slope_shape, negative_slope_offset, negative_slope_size) represents the data type and shape, offset address in memory and total data size of negative coefficient. Similarly, negative_slope_scale and negative_slope_shift mean the parameter during quantization as well. Note that the shape of this 'negative_slope_shape' can be smaller than input tensor (X), and if so, its shape must be unidirectional broadcastable to input tensor (X). Generally, negative_slop_shape = [out_channel].

- scale: scale_type, scale_value

  The scale is a vector with more than two elements, and its number of elements should be equal to the sum of output and input tensors. Where, scale_type and scale_value are the data type, value of the scale during quantization. Generally, scale_type has the int8, uint8 and int16 options. Furthermore, the scales type or value must be in order of 'output_scale, input_scale[i]'.

- shift: shift_type, shift_value

  It can be a scalar, which means an output shift operation during quantization. Where, shift_type and shift_value are the data type of the shift operation.

ElementwiseMax

Performs elementwise maximum of each of the input tensors (with NumPy-style broadcasting support). All inputs and outputs must have the same data type. This operator supports multidirectional (that is, NumPy-style) broadcasting. The input tensor can be a list of tensors.

Inputs

- Input data tensor X1, data tensor X2.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = Eltwise'.

- method: MAX

  The elementwise MAX of input tensors into the output tensor.

- with_activation: NONE, CLIP, RELU, RELU6, LEAKYRELU, PRELU

  Means whether to append an activation fusion operation to the current layer (such as Convolution, FullyConnected, Elementwise). 'None' means no this fusion.

  - Clip: clip_min, clip_max
    Where, clip_min (or clip_max) means the minimum (or maximum) saturation threshold during the clip operation to produce output tensor after convolution.

  - LeakyRelu: negative_slope_type, negative_slope_value, negative_slope_scale, negative_slope_shift
    LeakyRelu takes input data (Tensor) and an argument alpha, then produces one output data (Tensor) where the function f(x) = alpha * x for x <0, f(x) = x for x >=0, is applied to the data tensor elementwise, where 'alpha' is the slope parameter. That is, 'negative_slope_type' and 'negative_slope_value' mean the negative slope coefficient data type (such as a uint8 data type) and value, which are all a scalar.
    Besides, 'negative_slope_scale' and 'negative_slope_shift' mean the parameter during quantization, which are all a scalar as well.
  - PRelu: negative_slope_type, negative_slope_shape, negative_slope_offset, negative_slope_size, negative_slope_scale, negative_slope_shift
    PRelu takes input data (Tensor) and slope tensor as input, then produces one output data (Tensor) where the function f(x) = slope * x for x < 0, f(x) = x for x >= 0, is applied to the data tensor elementwise. This operator supports unidirectional broadcasting (tensor slope should be unidirectional broadcastable to the input tensor). Where negative_slope_type (or negative_slope_shape, negative_slope_offset, negative_slope_size) represents the data type and shape, offset address in memory and total data size of negative coefficient. Similarly, negative_slope_scale and negative_slope_shift mean the parameter during quantization as well. Note that the shape of this 'negative_slope_shape' can be smaller than input tensor (X), and if so, its shape must be unidirectional broadcastable to input tensor (X). Generally, negative_slop_shape = [out_channel].

- scale: scale_type, scale_value

  The scale is a vector with more than two elements, and its number of elements should be equal to the sum of output and input tensors. Where, scale_type and scale_value are the data type, value of the scale during quantization. Generally, scale_type has the int8, uint8 and int16 options. Furthermore, the scales type or value must be in order of 'output_scale, input_scale[i]'.

- shift: shift_type, shift_value

  It can be a scalar, which means an output shift operation during quantization. Where, shift_type and shift_value are the data type of the shift operation.

ElementwiseMin

Performs elementwise minimum of each of the input tensors (with NumPy-style broadcasting support). All inputs and outputs must have the same data type. This operator supports multidirectional (that is, NumPy-style) broadcasting. The input tensor can be a list of tensors.

Inputs

- Input data tensor X1, data tensor X2.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = Eltwise'.

- method: MIN

  The elementwise MIN of input tensors into the output tensor.

- with_activation: NONE, CLIP, RELU, RELU6, LEAKYRELU, PRELU

  Means whether to append an activation fusion operation to the current layer (such as Convolution, FullyConnected, Elementwise). 'None' means no this fusion.

  - Clip: clip_min, clip_max
    Where, clip_min (or clip_max) means the minimum (or maximum) saturation threshold during the clip operation to produce output tensor after convolution.

  - LeakyRelu: negative_slope_type, negative_slope_value, negative_slope_scale, negative_slope_shift
    LeakyRelu takes input data (Tensor) and an argument alpha, then produces one output data (Tensor) where the function f(x) = alpha * x for x <0, f(x) = x for x >=0, is applied to the data tensor elementwise, where 'alpha' is the slope parameter. That is, 'negative_slope_type' and 'negative_slope_value' mean the negative slope coefficient data type (such as a uint8 data type) and value, which are all a scalar.
    Besides, 'negative_slope_scale' and 'negative_slope_shift' mean the parameter during quantization, which are all a scalar as well.
  - PRelu: negative_slope_type, negative_slope_shape, negative_slope_offset, negative_slope_size, negative_slope_scale, negative_slope_shift
    PRelu takes input data (Tensor) and slope tensor as input, then produces one output data (Tensor) where the function f(x) = slope * x for x < 0, f(x) = x for x >= 0, is applied to the data tensor elementwise. This operator supports unidirectional broadcasting (tensor slope should be unidirectional broadcastable to the input tensor). Where negative_slope_type (or negative_slope_shape, negative_slope_offset, negative_slope_size) represents the data type and shape, offset address in memory and total data size of negative coefficient. Similarly, negative_slope_scale and negative_slope_shift mean the parameter during quantization as well. Note that the shape of this 'negative_slope_shape' can be smaller than input tensor (X), and if so, its shape must be unidirectional broadcastable to input tensor (X). Generally, negative_slop_shape = [out_channel].

- scale: scale_type, scale_value

  The scale is a vector with more than two elements, and its number of elements should be equal to the sum of output and input tensors. Where, scale_type and scale_value are the data type, value of the scale during quantization. Generally, scale_type has the int8, uint8 and int16 options. Furthermore, the scales type or value must be in order of 'output_scale, input_scale[i]'.

- shift: shift_type, shift_value

  It can be a scalar, which means an output shift operation during quantization. Where, shift_type and shift_value are the data type of the shift operation.

ElementwiseMul

Performs element-wise multiplication of the input tensors. All inputs and outputs must have the same data type. Besides, this operator supports multidirectional (that is, NumPy-style) broadcasting multiplication of another input tensor into the output tensor under the OPERATOR LIBRARY support.

Inputs

- Input data tensor X1, data tensor X2.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = Eltwise'.

- method: MUL

  The elementwise multiplication for computing input tensors.

- with_activation: NONE, CLIP, RELU, RELU6, LEAKYRELU, PRELU

  Means whether to append an activation fusion operation to the current layer (such as Convolution, FullyConnected, Elementwise). 'None' means no this fusion.

  - Clip: clip_min, clip_max

Where, clip_min (or clip_max) means the minimum (or maximum) saturation threshold during the clip operation to produce the output tensor after convolution.

- LeakyRelu: negative_slope_type, negative_slope_value, negative_slope_scale, negative_slope_shift
  LeakyRelu takes input data (Tensor) and an argument alpha, then produces one output data (Tensor) where the function f(x) = alpha * x for x <0, f(x) = x for x >=0, is applied to the data tensor elementwise, where 'alpha' is the slope parameter. That is, 'negative_slope_type' and 'negative_slope_value' mean the negative slope coefficient data type (such as a uint8 data type) and value, which are all a scalar.
  Besides, 'negative_slope_scale' and 'negative_slope_shift' mean the parameter during quantization, which are all a scalar as well.
- PRelu: negative_slope_type, negative_slope_shape, negative_slope_offset, negative_slope_size, negative_slope_scale, negative_slope_shift
  PRelu takes input data (Tensor) and slope tensor as input, then produces one output data (Tensor) where the function, f(x) = slope * x for x < 0, f(x) = x for x >= 0, is applied to the data tensor elementwise. This operator supports unidirectional broadcasting (tensor slope should be unidirectional broadcastable to the input tensor). Where negative_slope_type (or negative_slope_shape, negative_slope_offset, negative_slope_size) represents the data type and shape, offset address in memory and total data size of negative coefficient. Similarly, negative_slope_scale and negative_slope_shift mean the parameter during quantization as well. Note that the shape of this 'negative_slope_shape' can be smaller than input tensor (X), and if so, its shape must be unidirectional broadcastable to input tensor (X). Generally, negative_slop_shape = [out_channel].

- scale: scale_type, scale_value

  The scale is a scalar, where scale_type and scale_value are the data type, value of the scale during quantization. Generally, the scale_type has the int8, uint8 and int16 options.

- shift: shift_type, shift_value

  It can be a scalar, which means an output shift operation during quantization. Where, shift_type and shift_value are the data type of the shift operation.

ElementwiseSub

Performs elementwise subtraction of each of the input tensors (with NumPy-style broadcasting support). All inputs and outputs must have the same data type. Besides, this operator supports multidirectional (that is, NumPy-style) broadcasting. The input tensor can be a list of tensors.

Inputs

- Input first operand tensor X1, second operand tensor X2.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = Eltwise'.

- method: SUB

  The elementwise subtraction of input tensors into the output tensor.

- with_activation: NONE, CLIP, RELU, RELU6, LEAKYRELU, PRELU

  Means whether to append an activation fusion operation to the current layer (such as Convolution, Full Connected, Elementwise). 'None' means no this fusion.

  - Clip: clip_min, clip_max

Where, 'clip_min' (or clip_max) means the minimum (or maximum) saturation threshold during clip operation to produce the output tensor after convolution.

- LeakyRelu: negative_slope_type, negative_slope_value, negative_slope_scale, negative_slope_shift
  LeakyRelu takes input data (Tensor) and an argument alpha, then produces one output data (Tensor) where the function 'f(x) = alpha * x for x <0, f(x) = x for x >=0', is applied to the data tensor elementwise. Where, 'alpha' is the slope parameter. That is, 'negative_slope_type' and 'negative_slope_value' mean the negative slope coefficient data type (such as a uint8 data type) and value, which are all a scalar.
  Besides, 'negative_slope_scale' and 'negative_slope_shift' mean the parameter during quantization, which are all a scalar as well.
- PRelu: negative_slope_type, negative_slope_shape, negative_slope_offset, negative_slope_size, negative_slope_scale, negative_slope_shift
  PRelu takes input data (Tensor) and slope tensor as the input, then produces one output data (Tensor) where the function, 'f(x) = slope * x for x < 0, f(x) = x for x >= 0', is applied to the data tensor elementwise. This operator supports unidirectional broadcasting (tensor slope should be unidirectional broadcastable to the input tensor). Where negative_slope_type (or negative_slope_shape, negative_slope_offset and negative_slope_size) represents the data type (or shape, offset address in memory and total data size of negative coefficient). Similarly, negative_slope_scale and negative_slope_shift mean the parameter during quantization as well. Note that the group parameter of negative_slope is a tensor with the same shape as the last dimension of the output tensor. That is, negative_slope_shape (or the shape of quantization parameters) = output channel.

- scale: scale_type, scale_value

  The scale is a vector with more than two elements, and its number of elements should be equal to the sum of output and input tensors, where scale_type and scale_value are the data type and value of the scale during quantization. Generally, the scale_type has the int8, uint8 and int16 options. Furthermore, the scales type or value must be in order of 'output_scale, input_scale[i]'.

- shift: shift_type, shift_value

  It can be a scalar, which means an output shift operation during quantization, where shift_type and shift_value are the data type of the shift operation.

Elu

Takes an input tensor (X) and produces one output tensor (Y). Where, the function f(x) = alpha * (exp(x) -1) for x < 0, f(x) = x for x >= 0, is applied to the tensor elementwise.

Inputs

- Input data tensor X.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = Activation'.

- method: ELU

  Method to perform the input tensor.

- alpha

  Coefficient parameter in ELU. It defaults to 1.

- lut: lut_type, lut_shape, lut_offset, lut_size

The *Look Up Table* (LUT) is an offline table created after quantization. Where, lut_type, lut_shape, lut_offset and lut_size are the data type, table shape, table offset address and table size. The following is a simple equation, lut_size = data_type * lut_shape. Where, data type = 1 (or 1, 2, 2, 4, 4) if lut_type = int8 (or uint8, int16, uint16, int32, uint32).

EmbeddingLookupSparse

This operator assumes that there is at least one ID for each row in the dense tensor represented by `sp_ids` (i.e. there are no rows with empty features), and that all the indices of `sp_ids` are in canonical row-major order.

`sp_ids` and `sp_weights` (if not None) are `SparseTensor`s or `RaggedTensor`s with rank of 2. For `SpareTensor`s with left-aligned non-zero entries which can be described as `RaggedTensor`s, use of `RaggedTensor`s can yield higher performance.

It also assumes that all ID values lie in the range [0, p0), where p0 is the sum of the size of `params` along dimension 0.

If `len(params)` > 1, each element of `sp_ids` is partitioned between the elements of `params` according to the 'div ' partition strategy, which means that IDs are assigned to partitions in a contiguous manner. For instance, 13 IDs are split across 5 partitions as: [[0, 1, 2], [3, 4, 5], [6, 7, 8], [9, 10], [11, 12]].

If the ID space does not evenly divide the number of partitions, each of the first (`max_id` + 1) % `len(params)` partitions will be assigned one more ID.

Note that, in the following Inputs sections:

- X1: `params`, a single tensor representing the complete embedding tensor.
- X2: `sp_ids`, a tensor indicating the output indices.
- X3: `sp_value`, a tensor indicating the `sp_value`, which means how to gather the `params`.
- X4: `weight_value`, a tensor representing the weight for the gathered `params` to output.

Inputs

- Input data tensor X1, X2, X3, X4.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = EmbeddingLookupSpare'.

- max_norm

  A float value. If not None, each embedding is clipped if its l2-norm is larger than this value, before combining .

- combiner: MEAN/SUM/SQRTN

  A string specifying the reduction operator. 'SUM' computes the weighted sum of the embedding results for each row. 'MEAN' is the weighted sum divided by the total weight. 'SQRTN' is the weighted sum divided by the square root of the sum of the squares of the weights. Defaults to 'mean'.

- scale: scale_type, scale_value

  It can be a scalar, which means an output scale operation during quantization. Generally, the scale_type has the int8, uint8 and int16 options.

- shift: shift_type, shift_value

It can be a scalar, which means an output shift operation during quantization, where shift_type and shift_value are the data type of the shift operation.

Erf

Computes the error function of the given input tensor element-wise.

Inputs

- Input data tensor X.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = Erf'.

- lut: lut_type, lut_shape, lut_offset, lut_size

  The *Look Up Table* (LUT) is an offline table created during quantization. Where, lut_type, lut_shape, lut_offset and lut_size are the data type, table shape, table offset address and table size. The following is a simple relationship, lut_size = data_type * lut_shape. Where, data type = 1 (or 2, 4) if lut_type = int8 (or int16, int32).

Erosion2D

Computes the grayscale erosion of 4-D input and 3-D filters tensors. The input tensor has shape [batch, in_height, in_width, depth] and the filters tensor has shape [filter_height, filter_width, depth], that is, each input channel is processed independently of the others with its own structuring function. The output tensor has shape [batch, out_height, out_width, depth]. The spatial dimensions of the output tensor depend on the padding algorithm. Currently only the default "NHWC" data_format is supported.

output[b, y, x, c] = min_{dy, dx} input[b, strides_y * y + dilation_y * dy, strides_x * x + dilation_x * dx,c] + weight[dy, dx, c]

Inputs

- Input data tensor X.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = Erosion'.

- kernel: kernel_x, kernel_y

  'kernel_x' ('kernel_y') is the kernel size along the 'width' ('height') axis. If not present, it should be inferred from inputs 'weights_size' and 'weight_shape'.

- stride: stride_x, stride_y

  Stride along the 'batch', 'height', 'width' and 'channel' axis. Generally, the 'stride_x' ('stride_y') stride is along the 'width' ('height') axis.

- pad: pad_bottom, pad_top, pad_left, pad_right

Padding for the beginning and ending along spatial axis, it can take any value greater than or equal to 0. The value represents the number of pixels added to the beginning and end part of the corresponding axis. The 'pad' parameter format should be as follows ('pad_bottom', 'pad_top', 'pad_left', 'pad_right'), where it means the pad pixels of the data cube (that is, bottom and top along the 'height' axis, left and right along the 'width' axis). If not present, compute and output the same shape as the input tensor.

- dilation: dilation_x, dilation_y

  The dilation value along the spatial axis of the filter.

- weights: weights_type, weights_offset, weights_size, weights_shape

  The weight tensor that will be used in the filter. Where, weights_type (or weights_offset, weights_size, weights_shape) means the weights data format (or weights address offset, total weights size which counted in bytes, and weights shape of the current layer). Generally, the 'weights' are stacked in order of [input_channel, kernel_y, kernel_x, 1].

- scale: scale_type, scale_value

  It is a 1-D tensor in order of [output_scale, input_scale, weight_scale]. Generally, the scale_type has the int8, uint8 and int16 options.

- shift: shift_type, shift_value

  It can be a scalar, which means an output shift operation during quantization, where shift_type and shift_value are the data type of the shift operation.

Exp

Calculates the exponential of the given input tensor (X), where the function f(x) = exp(x), is applied to the tensor elementwise.

Inputs

- Input data tensor X.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = Exp'.

- lut: lut_type, lut_shape, lut_offset, lut_size

  The *Look Up Table* (LUT) is an offline table created after quantization. Where, lut_type, lut_shape, lut_offset and lut_size are the data type, table shape, table offset address and table size. The following is a simple equation, lut_size = data_type * lut_shape. Where, data type = 1 (or 2, 4) if lut_type = int8 (or int16, int32).

Filter

Takes one input data tensor (X) and filter tensor (B) to return an output tensor (Y) after filtration. Where, the element will be filtered with zero selector accordingly in filter tensor B, otherwise remaining. The filter tensor is a vector which is composed of '0' or '1' with the same length as the specified axis in data tensor X. Similarly, multiple input tensors (X[i]) will return multiple outputs after the same filtration.

Inputs

- Input data tensor X[i] ('i' means the index of input tensors), filter tensor B (a 1-D tensor).

Outputs

- Output tensor Y[i] ('i' means the index of output tensors), effective length (a scalar with an empty shape).

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = Filter'.

- num

  Indicates the number of input tensors.

- axis

  A scalar of integer to indicate the axis to filter. A negative value means counting the dimensions from the back. Generally, the axis can be '0' when you are not sure how to use it. The accepted range is [-1, r-1] where r = rank(input data).

Floor

The floor operation takes one input tensor (X) and produces one output tensor (Y). Where the floor function 'y = floor(x)' is applied to the tensor elementwise.

Inputs

- Input data tensor X.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = Floor'.

- lut: lut_type, lut_shape, lut_offset, lut_size

  The *Look Up Table* (LUT) is an offline table created after quantization. Where, lut_type, lut_shape, lut_offset and lut_size are the data type, table shape, table offset address and table size. The following is a simple equation, lut_size = data_type * lut_shape. Where, data type = 1 (or 2, 4) if lut_type = int8 (or int16, int32).

FractionalPool

Fractional average pooling is similar to Fractional max pooling in the pooling region generation step. The only difference is that after pooling regions are generated, a mean operation is performed instead of a max operation in each pooling region.

Note that, in the following Inputs or Outputs sections:

- X means the input data.
- Y1 means the output data after fractional avgerage pooling.
- Y2 means row_pooling_sequence.
- Y3 means col_pooling_sequence.

Inputs

- Input data tensor X.

Outputs

- Output tensor Y1, Y2, Y3.

Attributes

- layer_type

The operation type of a layer. Here is 'layer_type = FractionalPool'.

- pseudo_random: true, false

  When set to `true`, generates the pooling sequence in a pseudorandom fashion, otherwise, in a random fashion.

- overlap: true, false

  When set to `true`, it means when pooling, the values at the boundary of adjacent pooling cells are used by both cells.

- method: AVG/MAX

  The method of pooling.

- seed: true, false

  An optional `int`. Defaults to `0`. If set to be non-zero, the random number generator is seeded by the given seed. Otherwise it is seeded by a random seed.

FullyConnected

Takes input tensor (X) and computes the class scores and outputs the 1-D array of size equal to the number of classes. In other words, it performs a 'dot(input_tensor, kernel)' operation with a bias addition, then outputs the tensor after the activation function.

Inputs

- Input data tensor X.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = FullyConnected'.

- weights: weights_type, weights_offset, weights_size, weights_shape

  The weight tensor that will be used in the FullyConnected. Where, weights_type (or weights_offset, weights_size, weights_shape) means the weights data format (or weights address offset, total weights size which counted in bytes, and weights shape of the current layer). Generally, the 'weights' are stacked in order of [out_channel, input_channel] (that is, weights_shape = [out_channel, input_channel]).

- biases: biases_type, biases_offset, biases_size, biases_shape

  Bias to be added to the FullyConnected computation. Where, biases_type, biases_offset, biases_size, and biases_shape mean the biases data format, biases address offset, total biases size (counted in bytes), and biases shape of the current layer. Generally, 'biases_shape' equals the number of output channels.

- scale: scale_value, scale_type

  It can be a scalar, which means a per-tensor/layer quantization. If it is a scalar, scale_value and scale_type mean the value and data type of the scale parameter during per-tensor/layer quantization.

- shift: shift_type, shift_value

  It can be a scalar, which means a per-tensor/layer quantization. If it is a scalar, shift_value and shift_type mean the value and data type of the shift parameter during per-tensor/layer quantization.

- with_activation: NONE, CLIP, RELU, RELU6, LEAKYRELU, PRELU

Means whether to append an activation fusion operation to the current layer (such as Convolution, FullyConnected, Elementwise). 'None' means no this fusion.

- Clip: clip_min, clip_max
  Where, clip_min (or clip_max) means the minimum (or maximum) saturation threshold during the clip operation to produce the output tensor after convolution.
- LeakyRelu: negative_slope_type, negative_slope_value, negative_slope_scale, negative_slope_shift (a scalar)
  LeakyRelu takes input data (Tensor) and an argument alpha, then produces one output data (Tensor) where the function f(x) = alpha * x for x <0, f(x) = x for x >=0, is applied to the data tensor elementwise, where 'alpha' is the slope parameter. That is, 'negative_slope_type' and 'negative_slope_value' mean the negative slope coefficient data type (such as a uint8 data type) and value, which are all a scalar.
  Besides, 'negative_slope_scale' and 'negative_slope_shift' mean the parameter during quantization, which are all a scalar as well.
- PRelu: negative_slope_type, negative_slope_shape, negative_slope_offset, negative_slope_size, negative_slope_scale, negative_slope_shift
  PRelu takes input data (Tensor) and slope tensor as input, then produces one output data (Tensor) where the function, f(x) = slope * x for x < 0, f(x) = x for x >= 0, is applied to the data tensor elementwise. This operator supports unidirectional broadcasting (tensor slope should be unidirectional broadcastable to the input tensor). Where negative_slope_type (or negative_slope_shape, negative_slope_offset, negative_slope_size) represents the data type and shape, offset address in memory and total data size of negative coefficient. Similarly, negative_slope_scale and negative_slope_shift mean the parameter during quantization as well. Note that the shape of this 'negative_slope_shape' can be smaller than input tensor (X), and if so, its shape must be unidirectional broadcastable to input tensor (X). Generally, negative_slop_shape = [out_channel].

- num_output

  Channels of the output tensor.

GRUv1

Computes a one-layer GRU with the first mode.

Notations:

x: Input tensor

z: Update gate

r: Reset gate

h: Hidden gate

t: Time step (t-1 means the previous time step)

W[r, z, c]: Parameter weight matrix for reset, update and hidden gates

R[r, z, c]: Recurrence weight matrix for reset, update and hidden gates

b[r, z, c, n]: Bias vectors for reset, update, and two hidden gates

H: Hidden state

f: Activation function

g: Activation function

Activation functions:

Relu: max(0, x)

Tanh: (1 - e^{-2x}) / (1 + e^{-2x})

Sigmoid: 1/(1 + e^{-x})

Affine: alpha * x + beta

LeakyRelu: x if x >= 0 else alpha * x

ThresholdedRelu: x if x >= alpha else 0

HardSighmoid: min(max (alpha * x + beta, 0), 1)

Elu: x if x >= 0 else alpha * (e^x -1)

Softsign: x / (1 + |x|)

Softplus: log(1 + e^x)

Clip: clip_max if x >= clip_max, x if clip_min < x < clip_max, clip_min if x <= clip_min

Equations (Default: f = SIGMOID, g = TANH):

$r_t = f(W_r x_t + R_r H_{t-1} + b_r)$

$z_t = f(W_z x_t + R_z H_{t-1} + b_z)$

$h_t = g(W_c x_t + (R_c H_{t-1} + b_n) \otimes r_t + b_c)$

$H_t = z_t \otimes H_{t-1} + (\mathbf{1} - z_t) \otimes h_t$

Where, $\otimes$ means element-wise multiplication (or Hadamard product) and 1 (in bold) means that all the j-th elements in a vector are one. During computation, this operator has some optional inputs. An empty string means using the default value or unspecified arguments.

Note that, in the following Inputs or Outputs sections:

- X means the input with the shape as [batch size, time step, input_size].
- H_0 means the initial hidden state for each element in the batch with the shape as [batch size, cell size].
- H_n means the hidden state for t= time_step with the shape as [batch size, cell size].
- H means all hidden state with the shape as [batch size, time step, cell_size].

Inputs

- Input data tensor X, H_0.

Outputs

- The output tensor is a certain combination of H and Hn tensor described as 'out_sequence' parameters.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = GRUv1'.

- out_sequence

  The output sequence is one of enumerated lists and in a fixed order. Here, enumerated list includes [H], [H_n], [H, H_n].

- activations

  A list of activation functions for update, reset and hidden gates in an order (f, g) in GRU. The activation must be one of the activation functions defined in the preceding section.

- activation_alpha

  Optional scaling parameters in some activation functions. The values are composed of an order in activation functions, such as (f, g) in GRU. Generally, the values are the same as the values of corresponding operators. For example, the alpha value in LeakyRelu is 0.01.

- activation_beta

  Optional scaling parameters in some activation functions. The values are composed of an order in activation functions, such as (f, g) in GRU. Generally, the values are the same as the values of corresponding operators.

- threshold

  Optional parameter in a float IR, which is applied to the input of activations (or input tensor X). Clipping bounds the elements of a tensor in the range of [- threshold, + threshold]. There is no clip if not specified.

- clip: clip_min, clip_max

  Optional parameter in a float IR, which is applied to the CLIP activation functions. Where, clip_min (or clip_max) means the minimum (or maximum) saturation threshold applied to the activation functions.

- direction: forward, reverse

  Indicates that the RNN is forward or reverse. It must be either of the directions-forward (default), or reverse.

- time_steps

  Time step.

- input_size

  Size of input tensor x.

- cell_size

  Size of hidden state h.

- weights: weights_type, weights_offset, weights_size, weights_shape

  The weight tensor that will be used in the computation. Where, weights_type, weights_offset, weights_size, and weights_shape mean the weights data format, weights address offset, total weights size (counted in bytes), and weights shape of the current layer. The corresponding equation is weights_size = data_type * 3 * h * (x + h), weights_shape = [3 * h, x + h]. Here, data type = 1 (or 2, 4) if weight_type = int8 (or int16, int32).

Note that 'wight_shape' is stacked in order of [[W_{r0} , R_{r0}], [W_{z0} , R_{z0}], [W_{c0} , W_{c0}]], ..., [[$W_{r(h-1)}$, $R_{r(h-1)}$], [$W_{z(h-1)}$, $R_{z(h-1)}$], [$W_{c(h-1)}$, $W_{c(h-1)}$]] (that is, weights_shape = [3 * h, x + h]). Where, 'x' and 'h' mean the input size and cell size as described in the preceding section.

- biases: biases_type, biases_offset, biases_size, biases_shape

  To be added to the computation. Where, biases_type, biases_offset, biases_size, and biases_shape mean the biases data format, biases address offset, total biases size (counted in bytes), and biases shape of the current layer. The corresponding equation is biases_size = data_type * 4 * h, biases_shape = 4 * h. Here, data type = 1 (or 2, 4) if weight_type = int8 (or int16, int32). Note that 'biases_shape' is stacked in order of [b_{r0}, ..., $b_{r(h-1)}$, b_{z0}, ..., $b_{z(h-1)}$, b_{c0}, ..., $b_{c(h-1)}$, b_{n0}, ..., $b_{n(h-1)}$] (that is, biases_shape = 4 * h). Where, 'h' means the cell size as described in the preceding section.

- scale: scale_type, scale_value

  It can be a vector during quantization in order of [H_{t-1}, r_t, R_c, h_t, (**1** - z_t), H_t]. Where scale_type and scale_value are the data type and value of the scale. Generally, scale_type has int8, uint8, int16, and other data types.

- shift: shift_type, shift_value

  It can be a vector during quantization in order of [H_{t-1}, r_t, R_c, h_t, (**1** - z_t), H_t]. Where shift_type and shift_value are the data type and value. Generally, the shift type has int8 (or int16, int32).

- lut_rt: lut_rt_type, lut_rt_shape, lut_rt_offset, lut_rt_size

  The *Look Up Table* (LUT) is an offline table created during activation quantization in 'r_t' calculation. Where, lut_rt_type, lut_rt_shape, lut_rt_offset and lut_rt_size are the data type, table shape, table offset address and table size. The following is a simple equation, lut_rt_size = data_type * lut_rt_shape. Here, data type = 1 (or 2, 4) if lut_rt_type = int8 (or int16, int32).

- lut_zt: lut_zt_type, lut_zt_shape, lut_zt_offset, lut_zt_size

  The *Look Up Table* (LUT) is an offline table created during activation quantization in 'z_t' calculation. Where, lut_zt_type, lut_zt_shape, lut_zt_offset and lut_zt_size are the data type, table shape, table offset address and table size. The following is a simple equation, lut_zt_size = data_type * lut_zt_shape. Here, data type = 1 (or 2, 4) if lut_zt_type = int8 (or int16, int32).

- lut_ht: lut_ht_type, lut_ht_shape, lut_ht_offset, lut_ht_size

  The *Look Up Table* (LUT) is an offline table created during activation quantization in 'h_t' calculation. Where, lut_ht_type, lut_ht_shape, lut_ht_offset and lut_ht_size are the data type, table shape, table offset address and table size. The following is a simple equation, lut_ht_size = data_type * lut_ht_shape. Here, data type = 1 (or 2, 4) if lut_ht_type = int8 (or int16, int32).

GRUv3

Computes a one-layer GRU.

Notations:

x: Input tensor

z: Update gate

r: Reset gate

h: Hidden gate

t: Time step (t-1 means the previous time step)

W[r, z, c]: Parameter weight matrix for reset, update and hidden gates

R[r, z, c]: Recurrence weight matrix for reset, update and hidden gates

b[r, z, c]: Bias vectors for reset, update and hidden gates

H: Hidden state

f: Activation function

g: Activation function

Activation functions:

Relu: max(0, x)

Tanh: (1 - e^{-2x}) / (1 + e^{-2x})

Sigmoid: 1/(1 + e^{-x})

Affine: alpha * x + beta

LeakyRelu: x if x >= 0 else alpha * x

ThresholdedRelu: x if x >= alpha else 0

HardSighmoid: min(max (alpha * x + beta, 0), 1)

Elu: x if x >= 0 else alpha * (e^x -1)

Softsign: x / (1 + |x|)

Softplus: log(1 + e^x)

Clip: clip_max if x >= clip_max, x if clip_min < x < clip_max, clip_min if x <= clip_min

Equations (Default: f = SIGMOID, g = TANH):

$r_t = f(W_r x_t + R_r H_{t-1} + b_r)$;

$z_t = f(W_z x_t + R_z H_{t-1} + b_z)$;

$h_t = g(W_c x_t + R_c (H_{t-1} \otimes r_t) + b_c)$;

$H_t = z_t \otimes H_{t-1} + (\mathbf{1} - z_t) \otimes h_t$

Where, $\otimes$ means element-wise multiplication (or Hadamard product) and 1 (in bold) means all the j-th elements in a vector are one. During computation, this operator has some optional inputs. An empty string means using the default value or not specified arguments.

Note that, in the following Inputs or Outputs sections:

- X means the input with the shape as [batch size, time step, input_size].
- H_0 means the initial hidden state for each element in the batch with the shape as [batch size, cell size].
- H_n means the hidden state for t= time_step with the shape as [batch size, cell size].
- H means all hidden state with the shape as [batch size, time step, cell_size].

Inputs

- Input data tensor X, H_0.

Outputs

- The output tensor is a certain combination of H and Hn tensor described as 'out_sequence' parameters.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = GRUv3'.

- out_sequence

  The output sequence is one of enumerated lists and in a fixed order. Here, enumerated list includes [H], [H_n], [H, H_n].

- activations

  A list of activation functions for update, reset and hidden gates in an order (f, g) in GRU. The activation must be one of the activation functions defined above.

- activation_alpha

  Optional scaling parameters in some activation functions. The values are composed of an order in activation functions, such as (f, g) in GRU. Generally, the values are the same as the values of corresponding operators. For example, the alpha value in LeakyRelu is 0.01.

- activation_beta

  Optional scaling parameters in some activation functions. The values are composed of an order in activation functions, such as (f, g) in GRU. Generally, the values are the same as the values of corresponding operators.

- threshold

  Optional parameter in a float IR, which is applied to the input of activations (or input tensor X). Clipping bounds the elements of a tensor in the range of [- threshold, + threshold]. No clip if not specified.

- clip: clip_min, clip_max

  Optional parameter in a float IR, which is applied to the CLIP activation functions. Where, clip_min (or clip_max) means the minimum (or maximum) saturation threshold applied to the activation functions.

- direction: forward, reverse

  Indicates that the RNN is forward or reverse. It must be one of forward as default, or reverse.

- time_steps

  Time step.

- input_size

  Size of input tensor x.

- cell_size

  Size of hidden state h.

- weights: weights_type, weights_offset, weights_size, weights_shape

  The weight tensor that will be used in the computation. Where, weights_type, weights_offset, weights_size, and weights_shape mean the weights data format, weights address offset, total weights size (counted in bytes), and weights shape of the current layer. The corresponding equation is weights_size = data_type * 3 * h * (x + h), weights_shape = [3 * h, x + h]. Here, data type = 1 (or 2, 4) if weight_type = int8 (or int16, int32).

  Note that 'wight_shape' is stacked in order of [[W_{r0} , R_{r0}], [W_{z0} , R_{z0}], [W_{c0} , W_{c0}]], ..., [[$W_{r(h-1)}$, $R_{r(h-1)}$], [$W_{z(h-1)}$, $R_{z(h-1)}$], [$W_{c(h-1)}$, $W_{c(h-1)}$]] (that is, weights_shape = [3 * h, x + h]). Where, 'x' and 'h' mean the input size and cell size as above.

- biases: biases_type, biases_offset, biases_size, biases_shape

  To be added to the computation. Where, biases_type, biases_offset, biases_size, and biases_shape mean the biases data format, biases address offset, total biases size (counted in bytes), and biases shape of the current layer. The corresponding equation is biases_size = data_type * 3 * h, biases_shape = 3 * h. Here, data type = 1 (or 2, 4) if weight_type = int8 (or int16, int32).

  Note that 'biases_shape' is stacked in order of [b_{r0}, ..., $b_{r(h-1)}$, b_{z0}, ..., $b_{z(h-1)}$, b_{c0}, ..., $b_{c(h-1)}$] (that is, biases_shape = 3 * h). Where, 'h' means the cell size as above.

- scale: scale_type, scale_value

  It can be a vector during quantization in order of [H_{t-1}, r_t, R_c, h_t, (1 - z_t), H_t]. Where scale_type and scale_value are the data type and value of the scale. Generally, scale_type has int8, uint8, int16, and other data types.

- shift: shift_type, shift_value

  It can be a vector during quantization in order of [H_{t-1}, r_t, R_c, h_t, (1 - z_t), H_t]. Where shift_type and shift_value are the data type and value. Generally, the shift type has int8 (or int16, int32).

- lut_rt: lut_rt_type, lut_rt_shape, lut_rt_offset, lut_rt_size

  The *Look Up Table* (LUT) is an offline table created during activation quantization in 'r_t' calculation. Where, lut_rt_type, lut_rt_shape, lut_rt_offset, and lut_rt_size are the data type, table shape, table offset address and table size. The following is a simple equation, lut_rt_size = data_type * lut_rt_shape. Here, data type = 1 (or 2, 4) if lut_rt_type = int8 (or int16, int32).

- lut_zt: lut_zt_type, lut_zt_shape, lut_zt_offset, lut_zt_size

  The *Look Up Table* (LUT) is an offline table created during activation quantization in 'z_t' calculation. Where, lut_zt_type, lut_zt_shape, lut_zt_offset and lut_zt_size are the data type, table shape, table offset address and table size. The following is a simple equation, lut_zt_size = data_type * lut_zt_shape. Here, data type = 1 (or 2, 4) if lut_zt_type = int8 (or int16, int32).

- lut_ht: lut_ht_type, lut_ht_shape, lut_ht_offset, lut_ht_size

  The *Look Up Table* (LUT) is an offline table created during activation quantization in 'h_t' calculation. Where, lut_ht_type, lut_ht_shape, lut_ht_offset and lut_ht_size are the data type, table shape, table offset address and table size. The following is a simple equation, lut_ht_size = data_type * lut_ht_shape. Here, data type = 1 (or 2, 4) if lut_ht_type = int8 (or int16, int32).

Gather

Given parameters and indices tensor, gathers entries of the parameters tensor indexed by indices and concatenates them in an output tensor. The indices are a tensor of integer, and there will be an error if any of the index values are out of bounds. Generally, a negative value in the indices tensor means counting the index from the back.

Inputs

- Input data tensor X, indices.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = Gather'.

- axis

  The axis to gather on. A negative value means counting the dimensions from the back. Generally, the axis can be '-1' when you are not sure how to use it. The accepted range is [-1, r-1] where r = rank(input data).

- batch_dims

  An integer to indicate the number of batch dimensions. It must be less than or equal to the rank (inputs).

GatherElements

GatherElements takes two inputs data and indices of the same rank r >= 1 and an optional attribute axis that identifies an axis of data (by default, the outer-most axis, that is axis 0). It is an indexing operation that produces its output by indexing into the input data tensor at index positions determined by elements of the indices tensor. Its output shape is the same as the shape of indices and consists of one value (gathered from the data) for each element in indices.

For instance, in the 3-D case (r = 3), the output produced is determined by the following equations:

$$
\begin{aligned}
out[i][j][k] &= input\big[index[i][j][k]\big][j][k] \quad && if\ axis = 0 \\
out[i][j][k] &= input[i]\big[index[i][j][k]\big][k] \quad && if\ axis = 1 \\
out[i][j][k] &= input[i][j]\big[index[i][j][k]\big] \quad && if\ axis = 2
\end{aligned}
$$

Inputs

- Input data tensor X, index.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = GatherElements'.

- axis

  The axis to gather on. A negative value means counting the dimensions from the back. Generally, the axis can be '-1' when you are not sure how to use it. The accepted range is [-1, r-1] where r = rank(input data).

GatherND

Given the input indices tensor, gathers slices from input tensor X into output tensor Y with the shape specified by indices. The output tensor has the shape 'indices.shape[:-1] + X.shape[indices.shape[-1] + batch_dims:]' with the left closed and right open interval. Generally, 'indices[-1] + batch_dims' should be less than or equal to 'rank(X)'. Note that the negative value in indices tensor means counting the index from the back. For example,

- If batch_dims = 0,

  Input tensor X = [[['a0', 'b0'], ['c0', 'd0']], [['a1', 'b1'], ['c1', 'd1']]], indices = [[[1, 0]], [[0, 1]]], then returns an output tensor as Y = [[['a1','b1']], [['c0','d0']]].

- If batch_dims = 1,

Input tensor X = [[['a0', 'b0'], ['c0', 'd0']], [['a1', 'b1'], ['c1', 'd1']]], indices = [[[1, 0]], [[0, 1]]], then returns an output tensor as Y = [['c0'], ['b1']].

Inputs

- Input data tensor X, indices.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = GatherND'.

- batch_dims

  A scalar of integer to indicate the number of batch dimensions. Generally, it defaults to 0 if you are not sure of it.

Gelu

Computes the *Gaussian Error Linear Unit* (GELU) activation function. GELU computes x * P(X <= x), where P(X) ~ N(0, 1). The (GELU) nonlinearity weights inputs by their values, rather than gates inputs by their signs as in ReLU.

Inputs

- Input data tensor X.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = Activation'.

- method: GELU

  Method to perform the input tensor.

- lut: lut_type, lut_shape, lut_offset, lut_size

  The *Look Up Table* (LUT) is an offline table created during quantization. Where, lut_type, lut_shape, lut_offset and lut_size are the data type, table shape, table offset address and table size. The following is a simple relationship, lut_size = data_type * lut_shape. Where, data type = 1 (or 2, 4) if lut_type = int8 (or int16, int32).

Gemm

General matrix multiplication. Here,

- A' = transpose(A) if 'trans_a' else A.
- B' = transpose(B) if 'trans_b' else B.
- Y = alpha * (A') * (B') + beta * C.

Where A (or B) should be transposed before performing the computation if 'trans_a' (or 'trans_b') is 'true'. The input tensor C is optional, and can be broadcast to the shape (M, N).

Note that, in the following Inputs or Outputs sections:

- A means the first tensor with a shape (M, K) if 'trans_a' is 'false', or a shape (K, M) if 'trans_a' is 'true'.

- B means the first tensor with a shape (K, N) if ‘trans_b’ is ‘false’, or a shape (N, K) if ‘trans_b’ is ‘true’.
- C means the optional tensor whose shape should be unidirectional broadcastable to (M, N). If not specified, the computation is performed after matrix multiplication.
- Y means the output tensor with a shape (M, N).

Inputs

- Input data tensor A, input data tensor B, optional data tensor C.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is ‘layer_type = Gemm’.

- alpha

  The scalar multiplier for the product of input tensors (A’ * B’).

- beta

  The scalar multiplier for the optional input tensors C.

- trans_a: true, false

  Indicates whether input tensor A should be transposed.

- trans_b: true, false

  Indicates whether input tensor B should be transposed.

- scale: scale_type, scale_value

  The scale is a vector with two elements. Where ‘scale_type’ and ‘scale_value’ are the data type and value of the scale during quantization. Generally, scale_type has the uint8, uint16 options. The scales type or value must be in order of [output_scale, (alpha * A’ * B’)].

- shift: shift_type, shift_value

  It can be a vector with two elements and means an output shift operation during quantization. Where, shift_type and shift_value are the data type of the shift operation. The shift type or value must be in order of [output_shift, (alpha * A’ * B’)].

GetValidCounts

Gets valid count of bounding boxes given a score threshold. Also moves valid boxes to the top of input data.

Note that, in the following Inputs or Outputs sections:

- X1 means input data with shape [batch_size, num_anchors, 6] or [batch_size, num_anchors, 5].
- Y1 means the valid number of boxes.
- Y2 means the rearranged data tensor.
- Y3 means the related index in input data.

Inputs

- Input tensor X1

Outputs

- Output tensor Y1, Y2, Y3

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = GetValidCounts'.

- score_threshold

  The threshold of score for valid bounding boxes.

- id_index : optional, int

  Index of the class categories. -1 means disabled.

- score_index: optional, int

  Index of the scores/confidence of boxes.

GridSample

Given an input and a flow-field grid, computes the output using input values and pixel locations from grid. Currently, only spatial (4-D) inputs are supported. For input with shape (N, C, H, W) and grid with shape (N, H_out, W_out, 2), the output will have shape (N, C, H_out, W_out). For each output location output[N, C, H_out, W_out], the size-2 vector grid[N, H_out, W_out] specifies input pixel locations x and y, which are used to interpolate the output value output[N, C, H_out, W_out].

Inputs

- Input data tensor X.
- Grid data tensor X1.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = GridSample'.

- align_coners: bool (default is false)

  If align_corners=true, the extrema (-1 and 1) are considered as referring to the center points of the input's corner pixels. If align_corners=false, they are instead considered as referring to the corner points of the input's corner pixels, making the sampling more resolution agnostic.

- padding_mode: string (default is zeros)

  Supported padding modes for outside grid values: 'zeros'(default), 'border', 'reflection'.

  - zeros: use 0 for out-of-bound grid locations.
  - border: use border values for out-of-bound grid locations.
  - reflection: use values at locations reflected by the border for out-of-bound grid locations.

  If index 0 represents the margin pixel, the reflected value at index -1 will be the same as the value at index 1. For location far away from the border, it will keep being reflected until becoming in bound. If pixel location x = -3.5 reflects by border -1 and becomes x' = 1.5, then reflects by border 1 and becomes x'' = 0.5.

- method: string (default is bilinear)

Three interpolation modes: bilinear (default), nearest, and bicubic.

- scale: scale_type, scale_value

  The scale is a scalar, where scale_type and scale_value are the data type and value of the scale during quantization. Generally, scale_type has int8, uint8, int16 and other data types.

- shift: shift_type, shift_value

  It can be a list, which means a shift operation during quantization, where shift_type and shift_value are the data type of the shift operation.

GroupConvolution

Performs a filter on the input tensor (X) and produces the output tensor (Y). If bias is necessary, a bias vector will be added to the output tensor. Similarly, if the activation function is necessary, it is applied to the output tensor as well.

Inputs

- Input data tensor X.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is ‘layer_type = Convolution’.

- kernel: kernel_x, kernel_y

  ‘kernel_x’ (‘kernel_y’) is the kernel size along the ‘width’ (‘height’) axis. If not present, it should be inferred from inputs ‘weights_ size’ and ‘weight_shape’.

- stride: stride_x, stride_y

  Stride along ‘batch’, ‘height’, ‘width’ and ‘channel’ axis. Generally, the ‘stride_x’ (‘stride_y’) stride is along the ‘width’ (‘height’) axis.

- pad: pad_bottom, pad_left, pad_right, pad_top

  Padding for the beginning and ending along spatial axis, it can take any value greater than or equal to 0. The value represents the number of pixels added to the beginning and end part of the corresponding axis. The ‘pad’ parameter format should be as follows (‘pad_bottom’, ‘pad_top’, ‘pad_left’, ‘pad_right’), which means the pad pixels of the data cube (that is, bottom and top along the ‘height’ axis, left and right along the ‘width’ axis). If not present, compute and output the same shape as the input tensor.

- dilation: dilation_x, dilation_y

  The dilation value along the spatial axis of the filter. If not present, the dilation default value is 1 along each spatial axis. Where dilation_x (dilation_y) means the ‘height’ (‘width’) axis of the filter in the NHWC data format.

- weights: weights_type, weights_offset, weights_size, weights_shape

  The weight tensor that will be used in the convolutions. Where, weights_type (or weights_offset, weights_size, weights_shape) means the weights data format (or weights address offset, total weights size which counted in bytes, and weights shape of the current layer). Generally, the ‘weights’ are stacked in order of [out_channel, kernel_y, kernel_x, input_channel / group] (that is, weights_shape = [out_channel, kernel_y, kernel_x, input_channel / group]).

- biases: biases_type, biases_offset, biases_size, biases_shape

Bias to be added to the convolution. Where, biases_type, biases_offset, biases_size, and biases_shape mean the biases data format, biases address offset, total biases size (counted in bytes), and biases shape of the current layer. Generally, 'biases_shape' equals the number of output channels.

- with_activation: NONE, CLIP, RELU, RELU6, LEAKYRELU, PRELU

  Means whether to append an activation fusion operation to the current layer (such as Convolution, FullyConnected, Elementwise). 'None' means no this fusion.

  - Clip: clip_min, clip_max
    Where, clip_min (or clip_max) means the minimum (or maximum) saturation threshold during the clip operation to produce the output tensor after convolution.
  - LeakyRelu: negative_slope_type, negative_slope_value, negative_slope_scale, negative_slope_shift (a scalar)
    LeakyRelu takes input data (Tensor) and an argument alpha, then produces one output data (Tensor) where the function f(x) = alpha * x for x <0, f(x) = x for x >=0, is applied to the data tensor elementwise, where 'alpha' is the slope parameter. That is, 'negative_slope_type' and 'negative_slope_value' mean the negative slope coefficient data type (such as a uint8 data type) and value, which are all a scalar.
    Besides, 'negative_slope_scale' and 'negative_slope_shift' mean the parameter during quantization, which are all a scalar as well.
  - PRelu: negative_slope_type, negative_slope_shape, negative_slope_offset, negative_slope_size, negative_slope_scale, negative_slope_shift
    PRelu takes input data (Tensor) and slope tensor as input, then produces one output data (Tensor) where the function, f(x) = slope * x for x < 0, f(x) = x for x >= 0, is applied to the data tensor elementwise. This operator supports unidirectional broadcasting (tensor slope should be unidirectional broadcastable to the input tensor). Where negative_slope_type (or negative_slope_shape, negative_slope_offset, negative_slope_size) represents the data type and shape, offset address in memory and total data size of negative coefficient. Similarly, negative_slope_scale and negative_slope_shift mean the parameter during quantization as well. Note that the shape of this 'negative_slope_shape' can be smaller than input tensor (X), and if so, its shape must be unidirectional broadcastable to input tensor (X). Generally, negative_slop_shape = [out_channel].
- num_output

  Channels of the output tensor.

- group

  Number of groups that input channels and output channels are divided into. 'group' is greater than 1.

- scales: scale_type, scale_shape, scale_offset, scale_size

  It is a 1-D tensor, which means a per-group layer quantization. Its number of elements should be equal to 'group', where scale_value, scale_type, scale_shape, scale_offset, and scale_size are the value, data type and numbers, offset address and data size of the scale coefficient during quantization.

  Besides, scale_type has the int8, uint8 and int16 options. Where, the relationship is scale_size= data_type * scale_shape. Here, data type = 1 (or 2, 4) if shift_type = int8 (or int16, int32).

- shifts: shift_type, shift_shape, shift_offset, shift_size

  It is a 1-D tensor, which means a per-group layer quantization. Its number of elements should be equal to 'group', where shift_value, shift_type, shift_shape, shift_offset, and shift_size are the value, data type and numbers, offset address and data size of the shift coefficient during quantization.

  Besides, shift_type is a 'int8' format. Where, the relationship is shift_size= data_type * shift_shape. Here, data type = 1 (or 2, 4) if shift_type = int8 (or int16, int32).

GroupNormalization

Computes the activation normalization of the previous layer for the given axis in a batch and a group independently. That is, y = gamma * (x -mean) / sqrt(variance + epsilon) + beta, where mean and variance are computed under all effective axis within each group. By default, the shape of 'gamma' and 'beta' is the batch size in NHWC format.

Inputs

- Input data tensor X.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = GroupNorm'.

- weights: weights_type, weights_offset, weights_size, weights_shape

  The 'gamma' tensor will be used in instance normalization computation. Where, weights_type, weights_offset, weights_size and weights_shape mean the weights data format, weights address offset, total weights size (counted in bytes) and weights shape of the current layer. Generally, 'weights_shape' is broadcastable or the same shape as the input tensor's dimension that 'axis' specifies.

- biases: biases_type, biases_offset, biases_size, biases_shape

  The 'beta' tensor will be used in instance normalization computation. Where, biases_type, biases_offset, biases_size and biases_shape mean the biases data format, biases address offset, total biases size (counted in bytes) and biases shape of the current layer. Generally, 'biases_shape' is broadcastable or the same shape as the input tensor's dimension that 'axis' specifies.

- scale: scale_type, scale_offset, scale_size, scale_shape

  The scale *Look Up Table* (LUT) is an offline table created after quantization. Where, lut_type, lut_shape, lut_offset and lut_size are the data type, table shape, table offset address and table size. The following is a simple equation, lut_size = data_type * lut_shape. Here, data type = 1 (or 2, 4) if lut_type = int8 (or int16, int32). The size of lut is c dim.

- shift: shift_type, shift_offset, shift_size, shift_shape

  The shift *Look Up Table* (LUT) is an offline table created after quantization. Where, lut_type, lut_shape, lut_offset and lut_size are the data type, table shape, table offset address and table size. The following is a simple equation, lut_size = data_type * lut_shape. Here, data type = 1 (or 2, 4) if lut_type = int8 (or int16, int32). The size of lut is c dim.

- lut: lut_type, lut_shape, lut_offset, lut_size

  The *Look Up Table* (LUT) is an offline table created after quantization. Where, lut_type, lut_shape, lut_offset and lut_size are the data type, table shape, table offset address and table size. The following is a simple equation, lut_size = data_type * lut_shape. Here, data type = 1 (or 2, 4) if lut_type = int8 (or int16, int32).

- norm_scale: norm_scale_type, norm_scale_value

  A scalar pair of quantization parameters related to the 'gamma' coefficient when the input data width is 16-bit.

- norm_shift: norm_shift_type, norm_shift_value

  A scalar pair of quantization parameters related to the 'gamma' coefficient.

- var_shift: var_shift_type, var_shift_value

  A scalar pair of quantization parameters related to the input data variance calculation when the input data width is 8-bit.

- axis

  A list of integers to indicate the axis to normalize across. Typically, this is the feature axis and the leaving dimensions are typically the batch axis. Generally, the value is -1 which represents the last dimension of the input tensor (X). Note that this parameter should be continuous from the last dimension. For example, axis = [1, 2, 3] means statistics are across the Height, Width and Channel dimensions in NHWC format.

- group

  Integer, the number of groups for Group Normalization. It can be in the range [1, N] where N is the input dimension. The input dimension must be divisible by the number of groups. The default value is 32.

- epsilon

  Float IR parameter. The epsilon value is used to avoid division by zero. Generally, the default value is less than 10^{-5}.

HardSigmoid

Applies the hardsigmoid function element-wise. That is, HardSigmoid(x) = max(0, min(1, alpha * X + beta)).

Inputs

- Input data tensor X.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = Activation'.

- method: HARDSIGMOID

  Method to perform the input tensor.

- alpha

  Means the coefficient in the HARDSIGMOID function in float IR. For example, the coefficient is 0.2 in TensorFlow while 0.1667 in ONNX and PyTorch frameworks.

- beta

  Means the coefficient in the HARDSIGMOID function in float IR. For example, the coefficient is 0.5 in TensorFlow, ONNX and PyTorch frameworks.

- lut: lut_type, lut_shape, lut_offset, lut_size

  The *Look Up Table* (LUT) is an offline table created during quantization. Where, lut_type, lut_shape, lut_offset and lut_size are the data type, table shape, table offset address and table size. The following is a simple relationship, lut_size = data_type * lut_shape. Where, data type = 1 (or 2, 4) if lut_type = int8 (or int16, int32).

HardSwish

Applies the hardswish function element-wise. That is, HardSwish(x) = 0 if x <= -3. HardSwish(x) = x if x >= 3. HardSwish(x) = (x * (x + 3)) /6 if -3 < x <3. Also, the formula can be expressed as HardSwish(x) = (x * Relu6(x + 3)) / 6.

Inputs

- Input data tensor X.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = Activation'.

- method: HARDSWISH

  Method to perform the input tensor.

- lut: lut_type, lut_shape, lut_offset, lut_size

  The *Look Up Table* (LUT) is an offline table created during quantization. Where, lut_type, lut_shape, lut_offset and lut_size are the data type, table shape, table offset address and table size. The following is a simple relationship, lut_size = data_type * lut_shape. Here, data type = 1 (or 2, 4) if lut_type = int8 (or int16, int32).

InTopK

Performs the input tensor (X) with 'batch_size' classes to check whether the targets are in the top k predictions, and outputs a 'batch_size' bool array. That is, the output Y[i] returns 'true' if the predication for the target class exists among the top 'k' predications.

Y[i] = 'true' if predications['target_classes_id (Si)'] ∈ {top-k predications}

Note that these classes are in the top 'k' when multiple classes have the same predication value and straddle the top 'k' boundary. For example, X1 = [[0.1, 0.8, 0.7, 0.7], [0.1, 0.6, 0.5, 0.4]], X2 = [3, 3], then returns Y = [true, false].

Inputs

- Input predication tensor X1, target tensor X2 (a 'batch_size' vector of target classes IDs).

Outputs

- Output indices tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = InTopK'.

- k

  A scalar of integer to indicate the number of top elements to look at for computing precision.

InstanceNormalization

Carries out instance normalization. That is, y = gamma * (x -mean) / sqrt(variance + epsilon) + beta, where mean and variance are computed per instance. Here, 'gamma' and 'beta' are the input dimensional scale and bias tensor of size channel in NHWC format.

Inputs

- Input data tensor X.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = InstanceNorm'.

- weights: weights_type, weights_offset, weights_size, weights_shape

  The 'gamma' tensor will be used in instance normalization computation. Where, weights_type (or weights_offset, weights_size, weights_shape) means the weights data format (or weights address offset, total weights size (counted in bytes), and weights shape of the current layer) after 'Gamma' and 'input_var' parameter conversion. Generally, 'weights_shape' equals the number of output channels.

- biases: biases_type, biases_offset, biases_size, biases_shape

  The 'beta' tensor will be used in instance normalization computation. Where, biases_type (or biases_offset, biases_size, biases_shape) means the biases data format (or biases address offset, total biases size which counted in bytes, and biases shape of the current layer) after 'Beta', 'input_mean', 'Gamma' and 'input_var' parameter conversion. Generally, 'biases_shape' equals the number of output channels.

- scale: scale_type, scale_value

  The scale is a scalar, where 'scale_type' and 'scale_value' are the data type and value of the scale during quantization. Generally, scale_type has the int8, uint8 and int16 options.

- shift: shift_type, shift_value

  It can be a scalar, which means an output shift operation during quantization, where 'shift_type' and 'shift_value' are the data type of the shift operation.

- lut: lut_type, lut_shape, lut_offset, lut_size

  The *Look Up Table* (LUT) is an offline table created during quantization. Where, lut_type, lut_shape, lut_offset and lut_size are the data type, table shape, table offset address and table size. The following is a simple relationship, lut_size = data_type * lut_shape. Here, data type = 1 (or 2, 4) if lut_type = int8 (or int16, int32).

- norm_scale: norm_scale_type, norm_scale_value

  A scalar pair of quantization parameters related to the 'gamma' coefficient when the input data width is 16-bit.

- norm_shift: norm_shift_type, norm_shift_value

  A scalar pair of quantization parameters related to the 'gamma' coefficient.

- var_shift: var_shift_type, var_shift_value

  A scalar pair of quantization parameters related to the input data variance calculation when the input data width is 8-bit.

- epsilon

  Float IR parameter. The epsilon value is used to avoid division by zero; Generally, the default value is less than 10^{-5}.

L1Normalization

Given a matrix, apply L1-normalization along the provided axis.

Inputs

- Input data tensor X.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = Normalization'.

- scale: scale_type, scale_value

  The scale is a scalar, where scale_type and scale_value are the data type and value of the scale during quantization. Generally, scale_type has int8, uint8, int16, and other data types.

- shift: shift_type, shift_value

  It can be a scalar, which means an output shift operation during quantization, where shift_type and shift_value are the data type of the shift operation.

- axis

  A list of integers to indicate the axis to normalize across. Typically, this is the feature axis and the leaving dimensions are typically the batch axis. Generally, the value is -1 which represents the last dimension of the input tensor (X). Note that this parameter should be continuous from the last dimension. For example, axis = [1, 2, 3] means statistics are across the Height, Width and Channel dimensions in NHWC format.

- method: L1

  The reduce method.

- epsilon

  Float IR parameter. The epsilon value is used to avoid division by zero. Generally, the default value is less than 10^{-5}.

L1Pooling2D

L1Pooling2D consumes an input tensor X and applies L1 pooling across the tensor according to kernel sizes, stride sizes, and pad lengths. L1 pooling consists of computing the L1 norm on all values of a subset of the input tensor according to the kernel size and downsampling the data into the output tensor Y for further processing.

Inputs

- Input data tensor X.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = Pooling'.

- kernel: kernel_x, kernel_y

  'kernel_x' ('kernel_y') is the kernel size along the 'width' ('height') axis.

- stride: stride_x, stride_y

  Stride along the 'batch', 'height', 'width', and 'channel' axis. Generally, the 'stride_x' ('stride_y') stride is along the width (height) axis. It defaults to 1 if not present.

- pad: pad_bottom, pad_top, pad_left, pad_right

  Padding for the beginning and ending along the spatial axis, it can take any value greater than or equal to 0. The value represents the number of pixels added to the beginning and end part of the corresponding axis. The 'pad' parameter format should be as follows (‘pad_bottom’, ‘pad_top’, ‘pad_left’, ‘pad_right’), where it means the pad pixels of the data cube (that is, bottom and top along the ‘height’ axis, left and right along the ‘width’ axis). If not present, compute and output the same shape as the input tensor.

- dilation: dilation_x, dilation_y

  The dilation value along the spatial axis of the filter. If not present, the dilation default value is 1 along each spatial axis. Where the dilation_x (dilation_y) means the height (width) axis of the filter in the NHWC data format. Generally, it defaults to 1 if not present.

- method: L1

  The downsampling method is average during computing pooling.

- ceil_mode: true, false

  Optional parameter. Where ‘true’ means using **Ceil** instead of the **Floor** function to compute the output shape. Generally, the default value is ‘false’.

- scale: scale_type, scale_value

  The scale is a scalar, where scale_type and scale_value are the data type and value of the scale during quantization. Generally, scale_type has int8, uint8, int16, and other data types.

- shift: shift_type, shift_value

  It can be a scalar, which means an output shift operation during quantization, where shift_type and shift_value are the data type of the shift operation.

L2Normalization

Given a matrix, apply L2-normalization along the provided axis.

Inputs

- Input data tensor X.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is ‘layer_type = Normalization’.

- scale: scale_type, scale_value

  The scale is a scalar, where scale_type and scale_value are the data type and value of the scale during quantization. Generally, scale_type has int8, uint8, int16, and other data types.

- shift: shift_type, shift_value

  It can be a scalar, which means an output shift operation during quantization, where shift_type and shift_value are the data type of the shift operation.

- axis

  A list of integers to indicate the axis to normalize across. Typically, this is the feature axis and the leaving dimensions are typically the batch axis. Generally, the value is -1 which represents the last dimension of the input tensor (X). Note that this parameter should be continuous from the last dimension. For example, axis = [1, 2, 3] means statistics are across the Height, Width and Channel dimensions in NHWC format.

- method: L2

  The reduce method.

- lut: lut_type, lut_shape, lut_offset, lut_size

  The *Look Up Table* (LUT) is an offline table created after quantization. Where, lut_type, lut_shape, lut_offset and lut_size are the data type, table shape, table offset address and table size. The following is a simple equation, lut_size = data_type * lut_shape. Here, data type = 1 (or 2, 4) if lut_type = int8 (or int16, int32).

- reciprocal_shift: reciprocal_shift_type, reciprocal_shift_value

  It can be a scalar during quantization, where 'reciprocal_shift_type' and 'reciprocal_shift_value' are the data type and value of the shift during reciprocal calculation quantization.

- epsilon

  Float IR parameter. The epsilon value is used to avoid division by zero. Generally, the default value is less than 10^{-5}.

L2Pooling2D

L2Pooling2D consumes an input tensor X and applies L2 pooling across the tensor according to kernel sizes, stride sizes, and pad lengths. L2 pooling consists of computing the L2 norm on all values of a subset of the input tensor according to the kernel size and downsampling the data into the output tensor Y for further processing.

Inputs

- Input data tensor X.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = Pooling'.

- kernel: kernel_x, kernel_y

  'kernel_x' ('kernel_y') is the kernel size along the 'width' ('height') axis.

- stride: stride_x, stride_y

  Stride along the 'batch', 'height', 'width', and 'channel' axis. Generally, the 'stride_x' ('stride_y') stride is along the width (height) axis. It defaults to 1 if not present.

- pad: pad_bottom, pad_top, pad_left, pad_right

  Padding for the beginning and ending along the spatial axis, it can take any value greater than or equal to 0. The value represents the number of pixels added to the beginning and end part of the corresponding axis. The 'pad' parameter format should be as follows ('pad_bottom', 'pad_top', 'pad_left', 'pad_right'), where it means the pad pixels of the data cube (that is, bottom and top along the 'height' axis, left and right along the 'width' axis). If not present, compute and output the same shape as the input tensor.

- dilation: dilation_x, dilation_y

  The dilation value along the spatial axis of the filter. If not present, the dilation default value is 1 along each spatial axis. Where the dilation_x (dilation_y) means the height (width) axis of the filter in the NHWC data format. Generally, it defaults to 1 if not present.

- method: L2

The downsampling method is average during computing pooling.

- ceil_mode: true, false

  Optional parameter. Where 'true' means using **Ceil** instead of the **Floor** function to compute the output shape. Generally, the default value is 'false'.

- scale: scale_type, scale_value

  The scale is a scalar, where scale_type and scale_value are the data type and value of the scale during quantization. Generally, scale_type has int8, uint8, int16, and other data types.

- shift: shift_type, shift_value

  It can be a scalar, which means an output shift operation during quantization, where shift_type and shift_value are the data type of the shift operation.

- sqrt_lut: sqrt_lut_type, sqrt_lut_shape, sqrt_lut_offset, sqrt_lut_size

  The *Look Up Table* (LUT) is an offline table created after quantization. Where, sqrt_lut_type, sqrt_lut_shape, sqrt_lut_offset and sqrt_lut_size are the data type, table shape, table offset address and table size. The following is a simple equation, sqrt_lut_size = sqrt_data_type * sqrt_lut_shape. Here, data type = 1 (or 2, 4) if l sqrt_ut_type = int8 (or int16, int32).

LRN

It normalizes over local input regions. The normalization function 'Y[i] = X[i] / (bias + (alpha/size) * square_sum(X[j]))^beta', is applied to the tensor elementwise. There are two methods of 'ACROSS_CHANNELS' and 'WITHIN_CHANNEL' during local region normalization.

- When method is ACROSS_CHANNELS,

  The local region of 'square_sum' is defined as '{max(0, c - depth_radius) <= i <= min(C - 1, c + depth_radius)}' under NHWC data format. Also, square_sum[i] = sum(input tensor X[N, H, W, i - depth_radius : i + depth_radius + 1]^2).

- When method is WITHIN_CHANNEL,

  The local region is rectangular and defined as 'max{0, h - depth_radius} <= y <= min(H - 1, h + depth_radius + 1)}, max{0, w - depth_radius} <= x <= min(W - 1, w + depth_radius + 1)}' under NHWC data format. Also, square_sum[i] = sum(input tensor X[N, (y - depth_radius) : (y + depth_radius + 1), (x - depth_radius) : (x + depth_radius + 1), C]^2).

Inputs

- Input data tensor X.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = LRN'.

- size

  The number of channels to sum over.

- alpha

  Scaling parameter. It defaults to '1' and is usually positive.

- beta

  The exponent. It defaults to 0.5.

- bias

  Optional factor. It defaults to 1 and an offset that is usually positive to avoid dividing by 0.

- method: ACROSS_CHANNELS, WITHIN_CHANNEL

  Indicates the local normalization region. It defaults to ACROSS_CHANNELS.

- scale_sum: scale_sum_type, scale_sum_value

  It can be a scalar during the sum of square calculation quantization, where scale_sum_type and scale_sum_value are the data type and value of the scale during per tensor or layer quantization.

- shift_sum: shift_sum_type, shift_sum_value

  It can be a scalar during the sum of square calculation quantization, where shift_sum_type, shift_sum_value are the data type and value of the shift during per tensor or layer quantization.

- scale: scale_type, scale_value

  It can be a vector during quantization, where scale_type and scale_value are the data type and value of the scale during per tensor or layer quantization.

- shift: shift_type, shift_value

  It can be a scalar during quantization, where shift_type and shift_value are the data type and value of the shift during per tensor or layer quantization.

- lut: lut_type, lut_shape, lut_offset, lut_size

  The *Look Up Table* (LUT) is an offline table created after quantization. Where, lut_type, lut_shape, lut_offset and lut_size are the data type, table shape, table offset address and table size. The following is a simple equation, lut_size = data_type * lut_shape. Where, data type = 1 (or 2, 4) if lut_type = int8 (or int16, int32).

LayerNormalization

Computes the activation normalization of the previous layer for the given axis in a batch independently. That is, y = gamma * (x -mean) / sqrt(variance + epsilon) + beta, where mean and variance are computed under all effective axis. By default, the shape of ‘gamma’ and ‘beta’ are the batch size in NHWC format.

Inputs

- Input data tensor X.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is ‘layer_type = LayerNorm’.

- weights: weights_type, weights_offset, weights_size, weights_shape

  The ‘gamma’ tensor will be used in instance normalization computation. Where, weights_type, weights_offset, weights_size and weights_shape mean the weights data format, weights address offset, total weights size (counted in bytes) and weights shape of the current layer. Generally, ‘weights_shape’ is broadcastable or the same shape as the input tensor’s dimension that ‘axis’ specifies.

- biases: biases_type, biases_offset, biases_size, biases_shape

The 'beta' tensor will be used in instance normalization computation. Where, biases_type, biases_offset, biases_size and biases_shape mean the biases data format, biases address offset, total biases size (counted in bytes) and biases shape of the current layer. Generally, 'biases_shape' is broadcastable or the same shape as the input tensor's dimension that 'axis' specifies.

- scale: scale_type, scale_value

  The scale is a scalar, where scale_type and scale_value are the data type and value of the scale during quantization. Generally, scale_type has int8, uint8, int16, and other data types.

- shift: shift_type, shift_value

  It can be a scalar, which means an output shift operation during quantization, where shift_type and shift_value are the data type of the shift operation.

- lut: lut_type, lut_shape, lut_offset, lut_size

  The *Look Up Table* (LUT) is an offline table created after quantization. Where, lut_type, lut_shape, lut_offset and lut_size are the data type, table shape, table offset address and table size. The following is a simple equation, lut_size = data_type * lut_shape. Here, data type = 1 (or 2, 4) if lut_type = int8 (or int16, int32).

- norm_scale: norm_scale_type, norm_scale_value

  A scalar pair of quantization parameters related to the 'gamma' coefficient when the input data width is 16-bit.

- norm_shift: norm_shift_type, norm_shift_value

  A scalar pair of quantization parameters related to the 'gamma' coefficient.

- var_shift: var_shift_type, var_shift_value

  A scalar pair of quantization parameters related to the input data variance calculation when the input data width is 8-bit.

- axis

  A list of integers to indicate the axis to normalize across. Typically, this is the feature axis and the leaving dimensions are typically the batch axis. Generally, the value is -1 which represents the last dimension of the input tensor (X). Note that this parameter should be continuous from the last dimension. For example, axis = [1, 2, 3] means statistics are across the Height, Width and Channel dimensions in NHWC format.

- epsilon

  Float IR parameter. The epsilon value is used to avoid division by zero. Generally, the default value is less than 10^{-5}.

LeakyRelu

Takes one input tensor (X) and produces one output tensor (Y), where the rectified linear function 'y = alpha * x for x < 0, y = x for x >= 0', is applied to the tensor elementwise.

Inputs

- Input data tensor X.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = Activation'.

- method: LEAKYRELU

Method to perform the input tensor.

- negative_slop: negative_slope_type, negative_slope_value

  It can be a scalar during quantization of alpha in the equation 'y = alpha * x for x < 0, y = x for x >= 0', where negative_slope_type and negative_slope_value are the data type and value of the shift during per tensor or layer quantization.

- scale: scale_type, scale_value

  It can be a scalar during quantization, where scale_type and scale_value are the data type and value of the scale during per tensor or layer quantization.

- shift: shift_type, shift_value

  It can be a scalar during quantization, where shift_type and shift_value are the data type and value of the shift during per tensor or layer quantization.

LeftShift

The left shift operator performs element-wise operations. For each input element, the bits of binary representation move toward the left side, which results in the increase of its actual value. The input X is the tensor to be shifted and another input Y specifies the amounts of shifting. For example, X=[1, 2] and S=[1, 2], the corresponding output Y will be [2, 8]. This operator supports multidirectional broadcasting.

Inputs

- Input data tensor X.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = Bitshift'.

- direction: LEFT

  Direction of moving bits.

Log

Calculates the natural log of the given input tensor (X), where the function f(x) = loge(x), is applied to the tensor elementwise.

Inputs

- Input data tensor X.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = Log'.

- lut: lut_type, lut_shape, lut_offset, lut_size

The *Look Up Table* (LUT) is an offline table created during quantization. Where, lut_type, lut_shape, lut_offset and lut_size are the data type, table shape, table offset address and table size. The following is a simple equation, lut_size = data_type * lut_shape. Where, data type = 1 (or 2, 4) if lut_type = int8 (or int16, int32).

LogSoftmax

Computes log softmax.

$$Y = x - \log\left(reduce\_sum(\exp(x)\,, axis)\right)$$

Inputs

- Input data tensor X1, second input data tensor X2 (optional).

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = LogSoftmax'.

- axis

  The dimension that Softmax will be performed on. The accepted range is [-1, r-1] where r = rank(input data).

- scale: scale_type, scale_value

  It can be a scalar during quantization, where scale_type and scale_value are the data type and value of the scale during per tensor or layer quantization.

- shift: shift_type, shift_value

  It can be a scalar during quantization, where scale_type and scale_value are the data type and value of the scale during per tensor or layer quantization.

- lut_exp: lut_exp _type, lut_exp _offset, lut_exp _size, lut_exp _shape

  The *Look Up Table* (LUT) is an offline table created during quantization. Where, lut_exp _type, lut_exp _shape, lut_exp _offset and lut_exp _size are the data type, table shape, table offset address and table size. The following is a simple equation lut_exp _size = data_type * lut_exp _shape. Where, data type = 1 (or 2, 4) if lut_exp _type = int8 (or int16, int32).

- lut_log: lut_log _type, lut_log _offset, lut_log _size, lut_log _shape

  The *Look Up Table* (LUT) is an offline table created during quantization. Where, lut_log _type, lut_log _shape, lut_log _offset and lut_log _size are the data type, table shape, table offset address and table size. The following is a simple equation, lut_log _size = data_type * lut_log _shape. Where, data type = 1 (or 2, 4) if lut_log _type = int8 (or int16, int32).

Logical

Returns the tensor resulted from performing logical operation elementwise on the input tensor. It supports multidirectional broadcasting as NumPy-style and returns a tensor of Boolean values.

Inputs

- First input data tensor X1, second input data tensor X2 (optional).

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = Logical'.

- method: EQUAL, NOT_EQUAL, GREATER, GREATER_EQUAL, LESS, LESS_EQUAL, OR, AND, XOR, NOT

  Logical operation method. Where, EQUAL returns the element-wise truth value of (x == y). NOT_EQUAL returns the element-wise truth value of (x != y). GREATER returns the element-wise truth value of (x > y). GREATER_EQUAL returns the element-wise truth value of (x >= y). LESS returns the element-wise truth value of (x < y). LESS_EQUAL returns the element-wise truth value of (x <= y). OR returns the element-wise truth value of 'x OR y'. AND returns the element-wise truth value of 'x AND y'. XOR returns the element-wise truth value of '(x OR y) AND (!(x AND y))'. NOT returns the element-wise truth value of 'NOT x'.

- scale: scale_type, scale_value

  It can be optional or a 1-D tensor during quantization in order of input tensor [X1, X2], where scale_type and scale_value are the data type and value of the scale during per tensor or layer quantization.

- shift: shift_type, shift_value

  It can be optional or a 1-D tensor during quantization in order of input tensor [X1, X2], where shift_type and shift_value are the data type and value of the shift during per tensor or layer quantization.

MatMul

Matrix product of two input tensors (A, B) as NumPy-style. If either argument is N-D, N>2, it is treated as a stack of matrices residing in the last two indexes and broadcast accordingly. Besides, multiplication by a scalar is not allowed.

Inputs

- Input data tensor A, data tensor B.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = MatMul'.

- trans_a: true, false

  If 'true', input tensor A is transposed before multiplication.

- tans_b: true, false

  If 'true', input tensor B is transposed before multiplication.

- scale: scale_type, scale_value

  It can be a scalar during quantization, where scale_type and scale_value are the data type and value of the scale during per tensor or layer quantization.

- shift: shift_type, shift_value

  It can be a scalar during quantization, where scale_type and scale_value are the data type and value of the shift during per tensor or layer quantization.

MaxPooling2D

MaxPooling-2D takes an input tensor X and applies max pooling across the tensor by the kernel size, stride size, and padding. MaxPooling-2D consists of computing the max on all values of a subset of the input tensor according to the kernel size and downsampling the data into the output tensor Y for further processing. If no padding is present, the 'pad' group parameters default to keep the shape the same as the input tensor during computing.

Inputs

- Input data tensor X.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = Pooling'.

- kernel: kernel_x, kernel_y

  'kernel_x' ('kernel_y') is the kernel size along the 'width' ('height') axis.

- stride: stride_x, stride_y

  Stride along the 'batch', 'height', 'width' and 'channel' axis. Where the 'stride_x' ('tride_y') stride is along the 'width' ('height') axis in the NHWC data format. It defaults to 1 if not present.

- pad: pad_bottom, pad_top, pad_left, pad_right

  Padding for the beginning and ending along spatial axis, it can take any value greater than or equal to 0. The value represents the number of pixels added to the beginning and end part of the corresponding axis. The 'pad' parameter format should be as follows 'pad_bottom', 'pad_top', 'pad_left', 'pad_right', where it means the pad pixels of the data cube (that is, bottom and top along the 'height' axis; left and right along the 'width' axis). If not present, compute and output the same shape as the input tensor.

- dilation: dilation_x, dilation_y

  The dilation value along the spatial axis of the filter. If not present, the dilation default value is 1 along each spatial axis. Where dilation_x (dilation_y) means the 'width' ('height') axis of the filter. It defaults to 1 if not present.

- method: MAX

  The downsampling method is 'Maximum' during computing pooling.

- ceil_mode: true, false

  Optional parameter. Where 'true' means using **Ceil** instead of the **Floor** function to compute the output shape. Generally, the default value is 'false'.

MaxPooling3D

MaxPooling-3D takes an input tensor X and applies max pooling across the tensor by the kernel size, stride size, and padding. MaxPooling-3D consists of computing the max on all values of a subset of the input tensor according to the kernel size and downsampling the data into the output tensor Y for further processing. If no padding is present, the 'pad' group parameters keep the shape as the input tensor by default during computing. Generally, the output shape can be calculated as:

output_shape[i] = round(input_shape + pad_begin[i] + pad_end[i] - ((kernel_size[i] -1) * dilation[i]+1) / stride[i] + 1)

Where 'round' represents the **floor** or **ceil** function during the round operation.

Inputs

- Input data tensor X.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = Pooling3D'.

- kernel: kernel_x, kernel_y, kernel_z

  'kernel_x' ('kernel_y', 'kernel_z') is the kernel size along the 'width' ('height', 'depth') axis.

- stride: stride_x, stride_y, stride_z

  The stride of the sliding window for each dimension of the input tensor. Where the 'stride_x' ('stride_y', 'stride_z') stride is along the 'width' ('height', 'depth') axis under the NDHWC data format.

- pad: pad_x_begin, pad_x_end, pad_y_begin, pad_y_end, pad_z_begin, pad_z_end

  Padding for the beginning and ending along the spatial axis, it can take any value greater than or equal to 0. The value represents the number of pixels added to the beginning and end part of the corresponding axis. If not present, it is provided by the shape of the output.

- dilation: dilation_x, dilation_y, dilation_z

  The dilation value along the spatial axis of the filter. If not present, the dilation default value is 1 along each spatial axis. Where, 'dilation_x' ('dilation_y' or 'dilation_z') means the Height (Width or Depth) dimension of the filter under the NDHWC data format.

- method: MAX

  The downsampling method is 'Maximum' during computing pooling.

- ceil_mode: true, false

  Optional parameter. Where 'true' means using **Ceil** instead of the **Floor** function to compute the output shape. Generally, the default value is 'false'.

MaxPoolingWithArgMax

Performs max pooling on the inputs and returns both max values and indices. Generally, the indices in ArgMax are flattened, so a maximum value at position '[n, h, w, c]' will be flattened as:

'indices = h * width + w' if flatten_dim = HW

'indices = (h * width + w) * channel + c' if flatten_dim = HWC

'indices = ((n * height + h) * width + w) * channel + c' if flatten_dim = NHWC

Where, height (or width, channel) represents the size of corresponding H (or W, C) dimension under NHWC data format. Generally, the indices are located at ([0, height], [0, width]) originally before flattening even though padding is involved. Where, the range is a left closed and right open interval.

Inputs

- Input data tensor X.

Outputs

- Output maximum value tensor Y1, argmax indices tensor Y2.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = MaxPoolingWithArgMax'.

- kernel: kernel_x, kernel_y

  'kernel_x' ('kernel_y') is the kernel size along the 'width' ('height') axis. If not present, it should be inferred from inputs 'weights_size' and 'weight_shape'.

- stride: stride_x, stride_y

  Stride along the 'batch', 'height', 'width' and 'channel' axis. Generally, the 'stride_x' ('stride_y') stride is along the 'width' ('height') axis.

- dilation: dilation_x, dilation_y

  The dilation value along the spatial axis of the filter. If not present, the dilation default value is 1 along each spatial axis. Where dilation_x (dilation_y) means the height (width) axis of the filter in the NHWC data format.

- flatten_dim: NHWC, HWC, HW, NCHW

  A flatten mode to output the indices. NHWC means including batch dimension in the flattened index of ArgMax. HWC means including Height and Width dimensions in the flattened index of ArgMax. HW means including the Width dimension in the flattened index of ArgMax. Generally, it defaults to HWC mode.

- pad: pad_bottom, pad_top, pad_left, pad_right

  Padding for the beginning and end along the spatial axis, it can take any value greater than or equal to 0. The value represents the number of pixels added to the beginning and end part of the corresponding axis. The 'pad' parameter format should be as follows ('pad_bottom', 'pad_top', 'pad_left', 'pad_right'), where it means the pad pixels of the data cube (that is, bottom and top along the 'height' axis, left and right along the 'width' axis). If not present, compute and output the same shape as the input tensor.

- storage_order:

  An int value of storage_order of the output index tensor. The default value is 0. 0 is row-major, and 1 is column-major.

- ceil_mode: true, false

  Optional parameter. Where 'true' means using **Ceil** instead of the **Floor** function to compute the output shape. Generally, the default value is 'false'.

MaxRoiPool

The *Region of Interest* (ROI) pooling is used for converting all the proposals to fixed shape as required by the next special layers, especially in object detection network. That is, ROI pooling produces the fixed size of feature maps from non-uniform inputs by commonly performing max-pooling on the input tensors. Generally, the number of output channels is equal to the number of input channels for this layer. It takes two inputs—one is the feature map obtained from a Convolutional Neural Network after multiple convolutions and pooling layers and the other is 'num_rois' proposal or ROIs from region proposal network (so-called bounding box). Each proposal has five values—the first one indicates the 'batch_indices' and the rest of the four are proposal coordinates in original image. The four coordinates indicate the top-left and bottom-right corner coordinates of the proposal and the shape will be [num_rois, 5]. Where the value '5' means [batch_indices, y1, x1, y2, x2].

Inputs

- Input data tensor X (the shape is [N, H, W, C]), bbox (bounding box and the shape is [num_rois, 5]).

Outputs

- Output tensor Y with the shape as [num_rois, H, W, C].

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = MaxRoiPool'.

- pooled_shape

  A list to indicate the output shape of ROI pooing, in order of [Height, Width].

- spatial: spatial_x, spatial_y

  A list which is in order of [spatial_y, spatial_x], means the multiplicative spatial scale factor to translate ROI coordinates from the input scale used during the pooling operation. That is, spatial_y (spatial_x) = feature_map_H (W) / original_image_H (W).

MaxUnpool

MaxUnpool essentially computes the partial inverse of the MaxPool operator. The input information to this operator is typically the output information from a MaxPool operator. The first input tensor X is the tensor that needs to be unpooled, which is typically the pooled tensor (first output) from MaxPool. The second input tensor, I, contains the indices to the (locally maximal) elements corresponding to the elements in the first input tensor X. Input tensor I is typically the second output of the MaxPool operator. The third (optional) input is a tensor that specifies the output size of the unpooling operation.

MaxUnpool is intended to perform 'partial' inverse of the MaxPool operator. The inverse is 'partial' because all the non-maximal values from the original input to MaxPool are set to zero in the output of the MaxUnpool operator. Pooling the result of an unpooling operation should give back the original input to the unpooling operator. MaxUnpool can produce the same output size for several input sizes, which makes the unpooling operator ambiguous. The third input argument, output_size, is meant to disambiguate the operator and produce the output tensor of known/predictable size. In addition to the inputs, MaxUnpool takes three attributes, namely kernel_shape, strides, and pads, which define the exact unpooling operator. The attributes typically have the same values as the corresponding pooling operator that the unpooling operator is trying to invert.

Inputs

- Input data tensor X, Y.

Outputs

- Output maximum value tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = MaxUnpool'.

- flatten_dim: NHWC, HWC, HW, NCHW

  A flatten mode to output the indices. NHWC means including batch dimension in the flattened index of ArgMax. HWC means including Height and Width dimensions in the flattened index of ArgMax. HW means including the Width dimension in the flattened index of ArgMax. Generally, it defaults to HWC mode.

- pad: pad_bottom, pad_top, pad_left, pad_right

  Padding for the beginning and end along the spatial axis, it can take any value greater than or equal to 0. The value represents the number of pixels added to the beginning and end part of the corresponding axis. The 'pad' parameter format should be as follows ('pad_bottom', 'pad_top', 'pad_left', 'pad_right'), where it means the pad pixels of the data cube (that is, bottom and top along the ‘height’ axis, left and right along the ‘width’ axis). If not present, compute and output the same shape as the input tensor. Only for float IR.

- storage_order:

  An int value of storage_order of the output index tensor. The default value is 0. 0 is row-major, and 1 is column-major.

MeanVarianceNormalization

Performs mean variance normalization on the input data tensor (X). The formula is ‘Y=(X-E(X)) / sqrt(Z), where Z = E(X - E(X))\^2’.

Inputs

- Input data tensor X.

Outputs

- Output data tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is ‘layer_type = MVN’.

- axis

  A list of integers, along which to reduce. A default value [0, 1, 2] means calculating mean and variance along 0, 1, 2 dimension. Two given variables with the same channel-coordinate share the same mean and variance parameters.

- epsilon

  A float IR parameter. The epsilon value is used to avoid division by zero. Generally, the default value is less than 10^{-5}.

- scale: scale_type, scale_value

  The scale is a scalar, where scale_type and scale_value are the data type and value of the scale during quantization. Generally, the scale_type has the int8, uint8, in16 and uint16 options.

- shift: shift_type, shift_value

  It can be a scalar, which means an output shift operation during quantization, where shift_type and shift_value are the data type of the shift operation.

- norm_scale: norm_scale_type, norm_scale_value

  A scalar pair of quantization parameters related to the ‘gamma’ coefficient when the input data width is 16-bit.

- norm_shift: norm_shift_type, norm_shift_value

  A scalar pair of quantization parameters related to the ‘gamma’ coefficient.

- var_shift: var_shift_type, var_shift_value

  A scalar pair of quantization parameters related to the input data variance calculation when the input data width is 8-bit.

- lut: lut_type, lut_shape, lut_offset, lut_size

  The *Look Up Table* (LUT) is an offline table created after quantization. Where, lut_type, lut_shape, lut_offset and lut_size are the data type, table shape, table offset address and table size. The following is a simple equation, lut_size = data_type * lut_shape. Where, data type = 1 (or 2, 4) if lut_type = int8 (or int16, int32).

Meshgrid

Returns coordinate matrices from coordinate vectors.

Makes N-D coordinate arrays for vectorized evaluations of N-D scalar/vector fields over N-D grids, given one-dimensional coordinate arrays x1, x2, …, xn.

This function supports both indexing conventions through the indexing keyword argument. Giving the string 'ij' returns a meshgrid with matrix indexing, while 'xy' returns a meshgrid with Cartesian indexing. In the 2-D case with inputs of length M and N, the outputs are of shape (N, M) for 'xy' indexing and (M, N) for 'ij' indexing. In the 3-D case with inputs of length M, N and P, outputs are of shape (N, M, P) for 'xy' indexing and (M, N, P) for 'ij' indexing.

Inputs

- Input data tensor X1, X2, …, XN.

Outputs

- Output tensor Y1, Y2, …, YN.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = Meshgrid'.

- indexing: xy/ij

  Cartesian ('xy', default) or matrix ('ij') indexing of output.

- sparse: true / false

  If it is 'true', the shape of the returned coordinate array for dimension i is reduced from (N1, …, Ni, … Nn) to (1, …, 1, Ni, 1, …, 1). These sparse coordinate grids are intended to be used with broadcasting. When all coordinates are used in an expression, broadcasting still leads to a fully-dimensional result array.

  By default, it is 'false'.

- copy: true / false

  If it is 'false', a view into the original arrays is returned to conserve memory. By default, it is 'true'. Note that sparse=false, copy=false will likely return non-contiguous arrays. Furthermore, more than one element of a broadcast array may refer to a single memory location. If you need to write to the arrays, make copies first.

Mish

A self-regularized non-monotonic neural activation function. Computes mish activation.

$$mish(x) = x * tanh\ (softplus(x))$$

Inputs

- Input data tensor X.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = Activation'.

- method: MISH

  Method to perform the input tensor.

- lut: lut_type, lut_shape, lut_offset, lut_size

  The *Look Up Table* (LUT) is an offline table created after quantization. Where, lut_type, lut_shape, lut_offset and lut_size are the data type, table shape, table offset address and table size. The following is a simple equation, lut_size = data_type * lut_shape. Where, data type = 1 (or 1, 2, 2, 4, 4) if lut_type = int8 (or uint8, int16, uint16, int32, uint32).

Mod

Computes the element-wise remainder of division. The sign of the remainder is the same as that of the dividend. Besides, the mod operation can also behave like the **fmod** function in C or NumPy. This operator also supports broadcasting and multidirectional broadcasting as NumPy style.

Inputs

- Input dividend tensor X1, input divisor tensor X2.

Outputs

- Output remainder tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = Mod'.

- fmod: bool (default is false)

  Whether the operator should behave like fmod (default=false means that it will perform integer mods). Set this to true to force fmod treatment.

- scale: scale_type, scale_value

  It can be a vector with two elements. Where scale_type and scale_value are the data type and value of the scale during quantization. Generally, scale_type has the int8, uint8 and int16 options. Besides, the scales type or value must be in order of [scale_X1, scale_X2].

- shift: shift_type, shift_value

  It can be a vector with two elements. Where shift_type and shift_value are the data type and value of the shift during quantization. Besides, the scales type or value must be in order of [shift_X1, shift_X2].

Moments

Computes the mean and variance of the input tensor (X) along the specified axis. Especially, if input tensor (X) is a 1-D format and 'axis=[0]', the result will be the mean and variance of a vector.

Inputs

- Input data tensor X.

Outputs

- Output mean tensor Y1, variance tensor Y2.

Attributes

- layer_type

The operation type of a layer. Here is ‘layer_type = Moments’.

- axis

  A list along which dimension to compute the mean and variance.

- keepdims

  Produces moments with the same dimensionality as the input tensor (X).

- var_scale: var_scale_type, var_scale_value

  It can be a scalar during quantization, where scale_type and scale_value are the data type and value of the scale during per var output tensor or layer quantization.

- var_shift: var_shift_type, var_shift_value

  It can be a scalar during quantization, where scale_type and scale_value are the data type and value of the scale during per var output tensor or layer quantization.

- input_scale: input_scale_type, input_scale_value

  It can be a scalar during quantization, where scale_type and scale_value are the data type and value of the scale during per input tensor or layer quantization.

- input_shift: input_shift_type, input_shift_value

  It can be a scalar during quantization, where scale_type and scale_value are the data type and value of the scale during per input tensor or layer quantization.

Mul

Performs multiplication of each of the input tensors (with NumPy-style broadcasting support). All inputs and outputs must have the same data type. Besides, this operator supports multidirectional (that is, NumPy-style) broadcasting.

Inputs

- Input data tensor X1, data tensor X2.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is ‘layer_type = Add’.

- scale: scale_type, scale_value

  The scale is a scalar, where scale_type and scale_value are the data type, value of the scale during quantization. Generally, the scale_type has the int8, uint8 and int16 options.

- shift: shift_type, shift_value

  It can be a scalar, which means an output shift operation during quantization. Where, shift_type and shift_value are the data type of the shift operation.

MultiboxTransformLoc

Location transformation for multibox detection.

Note that, in the following Inputs or Outputs sections:

- X1 means the input class probabilities.
- X2 means the input location regression predictions.
- X3 means the input prior anchor boxes.
- Y1 means the output box.
- Y2 means the valid count of output box.

Inputs

- Input tensor X1, X2, X3.

Outputs

- Output tensor Y1, Y2.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = MultiboxTransformLoc'.

- th_lut: th_lut_type, th_lut_shape, th_lut_offset, th_lut_size

  The look up table (th_lut) is an offline table created after quantization for the gaussian method. Where, th_lut_type, th_lut_shape, th_lut_offset, and th_lut_size are the data type, table shape, table offset address, and table size. The following is a simple equation, lut_size = data_type * lut_shape. Where, data type = 1 (or 2, 4) if lut_type = int8 (or int16, int32).

- tw_lut: tw_lut_type, tw_lut_shape, tw_lut_offset, tw_lut_size

  The look up table (tw_lut) is an offline table created after quantization for the gaussian method. Where, th_lut_type, th_lut_shape, th_lut_offset, and th_lut_size are the data type, table shape, table offset address, and table size. The following is a simple equation, lut_size = data_type * lut_shape. Where, data type = 1 (or 2, 4) if lut_type = int8 (or int16, int32).

- box_scale : box_scale_type, box_scale_value

  The scale is a vector with five elements in order of [y, x, h, w, anchor]. Where, scale_type and scale_value are the data type, value of the scale during quantization. Generally, scale_type has the int8, uint8 and int16 options.

- box_shift: box_shift_type, box_shift_value

  The shift is a vector with five elements in order of [y, x, h, w, anchor]. Where, shift_type and shift_value are the data type, value of the shift during quantization.

- score_threshold: score_threshold_type, score_threshold_value

  The threshold for deciding when to remove boxes based on the score. It is a scalar. Where, score_threshold_type and score_threshold_value are the data type, value of the shift during quantization.

- score_scale : score_scale_type, score_scale_value

  The score_scale is a scalar value. Where, score_scale_type and score_scale_value are the data type, value of the scale during quantization. Generally, scale_type has the int8, uint8 and int16 options.

- score_shift : score_shift_type, score_shift_value

  The score_shift is a scalar value. Where, score_shift_type and score_shift_value are the data type, value of the shift during quantization.

- delta_shift

  It is a scalar value during quantization for box regression to avoid the result exceeding 32 bits.

- variances : float value

  A list of variances to be decoded from box regression output in order of [y, x, h, w].

NMS

The *Non-Maximum Suppression* (NMS) filters out boxes that have high *Intersection-Over-Union* (IOU) overlap with previously selected boxes. Bounding boxes with scores less than Score_Threshold are dropped or discarded in the last output. Bounding box are supplied as [y1, x1, y2, x2], where (y1, x1) and (y2, x2) are the coordinates of any diagonal pair of box corners and the coordinates can be provided as normalized (that is, lying in the interval [0, 1]) or absolute. Note that this algorithm is agnostic to where the origin is in the coordinate system and more generally is invariant to orthogonal transformations and translations of the coordinate system. Therefore, translating or reflections of the coordinate system result in the same boxes being selected by the algorithm. The selected output is a set of integers indexing into the input collection of bounding boxes representing the selected boxes. The bounding box coordinates corresponding to the selected indices can be obtained using the Gather or GatherND operation in the end. This operator also supports a Soft-NMS (with Gaussian weighting) mode (c.f. Bodla et al, https://arxiv.org/abs/1704.04503) where boxes reduce the score of other overlapping boxes instead of directly causing them to be pruned. To enable this Soft-NMS mode, set the soft_nms_sigma parameter to be larger than 0.

Note that, in the following Inputs or Outputs sections:

- X1 means the coordinates of the maximum output boxes per batch, of which the shape is [batch_size, num_boxes, 4].
- X2 means the boxes number of every class per batch, of which the shape is [batch_size, num_classes].
- X3 means the valid classes of every batch, of which the shape is [batch_size, 1].
- X4 means the scores of all the boxes per batch, of which the shape is [batch_size, num_boxes].
- Y1 means the coordinates of the selected boxes per batch, of which the shape is the same as X1.
- Y2 means the boxes number of every class per batch after selection, of which the shape is the same as X2.
- Y3 means the scores of all the selected boxes per batch, of which the shape is the same as X4.
- Y4 means the indices of the selected boxes per batch in input tensor X4, of which the shape is the same as X4 as well.

Inputs

- Input tensor X1, X2, X3, X4.

Outputs

- Output tensor Y1, Y2, Y3, Y4.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = NMS'.

- center_point_box: 0, 1

  An integer to indicate the format of the box data. Generally, '0' means the box data is supplied as [y1, x1, y2, x2], where (y1, x1) and (y2, x2) are the coordinates of any diagonal pair of box corners and the coordinates can be provided as normalized or absolute (mainly for TensorFlow framework). '1'means the box data is supplied as [y_center, x_center, height, width] (mainly for PyTorch framework).

- image: image_width, image_height

  The float IR parameters during normalized coordinate quantization, which indicate the width and height size of the input feature map.

- iou_threshold

  Means the threshold for deciding whether boxes overlap too much with respect to IOU.

- iou_threshold_shift

It can be a scalar during 'iou_threshold' quantization. Generally, the data type is int8.

- method: HARD/GAUSSIAN/LINEAR

  The method for NMS to update the box score.

- score_threshold

  The threshold for deciding when to remove boxes based on the score. It is a scalar.

- max_output_size

  Integer representing the maximum number of boxes to be selected per batch per class. It is a scalar.

- areas_shift

  It can be a scalar during quantization for IOU.

- soft_nms_sigma

  A float value of gaussian NMS for the sigma parameter.

- soft_nms_sigma_in_shift

  An optional parameter used for gaussian NMS quantization.

- scale: scale_type, scale_value

  It can be a scalar during quantization, where scale_type and scale_value are the data type and value of the scale during box coordinates quantization.

- shift: shift_type, shift_value

  It can be a scalar during quantization, where scale_type and scale_value are the data type and value of the scale during box coordinates quantization.

- gaussian_scale_lut: gaussian_scale_lut_type, gaussian_scale_lut_shape, gaussian_scale_lut_offset, gaussian_scale_lut_size

  The look up table (gaussian_scale_lut) is an offline table created after quantization for the gaussian method. Where, gaussian_scale_lut_type, gaussian_scale_lut_shape, gaussian_scale_lut_offset, and gaussian_scale_lut_size are the data type, table shape, table offset address, and table size. The following is a simple equation, lut_size = data_type * lut_shape. Where, data type = 1 (or 2, 4) if lut_type = int8 (or int16, int32).

- gaussian_shift_lut: gaussian_shift_lut_type, gaussian_shift_lut_shape, gaussian_shift_lut_offset, gaussian_shift_lut_size

  The look up table (gaussian_shift_lut) is an offline table created after quantization for the gaussian method. Where, gaussian_shift_lut_type, gaussian_shift_lut_shape, gaussian_shift_lut_offset, and gaussian_shifte_lut_size are the data type, table shape, table offset address, and table size. The following is a simple equation, lut_size = data_type * lut_shape. Where, data type = 1 (or 2, 4) if lut_type = int8 (or int16, int32).

Negative

Computes numerical negative value to the input tensor X by elements-wise and returns the same type output tensor Y. The equation is Y(i) = -X(i), where 'i' means the index.

Inputs

- Input data tensor X.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is ‘layer_type = Negative’.

- scale: scale_type, scale_value

  It can be a scalar during quantization, where scale_type and scale_value are the data type and value of the scale during per tensor or layer quantization.

- shift: shift_type, shift_value

  It can be a scalar during quantization, where scale_type and scale_value are the data type and value of the scale during per tensor or layer quantization.

NormalizedMoments

Calculates the mean and variance based on the sufficient statistics.

Note that, in the following Inputs or Outputs sections:

- X1 means the input sufficient statistics mean.
- X2 means the input sufficient statistics variance.
- X3 means the input shift value for input mean and variance data.
- Y1 means the output mean value.
- Y2 means the output variance value.

Inputs

- Input data tensor X1, X2, X3.

Outputs

- Output tensor Y1, Y2.

Attributes

- layer_type

  The operation type of a layer. Here is ‘layer_type = NormalizedMoments’.

- counts:

  A parameter containing the total count of the data.

- mean_scale: mean_scale_type, mean_scale_value

  The mean_scale is a vector with more than two elements, and its number of elements should be equal to the sum of output and input tensors. Where, mean_scale_type and mean_scale_value are the data type, value of the mean_scale during quantization. Generally, mean_scale_type has the int8, uint8 and int16 options. Furthermore, the scales type or value must be in order of ‘output_scale, input0_scale, input1_scale’.

- mean_shift: mean_shift_type, mean_shift_value

  It can be a scalar, which means an output shift operation during mean quantization. Where, mean_shift_type and mean_shift_value are the data type of the shift operation.

- var_scale: var_scale_type, var_scale_value

The var_scale is a vector with more than two elements, and its number of elements should be equal to the sum of output and input tensors. Where, var_scale_type and var_scale_value are the data type, value of the var_scale during quantization. Generally, var_scale_type has the int8, uint8 and int16 options. Furthermore, the scales type or value must be in order of 'output_scale, input0_scale, input2_scale'.

- var_shift: var_shift_type, var_shift_value

  It can be a scalar, which means an output shift operation during variance quantization. Where, var_shift_type and var_shift_value are the data type of the shift operation.

- lut: lut_type, lut_shape, lut_offset, lut_size

  The *Look Up Table* (LUT) is an offline table created after quantization. Where, lut_type, lut_shape, lut_offset, and lut_size are the data type, table shape, table offset address, and table size. The following is a simple equation, lut_size = data_type * lut_shape. Where, data type = 1 (or 1, 2, 2, 4, 4) if sqrt_lut_type = int8 (or uint8, int16, uint16, int32, uint32).

OneHot

Returns a one-hot tensor (Y) based on the input 'indices'tensor and input scalar 'depth'. Generally, the index value in the 'indices' input tensor means there is a 'on_value' in that location and the other will have a value of 'off_value' in the output tensor. Where, 'on_value' and 'off_value' are the parameters of the two-element value tensor in order of '[off_value, on_value]'. Besides, the dimension of the output tensor is 'rank(input tensor) + 1', where the additional dimension is for one-hot representation extension along the specified 'axis'. The size of the additional dimension is the same as the value of parameter 'depth'. Any entries in the 'indices' input tensor with values outside the range '[-depth, depth - 1]' will result in all 'off_value' values in the output tensor.

Generally, if 'indices' is a scalar, the shape of the output tensor will be a vector of length 'depth'. Or if the 'indices' is a vector of length 'features', the shape of the output tensor will be:

'features * depth if axis = -1'

'depth * features if axis = 0'

Where, * means the multiplication operator. If the 'indices' is a matrix (in other words, with a batch) with shape [batch, features], the shape of the output tensor will be:

'batch * features * depth if axis = -1'

'depth * batch * features if axis = 0'

'batch * depth * features if axis = 1'

For example, 'indices = [[0, 2], [1, -1]]', and 'depth = 3', then the result tensor will be '[[[1, 0, 0], [0, 0 ,1]], [[0, 1, 0], [0, 0, 0]]]'.

Inputs

- Input tensor indices.

Outputs

- Output tensor Y, of which the dimension is one greater than the input tensor 'indices'.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = OneHot'.

- values

  An optional two-element tensor in an order as [off_value, on_value]. Where, 'on_value' is the value to fill in the specified locations of the 'indices', and 'off_value' means filling in the locations other than the locations specified in the 'indices' tensor. For example, values = [0, 1].

- depth

  A scalar to specify the number of classes in the one-hot tensor. That is, it is the size of the one-hot dimension which is specified by 'axis' and added in the output tensor.

- axis

  An optional scalar parameter to specify which dimension to insert to the input tensor 'indices'. A negative value means inserting dimensions from the back. It defaults to -1, which means the insertion is the last dimension. The accepted range is [-1, r-1] where r = rank(input data).

PRelu

Takes one input tensor (X) and produces one output tensor (Y), where the rectified linear function y = alpha * x for x < 0, y = x for x >= 0, is applied to the tensor elementwise. But the difference with RELU Operation is that 'alpha' is a learned array with the same shape as the input tensor (X). Especially, this operator supports unidirectional broadcasting (tensor alpha should be unidirectional broadcastable to input tensor X).

Inputs

- Input data tensor X.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = Activation'.

- method: PRELU

  Method to perform the input tensor.

- negative_slope: negative_slope_type, negative_slope_shape, negative_slope_offset, negative_slope_size, negative_slope_shift

  Represents the data type, shape and offset address and number size of coefficient of leakage, or the slope of the activation function at x < 0. The shape of this parameter can be smaller than input tensor (X), and if so, its shape must be unidirectional broadcastable to input tensor (X). Generally, 'negative_slop_shape' is the same shape as the output channel.

  Besides, negative_slope_shift can be a scalar during quantization.

- scale: scale_type, scale_value

  It can be a scalar during quantization, where scale_type and scale_value are the data type and value of the scale during per tensor or layer quantization.

- shift: shift_type, shift_value

  It can be a scalar during quantization, where shift_type and shift_value are the data type and value of the shift during per tensor or layer quantization.

Pad

Given the input tensor containing the data to be padded, a tensor containing the number of the start and end pad values for specified axis and mode.

Inputs

- Input data tensor X.

Outputs

- Output tensor Y after padding.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = Pad'.

- pads

  A nested list means the padding number with shape [n, 2] to a N dimension data tensor. In terms of dimension D of input data tensor, pads[D, 0] means how many values to add or remove (if negative) before the contents of tensor in that dimension, and pads[D, 1] means how many values to add or remove (if negative) after the contents of the input tensor in that dimension. It can take any value greater than or equal to 0. The value represents the number of pixels added to the beginning and end part of the corresponding dimension or axis. It defaults to an all 0 value in list (that is, no pad addition). For example, pads = [[0, 0], [1, 2], [2, 1], [0, 0]] represents adding one and two rows at the top and bottom respectively in Height dimension, and two and one columns at the left and right respectively in Width dimension in [N, H, W, C] data format.

  The padded size of each dimension D of the output tensor can be calculated by pads[D, 0] + X.dim_size(D) + pads[D, 1].

- mode: CONSTANT, REFLECT, SYMMETRIC

  Mode to add padding. The default value is CONSTANT. CONSTANT pads with a given value as specified by constant_value. REFLECT pads with the reflection of the vector mirrored on the first and last values of the vector along each axis. SYMMETRIC pads with the symmetric value.

- constant_value

  An optional parameter of the scalar value to be used if the mode chosen is CONSTANT. The default value is 0, empty string or 'false'.

Pow

Takes input data tensor (X) and exponent tensor (Y), then produces an output tensor (Z) where the function 'z = x ^ exponent' (that is, exponent is from exponent tensor Y), is applied to the data tensor elementwise. This operator also supports multidirectional (that is, NumPy-style) broadcasting.

Inputs

- Input data tensor X, exponent tensor Y.

Outputs

- Output tensor Z.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = Pow'.

- lut_exp: lut_exp_type, lut_exp_shape, lut_exp_offset, lut_exp_size

  The *Look Up Table* (LUT) is an offline table created during exponential function quantization. Where, lut_exp_type, lut_exp_shape, lut_exp_offset and lut_exp_size are the data type, table shape, table offset address and table size. The following is a simple equation, lut_exp_size = data_type * lut_exp_shape. Where, data type = 1 (or 2, 4) if lut_exp_type = int8 (or int16, int32).

- lut_log: lut_log_type, lut_log_shape, lut_log_offset, lut_log_size

  The *Look Up Table* (LUT) is an offline table created during logarithmic function quantization. Where, lut_log_type, lut_log_shape, lut_log_offset and lut_log_size are the data type, table shape, table offset address and table size. The following is a simple equation, lut_log_size = data_type * lut_log_shape. Where, data type = 1 (or 2, 4) if lut_log_type = int8 (or int16, int32).

- scale: scale_type, scale_value

  It can be a scalar during quantization, where scale_type and scale_value are the data type and value of the scale during box coordinates quantization.

- shift: shift_type, shift_value

  It can be a scalar during quantization, where scale_type and scale_value are the data type and value of the scale during box coordinates quantization.

Reciprocal

Reciprocal takes one input data (Tensor) and produces one output data (Tensor) where the reciprocal is, y = 1/x, is applied to the tensor elementwise.

Inputs

- Input data tensor X.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = Reciprocal'.

- lut: lut_type, lut_shape, lut_offset, lut_size

  The *Look Up Table* (LUT) is an offline table created after quantization. Where, lut_type, lut_shape, lut_offset and lut_size are the data type, table shape, table offset address and table size. The following is a simple equation, lut_size = data_type * lut_shape. Where, data type = 1 (or 2, 4) if lut_type = int8 (or int16, int32).

ReduceAll

Computes the 'Logical AND' value of the input tensor's element along the provided axis. By default, the output tensor will keep the dimension as input tensor X.

Inputs

- Input data tensor X.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is ‘layer_type = Reduce’.

- method: ALL

  The reduce method.

- axis

  It indicates along which axis to reduce. The value cannot be null. For example, axis = [1], or [1, 2], …. The accepted range is [-1, r-1] where r = rank(input data).

- scale: scale_type, scale_value

  It can be a scalar during quantization, where scale_type and scale_value are the data type and value of the scale during per tensor or layer quantization.

- shift: shift_type, shift_value

  It can be a scalar during quantization, where scale_type and scale_value are the data type and value of the scale during per tensor or layer quantization.

ReduceAny

Computes the ‘Logical OR’ value of the input tensor’s element along the provided axis. By default, the output tensor will keep the dimension as input tensor X.

Inputs

- Input data tensor X.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is ‘layer_type = Reduce’.

- method: ANY

  The reduce method.

- axis

  It indicates along which axis to reduce. The value cannot be null. For example, axis = [1], or [1, 2], …. The accepted range is [-1, r-1] where r = rank(input data).

- scale: scale_type, scale_value

  It can be a scalar during quantization, where scale_type and scale_value are the data type and value of the scale during per tensor or layer quantization.

- shift: shift_type, shift_value

  It can be a scalar during quantization, where scale_type and scale_value are the data type and value of the scale during per tensor or layer quantization.

ReduceL1

Computes the L1 norm value of the input tensor's element along the axis. By default, the output tensor will keep the dimension as input tensor X.

Inputs

- Input data tensor X.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = Reduce'.

- method: L1

  The reduce method.

- axis

  It indicates along which axis to reduce. The value cannot be null. For example, axis = [1], or [1, 2], …. The accepted range is [-1, r-1] where r = rank(input data).

- scale: scale_type, scale_value

  It can be a scalar during quantization, where scale_type and scale_value are the data type and value of the scale during per tensor or layer quantization.

- shift: shift_type, shift_value

  It can be a scalar during quantization, where scale_type and scale_value are the data type and value of the scale during per tensor or layer quantization.

ReduceL2

Computes the L2 norm value of the input tensor's element along the axis. By default, the output tensor will keep the dimension as input tensor X.

Inputs

- Input data tensor X.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = Reduce'.

- method: L2

  The reduce method.

- axis

It indicates along which axis to reduce. The value cannot be null. For example, axis = [1], or [1, 2], …. The accepted range is [-1, r-1] where r = rank(input data).

- scale: scale_type, scale_value

  It can be a scalar during quantization, where scale_type and scale_value are the data type and value of the scale during per tensor or layer quantization.

- shift: shift_type, shift_value

  It can be a scalar during quantization, where scale_type and scale_value are the data type and value of the scale during per tensor or layer quantization.

- sqrt_lut: sqrt_lut_type, sqrt_lut_shape, sqrt_lut_offset, sqrt_lut_size

  The *Look Up Table* (LUT) is an offline table created after quantization. Where, sqrt_lut_type, sqrt_lut_shape, sqrt_lut_offset, and sqrt_lut_size are the data type, table shape, table offset address, and table size. The following is a simple equation, sqrt_lut_size = data_type * sqrt_lut_shape. Where, data type = 1 (or 1, 2, 2, 4, 4) if sqrt_lut_type = int8 (or uint8, int16, uint16, int32, uint32).

ReduceMax

Computes the maximum value of the input tensor's element along the axis. By default, the output tensor will keep the dimension as input tensor X.

Inputs

- Input data tensor X.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = Reduce'.

- method: MAX

  The reduce method.

- axis

  It indicates along which axis to reduce. The value cannot be null. For example, axis = [1], or [1, 2], …. The accepted range is [-1, r-1] where r = rank(input data).

ReduceMean

Computes the mean value of the input tensor's element along the axis. By default, the output tensor will keep the dimension as input tensor X.

Inputs

- Input data tensor X.

Outputs

- Output tensor Y.

Attributes

- layer_type

The operation type of a layer. Here is ‘layer_type = Reduce’.

- method: MEAN

  The reduce method.

- axis

  It indicates along which axis to reduce. The value cannot be null. For example, axis = [1], or [1, 2], …. The accepted range is [-1, r-1] where r = rank(input data).

- scale: scale_type, scale_value

  It can be a scalar during quantization, where scale_type and scale_value are the data type and value of the scale during per tensor or layer quantization.

- shift: shift_type, shift_value

  It can be a scalar during quantization, where scale_type and scale_value are the data type and value of the scale during per tensor or layer quantization.

ReduceMin

Computes the minimum value of the input tensor’s element along the axis. By default, the output tensor will keep the dimension as input tensor X.

Inputs

- Input data tensor X.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is ‘layer_type = Reduce’.

- method: MIN

  The reduce method.

- axis

  It indicates along which axis to reduce. The value cannot be null. For example, axis = [1], or [1, 2], …. The accepted range is [-1, r-1] where r = rank(input data).

ReduceProd

Computes the multiplication value of the input tensor’s element along the axis. By default, the output tensor will keep the dimension as input tensor X.

Inputs

- Input data tensor X.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = Reduce'.

- method: PROD

  The reduce method.

- axis

  It indicates along which axis to reduce. The value cannot be null. For example, axis = [1], or [1, 2], ... The accepted range is [-1, r-1] where r = rank(input data).

- scale: scale_type, scale_value

  It can be a scalar during quantization, where scale_type and scale_value are the data type and value of the scale during per tensor or layer quantization.

- shift: shift_type, shift_value

  It can be a scalar during quantization, where scale_type and scale_value are the data type and value of the scale during per tensor or layer quantization.

ReduceSum

Computes the sum value of the input tensor's element along the axis. By default, the output tensor will keep the dimension as input tensor X.

Inputs

- Input data tensor X.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = Reduce'.

- method: SUM

  The reduce method.

- axis

  It indicates along which axis to reduce. The value cannot be null. For example, axis = [1], or [1, 2], ... The accepted range is [-1, r-1] where r = rank(input data).

- scale: scale_type, scale_value

  It can be a scalar during quantization, where scale_type and scale_value are the data type and value of the scale during per tensor or layer quantization.

- shift: shift_type, shift_value

  It can be a scalar during quantization, where scale_type and scale_value are the data type and value of the scale during per tensor or layer quantization.

ReduceUnbiasedVariance

Computes the unbiased variance value of the input tensor's element along the axis. By default, the output tensor will keep the dimension as input tensor X.

Inputs

- Input data tensor X.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = Reduce'.

- method: UNBIASED_VARIANCE

  The reduce method.

- axis

  It indicates along which axis to reduce. The value cannot be null. For example, axis = [1], or [1, 2], …. The accepted range is [-1, r-1] where r = rank(input data).

- scale: scale_type, scale_value

  It can be a scalar during quantization, where scale_type and scale_value are the data type and value of the scale during per tensor or layer quantization.

- shift: shift_type, shift_value

  It can be a scalar during quantization, where scale_type and scale_value are the data type and value of the scale during per tensor or layer quantization.

ReduceVariance

Computes the variance value of the input tensor's element along the axis. By default, the output tensor will keep the dimension as input tensor X.

Inputs

- Input data tensor X.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = Reduce'.

- method: VARIANCE

  The reduce method.

- axis

It indicates along which axis to reduce. The value cannot be null. For example, axis = [1], or [1, 2], …. The accepted range is [-1, r-1] where r = rank(input data).

- scale: scale_type, scale_value

  It can be a scalar during quantization, where scale_type and scale_value are the data type and value of the scale during per tensor or layer quantization.

- shift: shift_type, shift_value

  It can be a scalar during quantization, where scale_type and scale_value are the data type and value of the scale during per tensor or layer quantization.

Region

Yolo network predicts 5 bounding boxes at each cell in the output feature map. The network predicts 5 coordinates for each bounding box, tx, ty, tw, th, and to. If the cell is offset from the top left corner of the image by (cx, cy) and the bounding box prior has width and height pw, ph, then the predictions correspond to:

```
bx = O(tx) + cx
by = O(ty) + cy
bw = pw * exp(tw)
bh = ph * exp(th)
Pr(object) * IOU(b, object) = O(to)
```

The region layer is to process the calculation and output the bounding boxes and confidence for NMS.

Note that, in the following Inputs or Outputs sections:

- X1 means the input data of which the shape is [batch_size, s, s, anchor_num, num_classes + 5]. Where s is the grid number of a feature map, and num_classes is the classification number of the network.
- Y1 means the scores of the selected boxes, of which the shape is [batch_size, max_box_num]. Where max_box_num is the configuration parameter of the network.
- Y2 means the coordinates of the filtered boxes, of which the shape is [batch_size, max_box_num, 4].
- Y3 means the boxes number of every class per batch, of which the shape is [batch_size, num_classes].
- Y4 means the valid classes label of every class, of which the shape is [batch_size, num_classes].
- Y5 means the classes number of every class per batch.

Inputs

- Input tensor X1.

Outputs

- Output tensor Y1, Y2, Y3, Y4, Y5.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = Region'.

- score_softmax_lut: score_softmax_lut_type, score_softmax_lut_shape, score_softmax_lut_offset, score_softmax_lut_size

The *Look Up Table* (LUT) is an offline table created during exponent computation quantization. Where score_softmax_lut_type, score_softmax_lut_shape, score_softmax_lut_offset and score_softmax_lut_size are the data type, table shape, table offset address and table size. The following is a simple equation, score_softmax_lut_size = data_type * score_softmax_lut_shape. Where, data type = 1 (or 2, 4) if score_softmax_lut_type = int8 (or int16, int32).

- conf_sigmoid_lut: conf_sigmoid_lut_type, conf_sigmoid_lut_shape, conf_sigmoid_lut_offset, conf_sigmoid_lut_size

  The *Look Up Table* (LUT) is an offline table created during sigmoid computation quantization. Where, conf_sigmoid_lut_type, conf_sigmoid_lut_shape, conf_sigmoid_lut_offset and conf_sigmoid_lut_size are the data type, table shape, table offset address and table size. The following is a simple equation, conf_sigmoid_lut_size = data_type * conf_sigmoid_lut_shape. Where, data type = 1 (or 1, 2, 2, 4, 4) if conf_sigmoid_lut_type = int8 (or uint8, int16, uint16, int32, uint32).

- wh_exp_lut: wh_exp_lut_type, wh_exp_lut_shape, wh_exp_lut_offset, wh_exp_lut_size

  The *Look Up Table* (LUT) is an offline table created during the box width and height exponent computation quantization. Where, wh_exp_lut_type, wh_exp_lut_shape, wh_exp_lut_offset and wh_exp_lut_size are the data type, table shape, table offset address and table size. The following is a simple equation, wh_exp_lut_size = data_type * wh_exp_lut_shape. Where, data type = 1 (or 1, 2, 2, 4, 4) if wh_exp_lut_type = int8 (or uint8, int16, uint16, int32, uint32).

- xy_sigmoid_lut: xy_sigmoid_lut_type, xy_sigmoid_lut_shape, xy_sigmoid_lut_offset, xy_sigmoid_lut_size

  The *Look Up Table* (LUT) is an offline table created during the box center sigmoid computation quantization. Where, xy_sigmoid_lut_type, xy_sigmoid_lut_shape, xy_sigmoid_lut_offset and xy_sigmoid_lut_size are the data type, table shape, table offset address and table size. The following is a simple equation, xy_sigmoid_lut_size = data_type * xy_sigmoid_lut_shape. Where, data type = 1 (or 1, 2, 2, 4, 4) if xy_sigmoid_lut_type = int8 (or uint8, int16, uint16, int32, uint32).

- qanchors_lut: qanchors_lut_type, qanchors_lut_shape, qanchors_lut_offset, qanchors_lut_size

  The *Look Up Table* (LUT) is an offline table created for anchor computation quantization. Where, qanchors_lut_type, qanchors_lut_shape, qanchors_lut_offset and qanchors_lut_size are the data type, table shape, table offset address and table size. The following is a simple equation, qanchors_lut_size = data_type * qanchors_lut_shape. Where, data type = 1 (or 1, 2, 2, 4, 4) if qanchors_lut_type = int8 (or uint8, int16, uint16, int32, uint32).

- obj_threshold

  The threshold for determining when to decide which score belongs to an object. It is a scalar.

- max_box_num

  Integer representing the maximum number of boxes to be selected per batch. It is a scalar.

- box_per_grid

  Integer representing the number of boxes to be presented by a grid. It is a scalar.

- class_num

  Integer representing the number of classes for the input. It is a scalar.

- grid_width

  Integer representing the grid number of the feature map width. It is a scalar.

- grid_height

  Integer representing the grid number of the feature map height. It is a scalar.

- anchors_exp_h_shift

  It can be a scalar during quantization for anchor height computation.

- anchors_exp_w_shift

It can be a scalar during quantization for anchor width computation.

- gird_w_scale

  It can be a scale scalar during quantization for input box width computation.

- gird_w_shift

  It can be a shift scalar during quantization for input box width computation.

- gird_h_scale

  It can be a scale scalar during quantization for input box height computation.

- gird_h_shift

  It can be a shift scalar during quantization for input box height computation.

- wh_exp_scale

  It can be a scale scalar during quantization for wh_exp_lut computation.

- wh_exp_shift

  It can be a shift scalar during quantization for wh_exp_lut computation.

RegionFuse

Merges two group input data from the Region layer into one output.

Note that, in the following Inputs or Outputs sections:

- X1 and X2 mean that the scores of all the boxes come from the first and second region layer output, of which the shape is [batch_size, secore_num].
- X3 and X4 mean that the coordinates of boxes come from the first and second region layer output, of which the shape is [batch_size, secore_num, 4].
- X5 and X6 mean that the boxes number of every class per batch comes from the first and second region layer output, of which the shape is [batch_size, class_num].
- X7 and X8 mean that the boxes label of every class per batch comes from the first and second region layer output, of which the shape is [batch_size, class_num].
- X9 and X10 mean that the classes number of every class per batch comes from the first and second region layer output, of which the shape is [batch_size, class_num].
- Y1 means the merged scores of the selected boxes.
- Y2 means the merged coordinates of the filtered boxes.
- Y3 means the total boxes number of every class per batch, of which the shape is [batch_size, num_classes].
- Y4 means the total valid classes label of every class, of which the shape is [batch_size, num_classes].
- Y5 means the maxmun class number of every class per batch.

Inputs

- Input tensor X1~X10.

Outputs

- Output tensor Y1, Y2, Y3, Y4, Y5.

Attributes

- layer_type

  The operation type of a layer. Here is ‘layer_type = RegionFuse’.

- class_num

  Integer representing the number of class for the input. It is a scalar.

- box_scale: box_scale_type, box_scale_value

  The box_scale is a vector with two elements for the first and second box coordinates merge calcaulation. Where, scale_type and scale_value are the data type and value of the scale during quantization. Generally, scale_type has the int8, uint8 and int16 options.

- box_shift: box_shift_type, box_shift_value

  The box_shift is a vector with two elements for the first and second box coordinates merge calcaulation shift operation during quantization. Where, shift_type and shift_value are the data type of the shift operation.

- score_scale: score_scale_type, score_scale_value

  The score_scale is a vector with two elements for the first and second score merge calcaulation. Where, scale_type and scale_value are the data type and value of the scale during quantization. Generally, scale_type has the int8, uint8 and int16 options.

- score_shift: score_shift_type, score_shift_value

  The score_shift is a vector with two elements for the first and second score merge calcaulation shift operation during quantization. Where, shift_type and shift_value are the data type of the shift operation.

Relu

Takes one input tensor and produces one output tensor, where the rectified linear function y = max(0, x), is applied to the tensor elementwise.

Inputs

- Input data tensor X.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is ‘layer_type = Activation’.

- method: RELU

  Method to perform the input tensor.

- scale: scale_type, scale_value

  It can be a scalar during quantization, where scale_type and scale_value are the data type and value of the scale during per tensor or layer quantization.

- shift: shift_type, shift_value

  It can be a scalar during quantization, where shift_type and shift_value are the data type and value of the shift during per tensor or layer quantization.

Relu6

Takes one input tensor and produces one output tensor, where the rectified linear function y = min(max(0, x),6), is applied to the tensor elementwise.

Inputs

- Input data tensor X.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = Activation'.

- method: RELU6

  Method to perform the input tensor.

- scale: scale_type, scale_value

  It can be a scalar during quantization, where scale_type and scale_value are the data type and value of the scale during per tensor or layer quantization.

- shift: shift_type, shift_value

  It can be a scalar during quantization, where shift_type and shift_value are the data type and value of the shift during per tensor or layer quantization.

Repeat

Repeats elements of input tensor X along the corresponding axis. A repeat tensor is a 1-D inter tensor which means the number of repetitions for each element. Besides, 'repeat' is broadcast to fit the shape of the given axis. The length of the 'repeat' tensor must equal the shape of the input tensor along the specified axis if the axis is not none. For example, input tensor X= [[1,2], [3,4]], repeats=[2,3] and axis=0, then the output tensor Y will be [[1,2], [1,2], [3,4], [3,4], [3,4]].

Inputs

- Input data tensor X1, repeat tensor X2.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = Repeat'.

- axis

  An optional parameter and a scalar of integer along which to repeat values. By default, there is no this attribution (that is, 'None'), which means using the flattened input tensor and returning a flat output array. Note that 'None' is not the same as 'axis = 0' or 'axis = -1'.

Reshape

Returns a tensor that has the same value as the input tensor with shape changing. At most one dimension of the new shape can be -1. In this special case, the value is inferred from the size of the input tensor and the remaining dimensions.

Inputs

- Input data tensor X.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = Reshape'.

- shape

  Specifies the shape of the output tensor.

Resize

Resizes the input tensor (X) using the specified method.

Inputs

- Input data tensor X.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = Resize'.

- method: BILINEAR, NEAREST

  The method to resize the input tensor.

- ratio: ratio_x, ratio_y

  Optional parameter indicates the scale array (ratio_x, ratio_y) along each dimension (width, height). The number of elements of 'ratio' should be equal to the 'rank(input tensor X)'. It is upsampling when it is greater than 1, otherwise, it is downsampling. Only one of 'ratio' and 'size' can be specified and the other should be an empty string at the same time. Most of all, there will be an error if both are specified.

- size

  Optional parameter indicates the size of the output tensor. The number of elements of 'sizes' should be equal to the 'rank(input tensor X)'. Only one of 'ratio' and 'size' can be specified and the other should be an empty string at the same time. Most of all, there will be an error if both are specified.

- mode: HALF_PIXEL, ALIGN_CORNERS, ASYMMETRIC, PYTORCH_HALF_PIXEL, TF_HALF_PIXEL_FOR_NN

  Defaults to 'half_pixel'. It defines how to transform the coordinate in the resized tensor (Y) to the coordinate in the original tensor (X). Generally,

  - half_pixel

x_original = (x_resized + 0.5) / scale - 0.5.

- align_corners
  x_original = x_resized * (length_original - 1) / (length_resized - 1).
- asymmetric
  x_original = x_resized / scale.
- pytorch_half_pixel
  x_original = length_resized > 1 ? (x_resized + 0.5)/ scale - 0.5 : 0.
- tf_half_pxiel_for_nn
  x_original = (x_resized + 0.5) / scale

Where, x_orignal is denoted as the coordinate of axis x in the original tensor, x_resized is denoted as the coordinate of axis x in the resized tensor, length_original is denoted as the length of the original tensor in axis x, and length_resized is denoted as the length of the resized tensor in axis x. Also, size = length_resized / length_original.

- Nearest_mode:

  Five modes: round_prefer_floor (default, as known as round half down), round_prefer_ceil (as known as round half up), floor, ceil, and simple (ceil when the resize is downsampling). Only used by nearest interpolation. It indicates how to get 'nearest' pixel in the input tensor from x_original, so this attribute is valid only if 'mode' is 'nearest'.

ReverseSequence

Reverses batch of sequences having different lengths specified by 'sequence_lens'. For each slice 'i' iterating on the batch axis, the operator reverses the first 'sequence_lens[i]' element to 'time_axis', and copies elements whose indexes are beyond 'sequence_lens[i]' to the output. So, the output slice 'i' includes reverse sequences elements in the first 'sequence_lens[i]' and original elements copied beyond the 'sequence_lens' index. Besides, another important parameter is **len** which indicates the effective length to reverse sequence. The [T, N, C] data format means [time_step, batch_size, channel].

Inputs

- Input data tensor X, sequence_lens (that is, effective ReverseSequence **len** with the same shape as the batch size in data tensor X).

Outputs

- Output tensor Y with the same shape as X.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = ReverseSequence'.

- batch_axis

  Specifies which axis is the batch axis. Generally, the default value is 1 in [T, N, C] data format.

- time_axis

  Specifies which axis is the time axis. Generally, the default value is 0 in [T, N, C] data format.

RgbToYuv

Conversion of one or more images from RGB to YUV. Generally, this operation is used for preprocessing. Here, the term YUV is commonly encoded using Y'CbCr, while the RGB term is encoded using R'dG'dB'd. There are three standards which are so-called 'BT.601', 'BT.709' and 'BT.2020'. For 8 bits color, the [0, 255] range is referred as full range, which is considered here, while the [16, 235] range for Y and [16, 240] for U/V are referred as narrow range. In addition, all the conversions are based on integral approximate (or using fixed-point arithmetic as an alternative formulation) in this operation.

For BT.601 with full range, conversion matrices with integral approximate are:

$$\begin{bmatrix} Y' \\ C_b \\ C_r \end{bmatrix} = \begin{bmatrix} 0 \\ 128 \\ 128 \end{bmatrix} + \left(\begin{bmatrix} 306 & 601 & 117 \\ -173 & -339 & 512 \\ 512 & -465 & -47 \end{bmatrix} \cdot \begin{bmatrix} R'_d \\ G'_d \\ B'_d \end{bmatrix} + \begin{bmatrix} 512 \\ 512 \\ 512 \end{bmatrix} \right) \gg 10$$

For BT.709 with full range, conversion matrices with integral approximate are:

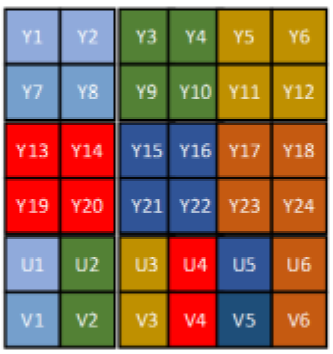
$$\begin{bmatrix} Y' \\ C_b \\ C_r \end{bmatrix} = \begin{bmatrix} 0 \\ 128 \\ 128 \end{bmatrix} + \left(\begin{bmatrix} 218 & 732 & 74 \\ -118 & -395 & 512 \\ 512 & -465 & -47 \end{bmatrix} \cdot \begin{bmatrix} R'_d \\ G'_d \\ B'_d \end{bmatrix} + \begin{bmatrix} 512 \\ 512 \\ 512 \end{bmatrix} \right) \gg 10$$

Where $[128, 128, 128]^T$ is the input offset, $[0, 128, 128]^T$ is the output offset, and [[54, 183, 18],[-29, -99, 128],[128, -116, -12]] are the coefficient. RGB files are typically encoded in 8, 12, 16, 24 bits per pixel. For example, 24 bits per pixel, which is written as RGB888, the standard byte format in the memory device is stacked as:

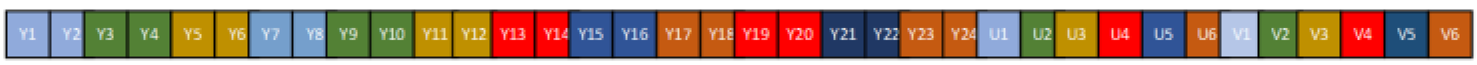
RGB888: r0, g0, b0, r1, g1, b1, …

YUV files can be encoded in 12, 16 or 24 bits per pixel. The common formats are YUV444, YUV422, YUV420p and YUV420sp. For example, YUV420p and YUV420sp with the planar format, the standard format in the memory device is stacked as follows:

I420: Single Frame

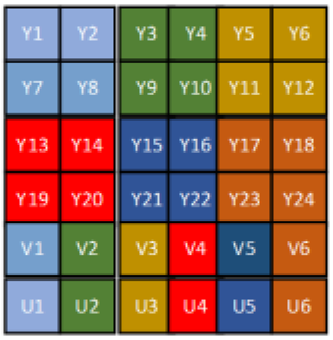


Position in the byte stream:



As shown in the preceding diagram, the Y, U and V components in YUV420p are encoded separately in sequential blocks. A Y value is stored for every pixel, followed by a U value for each 2x2 square block. The corresponding Y, U and V values are shown using the same color in the preceding diagram. When read line by line as the byte stream from a device, the Y block will be found at position 0, the U block at position xy *(6x4 = 24 in this case) and the V block at position* xy + (x * y)/4 (6x4 + (6x4)/4 = 30 in this case).

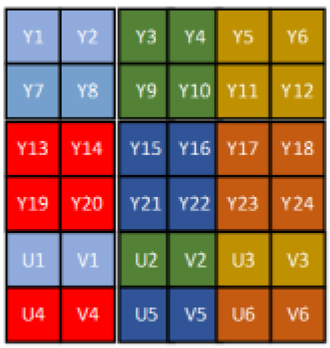
YV12: Single Frame



Position in the byte stream:

As shown in the preceding diagram, the Y, U and V components in YUV420p are encoded separately in sequential blocks. A Y value is stored for every pixel, followed by a U value for each 2x2 square block. The corresponding Y, U and V values are shown using the same color in the preceding diagram. When read line by line as the byte stream from a device, the Y block will be found at position 0, the V block at position x*y (6x4 = 24 in this case) and the U block at position* xy + (x * y)/4 (6x4 + (6x4)/4 = 30 in this case).

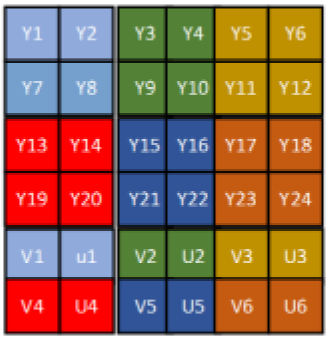
NV12: Single Frame



Position in the byte stream:

As shown in the preceding diagram, the Y, U and V components in NV12 are encoded separately in sequential blocks. A Y value is stored for every pixel, followed by a interleave of U/V for each 2x2 square block. The corresponding Y, U and V values are shown using the same color in the preceding diagram. When read line by line as the byte stream from a device, the Y block will be found at position 0, followed by interleaved U/V plane with 8 bits subsampled chroma samples from the position x*y (6x4 = 24 in this case).

NV21: Single Frame



Position in the byte stream:

This format is the standard picture on Android camera preview. As shown in the preceding diagram, the Y, U and V components in NV21 are encoded separately in sequential blocks. A Y value is stored for every pixel, followed by a interleave of V/U for each 2x2 square block. The corresponding Y, U and V values are shown using the same color in the preceding diagram. When read line by line as the byte stream from a device, the Y block will be found at position 0, followed by interleaved V/U plane with 8 bits subsampled chroma samples from the position x*y (6x4 = 24 in this case).

Inputs

- Input RGB image tensor, with a shape of [N, H, W, 3].

Outputs

- Output YUV image tensor, with a shape of [N, D].

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = RgbToYuv'.

- format: I420, YV12, NV12, NV21

  The output image format which will be converted. Generally, the formats of 'I420' and 'YV12' mean that the Y, U and V values are grouped together instead of interspersed. When given an array of an image in the 'I420' format, all the Y values come first, followed by all the U values, and then followed by all the V values. While in the 'YV12' format, all the Y values come first, followed by all the V values, and then followed by all the U values.

  The others are the formats of 'NV12' and 'NV21', which mean that Y values are grouped together, but U and V values are interspersed. When given an array of an image in the 'NV12' format, all the Y values come first, followed by the interleaved U and V value. While in the 'NV21' format, all the Y values come first, followed by the interleaved V and U value as the replacement.

- bits: 8, 10

  The bits to store Y, U or V values in memory, such as the DDR device.

- conversion: BT601, BT709, BT2020, SELF

  Means the different coefficient matrix to convert between the YUV and RGB formats. Especially, 'SELF' means that the conversion coefficients are customized, which is from the configuration.

- coefficient

  A list of color space coefficient (CSC) for conversion, which is in order of [input_offset, output_offset, coefficient]. For example, the list = [0, 0, 0, 0, 128, 128, 218, 732, 74, -118, -395, 512, 512, -465, -47] in BT709 format with integral approximate. While, the list = [0, 0, 0, 0, 128, 128, 0.212600, 0.715200, 0.072200, -0.114572, -0.385428, 0.500000, 0.500000, -0.454153, -0.045847] in BT709 format with float coefficient.

- coefficient_dtype: int8, int10, int16, float32

  The data type of coefficient.

- coefficient_shift

  An integer of the shift value depends on the data type of the ‘coefficient’ parameter, which means the shift value before the ‘output_offset’ operation. For example, coefficient_shift = 10 under integral approximate and coefficient_shift = 0 under float type.

RightShift

The right shift operator performs element-wise operations. For each input element, this operator moves its binary representation toward the right side so that the input value is effectively decreased. The input X is the tensor to be shifted and another input Y specifies the amounts of shifting. For example, X is [1, 4], and S is [1, 1], the corresponding output Z will be [0, 2]. This operator supports multidirectional broadcasting.

Inputs

- Input data tensor X.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is ‘layer_type = Bitshift’.

- direction: RIGHT

  Direction of moving bits.

RoiAlign

Region of Interest (ROI) align is used for converting all the proposals to the fixed shape as required by the next special layers, especially in an object detection network. ROI align is proposed to avoid the misalignment by removing quantization while converting from the original image into the feature map and from the feature map into the ROI feature. In each ROI bin, the value of the sampled locations is computed directly through bilinear interpolation other than quantization.

It takes two inputs, one is the input data tensor from the previous operator or layer, which has a shape of [N, H, W, C], and the other is ROIs proposal with a 2-D shape of [num_rois, 5] by giving as [batch_indices, y1, x1, y2, x2]. Meanwhile, the ‘batch_indices’ denotes the index of the corresponding image in the batch. The output tensor Y has a shape of [num_rois, output_height, output_width, C], while the **i**-th batch element Y[i-1] is a pooled feature map corresponding to the **i**-th ROIs tensor X[i-1].

Inputs

- Input data tensor (X), ROIs (ROIs proposal and the shape is [num_rois, 5]).

Outputs

- Output tensor Y.

Attributes

- layer_type

The operation type of a layer. Here is ‘layer_type = RoiAlign’.

- method: AVG, MAX

  Means the pooling method during sampling.

- pooled_shape

  A list to indicate the output shape of ROI pooing which is in order of [Height, Width].

- coordinate_transformation_mode : string (default is HALF_PIXEL)

  Allowed values are ‘HALF_PIXEL’ and ‘OUTPUT_HALF_PIXEL’. Use the value ‘HALF_PIXEL’ to pixel shift the input coordinates by -0.5 (the recommended behavior). Use the value ‘output_half_pixel’ to omit the pixel shift for the input (use this for a backward-compatible behavior).

- sample: sample_x, sample_y

  A list which is in order of [sample_y, sample_x]. It means the number of sampling points in the interpolation grid used to compute the output value of each pooled output bin. Generally, the calculation can be ‘sample_x = ceil(roi_width / output_width)’, and likewise for height.

- spatial_scale_value: spatial_scale_value_x, spatial_scale_value_y

  A list which is in order of [spatial_scale_value_y, spatial_scale_value_x]. It means the multiplicative spatial scale factor to translate the ROI coordinates from the input spatial scale to the scale used in the pooling operation. Generally, it can be a float factor, but will be a fixed-point parameter after quantization. For example, ‘spatial_scale_value_x = feature_map_width / image_width’ under non-quantization, and likewise for height.

- spatial_scale_type

  It can be a scalar during quantization, where ‘spatial_scale_type’ is the data type of the ‘spatial_scale_value_x’ and ‘spatial_scale_value_y’ during spatial (including ‘spatial_scale_value_x’ and ‘spatial_scale_value_y’) parameter quantization.

- spatial_shift: spatial_shift_type, spatial_shift_value

  It can be a scalar during quantization, where ‘spatial_shift_type’ and ‘spatial_shift_value’ are the data type and value of the shift during spatial (including ‘spatial_x’ and ‘spatial_y’) parameter quantization.

- scale_value: [featuremap_scale_value, roi_scale_value]

  A list which is in order of [featuremap_scale_value, roi_scale_value]. It means the value of scale during per tensor or layer quantization of the feature map and input ROIs.

- scale_type:

  It can be a list during quantization, where scale_type is the data type of ‘featuremap_scale_value’ and ‘roi_scale_value’ during the output feature map and input ROI parameters quantization.

- shift_value:

  A list of values during quantization, where shift_value is the shift value of ‘featuremap_scale_value’ and ‘roi_scale_value’ parameters quantization.

- shift_type:

  It can be a list during output tensor quantization, where shift_type is the type of shift_value.

- bin_scale_value: [h_bin_scale_value, w_bin_scale_value]

  A list which is in order of [h_bin_scale_value, w_bin_scale_value]. It means the value of scale during per tensor or layer quantization of input ROI parameters quantization when the ‘sample’ attributes is nonzero.

- bin_scale_type:

It can be a list during quantization, where bin_scale_type is the data type of 'h_bin_scale_value' and 'w_bin_scale_value' during input ROI parameters quantization when the 'sample' attributes is nonzero.

- bin_shift_value:

  A list of values during quantization, where bin_shift_value is the shift value of 'h_bin_scale_value' and 'w_bin_scale_value' during input ROI parameters quantization when the 'sample' attributes is nonzero.

- bin_shift_type:

  It can be a list during output tensor quantization, where bin_shift_type is the type of 'h_bin_scale_value' and 'w_bin_scale_value' during input ROI parameters quantization when the 'sample' attributes is nonzero.

- grid_scale_value: [h_grid_scale_value, w_grid_scale_value]

  A list which is in order of [h_grid_scale_value, w_grid_scale_value]. It means the value of scale during per tensor or layer quantization of input ROI parameters quantization when the 'sample' attributes is nonzero.

- grid_scale_type:

  It can be a list during quantization, where grid_scale_type is the data type of 'h_grid_scale_value' and 'w_grid_scale_value' during input ROI parameters quantization when the 'sample' attributes is nonzero.

- grid_shift_value:

  A list of values during quantization, where grid_shift_value is the shift value of 'h_grid_scale_value' and 'w_grid_scale_value' during input ROI parameters quantization when the 'sample' attributes is nonzero.

- grid_shift_type:

  It can be a list during output tensor quantization, where grid_shift_type is the type of 'h_grid_scale_value' and 'w_grid_scale_value' during input ROI parameters quantization when the 'sample' attributes is nonzero.

Round

Round takes one input Tensor and rounds the values, element-wise, meaning it finds the nearest integer for each value. In case of halves, the rule is to round them to the nearest even integer. The output tensor has the same shape and type as the input.

Float examples:

$$\begin{aligned} round([0.9]) &= [1.0] \\ round([2.5]) &= [2.0] \\ round([2.3]) &= [2.0] \\ round([1.5]) &= [2.0] \\ round([-4.5]) &= [-4.0] \end{aligned}$$

Inputs

- Input data tensor X.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = Round'.

- lut: lut_type, lut_shape, lut_offset, lut_size

The *Look Up Table* (LUT) is an offline table created after quantization. Where, lut_type, lut_shape, lut_offset and lut_size are the data type, table shape, table offset address and table size. The following is a simple equation, lut_size = data_type * lut_shape. Where, data type = 1 (or 2, 4) if lut_type = int8 (or int16, int32).

Rsqrt

Computes reciprocal of square root of the input tensor (X) element-wise as the output tensor (Y), where the equation, y = 1/ (x\^0.5), is applied to the tensor elementwise. If X is zero or negative, it will return a reciprocal of random value (for example, returns -128 in int8 format).

Inputs

- Input data tensor X.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = Rsqrt'.

- lut: lut_type, lut_shape, lut_offset, lut_size

  The *Look Up Table* (LUT) is an offline table created after quantization. Where, lut_type, lut_shape, lut_offset and lut_size are the data type, table shape, table offset address and table size. The following is a simple equation, lut_size = data_type * lut_shape. Where, data type = 1 (or 2, 4) if lut_type = int8 (or int16, int32).

ScatterElements

ScatterElements takes three inputs data, updates, and indices of the same rank r >= 1 and an optional attribute axis that identifies an axis of data (by default, the outer-most axis, that is axis 0). The output of the operation is produced by creating a copy of the input data, and then updating its value to values specified by updates at specific index positions specified by indices. Its output shape is the same as the shape of data.

For each entry in updates, the target index in data is obtained by combining the corresponding entry in indices with the index of the entry itself: the index-value for dimension = axis is obtained from the value of the corresponding entry in indices and the index-value for dimension != axis is obtained from the index of the entry itself.

Inputs

- Params: tensor of rank r>=1
- Indices: tensor of rank r>=1
- Updates: tensor of rank r>=1

Outputs

- Output tensor Y of rank r>=1

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = ScatterElements'.

- axis: int

Which axis to scatter on. A negative value means counting dimensions from the back. The accepted range is [-r, r-1] where r =rank(data).

- reduction: MUL ,ADD, NONE

  Reduction allows specification of an optional reduction operation, which is applied to all values in updates tensor into output at the specified indices. In cases where reduction is set to 'none', indices should not have duplicate entries, that is, if idx1 != idx2, then indices[idx1] != indices[idx2]. For instance, in a 2-D tensor case, the update corresponding to the [i][j] entry is performed as follows:

$$
\begin{aligned}
out[indices[i][j][k]][j] &= updates[i][j] \quad if\ axis = 0 \\
out[i][indices[i][j][k]] &= updates[i][j] \quad if\ axis = 1
\end{aligned}
$$

  When reduction is set to 'add', the update corresponding to the [i][j] entry is performed as follows:

$$
\begin{aligned}
out[indices[i][j][k]][j] &\mathrel{+}= updates[i][j] \quad if\ axis = 0 \\
out[i]\big[indices[i][j][k]\big] &\mathrel{+}= updates[i][j] \quad if\ axis = 1
\end{aligned}
$$

  When reduction is set to 'mul', the update corresponding to the [i][j] entry is performed as follows:

$$
\begin{aligned}
out[indices[i][j][k]][j] &\mathrel{*}= updates[i][j] \quad if\ axis = 0 \\
out[i]\big[indices[i][j][k]\big] &\mathrel{*}= updates[i][j] \quad if\ axis = 1
\end{aligned}
$$

- scale: scale_type, scale_value

  The scale is a vector with three elements. Where, scale_type and scale_value are the data type and value of the scale during quantization. Generally, scale_type has the int8, uint8, and int16 options. Furthermore, the scales type or value must be in order of 'output_scale, input_scale[i]'.

- shift: shift_type, shift_value

  The shift can be a vector with not more than three elements. Where, shift_type and shfit_value are the data type and value of the shift during quantization. Generally, shift_type is int8. Furthermore, the shift type or value must be in order of 'output_shift, input_shift[i]'. If there is only one value, it means output_shift.

ScatterND

ScatterND takes three inputs data tensor of rank r >= 1, indices tensor of rank q >= 1, and updates tensor of rank q + r - indices.shape[-1] - 1. The output of the operation is produced by creating a copy of the input data, and then updating its value to values specified by updates at specific index positions specified by indices. Its output shape is the same as the shape of data. Note that indices should not have duplicate entries. That is, two or more updates for the same index-location is not supported.

Inputs

- Params: tensor of rank r >= 1
- Indices: tensor of rank q >= 1
- Updates: tensor of rank q + r - indices_shape[-1] - 1

Outputs

- Output tensor Y of rank r >= 1

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = ScatterND'.

- reduction: MUL ,ADD, NONE

  A string. The type of reduction to apply: none (default), add, mul.

  - 'none': no reduction applied.
  - 'add': reduction using the addition operation.
  - 'mul': reduction using the multiplication operation.

  When reduction = NONE:

  output = np.copy(data)

  update_indices = indices.shape[:-1]

  for idx in np.ndindex(update_indices):

  output[indices[idx]] = updates[idx]

  If reduction = ADD:

  output = np.copy(data)

  update_indices = indices.shape[:-1]

  for idx in np.ndindex(update_indices):

  output[indices[idx]] += updates[idx]

  If reduction = MUL:

  output = np.copy(data)

  update_indices = indices.shape[:-1]

  for idx in np.ndindex(update_indices):

  *output[indices[idx]] *= updates[idx]*

- scale: scale_type, scale_value

  The scale is a vector with two elements. Where, scale_type and scale_value are the data type and value of the scale during quantization. Generally, scale_type has the int8, uint8, and int16 options. Furthermore, the scales type or value must be in order of 'output_scale, input_scale[i]'.

- shift: shift_type, shift_value

  The shift can be a vector with not more than two elements. Where, shift_type and shift_value are the data type and value of the shift during quantization. Generally, shift_type is int8. Furthermore, the shift type or value must be in order of 'output_shift, input_shift[i]'. If there is only one value, it means output_shift.

SegmentSumReduce

Reduces a tensor along the segment.

Inputs

- Input data tensor X.
- Segment_ids: Values should be sorted and can be repeated.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = SegmentReduce'.

- method: SUM

  The segment reduce method to compute the sum along segments of a tensor.

- scale: scale_type, scale_value

  It can be a scalar during quantization, where scale_type (or scale_value) is the data type (or value) of the scale during per tensor or layer quantization.

- shift: shift_type, shift_value

  It can be a scalar during quantization, where shift_type (or shift_value) is the data type (or value) of the shift during per tensor or layer quantization.

Selu

Takes an input tensor (X) and produces one output tensor (Y), where the scaled exponential linear unit function 'f(x) = gamma * (alpha * (exp(x) -1)) for x <= 0, f(x) = gamma * x for x > 0', is applied to the tensor elementwise.

Inputs

- Input data tensor X.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = Activation'.

- method: SELU

  Method to perform the input tensor.

- alpha

  A coefficient parameter of the activation function in a float IR.

- gamma

  A coefficient parameter of the activation function in a float IR.

- lut: lut_type, lut_shape, lut_offset, lut_size

  The *Look Up Table* (LUT) is an offline table created after quantization. Where, lut_type, lut_shape, lut_offset and lut_size are the data type, table shape, table offset address and table size. The following is a simple equation, lut_size = data_type * lut_shape. Where, data type = 1 (or 1, 2, 2, 4, 4) if lut_type = int8 (or uint8, int16, uint16, int32, uint32).

Shrink

Shrink takes one input data (Tensor) and produces one Tensor output, having the same datatype and shape with the input. It has two attributes, lambd and bias. The formula of this operator is: If x < -lambd, y = x + bias. If x > lambd, y = x - bias. Otherwise, y = 0.

Inputs

- Input data tensor X.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = Activation'.

- method: SHRINK

  Method to perform the input tensor.

- bias: float(default is 0.0)

  The bias value added to the output. Only needed in float IR. The default value is 0.

- lambd: float(default is 0.5)

  The lambd value for the Shrink formulation. Only needed in float IR. The default value is 0.5.

- lut: lut_type, lut_shape, lut_offset, lut_size

  The *Look Up Table* (LUT) is an offline table created after quantization. Where, lut_type, lut_shape, lut_offset and lut_size are the data type, table shape, table offset address and table size. The following is a simple equation, lut_size = data_type * lut_shape. Where, data type = 1 (or 1, 2, 2, 4, 4) if lut_type = int8 (or uint8, int16, uint16, int32, uint32).

Sigmoid

Takes one input tensor and produces one output tensor, where the sigmoid function y = 1/ (1 + exp(-x)), is applied to the tensor elementwise.

Inputs

- Input data tensor X.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = Activation'.

- method: SIGMOID

  Method to perform the input tensor.

- lut: lut_type, lut_shape, lut_offset, lut_size

  The *Look Up Table* (LUT) is an offline table created after quantization. Where, lut_type, lut_shape, lut_offset and lut_size are the data type, table shape, table offset address and table size. The following is a simple equation, lut_size = data_type * lut_shape. Where, data type = 1 (or 1, 2, 2, 4, 4) if lut_type = int8 (or uint8, int16, uint16, int32, uint32).

Sign

Calculates the sign of the given input tensor (X) element-wise. That is, f(x) = 1 if x > 0, f(x) = 0 if x == 0, and f(x) = -1 if x < 0.

Inputs

- Input data tensor X.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = Sign'.

Silu

Takes one input tensor and produces one output tensor, where the SiLU or Swish function y = x * sigmoid(x), is applied to the tensor elementwise and 'sigmoid(x)' is the logistic sigmoid function.

Inputs

- Input data tensor X.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = Activation'.

- method: SILU

  Method to perform the input tensor.

- lut: lut_type, lut_shape, lut_offset, lut_size

  The *Look Up Table* (LUT) is an offline table created after quantization. Where, lut_type, lut_shape, lut_offset and lut_size are the data type, table shape, table offset address and table size. The following is a simple equation, lut_size = data_type * lut_shape. Where, data type = 1 (or 1, 2, 2, 4, 4) if lut_type = int8 (or uint8, int16, uint16, int32, uint32).

Sine

Calculates the sine of the given input tensor by element-wise.

Inputs

- Input data tensor X.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = Sine'.

- lut: lut_type, lut_shape, lut_offset, lut_size

The *Look Up Table* (LUT) is an offline table created after quantization. Where, lut_type, lut_shape, lut_offset and lut_size are the data type, table shape, table offset address and table size. The following is a simple equation, lut_size = data_type * lut_shape. Where, data type = 1 (or 1, 2, 2, 4, 4) if lut_type = int8 (or uint8, int16, uint16, int32, uint32).

Sinh

Calculates the hyperbolic sine of the given input tensor element-wise.

Inputs

- Input data tensor X.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = Sinh'.

- lut: lut_type, lut_shape, lut_offset, lut_size

  The *Look Up Table* (LUT) is an offline table created after quantization. Where, lut_type, lut_shape, lut_offset and lut_size are the data type, table shape, table offset address and table size. The following is a simple equation, lut_size = data_type * lut_shape. Where, data type = 1 (or 1, 2, 2, 4, 4) if lut_type = int8 (or uint8, int16, uint16, int32, uint32).

Slice

Produces a slice of the input tensor (X) along multiple axes. Slice uses begin, end, strides and axis inputs to specify the start and end dimension and stride for each axis in the list of axes. It uses the information to slice the input data tensor (X). If a negative value is passed for any of the start or end indices, it represents slicing backward of that dimension. If the value passed to start or end is larger than 'n', which is the number of elements in this dimension, it acts as 'n'. If a negative value is passed for stride, it means slicing backward. For example, X=[[1, 2, 3, 4], [5, 6, 7, 8]], begin = [0, 1], end = [-1, 5], then the slicing result will be [[2, 3, 4]].

Inputs

- Input data tensor X.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = Slice'.

- begin

  A list of integers to indicate the starting indices.

- end

  A list of integers to indicate the ending indices (exclusive).

- strides

  A list of integers to indicate the slicing stride. A negative value means slicing backward. 'strides' cannot be 0 and the default value is 1.

Softmax

The operator computes the normalized exponential values for the given input: Softmax (input, axis) = Exp(input) / Reduce_Sum (Exp(input), axis = axis, keepdims = 1). The input does not need to explicitly be a 2D vector. The 'axis' attribute indicates the dimension along which softmax will be performed. The output tensor has the same shape and contains the softmax values of the corresponding input.

Inputs

- Input data tensor X.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = Softmax'.

- axis

  Describes the dimension that Softmax will be performed on. A negative value means counting dimensions from the back. The default value is axis = -1. The accepted range is [-1, r-1] where r = rank(input data).

- lut: lut_type, lut_shape, lut_offset, lut_size

  The *Look Up Table* (LUT) is an offline table created during exponent computation quantization. Where lut_type, lut_shape, lut_offset and lut_size are the data type, table shape, table offset address and table size. The following is a simple equation, lut_size = data_type * lut_shape. Where, data type = 1 (or 2, 4) if lut_type = int8 (or int16, int32).

- scale: scale_type, scale_value

  It can be a scalar quantization parameter before the output tensor, where scale_type and scale_value are the data type and value of the scale during per tensor or layer quantization.

- shift: shift_type, shift_value

  It can be a scalar quantization parameter before the output tensor, where shift_type and shift_value are the data type and value of the shift during per tensor or layer quantization.

Softplus

Takes one input tensor (X) and produces one output tensor (Y), where the function y = loge(exp(x) + 1), is applied to the tensor elementwise.

Inputs

- Input data tensor X.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = Activation'.

- method: SOFTPLUS

Method to perform the input tensor.

- lut: lut_type, lut_shape, lut_offset, lut_size

  The *Look Up Table* (LUT) is an offline table created during quantization. Where lut_type, lut_shape, lut_offset and lut_size are the data type, table shape, table offset address and table size. The following is a simple equation, lut_size = data_type * lut_shape. Where, data type = 1 (or 2, 4) if lut_type = int8 (or int16, int32).

Softsign

Takes one input tensor (X) and produces one output tensor (Y), where the function 'y = x / (abs(x) + 1)', is applied to the tensor elementwise.

Inputs

- Input data tensor X.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = Activation'.

- method: SOFTSIGN

  Method to perform the input tensor.

- lut: lut_type, lut_shape, lut_offset, lut_size

  The *Look Up Table* (LUT) is an offline table created during quantization. Where lut_type, lut_shape, lut_offset and lut_size are the data type, table shape, table offset address and table size. The following is a simple equation, lut_size = data_type * lut_shape. Where, data type = 1 (or 2, 4) if lut_type = int8 (or int16, int32).

SpaceToBatch

Rearranges (permutes) data from blocks of spatial data into batch. This is the reverse transformation of BatchToSpace. More specifically, this operator outputs a copy of the input tensor where values from the height and width dimensions are moved to the batch dimensions.

Inputs

- Input data tensor X.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = SpaceToBatch'.

- block_size: block_size_x, block_size_y

  It means that blocks of [block_size_x, block_size_y] are moved. Where block_size_x (block_size_y) is the block size along with the width (heights) axis. Generally, the output tensor will be [N * block_size_x * block_size_y), H + pad_top + pad_bottom / block_size_h, W + pad_left + pad_right / block_size_x, C].

- pad: pad_bottom, pad_top, pad_left, pad_right

Padding for the beginning and ending along spatial axis, it can take any value greater than or equal to 0. The value represents the number of pixels added to the beginning and end part of the corresponding axis. The 'pad' parameter format should be as follows 'pad_bottom', 'pad_top', 'pad_left', 'pad_right', where it means the pad pixels of the data cube (that is, bottom and top along the 'height' axis, left and right along the 'width' axis). If not present, compute and output the same shape as the input tensor.

SpaceToDepth

Rearranges (permutes) data from blocks of spatial data into depth. This is the reverse transformation of DepthToSpace. More specifically, this operator outputs a copy of the input tensor where values from the height and width dimensions are moved to the depth dimensions.

Inputs

- Input data tensor X.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = SpaceToDepth'.

- block_size: block_size_x, block_size_y

  It means that blocks of [block_size_y, block_size_x] are moved. Where block_size_x (block_size_y) is the block size along with the width (heights) axis. Generally, the output tensor will be [N, block_size_y * block_size_x, H / block_size_y, W / block_size_x, C].

Split

Splits a tensor into a list of tensors along the specified 'axis' direction. Lengths of the parts can be specified using input 'split'. By default, the tensor is split to the equal sized parts. For example, input tensor X = [[0, 1, 2, 3, 4, 5], [6, 7, 8, 9, 10, 11]], axis = 1 and splits = [2, 4], then returns an output tensor Y1 = [[0, 1], [6, 7]], Y2 = [[2, 3, 4, 5], [8, 9, 10, 11]].

Inputs

- Input data tensor X.

Outputs

- Output tensor Y1, Y2, …, Yi.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = Split'.

- axis

  The axis to split an operation. A negative value means counting dimensions from the back of the input tensor. The accepted range is [-1, r-1] where r = rank(input data).

- splits

  A list of integers to indicate the length of each output. Generally, it is a value greater than 0. Besides, the sum of split value should be equal to the dimension value at the corresponding 'axis' specified.

Sqrt

Square root takes one input tensor (X) and produces one output tensor (Y), where the square root is, y = x\^0.5, is applied to the tensor elementwise. If X is negative, then it will return a random value (for example, -128 in int8 format).

Inputs

- Input data tensor X.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = Sqrt'.

- lut: lut_type, lut_shape, lut_offset, lut_size

  The *Look Up Table* (LUT) is an offline table created after quantization. Where, lut_type, lut_shape, lut_offset and lut_size are the data type, table shape, table offset address and table size. The following is a simple equation, lut_size = data_type * lut_shape. Where, data type = 1 (or 1, 2, 2, 4, 4) if lut_type = int8 (or uint8, int16, uint16, int32, uint32).

Square

Computes square of input tensor (X) elements-wise, where the equation is y = x^2.

Inputs

- Input data tensor X.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = Square'.

- lut: lut_type, lut_shape, lut_offset, lut_size

  The *Look Up Table* (LUT) is an offline table created after quantization. Where, lut_type, lut_shape, lut_offset and lut_size are the data type, table shape, table offset address and table size. The following is a simple equation, lut_size = data_type * lut_shape. Where, data type = 1 (or 1, 2, 2, 4, 4) if lut_type = int8 (or uint8, int16, uint16, int32, uint32).

SquaredDifference

Returns (x1 - x2) * (x1 - x2) element-wise.

Inputs

- Input first operand tensor X1, second operand tensor X2.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = SquaredDifference'.

- scale_value: [out_scale_value, x1_scale_value, x2_scale_value, sub_scale_value]

  A list which is in order of [out_scale_value, x1_scale_value, x2_scale_value, sub_scale_value]. It means the value of scale during per tensor or layer quantization squred_difference calculation.

- scale_type:

  It can be a list during quantization, where scale_type is the data type of 'out_scale_value', 'x1_scale_value', 'x2_scale_value' and 'sub_scale_value' during parameters quantization.

- shift_value:

  A list of values during quantization, where shift_value is the shift value of 'out_scale_value' and 'sub_scale_value' parameters quantization.

- shift_type:

  It can be a list during output tensor quantization, where shift_type is the type of shift_value.

Sub

Performs subtraction of each of the input tensors (with NumPy-style broadcasting support). All inputs and outputs must have the same data type. Besides, this operator supports multidirectional (that is, NumPy-style) broadcasting.

Inputs

- Input data tensor X1, data tensor X2.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = Sub'.

- scale: scale_type, scale_value

  The scale is a vector with more than two elements, and its number of elements should be equal to the sum of output and input tensors. Where, scale_type and scale_value are the data type and value of the scale during quantization. Generally, scale_type has int8, uint8, and int16 options. Furthermore, the scales type or value must be in order of 'output_scale, input_scale[i]'.

- shift: shift_type, shift_value

  It can be a scalar, which means an output shift operation during quantization. Where, shift_type and shift_value are the data type of the shift operation.

SufficientStatistics

Calculates the sufficient statistics for the mean and variance of x.

Note that, in the following Inputs or Outputs sections:

- X1: The input tensor.

- X2: A Tensor containing the value by which to shift the data for numerical stability.
- Y1: The shifted sum of the elements of the input tensor along the axis.
- Y2: The shifted sum of squares of the elements of the input tensor along the axis.

Inputs

- Input data tensor X1, X2.

Outputs

- Output tensor Y1, Y2.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = SufficientStatistics'.

- axis

  A list of integers to indicate the axis to calcaulate across. Typically, this is the feature axis and the leaving dimensions are typically the batch axis. Generally, the value is -1 which represents the last dimension of the input tensor (X).

- scale: scale_type, scale_value

  The scale is a vector with four elements. Where, scale_type and scale_value are the data type and value of the scale during quantization. Generally, scale_type has the int8, uint8, and int16 options. Furthermore, the scales type or value must be in order of [squre_scale, sum_scale, input_scale, shift_scale].

- shift: shift_type, shift_value

  The shift is a vector with four elements. Where, shift_type and shift_value are the data type and value of the scale during quantization. Generally, shift_type has the int8 and uint8 options. Furthermore, the shifts type or value must be in order of [squre_shift, sum_shift, input_shift,shift_shift].

Swish

Takes one input tensor and produces one output tensor, where the SiLU or Swish function y = x * sigmoid(alpha*x), is applied to the tensor elementwise and 'sigmoid(alphax)' is the logistic sigmoid function.

Inputs

- Input data tensor X.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = Activation'.

- method: SWISH

  Method to perform the input tensor.

- alpha

  A coefficient parameter of the activation function in a float IR.

- lut: lut_type, lut_shape, lut_offset, lut_size

The *Look Up Table* (LUT) is an offline table created after quantization. Where, lut_type, lut_shape, lut_offset and lut_size are the data type, table shape, table offset address and table size. The following is a simple equation, lut_size = data_type * lut_shape. Where, data type = 1 (or 1, 2, 2, 4, 4) if lut_type = int8 (or uint8, int16, uint16, int32, uint32).

Tan

Calculates the tangent of the given input tensor, element-wise.

Inputs

- Input data tensor X.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = Tan'.

- lut: lut_type, lut_shape, lut_offset, lut_size

  The *Look Up Table* (LUT) is an offline table created after quantization. Where, lut_type, lut_shape, lut_offset and lut_size are the data type, table shape, table offset address and table size. The following is a simple equation, lut_size = data_type * lut_shape. Where, data type = 1 (or 1, 2, 2, 4, 4) if lut_type = int8 (or uint8, int16, uint16, int32, uint32).

Tanh

Calculates the hyperbolic tangent of the given input tensor, element-wise.

Inputs

- Input data tensor X.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = Activation'.

- method: TANH

  Method to perform the input tensor.

- lut: lut_type, lut_shape, lut_offset, lut_size

  The *Look Up Table* (LUT) is an offline table created after quantization. Where, lut_type, lut_shape, lut_offset and lut_size are the data type, table shape, table offset address and table size. The following is a simple equation, lut_size = data_type * lut_shape. Where, data type = 1 (or 1, 2, 2, 4, 4) if lut_type = int8 (or uint8, int16, uint16, int32, uint32).

ThresholdedRelu

ThresholdedRelu takes one input data (Tensor) and produces one output data (Tensor) where the rectified linear function, y = x for x > alpha, y = 0 otherwise, is applied to the tensor elementwise.

Inputs

- Input data tensor X.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = Activation'.

- method: THRESHOLDEDRELU

  Method to perform the input tensor.

- alpha: float(default is 1.0)

  Threshold value. Only needed in float IR. The default value is 1.0.

- lut: lut_type, lut_shape, lut_offset, lut_size

  The *Look Up Table* (LUT) is an offline table created after quantization. Where, lut_type, lut_shape, lut_offset and lut_size are the data type, table shape, table offset address and table size. The following is a simple equation, lut_size = data_type * lut_shape. Where, data type = 1 (or 1, 2, 2, 4, 4) if lut_type = int8 (or uint8, int16, uint16, int32, uint32).

Tile

Constructs a tensor by tiling a given tensor. This is a function similarly as NumPy-style, but no broadcast. For example, A = [[0,1], [2, 3]], B = [1, 2], tile(A, B) = [[0, 1, 0, 1], [2, 3, 2, 3]].

Inputs

- Input data tensor X.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = Tile'.

- repeats

  A 1D tensor of the same length as the input tensor's dimension number, including numbers of repeated copies along the input's dimensions.

TopK

Retrieves the top-K largest or smallest elements along a specified axis. Given an input tensor of shape [a_1, …, a_i, …, a_n] and integer argument K, it returns two outputs:

- 'Values' tensor of shape [a_1, …, a_i, …, a_k] which contains the values of the top K elements along the specified axis.
- 'Indices' tensor of shape [j_1, …, j_i, …, j_k] which contains the indices of the top K elements (original indices from the input tensor).

Inputs

- Input data tensor X.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = TopK'.

- k

  A scalar containing a single positive value corresponding to the number of top elements to retrieve.

- axis

  A scalar means dimension on which to perform the sort. A negative value means counting dimensions from the back, where axis < rank(input tensor). Generally, '-1' means the last dimension. The accepted range is [-1, r-1] where r = rank(input data).

- largest: true, false

  Indicates whether to return the top-K largest or smallest elements. The default value is **true**. If 'largest' is **true**, the K largest elements are returned.

- sorted: true, false

  Indicates whether to return the elements in sorted order. The default value is **true**. If 'sorted' is **true**, the resulting K elements will be sorted. If 'sorted' is **false**, the order of returned 'value' and 'index' is undefined.

- select_index: first, last, random

  An optional parameter during quantization in an int IR. To select the last index or the first index if the maximum element appears many times. It defaults to **last**.

Transpose

Permutes the input tensor similarly to NumPy-style. For example, given an input tensor with the shape [1, 2, 3, 4] and perm = [1, 0, 2, 3], it returns an output tensor with the shape as [2, 1, 3, 4].

Inputs

- Input data tensor X.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = Transpose'.

- perm

  A list of integers to indicates the permutation of the axis according to the value given.

Trunc

The truncated value of the input x is the nearest integer i which is closer to zero than x, element-wise.

Inputs

- Input data tensor X.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = Trunc'.

- lut: lut_type, lut_shape, lut_offset, lut_size

  The *Look Up Table* (LUT) is an offline table created after quantization. Where, lut_type, lut_shape, lut_offset and lut_size are the data type, table shape, table offset address and table size. The following is a simple equation, lut_size = data_type * lut_shape. Where, data type = 1 (or 1, 2, 2, 4, 4) if lut_type = int8 (or uint8, int16, uint16, int32, uint32).

UpsampleByIndex

Performs the upsample operation on the inputs according to the index in the corresponding indices tensor and returns the output tensor Y. Generally, the input indices tensor has three flattened formats. For example, a maximum value in the original tensor at position '[n, h, w, c]' will be flattened as:

'indices = h * width + w' if flatten_dim = HW

'indices = (h * width + w) * channel + c' if flatten_dim = HWC

'indices = ((n * height + h) * width + w) * channel + c' if flatten_dim = NHWC

Where, height (or width, channel) represents the size of corresponding H (or W, C) dimension under NHWC data format. Generally, the indices are located at ([0, height], [0, width]) originally before flattening even though padding is involved. Where the range is a left closed and right open interval.

Inputs

- Input data tensor X1, indices tensor X2.

Outputs

- Output data tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = UpsampleByIndex'.

- flatten_dim: NHWC, HWC, HW

  Indicates the flatten mode in the indices tensor. NHWC means including batch dimension in the flattened index during upsampling. HWC means including Height and Width dimension in the flattened index during upsampling. HW means including the Width dimension in the flattened index during upsampling. Generally, it defaults to HWC mode.

- storage_order:

  An int value of storage_order of the output index tensor. The default value is 0. 0 is row-major, and 1 is column-major.

- output_shape

  Specifies the shape of the output tensor.

Where

Selects elements from x or y, depending on different conditions. That is, the condition tensor acts as a mask that chooses, based on the value at each element, whether the corresponding element or row in the output should be taken from x (condition 'True') or y (condition 'false'). The parameters are in order of [condition, X1, X2]. Generally, the data type of input condition tensor (C1) is an integer in float or int IR, and its value is 0 or 1. In addition, the shape of all the three input tensors is broadcastable. For example,

- Non-broadcast mode:

  If input tensor C1 = [1, 0, 0, 1], X1 = [1, 2, 3, 4], X2 = [10, 20, 30, 40], it returns output tensor Y = [1, 20, 30, 4].

- Broadcast mode:
  - If input tensor C1 = [1, 0, 0, 1], X1 = [1, 2, 3, 4], X2 = [10], it returns output tensor Y = [1, 10, 10, 4].
  - If input tensor C1 = [1, 0, 0, 1], X1 = [1], X2 = [10], it returns output tensor Y = [1, 10, 10, 1].
  - If input tensor C1 = [1], X1 = [1, 2, 3, 4], X2 = [10], it returns output tensor Y = [1, 2, 3, 4].
  - If input tensor C1 = [0], X1 = [1, 2, 3, 4], X2 = [10], it returns output tensor Y = [10, 10, 10, 10].

Inputs

- Input condition tensor C1, data tensor X1, data tensor X2.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = Where'.

- scale: scale_type, scale_value

  It can be a 1-D tensor during quantization in order of input tensor [X1, X2], where scale_type and scale_value are the data type and value of the scale during per tensor or layer quantization.

- shift: shift_type, shift_value

  It can be a 1-D tensor during quantization in order of input tensor [X1, X2], where scale_type and scale_value are the data type and value of the scale during per tensor or layer quantization.

YuvToRgb

Conversion of one or more images from YUV to RGB. Generally, this operation is used for preprocessing. Here, the term YUV is commonly encoded using Y'CbCr, while the RGB term is encoded using R'dG'dB'd. There are three standards which are so-called 'BT.601', 'BT.709' and 'BT.2020'. For 8 bits color, the [0, 255] range is referred as full range, which is considered here, while the [16, 235] range for Y and [16, 240] for U/V are referred as narrow range. In addition, all the conversions are based on integral approximate (or using fixed-point arithmetic as an alternative formulation) in this operation.

For BT.601 with full range, conversion matrices with integral approximate are:

$$\begin{bmatrix} R'_d \\ G'_d \\ B'_d \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} + \left(\begin{bmatrix} 256 & 0 & 359 \\ 256 & -88 & -183 \\ 256 & 454 & 0 \end{bmatrix} \cdot \left(\begin{bmatrix} Y' \\ C_b \\ C_r \end{bmatrix} - \begin{bmatrix} 0 \\ 128 \\ 128 \end{bmatrix} \right) + \begin{bmatrix} 128 \\ 128 \\ 128 \end{bmatrix} \right) \gg 8$$

For BT.709 with full range, conversion matrices with integral approximate are:

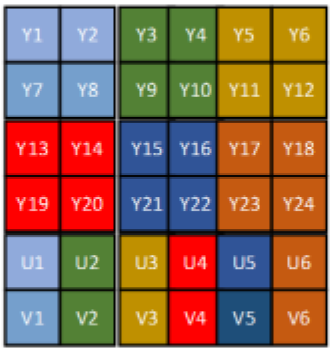
$$\begin{bmatrix} R'_d \\ G'_d \\ B'_d \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} + \left(\begin{bmatrix} 256 & 0 & 403 \\ 256 & -48 & -120 \\ 256 & 475 & 0 \end{bmatrix} \cdot \left(\begin{bmatrix} Y' \\ C_b \\ C_r \end{bmatrix} - \begin{bmatrix} 0 \\ 128 \\ 128 \end{bmatrix} \right) + \begin{bmatrix} 128 \\ 128 \\ 128 \end{bmatrix} \right) \gg 8$$

Where [0, 128, 128] is the input offset, $[0, 0, 0]^T$ is the output offset, and [[256, 0, 403],[256 -48, -120],[256, 475,0]] are the coefficient. RGB files are typically encoded in 8, 12, 16, 24 bits per pixel. For example, 24 bits per pixel, which is written as RGB888, the standard byte format in the memory device is stacked as:

RGB888: r0, g0, b0, r1, g1, b1, …

YUV files can be encoded in 12,16 or 24 bits per pixel. The common formats are YUV444, YUV422, YUV420p and YUV420sp. For example, YUV420p and YUV420sp with planar format, the standard format in memory device is stacked as follows:

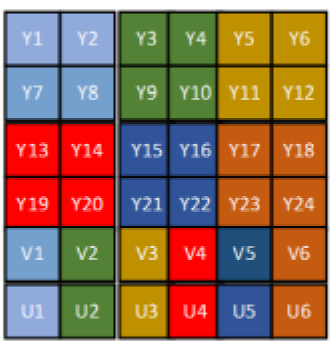
I420: Single Frame



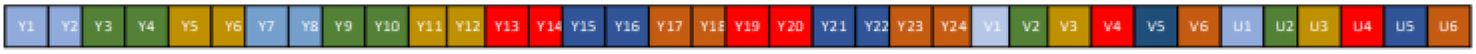
Position in the byte stream:

As shown in the preceding diagram, the Y, U and V components in YUV420p are encoded separately in sequential blocks. A Y value is stored for every pixel, followed by a U value for each 2x2 square block. The corresponding Y, U and V values are shown using the same color in the preceding diagram. When read line by line as the byte stream from a device, the Y block will be found at position 0, the U block at position xy *(6x4 = 24 in this case) and the V block at position* xy + (x * y)/4 (6x4 + (6x4)/4 = 30 in this case).

YV12: Single Frame

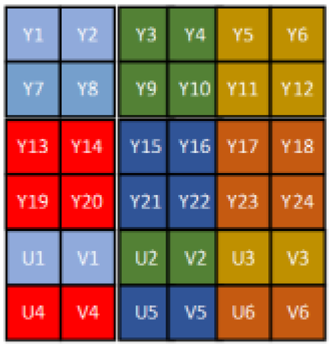


Position in the byte stream:



As shown in the preceding diagram, the Y, U and V components in YUV420p are encoded separately in sequential blocks. A Y value is stored for every pixel, followed by a U value for each 2x2 square block. The corresponding Y, U and V values are shown using the same color in the preceding diagram. When read line by line as the byte stream from a device, the Y block will be found at position 0, the V block at position x*y (6x4 = 24 in this case) and the U block at position* xy + (x * y)/4 (6x4 + (6x4)/4 = 30 in this case).

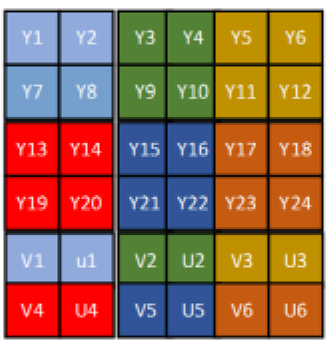
NV12: Single Frame



Position in the byte stream:

As shown in the preceding diagram, the Y, U and V components in NV12 are encoded separately in sequential blocks. A Y value is stored for every pixel, A Y value is stored for every pixel, followed by a interleave of U/V for each 2x2 square block. The corresponding Y, U and V values are shown using the same color in the preceding diagram. When read line by line as the byte stream from a device, the Y block will be found at position 0, followed by interleaved U/V plane with 8 bits subsampled chroma samples from the position x*y (6x4 = 24 in this case).

NV21: Single Frame



Position in byte the stream:

This format is the standard picture on Android camera preview. As shown in the preceding diagram, the Y, U and V components in NV21 are encoded separately in sequential blocks. A Y value is stored for every pixel, followed by a interleave of V/U for each 2x2 square block. The corresponding Y, U and V values are shown using the same color in the preceding diagram. When read line by line as the byte stream from a device, the Y block will be found at position 0, followed by interleaved V/U plane with 8 bits subsampled chroma samples from the position x*y (6x4 = 24 in this case).

Inputs

- Input YUV image tensor, with a shape of [N, D].

Outputs

- Output RGB image tensor, with a shape of [N, H, W, 3].

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = YuvToRgb'.

- shape

  The shape of the output tensor.

- format: I420, YV12, NV12, NV21

  The input image format which will be converted. Generally, the formats of 'I420' and 'YV12' mean that the Y, U and V values are grouped together instead of interspersed. When given an array of an image in the 'I420' format, all the Y values come first, followed by all the U values, and then followed by all the V values. While in the 'YV12' format, all the Y values come first, followed by all the V values, and then followed by all the U values.

  The others are the formats of 'NV12' and 'NV21', which mean that Y values are grouped together, but U and V values are interspersed. When given an array of an image in the 'NV12' format, all the Y values come first, followed by the interleaved U and V value. While in the 'NV21' format, all the Y values come first, followed by the interleaved V and U value as the replacement.

- bits: 8, 10

  The bits to store Y, U or V values in memory, such as the DDR device.

- conversion: BT601, BT709, BT2020, SELF

  Means the different coefficient matrix to convert between the YUV and RGB formats. Especially, 'SELF' means that the conversion coefficients are customized, which is from the configuration.

- coefficient

  A list of color space coefficient (CSC) for conversion, which is in order of [input_offset, output_offset, coefficient]. For example, the list = [0, 128, 128, 0, 0, 0, 256, 0, 403, 256 -48, -120, 256, 475,0] in BT709 format with integral approximate. While, the list = [0, 128, 128, 0, 0, 0, 1.000000, 0.000000, 1.574800, 1.000000, -0.187324, -0.468124, 1.000000, 1.855600, 0.000000] in BT709 format with float coefficient.

- coefficient_dtype: int8, int10, int16, float32

  The data type of coefficient.

- coefficient_shift

  An integer of the shift value depends on the data type of the 'coefficient' parameter, which means the shift value before the 'output_offset' operation. For example, coefficient_shift = 8 under integral approximate and coefficient_shift = 0 under float type.

ZeroFraction

Returns a fraction of zeros in input tensor (X). This is useful in summaries to measure and report sparsity. For example, the input tensor 'X = [[0, 0, 1, 2], [10, 0, 11, 21]]', then the output tensor 'Y = 0.375'.

Inputs

- Input data tensor X.

Outputs

- Output tensor Y.

Attributes

- layer_type

  The operation type of a layer. Here is 'layer_type = ZeroFraction'.

4.4 Float IR and int IR

Float IR and int IR are IR descriptions with different data types, such as different precision attributes. Generally, most of the parameters between them are the same. A float IR can be transformed into an int IR with quantization processing in the NN compiler.

4.4.1 Float IR

A float IR is an IR description with the float data type. A float IR can be generated from the parser tool and used for quantization or the optimization tool in the NN compiler. It includes the basic common and layer parameters and some other special parameters in a float IR. The following are some examples:

- For some quantization models (such as, TFLite framework), 'layer_top_range' and 'weights_range' maybe exist in a float IR and will be deleted from an int IR. For example,
  - layer_top_range=[-1.52, 2.33]
  - weights_range=[-0.544, 0.8423]
- For some special parameters in LayerNormalization, the 'epsilon' parameter is used to avoid division by zero. So, it exists in a float IR and will be deleted from an int IR.

4.4.2 Int IR

An int IR is an IR description with int8 (or int4, uint4, uint8, int16, uint16) data type according to different quantization implementation. Generally, an int IR can be generated from a float IR and used for the graph compiler tool in the NN compiler. It includes the basic common and layer parameters and some other special parameters in an int IR. The following are some examples:

- For most linear operators that need to be quantized by tensor, like Convolution and Eltwise, several parameters such as 'scale_type', 'scale_value', 'shift_type' and 'shift_value' will be added in an int IR during quantization.
- For some operators that need to be quantized by channel, like the DepthwiseConv operator, several parameters such as 'scale_type', 'scale_shape', scale_offset', 'scale_size', 'shift _type', 'shift _shape', 'shift _offset' and 'shift _size' will be added in an int IR during quantization. For example,

```
scale_type=int8
scale_offset=1056
scale_size=96
scale_shape=[24]
shift_type=int8
shift_offset=1152
shift_size=24
shift_shape=[24]
```

- For some activation operators that need to be quantized with the LUT strategy, like Sigmoid and Tanh, ‘lut_type’, ‘lut_shape’, ‘lut_offset’ and ‘lut_size’ will be added in an int IR during quantization. For example,

```
lut_type=int8
lut_offset=3456
lut_size=256
lut_shape=[256]
```

- For some activation ranges that need to be quantized, like **Clip**, the range of ‘clip_min’ and ‘clip_max’ values will be changed in an int IR during quantization.

4.5 Rules

The Zhouyi Compass IR is a specification format for the AIPU NN compiler. So, the following basic rules must be obeyed:

- The valid characters are `a-z`, `A-Z`, `0-9`, `.`, `:`, `/`, `_`, `;`, and `‘`.
- No annotations are allowed.
- A parameter with prefix 'layer_' is the keyword in the IR definition, so a customer parameter should not start with this keyword. For example, strings with prefix 'op_' or 'tensor_' are alternatives to avoid the conflict during user-defined development.
- A list parameter should be in brackets []. For example,

```
top_shape = [1,224,224,3]
```

- All parameters with suffix '_shape' should be in brackets. For example,

```
biases_shape=[96]
weight_shape=[4,16,16,32]
```

- A tensor like object is described by at least three keys including name, type and shape. In addition, some constant tensors, like size and offset, are also expected. All these keys share the same prefix with a '_' separator. For example,
  - A tensor described by name:

```
layer_top=MobilenetV2/depthwise6/Relu6_0
layer_top_shape=[1,28,28,192]
layer_top_type=uint8
```

  - A tensor described by data:

```
weights_type=int8
weights_offset=21431236
weights_size=393216
weights_shape=[192,1,1,2048]
```

- A value with type is combined by two keys—value and type with the same prefix separated by a '_'. For example,

```
shift_value=6
shift_type=int8
```

- For some optional parameters in a float or int IR, it is recommended to set the attribute with an empty value. For example,

```
layer_bottom=
layer_bottom_shape=
layer_bottom_type=
```

5 Build-in operator examples

The Zhouyi Compass IR is a specification format for the Zhouyi Compass NN compiler. Therefore, some rules must be followed.

The following are some NN compiler build-in int IR examples for your reference.

Abs

```
layer_id=1
layer_name=Abs
layer_type=Abs
layer_bottom=[Placeholder_0]
layer_bottom_shape=[[3,75,11,7]]
layer_bottom_type=[int8]
layer_top=[Abs_0]
layer_top_shape=[[3,75,11,7]]
layer_top_type=[uint8]
layer_top_datalayout=[NHWC]
layer_top_scale=[1.000000]
layer_top_zp=[0]
```

AccidentalHits

```
layer_id=4
layer_name=ComputeAccidentalHits
layer_type=AccidentalHits
layer_bottom=[reshape0,reshape1]
layer_bottom_shape=[[4,5],[30]]
layer_bottom_type=[uint16,uint16]
layer_top=[ComputeAccidentalHits_0,ComputeAccidentalHits_1,ComputeAccidentalHits_2]
layer_top_shape=[[20],[20],[1]]
layer_top_type=[uint16,uint16,uint16]
layer_top_datalyout=[NHWC,NHWC,NHWC]
layer_top_scale=[1.000000,1.000000,1.000000]
layer_top_zp=[0,0,0]
```

Acos

```
layer_id=1
layer_name=Acos_0
layer_type=Acos
```

```
layer_bottom=[Placeholder_0]
layer_bottom_shape=[[2,3]]
layer_bottom_type=[int8]
layer_top=[Acos_0]
layer_top_shape=[[2,3]]
layer_top_type=[uint8]
layer_top_scale=[181.162262]
layer_top_zp=[0]
lut_type=uint8
lut_offset=0
lut_size=256
lut_shape=[256]
```

Acosh

```
layer_id=1
layer_name=Acosh_0
layer_type=Acosh
layer_bottom=[Placeholder_0]
layer_bottom_shape=[[2,3,4]]
layer_bottom_type=[int8]
layer_top=[Acosh_0]
layer_top_shape=[[2,3,4]]
layer_top_type=[uint8]
layer_top_scale=[177.340149]
layer_top_zp=[0]
lut_type=uint8
lut_offset=0
lut_size=256
lut_shape=[256]
```

Add

```
layer_id=3
layer_name=Add_
layer_type=Add
layer_bottom=[input_0,input_1]
layer_bottom_shape=[[2,256],[256]]
layer_bottom_type=[int8,int8]
```

```
layer_top=[output]
layer_top_shape=[[2,256]]
layer_top_type=[int8]
layer_top_scale=[0.001733]
layer_top_zp=[0]
scale_type=[uint8,uint16,uint16]
scale_value=[188,32767,256]
shift_type=int8
shift_value=29
layer_top_scale=[1.000000]
layer_top_zp=[0]
```

ArgMax

```
layer_id=1
layer_name=ArgMax
layer_type=ArgMinMax
layer_bottom=[Placeholder_0]
layer_bottom_shape=[[5,40,15,8]]
layer_bottom_type=[int8]
layer_top=[ArgMax_0]
layer_top_shape=[[1,40,15,8]]
layer_top_type=[uint16]
layer_top_datalayout=[NHWC]
layer_top_scale=[1.000000]
layer_top_zp=[0]
axis=0
method=MAX
select_last_index=false
```

ArgMin

```
layer_id=1
layer_name=ArgMin
layer_type=ArgMinMax
layer_bottom=[Placeholder_0]
layer_bottom_shape=[[1,91,66,4]]
layer_bottom_type=[int8]
layer_top=[ArgMin_0]
```

```
layer_top_shape=[[1,91,66,4]]
layer_top_type=[uint16]
layer_top_datalayout=[NHWC]
layer_top_scale=[1.000000]
layer_top_zp=[0]
axis=0
method=MIN
select_last_index=false
```

Asin

```
layer_id=1
layer_name=Asin_0
layer_type=Asin
layer_bottom=[Placeholder_0]
layer_bottom_shape=[[2,3,4]]
layer_bottom_type=[int8]
layer_top=[Asin_0]
layer_top_shape=[[2,3,4]]
layer_top_type=[int8]
layer_top_scale=[144.570541]
layer_top_zp=[0]
lut_type=int8
lut_offset=0
lut_size=256
lut_shape=[256]
```

Asinh

```
layer_id=1
layer_name=Asinh_0
layer_type=Asinh
layer_bottom=[Placeholder_0]
layer_bottom_shape=[[2,3]]
layer_bottom_type=[int8]
layer_top=[Asinh_0]
layer_top_shape=[[2,3]]
layer_top_type=[int8]
layer_top_scale=[126.841988]
```

```
layer_top_zp=[0]
lut_type=int8
lut_offset=0
lut_size=256
lut_shape=[256]
```

AveragePooling2D

```
layer_id=1
layer_name=AvgPool
layer_type=Pooling
layer_bottom=[Placeholder_0]
layer_bottom_shape=[[1,224,224,3]]
layer_bottom_type=[int8]
layer_top=[AvgPool_0]
layer_top_shape=[[1,1,32,3]]
layer_top_type=[int8]
layer_top_datalayout=[NHWC]
layer_top_scale=[27.469427]
layer_top_zp=[0]
dilation_x=1
dilation_y=1
kernel_x=5
kernel_y=3
method=AVG
pad_bottom=0
pad_left=0
pad_right=0
pad_top=0
stride_x=7
stride_y=300
ceil_mode=false
```

AveragePooling3D

```
layer_id=1
layer_name=AvgPool3D
layer_type=Pooling3D
layer_bottom=[Placeholder_0]
```

```
layer_bottom_shape=[[2,21,22,23,3]]
layer_bottom_type=[int8]
layer_top=[AvgPool3D_0]
layer_top_shape=[[2,21,11,8,3]]
layer_top_type=[int8]
layer_top_scale=[28.085308]
layer_top_zp=[0]
ceil_mode=false
count_include_pad=false
dilation_x=1
dilation_y=1
dilation_z=1
kernel_x=7
kernel_y=5
kernel_z=3
method=AVG
pad_x_begin=2
pad_x_end=3
pad_y_begin=1
pad_y_end=2
pad_z_begin=1
pad_z_end=1
stride_x=3
stride_y=2
stride_z=1
ceil_mode=false
```

BNLL

```
layer_id=1
layer_name=bnll
layer_type=BNLL
layer_bottom=[Placeholder]
layer_bottom_shape=[[8,6,92,3]]
layer_bottom_type=[int8]
layer_top=[bnll]
layer_top_shape=[[8,6,92,3]]
layer_top_type=[uint8]
```

```
layer_top_datalayout=[NHWC]
layer_top_scale=[69.139595]
layer_top_zp=[0]
lut_type=uint8
lut_offset=0
lut_size=256
lut_shape=[256]
```

BasicLSTM

```
layer_id=6
layer_name=LSTM_Layer
layer_type=BasicLSTM
layer_bottom=[input_00,tensor_1,tensor_2]
layer_bottom_shape=[[1,49,13],[1,256],[1,256]]
layer_bottom_type=[int8,int8,int8]
layer_top=[LSTM-Layer/lstm/rnn/while/Exit_4_0]
layer_top_shape=[[1,256]]
layer_top_type=[int8]
layer_top_datalayout=[NHWC]
layer_top_scale=[127.500977]
layer_top_zp=[0]
biases_type=int32
biases_offset=512
biases_size=4096
biases_shape=[1024]
lut_ct_type=int8
lut_ct_offset=4608
lut_ct_size=256
lut_ct_shape=[256]
lut_ft_type=int8
lut_ft_offset=4864
lut_ft_size=256
lut_ft_shape=[256]
lut_h_type=int8
lut_h_offset=5120
lut_h_size=256
lut_h_shape=[256]
```

```
lut_it_type=int8
lut_it_offset=5376
lut_it_size=256
lut_it_shape=[256]
lut_ot_type=int8
lut_ot_offset=5632
lut_ot_size=256
lut_ot_shape=[256]
scale_type=uint8
scale_offset=5888
scale_size=103
scale_shape=[103]
shift_type=int8
shift_offset=5991
shift_size=103
shift_shape=[103]
weights_type=int8
weights_offset=6094
weights_size=275456
weights_shape=[1024,269]
activations=[SIGMOID,TANH,TANH]
cell_size=256
direction=forward
input_size=13
lut_shift_type=int8
lut_shift_value=0
out_sequence=[H]
time_steps=49
```

BatchNormalization

```
layer_id=1
layer_name=batch_normalization/FusedBatchNorm
layer_type=BatchNorm
layer_bottom=[Placeholder_0]
layer_bottom_shape=[[4,48,5,1]]
layer_bottom_type=[int8]
layer_top=[batch_normalization/FusedBatchNorm_0]
```

```
layer_top_shape=[[4,48,5,1]]
layer_top_type=[int8]
layer_top_datalayout=[NHWC]
layer_top_scale=[37.027405]
layer_top_zp=[0]
biases_type=int32
biases_offset=0
biases_size=4
biases_shape=[1]
weights_type=int16
weights_offset=4
weights_size=2
weights_shape=[1]
axis=3
scale_type=uint8
scale_value=128
shift_type=int8
shift_value=22
```

BatchToSpace

```
layer_id=1
layer_name=BatchToSpaceND
layer_type=BatchToSpace
layer_bottom=[Placeholder_0]
layer_bottom_shape=[[4,1,1,1]]
layer_bottom_type=[int8]
layer_top=[BatchToSpaceND_0]
layer_top_shape=[[1,2,2,1]]
layer_top_type=[int8]
layer_top_datalayout=[NHWC]
layer_top_scale=[142.575928]
layer_top_zp=[0]
block_size_x=2
block_size_y=2
crop_bottom=0
crop_left=0
crop_right=0
```

```
crop_top=0
```

BiasAdd

```
layer_id=1
layer_name=bias_add0
layer_type=BiasAdd
layer_bottom=[Input0]
layer_bottom_shape=[[1,25,16,64]]
layer_bottom_type=[int8]
layer_top=[bias_add0]
layer_top_shape=[[1,25,16,64]]
layer_top_type=[int8]
biases_type=int32
biases_offset=0
biases_size=256
biases_shape=[64]
scale_type=uint8
scale_value=192
shift_type=int8
shift_value=-7
```

BitwiseAnd

```
layer_id=2
layer_name=BitwiseAnd
layer_type=Bitwise
layer_bottom=[Placeholder,Placeholder_1]
layer_bottom_shape=[[2,3,4,5],[2,3,4,5]]
layer_bottom_type=[uint8,uint8]
layer_top=[BitwiseAnd]
layer_top_shape=[[2,3,4,5]]
layer_top_type=[uint8]
method=AND
```

BitwiseNot

```
layer_id=2
layer_name=BitwiseNot
layer_type=Bitwise
layer_bottom=[Placeholder]
```

```
layer_bottom_shape=[[2,3,4,5]]
layer_bottom_type=[int8]
layer_top=[BitwiseNot]
layer_top_shape=[[2,3,4,5]]
layer_top_type=[int8]
method=NOT
```

BitwiseOr

```
layer_id=2
layer_name=BitwiseOr
layer_type=Bitwise
layer_bottom=[Placeholder,Placeholder_1]
layer_bottom_shape=[[2,3,4,5],[2,3,4,5]]
layer_bottom_type=[uint8,uint8]
layer_top=[BitwiseOr]
layer_top_shape=[[2,3,4,5]]
layer_top_type=[uint8]
method=OR
```

BitwiseXor

```
layer_id=2
layer_name=BitwiseXor
layer_type=Bitwise
layer_bottom=[Placeholder,Placeholder_1]
layer_bottom_shape=[[2,3,4,5],[2,3,4,5]]
layer_bottom_type=[uint8,uint8]
layer_top=[BitwiseXor]
layer_top_shape=[[2,3,4,5]]
layer_top_type=[uint8]
method=XOR
```

BoundingBox

```
layer_id=3
layer_name=boundingbox
layer_type=BoundingBox
layer_bottom=[box,delta]
layer_bottom_shape=[[1,6000,4],[1,6000,4]]
layer_bottom_type=[int16,int8]
```

```
layer_top=[out0,out1]
layer_top_shape=[[1,6000,4]]
layer_top_type=[int16]
th_lut_type=int16
th_lut_offset=0
th_lut_size=512
th_lut_shape=[256]
tw_lut_type=int16
tw_lut_offset=512
tw_lut_size=512
tw_lut_shape=[256]
box_scale_value=[20647,20647,30615,30615,16352]
box_scale_type=[uint16,uint16,uint16,uint16,uint16]
box_shift_value=[23,23,30,30,17]
box_shift_type=[int8,int8,int8,int8,int8]
delta_shift=9
```

CRelu

```
layer_id=1
layer_name=CRelu
layer_type=Activation
layer_bottom=[Placeholder_0]
layer_bottom_shape=[[4,42,90,5]]
layer_bottom_type=[int8]
layer_top=[CRelu/Relu_0]
layer_top_shape=[[4,42,90,10]]
layer_top_type=[uint8]
layer_top_datalayout=[NHWC]
layer_top_scale=[51.060760]
layer_top_zp=[0]
axis=3
method=CRELU
scale_type=uint8
scale_value=128
shift_type=int8
shift_value=6
```

CTCGreedyDecoder

```
layer_id=2
layer_name=CTCGreedyDecoder
layer_type=CTCGreedyDecoder
layer_bottom=[Placeholder_0,seq_len_0]
layer_bottom_shape=[[1,20,333],[1]]
layer_bottom_type=[uint8,uint8]
layer_top=[CTCGreedyDecoder_0]
layer_top_shape=[[1,100,1,1]]
layer_top_type=[uint16]
layer_top_datalayout=[NHWC]
layer_top_scale=[1.000000]
layer_top_zp=[0]
```

Cast

```
layer_id=1
layer_name=CastV2
layer_type=Cast
layer_top=[InceptionV2/InceptionV2/Mixed_5b/CastV2_0]
layer_top_shape=[[1,32,32,32]]
layer_top_type=[uint8]
layer_bottom=[input_u8_1]
layer_bottom_shape=[[1,32,32,32]]
layer_bottom_type=[int8]
to_dtype=uint8
scale_type=uint8
scale_value=128
shift_type=int8
shift_value=6
layer_top_scale=[1.0]
layer_top_zp=[0]
ignore_scale_zp=false
clip_mode=SATURETION
```

Ceil

```
layer_id=1
layer_name=Ceil
```

```
layer_type=Ceil
layer_bottom=[Placeholder_0]
layer_bottom_shape=[[5,53,74,2]]
layer_bottom_type=[int8]
layer_top=[Ceil_0]
layer_top_shape=[[5,53,74,2]]
layer_top_type=[int8]
layer_top_scale=[31.875000]
layer_top_zp=[0]
lut_type=int8
lut_offset=0
lut_size=256
lut_shape=[256]
```

Celu

```
layer_id=1
layer_name=Celu_0
layer_type=Activation
layer_bottom=[Placeholder_0]
layer_bottom_shape=[[2,3]]
layer_bottom_type=[int8]
layer_top=[Celu_0]
layer_top_shape=[[2,3]]
layer_top_type=[int8]
layer_top_scale=[95.994194]
layer_top_zp=[0]
lut_type=int8
lut_offset=0
lut_size=256
lut_shape=[256]
method=CELU
```

ChannelShuffle

```
layer_id=1
layer_name=shuffle2
layer_type=ChannelShuffle
layer_bottom=[data]
```

```
layer_bottom_shape=[[1,28,28,60]]
layer_bottom_type=[int8]
layer_top=[shuffle2_0]
layer_top_shape=[[1,28,28,60]]
layer_top_type=[int8]
layer_top_scale=[29.555828]
layer_top_zp=[0]
group=3
splits=1
```

Clip

```
layer_id=1
layer_name=clip
layer_type=Activation
layer_bottom=[Placeholder]
layer_bottom_shape=[[2,9,6,1]]
layer_bottom_type=[int8]
layer_top=[clip]
layer_top_shape=[[2,9,6,1]]
layer_top_type=[int8]
layer_top_datalayout=[NHWC]
layer_top_scale=[59.420982]
layer_top_zp=[0]
clip_max=65972.000000
clip_min=-18784.000000
method=CLIP
scale_type=uint16
scale_value=17649
shift_type=int8
shift_value=6
```

Compress

```
layer_id=2
layer_name=Compress_0
layer_type=Compress
layer_bottom=[Placeholder_0,Placeholder_1_Initializer]
layer_bottom_shape=[[5,6,7],[6]]
```

```
layer_bottom_type=[int8,uint8]
layer_top=[Compress_0]
layer_top_shape=[[5,5,7]]
layer_top_type=[int8]
layer_top_scale=[37.79585]
layer_top_zp=[0]
axis=1
```

Concat

```
layer_id=2
layer_name=concat
layer_type=Concat
layer_bottom=[Placeholder_0,Placeholder_1_0]
layer_bottom_shape=[[4,11,32,9],[4,11,32,9]]
layer_bottom_type=[int8,int8]
layer_top=[concat_0]
layer_top_shape=[[8,11,32,9]]
layer_top_type=[int8]
layer_top_datalayout=[NHWC]
layer_top_scale=[29.451674]
layer_top_zp=[0]
axis=0
scale_type=[uint8,uint8]
scale_value=[138,234]
shift_type=[int8,int8]
shift_value=[7,8]
```

Constant

```
layer_id=1
layer_name=GatherV2/indices
layer_type=Constant
layer_bottom=[]
layer_bottom_shape=[[]]
layer_bottom_type=[]
layer_top=[GatherV2/indices]
layer_top_shape=[[17,19]]
layer_top_type=[uint16]
```

```
weights_type=uint16
weights_offset=0
weights_size=646
weights_shape=[17,19]
```

ConvTranspose2D

```
layer_id=1
layer_name=conv2d_transpose
layer_type=ConvTranspose
layer_bottom=[Placeholder_0]
layer_bottom_shape=[[1,200,100,5]]
layer_bottom_type=[int8]
layer_top=[conv2d_transpose_0]
layer_top_shape=[[1,200,100,1]]
layer_top_type=[int8]
layer_top_datalayout=[NHWC]
layer_top_scale=[159.349762]
layer_top_zp=[0]
biases_type=int32
biases_offset=0
biases_size=4
biases_shape=[1]
weights_type=int8
weights_offset=4
weights_size=5
weights_shape=[1,1,1,5]
dilation_x=1
dilation_y=1
group=1
kernel_x=1
kernel_y=1
num_output=1
pad_bottom=0
pad_left=0
pad_right=0
pad_top=0
scale_type=uint8
```

```
scale_value=192
shift_type=int8
shift_value=15
stride_x=1
stride_y=1
with_activation=NONE
```

ConvTranspose3D

```
layer_id=1
layer_name=conv3d_transpose
layer_type=ConvTranspose3D
layer_bottom=[Placeholder_0]
layer_bottom_shape=[[2,16,32,64,3]]
layer_bottom_type=[int8]
layer_top=[conv3d_transpose_0]
layer_top_shape=[[2,16,32,64,4]]
layer_top_type=[int8]
layer_top_scale=[7.715990]
layer_top_zp=[0]
biases_type=int32
biases_offset=0
biases_size=16
biases_shape=[4]
weights_type=int8
weights_offset=16
weights_size=6048
weights_shape=[4,8,7,9,3]
dilation_x=1
dilation_y=1
dilation_z=1
group=1
kernel_x=7
kernel_y=8
kernel_z=9
num_output=4
pad_x_begin=3
pad_x_end=3
```

```
pad_y_begin=3
pad_y_end=4
pad_z_begin=4
pad_z_end=4
scale_type=uint8
scale_value=234
shift_type=int8
shift_value=19
stride_x=1
stride_y=1
stride_z=1
with_activation=NONE
```

Convolution2D

```
layer_id=1
layer_name=Conv2D
layer_type=Convolution
layer_bottom=[Placeholder_0]
layer_bottom_shape=[[6,48,52,2]]
layer_bottom_type=[int8]
layer_top=[Conv2D_0]
layer_top_shape=[[6,12,26,1]]
layer_top_type=[int8]
layer_top_datalayout=[NHWC]
layer_top_scale=[31.019745]
layer_top_zp=[0]
biases_type=int32
biases_offset=0
biases_size=4
biases_shape=[1]
weights_type=int8
weights_offset=4
weights_size=372
weights_shape=[1,31,6,2]
dilation_x=1
dilation_y=1
group=1
```

```
kernel_x=6
kernel_y=31
num_output=1
pad_bottom=14
pad_left=2
pad_right=2
pad_top=13
scale_type=uint8
scale_value=152
shift_type=int8
shift_value=17
stride_x=2
stride_y=4
with_activation=NONE
```

Convolution3D

```
layer_id=1
layer_name=Conv3D
layer_type=Convolution3D
layer_bottom=[Placeholder_0]
layer_bottom_shape=[[2,50,53,64,3]]
layer_bottom_type=[int8]
layer_top=[Conv3D_0]
layer_top_shape=[[2,17,14,13,4]]
layer_top_type=[int8]
layer_top_scale=[31.548759]
layer_top_zp=[0]
biases_type=int32
biases_offset=0
biases_size=16
biases_shape=[4]
weights_type=int8
weights_offset=16
weights_size=6048
weights_shape=[4,8,7,9,3]
dilation_x=1
dilation_y=1
```

```
dilation_z=1
group=1
kernel_x=7
kernel_y=8
kernel_z=9
num_output=4
pad_x_begin=1
pad_x_end=2
pad_y_begin=3
pad_y_end=4
pad_z_begin=3
pad_z_end=4
scale_type=uint8
scale_value=136
shift_type=int8
shift_value=18
stride_x=5
stride_y=4
stride_z=3
with_activation=NONE
```

Cosh

```
layer_id=1
layer_name=Cosh_0
layer_type=Cosh
layer_bottom=[Placeholder_0]
layer_bottom_shape=[[2,3,4,5]]
layer_bottom_type=[int8]
layer_top=[Cosh_0]
layer_top_shape=[[2,3,4,5]]
layer_top_type=[uint8]
layer_top_scale=[38.461384]
layer_top_zp=[0]
lut_type=uint8
lut_offset=0
lut_size=256
lut_shape=[256]
```

Cosine

```
layer_id=1
layer_name=Cos
layer_type=Cosine
layer_bottom=[Placeholder_0]
layer_bottom_shape=[[9,70,90,6]]
layer_bottom_type=[int8]
layer_top=[Cos_0]
layer_top_shape=[[9,70,90,6]]
layer_top_type=[int8]
layer_top_datalayout=[NHWC]
layer_top_scale=[127.500000]
layer_top_zp=[0]
lut_type=int8
lut_offset=0
lut_size=256
lut_shape=[256]
```

Count

```
layer_id=1
layer_name=Count
layer_type=Count
layer_bottom=[Placeholder]
layer_bottom_shape=[[5,1000]]
layer_bottom_type=[int8]
layer_top=[Histogram]
layer_top_shape=[[5,50]]
layer_top_type=[uint16]
layer_top_datalayout=[NHWC]
layer_top_scale=[34.764908]
layer_top_zp=[0]
discrete=false
max=49
min=0
nbins=50
```

Crop

```
layer_id=1
layer_name=crop
layer_type=Crop
layer_bottom=[Placeholder]
layer_bottom_shape=[[10,20,30,40,50]]
layer_bottom_type=[int8]
layer_top=[crop]
layer_top_shape=[[10,20,30,37,6]]
layer_top_type=[int8]
layer_top_scale=[23.67497]
layer_top_zp=[0]
crops=[[0,10],[0,20],[0,30],[2,39],[2,8]]
```

CropAndResize

```
layer_id=4
layer_name=CropAndResize
layer_type=CropAndResize
layer_bottom=[Placeholder_0,Placeholder_1_0,Placeholder_2_0]
layer_bottom_shape=[[10,224,224,3],[10,4],[10]]
layer_bottom_type=[uint8,uint16,uint8]
layer_top=[CropAndResize_0]
layer_top_shape=[[10,200,100,3]]
layer_top_type=[uint8]
layer_top_scale=[51.000000]
layer_top_zp=[0]
crop_size=[200,100]
extrapolation_value=38
method=BILINEAR
scale_type=uint8
scale_value=160
shift_type=int8
shift_value=12
```

CumProd

```
layer_id=1
layer_name=Cumulative
```

```
layer_type=Cumulate
layer_bottom=[feature_In]
layer_bottom_shape=[[1,7,7,256]]
layer_bottom_type=[uint8]
layer_top=[cum_0]
layer_top_shape=[[1,7,7,256]]
layer_top_type=[uint8]
method=PROD
axis=3
exclusive=1
reverse=1
scale_type=uint16
scale_offset=0
scale_size=512
scale_shape=[256]
shift_type=int32
shift_offset=512
shift_size=1024
shift_shape=[256]
layer_top_zp=[0]
layer_top_scale=[1.0]
```

CumSum

```
layer_id=1
layer_name=Cumsum
layer_type=Cumulate
layer_bottom=[Placeholder]
layer_bottom_shape=[[2,3,4,5,6]]
layer_bottom_type=[int8]
layer_top=[Cumsum]
layer_top_shape=[[2,3,4,5,6]]
layer_top_type=[int8]
layer_top_scale=[43.060646057128906]
layer_top_zp=[0]
axis=0
exclusive=true
method=SUM
```

```
reverse=true
unquantifiable=false
shift_value=14
shift_type=int8
scale_value=17617
scale_type=uint16
```

DataStride

```
layer_id=1
layer_name=Datastride
layer_type=DataStride
layer_bottom=[Placeholder]
layer_bottom_shape=[[2,9,9,1]]
layer_bottom_type=[int8]
layer_top=[Datastride]
layer_top_shape=[[2,6,1,6]]
layer_top_type=[int8]
layer_top_datalayout=[NHWC]
layer_top_scale=[41.023846]
layer_top_zp=[0]
kernel_x=2
kernel_y=2
stride_x=3
stride_y=3
```

DecodeBox

```
layer_id=2
layer_name=SSD_DecodeBox
layer_type=DecodeBox
layer_bottom=[box_score,box_encoding]
layer_bottom_shape=[[1,1917,91],[1,1917,4]]
layer_bottom_type=[uint8,int8]
layer_top=[SSD_DecodeBox_box,box_num_pre_class,total_class_num,SSD_DecodeBox_out_
score,class_label]
layer_top_shape=[[1,5000,4],[1,5000],[1,1],[1,5000],[1,5000]]
layer_top_type=[int16,uint16,uint16,uint8,uint16]
weights_type=int16
weights_offset=0
```

```
weights_size=17384
weights_shape=[8692]
width=16384
height=16384
score_threshold_uint8=127
box_shift=13
class_num=90
```

DepthToSpace

```
layer_id=1
layer_name=DepthToSpace
layer_type=DepthToSpace
layer_bottom=[Placeholder_0]
layer_bottom_shape=[[1,1,1,4]]
layer_bottom_type=[int8]
layer_top=[DepthToSpace_0]
layer_top_shape=[[1,2,2,1]]
layer_top_type=[int8]
layer_top_datalayout=[NHWC]
layer_top_scale=[240.272781]
layer_top_zp=[0]
block_size_x=2
block_size_y=2
mode=DCR
```

DepthwiseConvolution

```
layer_id=1
layer_name=depthwise
layer_type=DepthwiseConv
layer_bottom=[Placeholder_0]
layer_bottom_shape=[[1,200,200,3]]
layer_bottom_type=[int8]
layer_top=[depthwise_0]
layer_top_shape=[[1,196,196,6]]
layer_top_type=[int8]
layer_top_datalayout=[NHWC]
layer_top_scale=[59.080753]
```

```
layer_top_zp=[0]
biases_type=int32
biases_offset=0
biases_size=24
biases_shape=[6]
scale_type=uint8
scale_offset=24
scale_size=6
scale_shape=[6]
shift_type=int8
shift_offset=30
shift_size=6
shift_shape=[6]
weights_type=int8
weights_offset=36
weights_size=150
weights_shape=[6,5,5,1]
dilation_x=1
dilation_y=1
group=3
kernel_x=5
kernel_y=5
multiplier=2
num_output=6
pad_bottom=0
pad_left=0
pad_right=0
pad_top=0
stride_x=1
stride_y=1
with_activation=NONE
```

Dilation2D

```
layer_id=1
layer_name=Dilation2D
layer_type=Dilation
layer_bottom=[Placeholder]
```

```
layer_bottom_shape=[[2,100,97,5]]
layer_bottom_type=[int8]
layer_top=[Dilation2D]
layer_top_shape=[[2,82,90,5]]
layer_top_type=[int8]
layer_top_scale=[28.538911819458008]
layer_top_zp=[0]
weights_type=int8
weights_offset=0
weights_size=280
weights_shape=[5,7,8,1]
dilation_x=1
dilation_y=3
kernel_x=8
kernel_y=7
pad_bottom=0
pad_left=0
pad_right=0
pad_top=0
stride_x=1
stride_y=1
unquantifiable=false
shift_value=19
shift_type=int8
scale_value=[150,3470,256]
scale_type=[uint8,uint16,uint16]
```

Div

```
layer_id=2
layer_name=div
layer_type=Div
layer_bottom=[Placeholder_0,Placeholder_1_0]
layer_bottom_shape=[[4,44,30,2],[4,44,30,2]]
layer_bottom_type=[uint8,uint8]
layer_top=[div_0]
layer_top_shape=[[4,44,30,2]]
layer_top_type=[uint8]
```

```
layer_top_datalayout=[NHWC]
layer_top_scale=[52.089363]
layer_top_zp=[0]
lut_type=uint16
lut_offset=0
lut_size=512
lut_shape=[256]
scale_type=uint8
scale_value=208
shift_type=int8
shift_value=18
```

ElementwiseAdd

```
layer_id=2
layer_name=Add
layer_type=Eltwise
layer_bottom=[Placeholder_0,Placeholder_1_0]
layer_bottom_shape=[[10,34,30,10],[10,34,30,10]]
layer_bottom_type=[int8,int8]
layer_top=[Add_0]
layer_top_shape=[[10,34,30,10]]
layer_top_type=[int8]
layer_top_datalayout=[NHWC]
layer_top_scale=[18.939608]
layer_top_zp=[0]
method=ADD
scale_type=[uint8,uint16,uint16]
scale_value=[172,281,256]
shift_type=int8
shift_value=16
with_activation=NONE
```

ElementwiseMax

```
layer_id=2
layer_name=Maximum
layer_type=Eltwise
layer_bottom=[Placeholder_0,Placeholder_1_0]
```

```
layer_bottom_shape=[[2,18,2,8],[2,18,2,8]]
layer_bottom_type=[int8,int8]
layer_top=[Maximum_0]
layer_top_shape=[[2,18,2,8]]
layer_top_type=[int8]
layer_top_datalayout=[NHWC]
layer_top_scale=[33.423695]
layer_top_zp=[0]
method=MAX
scale_type=[uint8,uint16,uint16]
scale_value=[232,256,282]
shift_type=int8
shift_value=16
with_activation=NONE
```

ElementwiseMin

```
layer_id=2
layer_name=Minimum
layer_type=Eltwise
layer_bottom=[Placeholder_0,Placeholder_1_0]
layer_bottom_shape=[[7,18,92,2],[7,18,92,2]]
layer_bottom_type=[int8,int8]
layer_top=[Minimum_0]
layer_top_shape=[[7,18,92,2]]
layer_top_type=[int8]
layer_top_datalayout=[NHWC]
layer_top_scale=[29.464540]
layer_top_zp=[0]
method=MIN
scale_type=[uint8,uint16,uint16]
scale_value=[128,266,256]
shift_type=int8
shift_value=15
with_activation=NONE
```

ElementwiseMul

```
layer_id=2
layer_name=Mul
layer_type=Eltwise
layer_bottom=[Placeholder_0,Placeholder_1_0]
layer_bottom_shape=[[6,5,55,9],[6,5,55,9]]
layer_bottom_type=[int8,int8]
layer_top=[Mul_0]
layer_top_shape=[[6,5,55,9]]
layer_top_type=[int8]
layer_top_datalayout=[NHWC]
layer_top_scale=[14.898222]
layer_top_zp=[0]
method=MUL
scale_type=uint8
scale_value=235
shift_type=int8
shift_value=14
with_activation=NONE
```

ElementwiseSub

```
layer_id=2
layer_name=Sub
layer_type=Eltwise
layer_bottom=[Placeholder_0,Placeholder_1_0]
layer_bottom_shape=[[1,28,47,3],[1,28,47,3]]
layer_bottom_type=[int8,int8]
layer_top=[Sub_0]
layer_top_shape=[[1,28,47,3]]
layer_top_type=[int8]
layer_top_datalayout=[NHWC]
layer_top_scale=[22.745680]
layer_top_zp=[0]
method=SUB
scale_type=[uint8,uint16,uint16]
scale_value=[166,256,264]
shift_type=int8
```

```
shift_value=16
with_activation=NONE
```

Elu

```
layer_id=1
layer_name=Elu
layer_type=Activation
layer_bottom=[Placeholder_0]
layer_bottom_shape=[[4,93,4,3]]
layer_bottom_type=[int8]
layer_top=[Elu_0]
layer_top_shape=[[4,93,4,3]]
layer_top_type=[int8]
layer_top_datalayout=[NHWC]
layer_top_scale=[35.270294]
layer_top_zp=[0]
lut_type=int8
lut_offset=0
lut_size=256
lut_shape=[256]
method=ELU
```

EmbeddingLookupSparse

```
layer_id=8
layer_name=embedding_lookup_sparse
layer_type=EmbeddingLookupSparse
layer_bottom=[Placeholder_0,sparse,value,weights]
layer_bottom_shape=[[41,53,69],[53],[53],[53]]
layer_bottom_type=[uint8,int16,int8,int8]
layer_top=[embedding_lookup_sparse]
layer_top_shape=[[50,53,69]]
layer_top_type=[int8]
layer_top_scale=[1.0]
layer_top_zp=[0]
combiner=SUM
max_norm=NONE
unquantifiable=false
```

```
shift_value=0
shift_type=int8
scale_value=1
scale_type=uint8
```

Erf

```
layer_id=1
layer_name=Erf_0
layer_type=Erf
layer_bottom=[Placeholder_0]
layer_bottom_shape=[[10,20,30,40]]
layer_bottom_type=[int8]
layer_top=[Erf_0]
layer_top_shape=[[10,20,30,40]]
layer_top_type=[int8]
layer_top_scale=[255.000000]
layer_top_zp=[0]
lut_type=uint8
lut_offset=0
lut_size=256
lut_shape=[256]
```

Erosion2D

```
layer_id=1
layer_name=erosion2d
layer_type=Erosion
layer_bottom=[Placeholder]
layer_bottom_shape=[[2,100,97,5]]
layer_bottom_type=[int8]
layer_top=[erosion2d]
layer_top_shape=[[2,88,69,5]]
layer_top_type=[int8]
layer_top_scale=[28.381811141967773]
layer_top_zp=[0]
weights_type=int8
weights_offset=0
weights_size=280
```

```
weights_shape=[5,7,8,1]
dilation_x=4
dilation_y=2
kernel_x=8
kernel_y=7
pad_bottom=0
pad_left=0
pad_right=0
pad_top=0
stride_x=1
stride_y=1
unquantifiable=false
shift_value=19
shift_type=int8
scale_value=[154,3862,256]
scale_type=[uint8,uint16,uint16]
```

Exp

```
layer_id=1
layer_name=Exp
layer_type=Exp
layer_bottom=[Placeholder_0]
layer_bottom_shape=[[1,51,3,7]]
layer_bottom_type=[int8]
layer_top=[Exp_0]
layer_top_shape=[[1,51,3,7]]
layer_top_type=[uint8]
layer_top_datalayout=[NHWC]
layer_top_scale=[10.650517]
layer_top_zp=[0]
lut_type=uint8
lut_offset=0
lut_size=256
lut_shape=[256]
```

Filter

```
layer_id=3
layer_name=filter_1
layer_type=Filter
layer_bottom=[Placeholder,Placeholder_1,selector]
layer_bottom_shape=[[50,6,6,6],[50,6,6,6],[50]]
layer_bottom_type=[int8,int8,int8]
layer_top=[filter_1_0,filter_1_1,effective_len]
layer_top_shape=[[50,6,6,6],[50,6,6,6],[1]]
layer_top_type=[int8,int8,uint16]
layer_top_datalayout=[NHWC,NHWC,NHWC]
layer_top_scale=[35.630684,32.889538,1.000000]
layer_top_zp=[0,0,0]
axis=0
num=2
```

Floor

```
layer_id=1
layer_name=Floor
layer_type=Floor
layer_bottom=[Placeholder_0]
layer_bottom_shape=[[3,38,43,8]]
layer_bottom_type=[int8]
layer_top=[Floor_0]
layer_top_shape=[[3,38,43,8]]
layer_top_type=[int8]
layer_top_scale=[31.875000]
layer_top_zp=[0]
lut_type=int8
lut_offset=0
lut_size=256
lut_shape=[256]
```

FractionalPool

```
layer_id=1
layer_name=FractionalAvgPool
layer_type=FractionalPool
```

```
layer_bottom=[Placeholder]
layer_bottom_shape=[[3,46,32,5]]
layer_bottom_type=[int8]
layer_top=[FractionalAvgPool_0,FractionalAvgPool_1,FractionalAvgPool_2]
layer_top_shape=[[3,20,10,5],[21],[11]]
layer_top_type=[int8,int16,int16]
layer_top_scale=[31.34306,1.0,1.0]
layer_top_zp=[0,0,0]
method=AVG
overlap=false
pseudo=false
seed=87654321
unquantifiable=false
```

FullyConnected

```
layer_id=1
layer_name=MatMul
layer_type=FullyConnected
layer_bottom=[Placeholder_0]
layer_bottom_shape=[[81,26]]
layer_bottom_type=[int8]
layer_top=[MatMul_0]
layer_top_shape=[[81,10]]
layer_top_type=[int8]
layer_top_datalayout=[NHWC]
layer_top_scale=[81.161186]
layer_top_zp=[0]
biases_type=int32
biases_offset=0
biases_size=40
biases_shape=[10]
weights_type=int8
weights_offset=40
weights_size=260
weights_shape=[10,26]
num_output=10
scale_type=uint8
```

```
scale_value=226
shift_type=int8
shift_value=16
with_activation=NONE
```

GRUv1

```
layer_id=4
layer_name=gru/while/add_3
layer_type=GRUv1
layer_bottom=[Placeholder_0_cast_tensor_0,gru/zeros_0_cast_tensor_1]
layer_bottom_shape=[[1,28,10],[1,10]]
layer_bottom_type=[int8,int8]
layer_top=[gru/strided_slice_15_0]
layer_top_shape=[[1,10]]
layer_top_type=[int8]
layer_top_datalayout=[NHWC]
layer_top_scale=[189.222443]
layer_top_zp=[0]
biases_type=int32
biases_offset=10
biases_size=160
biases_shape=[40]
lut_ht_type=int8
lut_ht_offset=170
lut_ht_size=256
lut_ht_shape=[256]
lut_rt_type=int8
lut_rt_offset=426
lut_rt_size=256
lut_rt_shape=[256]
lut_zt_type=int8
lut_zt_offset=682
lut_zt_size=256
lut_zt_shape=[256]
weights_type=int8
weights_offset=938
weights_size=600
```

```
weights_shape=[30,20]
activations=[SIGMOID,TANH]
cell_size=10
direction=forward
input_size=10
out_sequence=[Hn]
scale_type=[uint8,uint8,uint8,uint8,uint8,uint8]
scale_value=[234,186,250,187,128,129]
shift_type=[int8,int8,int8,int8,int8,int8]
shift_value=[9,17,16,16,7,14]
time_steps=28
```

GRUv3

```
layer_id=4
layer_name=gru/while/add_3
layer_type=GRUv3
layer_bottom=[Placeholder_0_cast_tensor_0,gru/zeros_0_cast_tensor_1]
layer_bottom_shape=[[1,28,10],[1,10]]
layer_bottom_type=[int8,int8]
layer_top=[gru/transpose_1_0]
layer_top_shape=[[1,28,10]]
layer_top_type=[int8]
layer_top_datalayout=[NHWC]
layer_top_scale=[169.931442]
layer_top_zp=[0]
biases_type=int32
biases_offset=10
biases_size=120
biases_shape=[30]
lut_ht_type=int8
lut_ht_offset=130
lut_ht_size=256
lut_ht_shape=[256]
lut_rt_type=int8
lut_rt_offset=386
lut_rt_size=256
lut_rt_shape=[256]
```

```
lut_zt_type=int8
lut_zt_offset=642
lut_zt_size=256
lut_zt_shape=[256]
weights_type=int8
weights_offset=898
weights_size=600
weights_shape=[30,20]
activations=[SIGMOID,TANH]
cell_size=10
direction=forward
input_size=10
out_sequence=[H]
scale_type=[uint8,uint8,uint8,uint8,uint8,uint8]
scale_value=[232,249,199,211,128,129]
shift_type=[int8,int8,int8,int8,int8,int8]
shift_value=[9,17,8,16,7,14]
time_steps=28
```

Gather

```
layer_id=2
layer_name=GatherV2
layer_type=Gather
layer_bottom=[Placeholder_0,GatherV2/indices_0]
layer_bottom_shape=[[10,100,100,5],[1]]
layer_bottom_type=[int8,uint8]
layer_top=[GatherV2_0]
layer_top_shape=[[10,100,100,1]]
layer_top_type=[int8]
layer_top_datalayout=[NHWC]
layer_top_scale=[26.546780]
layer_top_zp=[0]
axis=3
batch_dims=0
```

GatherElements

```
layer_id=2
layer_name=GatherElements_0
layer_type=GatherElements
layer_bottom=[Placeholder_0,Placeholder_1_Initializer]
layer_bottom_shape=[[3,4,5,6,7],[2,3,3,5,4]]
layer_bottom_type=[int8,int8]
layer_top=[GatherElements_0]
layer_top_shape=[[2,3,3,5,4]]
layer_top_type=[int8]
layer_top_scale=[32.680984]
layer_top_zp=[0]
axis=4
```

GatherND

```
layer_id=4
layer_name=gather
layer_type=GatherND
layer_bottom=[Placeholder1_cast_tensor_0,Placeholder2_cast_tensor_1]
layer_bottom_shape=[[60,60,60],[100,2]]
layer_bottom_type=[int8,uint16]
layer_top=[gather]
layer_top_shape=[[100,60]]
layer_top_type=[int8]
layer_top_datalayout=[NHWC]
layer_top_scale=[29.030643]
layer_top_zp=[0]
batch_dims=0
```

Gelu

```
layer_id=1
layer_name=Gelu_0
layer_type=Activation
layer_bottom=[Placeholder_0]
layer_bottom_shape=[[10,20,30,40]]
layer_bottom_type=[int8]
layer_top=[Gelu_0]
```

```
layer_top_shape=[[10,20,30,40]]
layer_top_type=[uint8]
layer_top_scale=[255.000000]
layer_top_zp=[0]
lut_type=uint8
lut_offset=0
lut_size=256
lut_shape=[256]
method=GELU
```

Gemm

```
layer_id=6
layer_name=Gemm_0
layer_type=Gemm
layer_bottom=[reshape0,reshape1,reshape2]
layer_bottom_shape=[[3,5],[8,5],[3,8]]
layer_bottom_type=[int8,int8,int8]
layer_top=[Gemm_0]
layer_top_shape=[[3,8]]
layer_top_type=[int8]
layer_top_scale=[12.476920]
layer_top_zp=[0]
alpha=1.400000
beta=3.300000
scale_type=[int8,int16,int16]
scale_value=[177,31477,256]
shift_type=int8
shift_value=8
trans_a=false
trans_b=true
```

GetValidCounts

```
layer_id=3
layer_name=get_valid_count_0
layer_type=GetValidCount
layer_bottom=[cls_prod]
layer_bottom_shape=[[2,81,6]]
```

```
layer_bottom_type=[int16]
layer_top=[valid_count,out_tensor,out_indices]
layer_top_shape=[[2],[2,81,6],[2,81]]
layer_top_type=[int16,int16,int16]
score_threshold_type=uint8
score_threshold=128
score_index=0
id_index=2
```

GridSample

```
layer_id=5
layer_name=GridSample_0
layer_type=GridSample
layer_bottom=[Placeholder_0_post_transpose_Cast_tensor_0_1654759566_749878_0_5813876958
407678_,Placeholder_1_Cast_tensor_1_1654759566_758774_0_4267371047474504_]
layer_bottom_shape=[[2,20,30,3],[2,68,72,2]]
layer_bottom_type=[int8,int16]
layer_top=[GridSample_0]
layer_top_shape=[[2,68,72,3]]
layer_top_type=[int8]
layer_top_scale=[36.659504]
layer_top_zp=[0]
align_corners=false
method=NEAREST
padding_mode=BORDER
scale_type=uint8
scale_value=32
shift_type=[int8,int8]
shift_value=[0,18]
```

GroupConvolution

```
layer_id=1
layer_name=Conv2D_group
layer_type=Convolution
layer_bottom=[Placeholder]
layer_bottom_shape=[[1,500,224,6]]
layer_bottom_type=[int8]
layer_top=[Conv2D_group]
```

```
layer_top_shape=[[1,500,224,10]]
layer_top_type=[int8]
layer_top_datalayout=[NHWC]
layer_top_scale=[39.33329]
layer_top_zp=[0]
biases_type=int32
biases_offset=0
biases_size=40
biases_shape=[10]
scale_type=uint8
scale_offset=40
scale_size=10
scale_shape=[10]
shift_type=int8
shift_offset=50
shift_size=10
shift_shape=[10]
weights_type=int8
weights_offset=60
weights_size=3960
weights_shape=[10,11,12,3]
dilation_x=1
dilation_y=1
group=2
kernel_x=12
kernel_y=11
num_output=10
pad_bottom=5
pad_left=5
pad_right=6
pad_top=5
stride_x=1
stride_y=1
with_activation=NONE
```

GroupNormalization

```
layer_id=1
layer_name=group_normalization/batchnorm/add_1
layer_type=GroupNorm
layer_bottom=[Placeholder]
layer_bottom_shape=[[1,70,41,48]]
layer_bottom_type=[uint8]
layer_top=[group_normalization/batchnorm/add_1]
layer_top_shape=[[1,70,41,48]]
layer_top_type=[int8]
layer_top_scale=[77.550247]
layer_top_zp=[0]
biases_type=int32
biases_offset=0
biases_size=192
biases_shape=48
lut_type=int16
lut_offset=192
lut_size=386
lut_shape=[193]
scale_type=uint8
scale_offset=290
scale_size=48
scale_shape=48
shift_type=int8
shift_offset=338
shift_size=48
shift_shape=48
weights_type=int8
weights_offset=386
weights_size=48
weights_shape=48
axis=3
epsilon=0.300000011921
group=24
var_shift_type=uint8
```

```
var_shift_value=18
norm_shift_type=int8
norm_shift_value=13
```

HardSigmoid

```
layer_id=1
layer_name=HardSigmoid_0
layer_type=Activation
layer_bottom=[Placeholder_0]
layer_bottom_shape=[[10,20,30,40]]
layer_bottom_type=[int8]
layer_top=[HardSigmoid_0]
layer_top_shape=[[10,20,30,40]]
layer_top_type=[uint8]
layer_top_scale=[255.000000]
layer_top_zp=[0]
lut_type=uint8
lut_offset=0
lut_size=256
lut_shape=[256]
alpha=-1.400000
beta=-3.300000
method=HARDSIGMOID
```

HardSwish

```
layer_id=1
layer_name=truediv
layer_type=Activation
layer_bottom=[Placeholder_0]
layer_bottom_shape=[[9,20,29,10]]
layer_bottom_type=[int8]
layer_top=[truediv_0]
layer_top_shape=[[9,20,29,10]]
layer_top_type=[int8]
layer_top_datalayout=[NHWC]
layer_top_scale=[27.288155]
layer_top_zp=[0]
```

```
lut_type=int8
lut_offset=0
lut_size=256
lut_shape=[256]
method=HARDSWISH
```

InTopK

```
layer_id=2
layer_name=in_top_k/InTopKV2
layer_type=InTopK
layer_bottom=[Placeholder_0,Placeholder_1_0]
layer_bottom_shape=[[2,10],[2]]
layer_bottom_type=[int8,uint8]
layer_top=[in_top_k/InTopKV2_0]
layer_top_shape=[[2]]
layer_top_type=[uint16]
layer_top_datalayout=[NHWC]
layer_top_scale=[1.000000]
layer_top_zp=[0]
k=3
```

InstanceNormalization

```
layer_id=1
layer_name=InstanceNorm/instancenorm/add_1
layer_type=InstanceNorm
layer_bottom=[Placeholder_0]
layer_bottom_shape=[[9,66,86,10]]
layer_bottom_type=[int8]
layer_top=[InstanceNorm/instancenorm/add_1_0]
layer_top_shape=[[9,66,86,10]]
layer_top_type=[int8]
layer_top_datalayout=[NHWC]
layer_top_scale=[28.117313]
layer_top_zp=[0]
biases_type=int32
biases_offset=0
biases_size=40
```

```
biases_shape=[10]
lut_type=int16
lut_offset=40
lut_size=386
lut_shape=[193]
weights_type=int8
weights_offset=426
weights_size=10
weights_shape=[10]
epsilon=0.000001
var_shift_type=int8
var_shift_value=132
norm_shift_type=int8
norm_shift_value=15
scale_type=uint8
scale_value=129
shift_type=int8
shift_value=14
```

L1Normalization

```
layer_id=1
layer_name=LpNormalization_0
layer_type=Normalization
layer_bottom=[Placeholder_0]
layer_bottom_shape=[[2,3,4]]
layer_bottom_type=[int8]
layer_top=[LpNormalization_0]
layer_top_shape=[[2,3,4]]
layer_top_type=[int8]
layer_top_scale=[127.9888]
layer_top_zp=[0]
axis=[0]
epsilon=0.0
method=L1
scale_type=uint8
scale_value=128
shift_type=int8
```

```
shift_value=0
unquantifiable=false
```

L1Pooling2D

```
layer_id=2
layer_name=LpPool_0
layer_type=Pooling
layer_bottom=[Placeholder_0_post_transpose]
layer_bottom_shape=[[2,123,224,5]]
layer_bottom_type=[int8]
layer_top=[LpPool_0]
layer_top_shape=[[2,123,224,5]]
layer_top_type=[uint8]
layer_top_scale=[3.815453]
layer_top_zp=[0]
ceil_mode=false
dilation_x=1
dilation_y=1
kernel_x=8
kernel_y=7
method=L1
pad_bottom=3
pad_left=4
pad_right=3
pad_top=3
scale_type=uint16
scale_value=18728
shift_type=int8
shift_value=17
stride_x=1
stride_y=1
```

L2Normalization

```
layer_id=1
layer_name=LpNormalization_0
layer_type=Normalization
layer_bottom=[Placeholder_0]
```

```
layer_bottom_shape=[[2,3,4]]
layer_bottom_type=[int8]
layer_top=[LpNormalization_0]
layer_top_shape=[[2,3,4]]
layer_top_type=[int8]
layer_top_scale=[127.9888]
layer_top_zp=[0]
lut_type=int16
lut_offset=0
lut_size=386
lut_shape=[193]
axis=[0]
epsilon=0.0
method=L2
reciprocal_shift_type=int8
reciprocal_shift_value=0
scale_type=uint8
scale_value=128
shift_type=int8
shift_value=0
unquantifiable=false
```

L2Pooling2D

```
layer_id=2
layer_name=LpPool_0
layer_type=Pooling
layer_bottom=[Placeholder_0_post_transpose]
layer_bottom_shape=[[2,123,224,5]]
layer_bottom_type=[int8]
layer_top=[LpPool_0]
layer_top_shape=[[2,62,74,5]]
layer_top_type=[uint8]
layer_top_scale=[24.206547]
layer_top_zp=[0]
sqrt_lut_type=uint8
sqrt_lut_offset=0
sqrt_lut_size=256
```

```
sqrt_lut_shape=[256]
ceil_mode=false
dilation_x=1
dilation_y=1
kernel_x=8
kernel_y=7
method=L2
pad_bottom=2
pad_left=3
pad_right=1
pad_top=4
scale_type=uint16
scale_value=27306
shift_type=int8
shift_value=15
stride_x=3
stride_y=2
```

LRN

```
layer_id=1
layer_name=lrn
layer_type=LRN
layer_bottom=[Placeholder]
layer_bottom_shape=[[2,2,28,10]]
layer_bottom_type=[int8]
layer_top=[lrn]
layer_top_shape=[[2,2,28,10]]
layer_top_type=[int8]
layer_top_datalayout=[NHWC]
layer_top_scale=[118.701920]
layer_top_zp=[0]
lut_type=uint16
lut_offset=0
lut_size=512
lut_shape=[256]
alpha=0.174000
beta=1.169000
```

```
bias=2.000000
method=ACROSS_CHANNELS
scale_sum_type=uint8
scale_sum_value=170
scale_type=uint8
scale_value=182
shift_sum_type=int8
shift_sum_value=9
shift_type=int8
shift_value=23
size=3
```

LayerNormalization

```
layer_id=1
layer_name=LayerNorm/batchnorm/add_1
layer_type=LayerNorm
layer_bottom=[Placeholder_0]
layer_bottom_shape=[[7,69,37,6]]
layer_bottom_type=[int8]
layer_top=[LayerNorm/batchnorm/add_1_0]
layer_top_shape=[[7,69,37,6]]
layer_top_type=[int8]
layer_top_datalayout=[NHWC]
layer_top_scale=[29.062611]
layer_top_zp=[0]
biases_type=int32
biases_offset=0
biases_size=61272
biases_shape=[69,37,6]
lut_type=int16
lut_offset=61272
lut_size=386
lut_shape=[193]
weights_type=int8
weights_offset=61658
weights_size=15318
weights_shape=[69,37,6]
```

```
axis=[1,2,3]
epsilon=0.000000
var_shift_type=uint8
var_shift_value=130
norm_shift_type=int8
norm_shift_value=15
scale_type=uint8
scale_value=129
shift_type=int8
shift_value=14
```

LeakyRelu

```
layer_id=1
layer_name=LeakyRelu
layer_type=Activation
layer_bottom=[Placeholder_0]
layer_bottom_shape=[[2,85,7,2]]
layer_bottom_type=[int8]
layer_top=[LeakyRelu_0]
layer_top_shape=[[2,85,7,2]]
layer_top_type=[int8]
layer_top_datalayout=[NHWC]
layer_top_scale=[8.902371]
layer_top_zp=[0]
method=LEAKYRELU
negative_slope_shift=8
negative_slope_type=uint8
negative_slope_value=1083
scale_type=uint16
scale_value=133
shift_type=int8
shift_value=9
```

LeftShift

```
layer_id=4
layer_name=BitShift_0
layer_type=BitShift
```

```
layer_bottom=[BitShift_0_pre_tile,BitShift_0_pre_tile_0]
layer_bottom_shape=[[2,3],[2,3]]
layer_bottom_type=[uint8,uint8]
layer_top=[BitShift_0]
layer_top_shape=[[2,3]]
layer_top_type=[uint8]
layer_top_scale=[1.000000]
layer_top_zp=[0]
direction=LEFT
```

Log

```
layer_id=1
layer_name=Log
layer_type=Log
layer_bottom=[Placeholder_0]
layer_bottom_shape=[[4,75,56,2]]
layer_bottom_type=[uint8]
layer_top=[Log_0]
layer_top_shape=[[4,75,56,2]]
layer_top_type=[int8]
layer_top_datalayout=[NHWC]
layer_top_scale=[12.393167]
layer_top_zp=[0]
lut_type=int8
lut_offset=0
lut_size=256
lut_shape=[256]
```

LogSoftmax

```
layer_id=1
layer_name=SoftmaxInt8
layer_type=LogSoftmax
layer_bottom=['score']
layer_bottom_shape=[[2,3,4,5,6]]
layer_bottom_type=['int8']
layer_top=['out_score_ptr']
layer_top_shape=[[2,3,4,5,6]]
```

```
layer_top_type=['int8']
lut_exp_type=uint32
lut_exp_size=1024
lut_exp_offset=0
lut_exp_shape=[256]
lut_log_type=int8
lut_log_size=256
lut_log_offset=1024
lut_log_shape=[256]
axis=-1
scale_type=uint16
scale_value=128
shift_type=int8
shift_value=7
layer_top_zp=[0]
```

Logical

```
layer_id=1
layer_name=logical
layer_type=Logical
layer_bottom=[Placeholder]
layer_bottom_shape=[[1,89,88,8]]
layer_bottom_type=[int8]
layer_top=[logical]
layer_top_shape=[[1,89,88,8]]
layer_top_type=[uint8]
layer_top_datalayout=[NHWC]
layer_top_scale=[1.000000]
layer_top_zp=[0]
method=NOT
```

MatMul

```
layer_id=2
layer_name=MatMul
layer_type=MatMul
layer_bottom=[Placeholder_0,Placeholder_1_0]
layer_bottom_shape=[[1,2,3,4],[1,2,4,3]]
```

```
layer_bottom_type=[int8,int8]
layer_top=[MatMul_0]
layer_top_shape=[[1,2,4,4]]
layer_top_type=[int8]
layer_top_datalayout=[NHWC]
layer_top_scale=[32.207657]
layer_top_zp=[0]
adj_x=true
adj_y=true
scale_type=uint8
scale_value=208
shift_type=int8
shift_value=14
```

MaxPooling2D

```
layer_id=1
layer_name=MaxPool
layer_type=Pooling
layer_bottom=[Placeholder_0]
layer_bottom_shape=[[1,224,224,3]]
layer_bottom_type=[int8]
layer_top=[MaxPool_0]
layer_top_shape=[[1,1,32,3]]
layer_top_type=[int8]
layer_top_datalayout=[NHWC]
layer_top_scale=[28.295528]
layer_top_zp=[0]
dilation_x=1
dilation_y=1
kernel_x=5
kernel_y=3
method=MAX
pad_bottom=0
pad_left=0
pad_right=0
pad_top=0
stride_x=7
```

```
stride_y=300
ceil_mode=false
```

MaxPooling3D

```
layer_id=1
layer_name=MaxPool3D
layer_type=Pooling3D
layer_bottom=[Placeholder_0]
layer_bottom_shape=[[1,100,224,200,3]]
layer_bottom_type=[int8]
layer_top=[MaxPool3D_0]
layer_top_shape=[[1,47,110,99,3]]
layer_top_type=[int8]
layer_top_scale=[22.744478]
layer_top_zp=[0]
ceil_mode=false
dilation_x=1
dilation_y=1
dilation_z=1
kernel_x=3
kernel_y=5
kernel_z=7
method=MAX
pad_x_begin=0
pad_x_end=0
pad_y_begin=0
pad_y_end=0
pad_z_begin=0
pad_z_end=0
stride_x=2
stride_y=2
stride_z=2
ceil_mode=false
```

MaxPoolingWithArgMax

```
layer_id=1
layer_name=pooling
layer_type=MaxPoolingWithArgMax
layer_bottom=[Placeholder]
layer_bottom_shape=[[3,14,15,9]]
layer_bottom_type=[int8]
layer_top=[pooling_0,pooling_1]
layer_top_shape=[[3,10,13,9],[3,10,13,9]]
layer_top_type=[int8,uint32]
layer_top_layout=[NHWC,NHWC]
layer_top_scale=[34.126915,1.000000]
layer_top_zp=[0,0]
dilation_x=1
dilation_y=1
flatten_dim=HW
kernel_x=3
kernel_y=5
pad_bottom=0
pad_left=0
pad_right=0
pad_top=0
stride_x=1
stride_y=1
storage_order=0
ceil_mode=false
```

MaxRoiPool

```
layer_id=4
layer_name=MaxRoiPool_0
layer_type=MaxRoiPool
layer_bottom=[Placeholder_0_cast_tensor_0,Placeholder_1_cast_tensor_1]
layer_bottom_shape=[[1,244,244,3],[10,5]]
layer_bottom_type=[int8,uint16]
layer_top=[MaxRoiPool_0]
layer_top_shape=[[10,100,100,3]]
layer_top_type=[int8]
```

```
layer_top_scale=[28.416882]
layer_top_zp=[0]
pooled_shape=[100,100]
spatial =[153,153]
```

MaxUnpool

```
layer_id=4
layer_name=MaxUnpool_0
layer_type=MaxUnpool
layer_bottom=[Placeholder_0_post_transpose,Placeholder_1_post_transpose]
layer_bottom_shape=[[2,99,196,3],[2,99,196,3]]
layer_bottom_type=[int8,int8]
layer_top=[MaxUnpool_0]
layer_top_shape=[[2,124,223,3]]
layer_top_type=[int8]
layer_top_scale=[29.18309]
layer_top_zp=[0]
flatten_dim=NCHW
output_shape=[2,124,223,3]
storage_order=0
```

MeanVarianceNormalization

```
layer_id=1
layer_name=MVN0
layer_type=MVN
layer_bottom=[Input0]
layer_bottom_shape=[[1,6,16,32]]
layer_bottom_type=[int8]
layer_top=[LayerNorm0]
layer_top_shape=[[1,6,16,32]]
layer_top_type=[int8]
axis=[1,2,3]
lut_type=int16
lut_size=98
lut_offset=0
lut_shape=[49]
scale_type=uint8
```

```
scale_value=192
shift_type=int8
shift_value=15
var_shift_type=uint16
var_shift_value=30720
norm_shift_type=int16
norm_shift_value=2
```

Meshgrid

```
layer_id=2
layer_name=meshgrid/mul
layer_type=Meshgrid
layer_bottom=[Placeholder,Placeholder_1]
layer_bottom_shape=[[3],[5]]
layer_bottom_type=[int8,int8]
layer_top=[meshgrid/mul_0,meshgrid/mul_1]
layer_top_shape=[[5,3],[5,3]]
layer_top_type=[int8,int8]
indexing=xy
sparse=false
copy=true
```

Mish

```
layer_name=mul
layer_type=Activation
layer_bottom=[Placeholder]
layer_bottom_shape=[[2,3]]
layer_bottom_type=[int8]
layer_top=[mul]
layer_top_shape=[[2,3]]
layer_top_type=[int8]
layer_top_scale=[89.506554]
layer_top_zp=[0]
lut_type=int8
lut_offset=0
lut_size=256
lut_shape=[256]
```

```
method=MISH
```

Mod

```
layer_id=4
layer_name=FloorMod
layer_type=Mod
layer_bottom=[Cast_0,Cast_1_0]
layer_bottom_shape=[[2,3,4,5],[2,3,4,5]]
layer_bottom_type=[int8,int8]
layer_top=[FloorMod_0]
layer_top_shape=[[2,3,4,5]]
layer_top_type=[int8]
layer_top_scale=[31.875000]
layer_top_zp=[0]
scale_type=[uint8,uint8]
scale_value=[160,160]
shift_type=[int8,int8]
shift_value=[7,7]
fmod=false
```

Moments

```
layer_id=1
layer_name=Moments_0
layer_type=Moments
layer_bottom=[input1]
layer_bottom_shape=[[2,6,16,32]]
layer_bottom_type=[int8]
layer_top=[mean0,variance0]
layer_top_shape=[[2,1,1,1],[2,1,1,1]]
layer_top_type=[int8,uint8]
axis=[1,2,3]
var_scale_type=uint8
var_scale_value=192
var_shift_type=int8
var_shift_value=15
input_scale_type=uint8
input_scale_value=20
```

input_shift_type=int8

input_shift_value=3

keepdims=True

Mul

layer_id=3

layer_name=Add_

layer_type=Add

layer_bottom=[input_0,input_1]

layer_bottom_shape=[[2,256],[256]]

layer_bottom_type=[int8,int8]

layer_top=[output]

layer_top_shape=[[2,256]]

layer_top_type=[int8]

layer_top_scale=[0.001733]

layer_top_zp=[0]

scale_type=[uint8]

scale_value=[188]

shift_type=int8

shift_value=29

layer_top_scale=[1.000000]

layer_top_zp=[0]

MultiboxTransformLoc

layer_id=3

layer_name=multiboxTransformLoc

layer_type=MultiboxTransformLoc

layer_bottom=[fingerprint_input_0_Cast_tensor_0_1668066133_705893_0_7448216298334006_,fingerprint_input_1_Cast_tensor_1_1668066133_707133_0_5258313655747363_,fingerprint_input_2_Cast_tensor_2_1668066133_707992_0_25741316059881236_]

layer_bottom_shape=[[1,8,1000],[1,4000],[1,1000,4]]

layer_bottom_type=[uint8,int8,int16]

layer_top=[out0,out1]

layer_top_shape=[[1,1000,6],[1]]

layer_top_type=[int16,uint16]

layer_top_scale=[32767,1]

layer_top_zp=[0,0]

th_lut_type=int16

```
th_lut_offset=0
th_lut_size=512
th_lut_shape=[256]
tw_lut_type=int16
tw_lut_offset=512
tw_lut_size=512
tw_lut_shape=[256]
box_scale_value=[20647,20647,30615,30615,16352]
box_scale_type=[uint16,uint16,uint16,uint16,uint16]
box_shift_value=[23,23,30,30,17]
box_shift_type=[int8,int8,int8,int8,int8]
delta_shift=9
score_threshold_value=1
score_threshold_type=uint8
score_scale_value=16448
score_scale_type=uint16
score_shift_value=7
score_shift_type=int8
```

NMS

```
layer_id=10
layer_name=NonMaxSuppression_0
layer_type=NMS
layer_bottom=[NMS_box,NMS_per_class_boxes_num,NMS_total_class,NMS_score]
layer_bottom_shape=[[1,6,4],[1,1],[1,1],[1,6]]
layer_bottom_type=[int16,uint16,uint16,uint8]
layer_top=[box_out,NonMaxSuppression_0_1,NonMaxSuppression_0_2,NonMaxSuppression_0_3]
layer_top_shape=[[1,3,4],[1,1],[1,3],[1,3]]
layer_top_type=[int16,uint16,uint8,uint16]
layer_top_scale=[1.0,1.0,151.4883,1.0]
layer_top_zp=[0,0,0,0]
areas_shift=13
center_point_box=0
image_height=300
image_width=300
iou_thresh_shift=8
iou_threshold=0
```

```
max_output_size=3
method=HARD
scale_type=uint16
scale_value=24550
score_threshold=0
shift_type=int8
shift_value=28
soft_nms_sigma=0.0
unquantifiable=false
```

Negative

```
layer_id=1
layer_name=Neg
layer_type=Negative
layer_bottom=[Placeholder_0]
layer_bottom_shape=[[2,60,43,3]]
layer_bottom_type=[int8]
layer_top=[Neg_0]
layer_top_shape=[[2,60,43,3]]
layer_top_type=[int8]
layer_top_datalayout=[NHWC]
layer_top_scale=[32.390423]
layer_top_zp=[0]
```

NormalizedMoments

```
layer_id=6
layer_name=Relu6
layer_type=NormalizedMoments
layer_bottom=[Placeholder0,Placeholder1,NormalizedMoments_shift]
layer_bottom_shape=[[2,3],[2,3],[2,3]]
layer_bottom_type=[uint8,int8,uint8]
layer_top=[Relu6_0,Relu6_1]
layer_top_shape=[[2,3],[2,3]]
layer_top_type=[uint8,int8]
layer_top_scale=[2188.2236328125,3682.852783203125]
layer_top_zp=[0,0]
lut_type=uint8
```

```
lut_offset=6
lut_size=256
lut_shape=[256]
counts=23.0
unquantifiable=false
mean_shift_value=10
mean_shift_type=int8
mean_scale_value=[128,8,17506]
mean_scale_type=[uint8,uint16,uint16]
var_shift_value=22
var_shift_type=int8
var_scale_value=[149,32767,5510]
var_scale_type=[uint8,uint16,uint16]
```

OneHot

```
layer_id=1
layer_name=one_hot_41
layer_type=OneHot
layer_bottom=[ArgMax_41_squeeze_dims_tensor]
layer_bottom_shape=[[1,8]]
layer_bottom_type=[uint16]
layer_top=[one_hot_41_0]
layer_top_shape=[[1,8,8]]
layer_top_type=[uint16]
axis=1
values=[0,1]
depth=8
```

PRelu

```
layer_id=1
layer_name=p_re_lu/add
layer_type=Activation
layer_bottom=[Placeholder]
layer_bottom_shape=[[5,68,4,5]]
layer_bottom_type=[int8]
layer_top=[p_re_lu/add]
layer_top_shape=[[5,68,4,5]]
```

```
layer_top_type=[int8]
layer_top_datalayout=[NHWC]
layer_top_scale=[10.710900]
layer_top_zp=[0]
negative_slope_type=int16
negative_slope_offset=0
negative_slope_size=2720
negative_slope_shape=[68,4,5]
method=PRELU
negative_slope_shift=12
scale_type=uint16
scale_value=165
shift_type=int8
shift_value=9
```

Pad

```
layer_id=1
layer_name=Pad
layer_type=Pad
layer_bottom=[Placeholder_0]
layer_bottom_shape=[[1,224,224,3]]
layer_bottom_type=[int8]
layer_top=[Pad_0]
layer_top_shape=[[4,231,235,18]]
layer_top_type=[int8]
layer_top_datalayout=[NHWC]
layer_top_scale=[29.434765]
layer_top_zp=[0]
constant_value=0
mode=CONSTANT
pads=[[1,2],[3,4],[5,6],[7,8]]
```

Pow

```
layer_id=2
layer_name=Pow_0
layer_type=Pow
layer_bottom=[Placeholder_0,Placeholder_1]
```

```
layer_bottom_shape=[[2,61,24,5],[2,61,24,5]]
layer_bottom_type=[uint8,int8]
layer_top=[Pow_0]
layer_top_shape=[[2,61,24,5]]
layer_top_type=[uint8]
layer_top_datalayout=[NHWC]
layer_top_scale=[18.227720]
layer_top_zp=[0]
lut_exp_type=uint8
lut_exp_offset=0
lut_exp_size=256
lut_exp_shape=[256]
lut_log_type=int8
lut_log_offset=256
lut_log_size=256
lut_log_shape=[256]
scale_type=uint8
scale_value=177
shift_type=int8
shift_value=8
```

Reciprocal

```
layer_id=1
layer_name=Reciprocal_0
layer_type=Reciprocal
layer_bottom=[Placeholder_0]
layer_bottom_shape=[[2,3,4]]
layer_bottom_type=[int8]
layer_top=[Reciprocal_0]
layer_top_shape=[[2,3,4]]
layer_top_type=[int8]
layer_top_scale=[4.623067]
layer_top_zp=[0]
lut_type=int8
lut_offset=0
lut_size=256
lut_shape=[256]
```

ReduceAll

```
layer_id=3
layer_name=All
layer_type=Reduce
layer_bottom=[All_pre_cast_0]
layer_bottom_shape=[[2,3,4,5]]
layer_bottom_type=[int8]
layer_top=[All]
layer_top_shape=[[2,3,4,1]]
layer_top_type=[uint8]
layer_top_scale=[1.000000]
layer_top_zp=[0]
axis=[3]
method=ALL
```

ReduceAny

```
layer_id=3
layer_name=Any
layer_type=Reduce
layer_bottom=[All_pre_cast_0]
layer_bottom_shape=[[2,3,4,5]]
layer_bottom_type=[int8]
layer_top=[All]
layer_top_shape=[[2,3,4,1]]
layer_top_type=[uint8]
layer_top_scale=[1.000000]
layer_top_zp=[0]
axis=[3]
method=ANY
```

ReduceL1

```
layer_id=1
layer_name=ReduceL1_0
layer_type=Reduce
layer_bottom=[Placeholder_0]
layer_bottom_shape=[[2,3,4,5]]
layer_bottom_type=[int8]
```

```
layer_top=[ReduceL1_0]
layer_top_shape=[[2,3,1,5]]
layer_top_type=[uint8]
layer_top_scale=[50.51838]
layer_top_zp=[0]
axis=[2]
method=L1
scale_type=uint16
scale_value=17401
shift_type=int8
shift_value=14
```

ReduceL2

```
layer_id=1
layer_name=ReduceL2_0
layer_type=Reduce
layer_bottom=[Placeholder_0]
layer_bottom_shape=[[2,3,4,5,6]]
layer_bottom_type=[int8]
layer_top=[ReduceL2_0]
layer_top_shape=[[2,1,4,5,6]]
layer_top_type=[uint8]
layer_top_scale=[68.44276]
layer_top_zp=[0]
sqrt_lut_type=uint8
sqrt_lut_offset=0
sqrt_lut_size=256
sqrt_lut_shape=[256]
axis=[1]
method=L2
scale_type=uint16
scale_value=26562
shift_type=int8
shift_value=13
```

ReduceMax

```
layer_id=1
layer_name=ReduceMax_0
layer_type=Reduce
layer_bottom=[Placeholder_0]
layer_bottom_shape=[[2,3]]
layer_bottom_type=[int8]
layer_top=[ReduceMax_0]
layer_top_shape=[[2,1]]
layer_top_type=[int8]
layer_top_scale=[113.206535]
layer_top_zp=[0]
axis=[1]
method=MAX
```

ReduceMean

```
layer_id=1
layer_name=ReduceMean_0
layer_type=Reduce
layer_bottom=[Placeholder_0]
layer_bottom_shape=[[2,3,4]]
layer_bottom_type=[int8]
layer_top=[ReduceMean_0]
layer_top_shape=[[1,3,4]]
layer_top_type=[int8]
layer_top_scale=[102.835449]
layer_top_zp=[0]
axis=[0]
method=MEAN
scale_type=uint8
scale_value=235
shift_type=int8
shift_value=8
```

ReduceMin

```
layer_id=1
layer_name=ReduceMin_0
layer_type=Reduce
layer_bottom=[Placeholder_0]
layer_bottom_shape=[[2,3,4]]
layer_bottom_type=[int8]
layer_top=[ReduceMin_0]
layer_top_shape=[[1,3,4]]
layer_top_type=[int8]
layer_top_scale=[38.168953]
layer_top_zp=[0]
axis=[0]
method=MIN
```

ReduceProd

```
layer_id=1
layer_name=Prod
layer_type=Reduce
layer_bottom=[Placeholder_0]
layer_bottom_shape=[[2,3,4]]
layer_bottom_type=[int8]
layer_top=[Prod_0]
layer_top_shape=[[1,1,4]]
layer_top_type=[int8]
layer_top_scale=[1615.338135]
layer_top_zp=[0]
axis=[1,0]
method=PROD
scale_type=uint16
scale_value=10196
shift_type=int8
shift_value=36
```

ReduceSum

```
layer_id=1
layer_name=Sum
layer_type=Reduce
layer_bottom=[Placeholder_0]
layer_bottom_shape=[[8,47,17,8]]
layer_bottom_type=[int8]
layer_top=[Sum_0]
layer_top_shape=[[1,1,1,1]]
layer_top_type=[int8]
layer_top_scale=[0.486710]
layer_top_zp=[0]
axis=[0,1,2,3]
method=SUM
scale_type=uint8
scale_value=144
shift_type=int8
shift_value=13
```

ReduceUnbiasedVariance

```
layer_id=1
layer_name=reduce_variance/Mean_1
layer_type=Reduce
layer_bottom=[Placeholder]
layer_bottom_shape=[[2,3,4,5]]
layer_bottom_type=[int8]
layer_top=[reduce_variance/Mean_1]
layer_top_shape=[[2,3,1,5]]
layer_top_type=[uint8]
layer_top_scale=[160.9791]
layer_top_zp=[0]
axis=[2]
method=UNBIASED_VARIANCE
scale_type=uint16
scale_value=27123
shift_type=int8
shift_value=21
```

ReduceVariance

```
layer_id=1
layer_name=reduce_variance/Mean_1
layer_type=Reduce
layer_bottom=[Placeholder]
layer_bottom_shape=[[2,3,4,5]]
layer_bottom_type=[int8]
layer_top=[reduce_variance/Mean_1]
layer_top_shape=[[2,3,1,5]]
layer_top_type=[uint8]
layer_top_scale=[160.9791]
layer_top_zp=[0]
axis=[2]
method=VARIANCE
scale_type=uint16
scale_value=27123
shift_type=int8
shift_value=21
unquantifiable=false
```

Region

```
layer_id=35
layer_name=yolo_v2_416_region
layer_type=Region
layer_bottom=[Cast_tensor_0]
layer_bottom_shape=[[1,13,13,5,25]]
layer_bottom_type=[int8]
layer_top=[region_0,region_1,region_2,region_3,region_4]
layer_top_shape=[[1,5000],[1,5000,4],[1,20],[1,20],[1,1]]
layer_top_type=[uint8,int16,uint16,int16,uint16]
layer_top_datalayout=[Flat,Flat,Flat,Flat,Flat]
layer_top_scale=[255.0,4096.0,1.0,1.0,1.0]
layer_top_zp=[0,0,0,0,0]
conf_sigmoid_lut_type=uint16
conf_sigmoid_lut_offset=50679520
conf_sigmoid_lut_size=512
conf_sigmoid_lut_shape=[256]
```

```
qanchors_lut_type=uint16
qanchors_lut_offset=50680032
qanchors_lut_size=20
qanchors_lut_shape=[10]
score_softmax_lut_type=uint32
score_softmax_lut_offset=50680052
score_softmax_lut_size=1024
score_softmax_lut_shape=[256]
wh_exp_lut_type=uint16
wh_exp_lut_offset=50681076
wh_exp_lut_size=512
wh_exp_lut_shape=[256]
xy_sigmoid_lut_type=int16
xy_sigmoid_lut_offset=50681588
xy_sigmoid_lut_size=512
xy_sigmoid_lut_shape=[256]
anchors_exp_h_shift=12
anchors_exp_w_shift=12
box_per_grid=5
class_num=20
col_shift=12
conf_sigmoid_shift=15
grid_compensate=true
grid_h_scale=2520
grid_h_shift=15
grid_height=13
grid_w_scale=2520
grid_w_shift=15
grid_width=13
max_box_num=5000
obj_thresh=76
row_shift=12
unquantifiable=false
wh_exp_scale=19068
wh_exp_shift=9
```

RegionFuse

```
layer_id=10
layer_name=RegionFuse
layer_type=RegionFuse
layer_bottom=[yolo_region_out_score_ptr1,yolo_region_out_score_ptr2,yolo_region_out_box_ptr1,yolo_region_out_box_ptr2,yolo_region_per_class_box_total1,yolo_region_per_class_box_total2,yolo_region_per_class_label1,yolo_region_per_class_label2,yolo_region_all_class_total1,yolo_region_all_class_total2]
layer_bottom_shape=[[1,100],[1,200],[1,100,4],[1,200,4],[1,90],[1,90],[1,90],[1,90],[1,1],[1,1]]
layer_bottom_type=[uint8,uint8,int16,int16,int16,int16,int16,int16,int16,int16]
layer_top=[out_score_ptr,out_box_ptr,per_class_box_total,per_class_label,all_class_total]
layer_top_shape=[[1,9000],[1,9000,4],[1,90],[1,90],[1,1]]
layer_top_type=[uint8,int16,int16,int16,int16]
box_scale_type=[uint16,uint16]
box_scale_value=[2048,4096]
box_shift_type=[int8,int8]
box_shift_value=[8,8]
score_scale_type=[uint8,uint8]
score_scale_value=[255,255]
score_shift_type=[int8,int8]
score_shift_value=[8,8]
class_num=90
```

Relu

```
layer_id=1
layer_name=Relu
layer_type=Activation
layer_bottom=[Placeholder_0]
layer_bottom_shape=[[2,53,24,2]]
layer_bottom_type=[int8]
layer_top=[Relu_0]
layer_top_shape=[[2,53,24,2]]
layer_top_type=[uint8]
layer_top_scale=[74.883003]
layer_top_zp=[0]
method=RELU
```

```
scale_type=uint8
scale_value=132
shift_type=int8
shift_value=6
```

Relu6

```
layer_id=1
layer_name=Relu6
layer_type=Activation
layer_bottom=[Placeholder_0]
layer_bottom_shape=[[3,68,61,1]]
layer_bottom_type=[int8]
layer_top=[Relu6_0]
layer_top_shape=[[3,68,61,1]]
layer_top_type=[uint8]
layer_top_scale=[69.849144]
layer_top_zp=[0]
method=RELU6
scale_type=uint8
scale_value=128
shift_type=int8
shift_value=6
```

Repeat

```
layer_id=4
layer_name=Repeat
layer_type=Repeat
layer_bottom=[Placeholder0_cast_tensor_0,Placeholder1_cast_tensor_1]
layer_bottom_shape=[[20],[20]]
layer_bottom_type=[int8,uint16]
layer_top=[Repeat]
layer_top_shape=[[1,190]]
layer_top_type=[int8]
layer_top_scale=[1.000000]
layer_top_zp=[0]
```

Reshape

```
layer_id=1
layer_name=Reshape
layer_type=Reshape
layer_bottom=[Placeholder_0]
layer_bottom_shape=[[1,100,3]]
layer_bottom_type=[int8]
layer_top=[Reshape_0]
layer_top_shape=[[1,10,10,3]]
layer_top_type=[int8]
layer_top_scale=[38.524994]
layer_top_zp=[0]
shape=[1,10,10,3]
```

Resize

```
layer_id=1
layer_name=ResizeBilinear
layer_type=Resize
layer_bottom=[Placeholder_0]
layer_bottom_shape=[[5,7,8,1]]
layer_bottom_type=[int8]
layer_top=[ResizeBilinear_0]
layer_top_shape=[[5,21,16,1]]
layer_top_type=[int8]
layer_top_scale=[45.564224]
layer_top_zp=[0]
method=BILINEAR
mode=ALIGN_CORNERS
ratio_x=2.000000
ratio_y=3.000000
```

ReverseSequence

```
layer_id=4
layer_name=ReverseSequence
layer_type=ReverseSequence
layer_bottom=[Placeholder_0_cast_tensor_0,ReverseSequence/seq_lengths_0_cast_tensor_1]
layer_bottom_shape=[[5,6,7],[5]]
```

```
layer_bottom_type=[int8,uint16]
layer_top=[ReverseSequence_0]
layer_top_shape=[[5,6,7]]
layer_top_type=[int8]
layer_top_datalayout=[NHWC]
layer_top_scale=[39.682415]
layer_top_zp=[0]
batch_axis=0
time_axis=1
```

RgbToYuv

```
layer_id=1
layer_name=Placeholder
layer_type=RgbToYuv
layer_bottom=[preprocess_Input]
layer_bottom_shape=[[1,244,244,3]]
layer_bottom_type=[uint8]
layer_top=[Placeholder_0]
layer_top_shape=[[1,89304]]
layer_top_type=[uint8]
layer_top_scale=[1.000000]
layer_top_zp=[0]
bits=8
coefficient=[0,0,0,0,128,128,218,732,74,-117,-395,512,512,-465,-47]
coefficient_dtype=int16
coefficient_shift=10
conversion=BT709
format=I420
shape=[[1,244,244,3]]
```

RightShift

```
layer_id=2
layer_name=RightShift
layer_type=BitShift
layer_bottom=[Input0,Input1]
layer_bottom_shape=[[2,3,4],[2,3,4]]
layer_bottom_type=[int16,uint8]
```

```
layer_top=[RightShift_0]
layer_top_shape=[[2,3,4]]
layer_top_type=[int16]
layer_top_scale=[1.000000]
layer_top_zp=[0]
direction=RIGHT
```

RoiAlign

```
layer_id=2
layer_name=RoiAlign0
layer_type=RoiAlign
layer_top_shape=[[0, 2, 2, 0]]
layer_top_type=[int8]
layer_top=[output]
layer_bottom=[input_i8_0,rois]
layer_bottom_shape=[[3, 5, 4, 5], [2, 5]]
layer_bottom_type=[int8,uint16]
scale_type=[uint8,uint16]
scale_value=[200,32768]
shift_type=[int8,int8]
shift_value=[7,17]
spatial_scale_type=[uint16,uint16]
spatial_scale_value=[2,1]
spatial_shift_type=uint16
spatial_shift_value=11
bin_scale_type=[uint16,uint16]
bin_scale_value=[100,200]
bin_shift_type=[int8,int8]
bin_shift_value=[7,8]
grid_scale_type=[uint16,uint16]
grid_scale_value=[100,300]
grid_shift_type=[int8,int8]
grid_shift_value=[5,8]
sample=[4,3]
method=AVG
coordinate_transformation_mode=OUTPUT_HALF_PIXEL
layer_top_scale=[1.000000]
```

```
layer_top_zp=[0]
pooled_shape=[2, 2]
```

Round

```
layer_id=1
layer_name=Round_0
layer_type=Round
layer_bottom=[Placeholder_0]
layer_bottom_shape=[[2,3,4,5,6]]
layer_bottom_type=[int8]
layer_top=[Round_0]
layer_top_shape=[[2,3,4,5,6]]
layer_top_type=[int8]
layer_top_scale=[31.875000]
layer_top_zp=[0]
lut_type=int8
lut_offset=0
lut_size=256
lut_shape=[256]
```

Rsqrt

```
layer_id=1
layer_name=Rsqrt
layer_type=Rsqrt
layer_bottom=[Placeholder_0]
layer_bottom_shape=[[3,45,8,9]]
layer_bottom_type=[uint8]
layer_top=[Rsqrt_0]
layer_top_shape=[[3,45,8,9]]
layer_top_type=[uint8]
layer_top_scale=[3.225544]
layer_top_zp=[0]
lut_type=uint8
lut_offset=0
lut_size=256
lut_shape=[256]
```

ScatterElements

```
layer_id=3
layer_name=ScatterElements_0
layer_type=ScatterElements
layer_bottom=[Placeholder_0,Placeholder_1_Initializer,Placeholder_2]
layer_bottom_shape=[[3,4,5,6,7],[3,2,3,3,7],[3,2,3,3,7]]
layer_bottom_type=[int8,int8,int8]
layer_top=[ScatterElements_0]
layer_top_shape=[[3,4,5,6,7]]
layer_top_type=[int8]
layer_top_scale=[22.773283]
layer_top_zp=[0]
axis=2
reduction=MUL
scale_type=[uint8,uint8,uint8]
scale_value=[230,182,188]
shift_type=[int8,int8,int8]
shift_value=[13,3,13]
```

ScatterND

```
layer_id=3
layer_name=ScatterND_0
layer_type=ScatterND
layer_bottom=[Placeholder_0,Placeholder_1_Initializer,Placeholder_2]
layer_bottom_shape=[[10,5],[33,2],[33]]
layer_bottom_type=[uint8,uint8,uint8]
layer_top=[ScatterND_0]
layer_top_shape=[[10,5]]
layer_top_type=[uint8]
layer_top_scale=[1.000000]
layer_top_zp=[0]
reduction=NONE
scale_type=[uint8,uint16,uint16]
scale_value=[128,256,256]
shift_type=int8
shift_value=15
```

SegmentSumReduce

```
layer_id=2
layer_name=SegmentSum
layer_type=SegmentReduce
layer_bottom=[Input0,Input1]
layer_bottom_shape=[[3,4],[3]]
layer_bottom_type=[int8,uint8]
layer_top=[SegmentSum_0]
layer_top_shape=[[18,4]]
layer_top_type=[int8]
layer_top_scale=[83.071983]
layer_top_zp=[0]
method=SUM
scale_type=uint16
scale_value=16384
shift_type=int8
shift_value=14
```

Selu

```
layer_id=1
layer_name=Selu
layer_type=Activation
layer_bottom=[Placeholder_0]
layer_bottom_shape=[[9,46,64,2]]
layer_bottom_type=[int8]
layer_top=[Selu_0]
layer_top_shape=[[9,46,64,2]]
layer_top_type=[int8]
layer_top_scale=[23.226768]
layer_top_zp=[0]
lut_type=int8
lut_offset=0
lut_size=256
lut_shape=[256]
method=SELU
```

Shrink

```
layer_id=1
layer_name=Shrink_0
layer_type=Activation
layer_bottom=[Placeholder_0]
layer_bottom_shape=[[2,3]]
layer_bottom_type=[int8]
layer_top=[Shrink_0]
layer_top_shape=[[2,3]]
layer_top_type=[int8]
layer_top_scale=[139.927094]
layer_top_zp=[0]
lut_type=int8
lut_offset=0
lut_size=256
lut_shape=[256]
method=SHRINK
```

Sigmoid

```
layer_id=1
layer_name=Sigmoid
layer_type=Activation
layer_bottom=[Placeholder_0]
layer_bottom_shape=[[7,65,28,3]]
layer_bottom_type=[int8]
layer_top=[Sigmoid_0]
layer_top_shape=[[7,65,28,3]]
layer_top_type=[uint8]
layer_top_scale=[259.193420]
layer_top_zp=[0]
lut_type=uint8
lut_offset=0
lut_size=256
lut_shape=[256]
method=SIGMOID
```

Sign

```
layer_id=1
layer_name=Sign
layer_type=Sign
layer_bottom=[Placeholder_0]
layer_bottom_shape=[[3,23,91,9]]
layer_bottom_type=[int8]
layer_top=[Sign_0]
layer_top_shape=[[3,23,91,9]]
layer_top_type=[int8]
layer_top_scale=[1.000000]
layer_top_zp=[0]
```

Silu

```
layer_id=1
layer_name=Silu
layer_type=Activation
layer_bottom=[Placeholder]
layer_bottom_shape=[[10,20,30,40]]
layer_bottom_type=[int8]
layer_top=[Silu]
layer_top_shape=[[10,20,30,40]]
layer_top_type=[int8]
layer_top_scale=[30.841106]
layer_top_zp=[0]
lut_type=int8
lut_offset=0
lut_size=256
lut_shape=[256]
method=SILU
```

Sine

```
layer_id=1
layer_name=Sin
layer_type=Sine
layer_bottom=[Placeholder_0]
layer_bottom_shape=[[5,59,8,4]]
```

```
layer_bottom_type=[int8]
layer_top=[Sin_0]
layer_top_shape=[[5,59,8,4]]
layer_top_type=[int8]
layer_top_scale=[127.500008]
layer_top_zp=[0]
lut_type=int8
lut_offset=0
lut_size=256
lut_shape=[256]
```

Sinh

```
layer_id=1
layer_name=Sinh_0
layer_type=Sinh
layer_bottom=[Placeholder_0]
layer_bottom_shape=[[2,3,4]]
layer_bottom_type=[int8]
layer_top=[Sinh_0]
layer_top_shape=[[2,3,4]]
layer_top_type=[int8]
layer_top_scale=[17.184174]
layer_top_zp=[0]
lut_type=int8
lut_offset=0
lut_size=256
lut_shape=[256]
```

Slice

```
layer_id=2
layer_name=Slice
layer_type=Slice
layer_bottom=[Placeholder_0,Const_0]
layer_bottom_shape=[[3,64,64,24],[4]]
layer_bottom_type=[int8,uint8]
layer_top=[Slice_0]
layer_top_shape=[[1,50,49,3]]
```

```
layer_top_type=[int8]
layer_top_scale=[28.505024]
layer_top_zp=[0]
begin=[1,8,15,20]
end=[2,58,64,23]
strides=[1,1,1,1]
```

Softmax

```
layer_id=1
layer_name=Softmax
layer_type=Softmax
layer_bottom=[Placeholder_0]
layer_bottom_shape=[[1,56,6,18]]
layer_bottom_type=[int8]
layer_top=[Softmax_0]
layer_top_shape=[[1,56,6,18]]
layer_top_type=[uint8]
layer_top_scale=[255.000000]
layer_top_zp=[0]
lut_type=uint32
lut_offset=0
lut_size=1024
lut_shape=[256]
axis=3
```

Softplus

```
layer_id=1
layer_name=Softplus
layer_type=Activation
layer_bottom=[Placeholder_0]
layer_bottom_shape=[[1,62,7,1]]
layer_bottom_type=[int8]
layer_top=[Softplus_0]
layer_top_shape=[[1,62,7,1]]
layer_top_type=[uint8]
layer_top_scale=[78.195892]
layer_top_zp=[0]
```

```
lut_type=uint8
lut_offset=0
lut_size=256
lut_shape=[256]
method=SOFTPLUS
```

Softsign

```
layer_id=1
layer_name=Softsign
layer_type=Activation
layer_bottom=[Placeholder_0]
layer_bottom_shape=[[8,70,28,10]]
layer_bottom_type=[int8]
layer_top=[Softsign_0]
layer_top_shape=[[8,70,28,10]]
layer_top_type=[int8]
layer_top_scale=[157.307587]
layer_top_zp=[0]
lut_type=int8
lut_offset=0
lut_size=256
lut_shape=[256]
method=SOFTSIGN
```

SpaceToBatch

```
layer_id=1
layer_name=SpaceToBatchND
layer_type=SpaceToBatch
layer_bottom=[Placeholder_0]
layer_bottom_shape=[[2,13,21,3]]
layer_bottom_type=[int8]
layer_top=[SpaceToBatchND_0]
layer_top_shape=[[8,7,11,3]]
layer_top_type=[int8]
layer_top_datalayout=[NHWC]
layer_top_scale=[34.113560]
layer_top_zp=[0]
```

```
block_size_x=2
block_size_y=2
pad_bottom=0
pad_left=1
pad_right=0
pad_top=1
```

SpaceToDepth

```
layer_id=1
layer_name=SpaceToDepth
layer_type=SpaceToDepth
layer_bottom=[Placeholder_0]
layer_bottom_shape=[[1,100,110,10]]
layer_bottom_type=[int8]
layer_top=[SpaceToDepth_0]
layer_top_shape=[[1,50,55,40]]
layer_top_type=[int8]
layer_top_layout=[NHWC]
layer_top_scale=[29.559605]
layer_top_zp=[0]
block_size_x=2
block_size_y=2
```

Split

```
layer_id=1
layer_name=split
layer_type=Split
layer_bottom=[Placeholder_0]
layer_bottom_shape=[[8,214,244,6]]
layer_bottom_type=[int8]
layer_top=[split_0,split_1]
layer_top_shape=[[8,214,122,6],[8,214,122,6]]
layer_top_type=[int8,int8]
layer_top_scale=[25.553686,25.553686]
layer_top_zp=[0,0]
axis=2
splits=[122,122]
```

Sqrt

```
layer_id=1
layer_name=Sqrt
layer_type=Sqrt
layer_bottom=[Placeholder_0]
layer_bottom_shape=[[9,1,78,1]]
layer_bottom_type=[uint8]
layer_top=[Sqrt_0]
layer_top_shape=[[9,1,78,1]]
layer_top_type=[uint8]
layer_top_scale=[140.828018]
layer_top_zp=[0]
lut_type=uint8
lut_offset=0
lut_size=256
lut_shape=[256]
```

Square

```
layer_id=1
layer_name=Square
layer_type=Square
layer_bottom=[Placeholder_0]
layer_bottom_shape=[[9,59,40,5]]
layer_bottom_type=[int8]
layer_top=[Square_0]
layer_top_shape=[[9,59,40,5]]
layer_top_type=[uint8]
layer_top_scale=[10.573904]
layer_top_zp=[0]
lut_type=uint8
lut_offset=0
lut_size=256
lut_shape=[256]
```

SquaredDifference

```
layer_id=4
layer_name=SquaredDifference
layer_type=SquaredDifference
layer_bottom=[SquaredDifference_pre_tile,SquaredDifference_pre_tile_0]
layer_bottom_shape=[[2,3,4],[2,3,4]]
layer_bottom_type=[uint8,uint8]
layer_top=[SquaredDifference]
layer_top_shape=[[2,3,4]]
layer_top_type=[uint8]
layer_top_scale=[1.000000]
layer_top_zp=[0]
scale_type=[uint8,uint16,uint16,uint16]
scale_value=[128,256,256,16384]
shift_type=[int8,int8]
shift_value=[7,22]
```

Sub

```
layer_id=3
layer_name=Sub_
layer_type=Sub
layer_bottom=[input_0,input_1]
layer_bottom_shape=[[2,256],[256]]
layer_bottom_type=[int8,int8]
layer_top=[output]
layer_top_shape=[[2,256]]
layer_top_type=[int8]
layer_top_scale=[0.001733]
layer_top_zp=[0]
scale_type=[uint8,uint16,uint16]
scale_value=[188,32767,256]
shift_type=int8
shift_value=29
layer_top_scale=[1.000000]
layer_top_zp=[0]
```

SufficientStatistics

```
layer_id=2
layer_name=Identity
layer_type=SufficientStatistics
layer_bottom=[Placeholder,Identity_inp1]
layer_bottom_shape=[[2,3,4,5],[2,3,4,5]]
layer_bottom_type=[int8,uint8]
layer_top=[Identity_0,Identity_1]
layer_top_shape=[[2,3,1,1],[2,3,1,1]]
layer_top_type=[int8,uint8]
layer_top_scale=[16.48089599609375,9.98042106628418]
layer_top_zp=[0,0]
axis=[2,3]
unquantifiable=false
shift_value=[16,23,14,8]
shift_type=[int8,int8,int8,int8]
scale_value=[16876,20440,23918,16384]
scale_type=[uint16,uint16,uint16,uint16]
```

Swish

```
layer_id=1
layer_name=Swish
layer_type=Activation
layer_bottom=[Placeholder]
layer_bottom_shape=[[10,20,30,40]]
layer_bottom_type=[int8]
layer_top=[Swish]
layer_top_shape=[[10,20,30,40]]
layer_top_type=[int8]
layer_top_scale=[30.841106]
layer_top_zp=[0]
lut_type=int8
lut_offset=0
lut_size=256
lut_shape=[256]
method=SWISH
```

Tan

```
layer_id=1
layer_name=Tan_0
layer_type=Tan
layer_bottom=[Placeholder_0]
layer_bottom_shape=[[2,3,4]]
layer_bottom_type=[int8]
layer_top=[Tan_0]
layer_top_shape=[[2,3,4]]
layer_top_type=[int8]
layer_top_scale=[0.215925]
layer_top_zp=[0]
lut_type=int8
lut_offset=0
lut_size=256
lut_shape=[256]
```

Tanh

```
layer_id=1
layer_name=Tanh
layer_type=Activation
layer_bottom=[Placeholder_0]
layer_bottom_shape=[[1,14,4,7]]
layer_bottom_type=[int8]
layer_top=[Tanh_0]
layer_top_shape=[[1,14,4,7]]
layer_top_type=[int8]
layer_top_scale=[127.722069]
layer_top_zp=[0]
lut_type=int8
lut_offset=0
lut_size=256
lut_shape=[256]
method=TANH
```

ThresholdedRelu

```
]layer_id=1
layer_name=ThresholdedRelu_0
layer_type=Activation
layer_bottom=[Placeholder_0]
layer_bottom_shape=[[2,3]]
layer_bottom_type=[int8]
layer_top=[ThresholdedRelu_0]
layer_top_shape=[[2,3]]
layer_top_type=[uint8]
layer_top_scale=[215.078720]
layer_top_zp=[0]
lut_type=uint8
lut_offset=0
lut_size=256
lut_shape=[256]
method=THRESHOLDEDRELU
```

Tile

```
layer_id=1
layer_name=Tile
layer_type=Tile
layer_bottom=[Placeholder_0]
layer_bottom_shape=[[7,10,57,1]]
layer_bottom_type=[int8]
layer_top=[Tile_0]
layer_top_shape=[[14,10,114,3]]
layer_top_type=[int8]
layer_top_scale=[36.911419]
layer_top_zp=[0]
repeats=[2,1,2,3]
```

TopK

```
layer_id=1
layer_name=TopKV2
layer_type=TopK
layer_bottom=[Placeholder_0]
```

```
layer_bottom_shape=[[8,10,71,3]]
layer_bottom_type=[int8]
layer_top=[TopKV2_0,TopKV2_1]
layer_top_shape=[[8,10,71,2],[8,10,71,2]]
layer_top_type=[int8,uint16]
layer_top_scale=[30.131403,1.000000]
layer_top_zp=[0,0]
axis=3
k=2
largest=true
select_index=last
sorted=true
```

Transpose

```
layer_id=1
layer_name=transpose
layer_type=Transpose
layer_bottom=[Placeholder_0]
layer_bottom_shape=[[7,19,3,8]]
layer_bottom_type=[int8]
layer_top=[transpose_0]
layer_top_shape=[[7,3,19,8]]
layer_top_type=[int8]
layer_top_scale=[33.210510]
layer_top_zp=[0]
perm=[0,2,1,3]
```

Trunc

```
layer_id=1
layer_name=Trunc_0
layer_type=Trunc
layer_bottom=[Placeholder_0]
layer_bottom_shape=[[2,3,4]]
layer_bottom_type=[int8]
layer_top=[Trunc_0]
layer_top_shape=[[2,3,4]]
layer_top_type=[int8]
```

```
layer_top_scale=[0.215925]
layer_top_zp=[0]
lut_type=int8
lut_offset=0
lut_size=256
lut_shape=[256]
```

UpsampleByIndex

```
layer_id=2
layer_name=upsample4_0_4
layer_type=UpsampleByIndex
layer_bottom=[input1,input2]
layer_bottom_shape=[[1,64,128,64],[1,64,128,64]]
layer_bottom_type=[int8,int32]
layer_top=[upsample4_0_4]
layer_top_shape=[[1,128,256,64]]
layer_top_type=[int8]
output_shape=[1,128,256,64]
flatten_dim=HW
storage_order=0
```

Where

```
layer_id=3
layer_name=Select
layer_type=Where
layer_bottom=[Cast/x_0,Placeholder_0,Placeholder_1_0]
layer_bottom_shape=[[4,29,22,6],[4,29,22,6],[4,29,22,6]]
layer_bottom_type=[uint8,int8,int8]
layer_top=[Select_0]
layer_top_shape=[[4,29,22,6]]
layer_top_type=[int8]
layer_top_scale=[27.380608]
layer_top_zp=[0]
scale_type=[uint8,uint8]
scale_value=[204,128]
shift_type=[int8,int8]
shift_value=[8,7]
```

YuvToRgb

```
layer_id=1
layer_name=Placeholder
layer_type=YuvToRgb
layer_bottom=[preprocess_Input]
layer_bottom_shape=[[1,86400]]
layer_bottom_type=[uint8]
layer_top=[Placeholder_0]
layer_top_shape=[[1,240,240,3]]
layer_top_type=[uint8]
layer_top_scale=[1.000000]
layer_top_zp=[0]
bits=8
coefficient=[0,128,128,0,0,0,256,0,403,256,-48,-120,256,475,0]
coefficient_dtype=int16
coefficient_shift=8
conversion=BT709
format=I420
```

ZeroFraction

```
layer_id=1
layer_name=zero_fraction/counts_to_fraction/truediv
layer_type=ZeroFraction
layer_bottom=[Placeholder_0]
layer_bottom_shape=[[10,20,30,40]]
layer_bottom_type=[int8]
layer_top=[zero_fraction/counts_to_fraction/truediv_0]
layer_top_shape=[[1]]
layer_top_type=[uint8]
layer_top_scale=[255.000000]
layer_top_zp=[0]
output_scale=255
```

Appendix A Revision history

This appendix describes the technical changes between released issues of this book.

Table A-1: Issue A (Version 1.0)

Change	Location
First release.	-

Table A-2: Issue B (Version 2.0)

Change	Location
Added more basic operator parameters and operator examples.	4.3.1 Basic operator parameters on page 12 5 Build-in operator examples on page 146

Table A-3: Issue C (Version 3.0)

Change	Location
Updated the common parameters, added more basic operator parameters, and updated some operator examples.	4.2 Common parameters on page 11 4.3.1 Basic operator parameters on page 12 5 Build-in operator examples on page 146

Table A-4: Issue D (Version 4.0)

Change	Location
Updated and added a few basic operator parameters.	4.3.1 Basic operator parameters on page 12

Table A-5: Issue E (Version 5.0)

Change	Location
Added more basic operator parameters and operator examples.	4.3.1 Basic operator parameters on page 12 5 Build-in operator examples on page 146

Table A-6: Issue F (Version 6.0)

Change	Location
Added the Add operator.	Add on page 15
Added the Compress operator.	Compress on page 31
Added the GroupNormalization operator.	GroupNormalization on page 75
Added the MaxUnpool operator.	MaxUnpool on page 91
Added the Mul operator.	Mul on page 95
Updated the NMS operator.	NMS on page 97
Added the ReduceL1 operator.	ReduceL1 on page 105
Added the ReduceL2 operator.	ReduceL2 on page 105
Added the ReduceUnbiasedVariance operator.	ReduceUnbiasedVariance on page 109
Added the ReduceVariance operator.	ReduceVariance on page 109
Added the Sub operator.	Sub on page 134
Added the Add operator example.	Add on page 147
Added the Compress operator example.	Compress on page 160

Change	Location
Added the GroupNormalization operator example.	GroupNormalization on page 189
Added the MaxUnpool operator example.	MaxUnpool on page 203
Added the Mul operator example.	Mul on page 206
Updated the NMS operator example.	NMS on page 207
Updated the OneHot operator example.	OneHot on page 209
Added the ReduceL1 operator example.	ReduceL1 on page 212
Added the ReduceL2 operator example.	ReduceL2 on page 213
Added the ReduceUnbiasedVariance operator example.	ReduceUnbiasedVariance on page 216
Added the ReduceVariance operator example.	ReduceVariance on page 217
Added the Sub operator example.	Sub on page 234

Table A-7: Issue G (Version 7.0)

Change	Location
Added the BitwiseAnd operator.	BitwiseAnd on page 25
Added the BitwiseNot operator.	BitwiseNot on page 26
Added the BitwiseOr operator.	BitwiseOr on page 26
Added the BitwiseXor operator.	BitwiseXor on page 26
Added the CumProd operator.	CumProd on page 44
Added the CumSum operator.	CumSum on page 45
Updated the Dilation2D operator.	Dilation2D on page 49
Added the Erf operator.	Erf on page 58
Added the Erosion2D operator.	Erosion2D on page 58
Added the Gelu operator.	Gelu on page 70
Added the GetValidCounts operator.	GetValidCounts on page 71
Added the Meshgrid operator.	Meshgrid on page 93
Added the MultiboxTransformLoc operator.	MultiboxTransformLoc on page 95
Added the Swish operator.	Swish on page 135
Added the Trunc operator.	Trunc on page 138
Added the BitwiseAnd operator example.	BitwiseAnd on page 155
Added the BitwiseNot operator example.	BitwiseNot on page 155
Added the BitwiseOr operator example.	BitwiseOr on page 156
Added the BitwiseXor operator example.	BitwiseXor on page 156
Added the CumProd operator example.	CumProd on page 168
Added the CumSum operator example.	CumSum on page 169
Updated the Dilation2D operator example.	Dilation2D on page 172
Added the Erf operator example.	Erf on page 178
Added the Erosion2D operator example.	Erosion2D on page 178
Added the Gelu operator example.	Gelu on page 185
Added the GetValidCounts operator example.	GetValidCounts on page 186
Added the Meshgrid operator example.	Meshgrid on page 204
Added the MultiboxTransformLoc operator example.	MultiboxTransformLoc on page 206
Added the Swish operator example.	Swish on page 235

Change	Location
Added the Trunc operator example.	Trunc on page 238

Table A-8: Issue H (Version 7.1)

Change	Location
Added a common parameter about compat_quantized_model.	4.2 Common parameters on page 11
Added the DecodeBox operator.	DecodeBox on page 46
Added the NormalizedMoments operator.	NormalizedMoments on page 99
Added the Region operator.	Region on page 110
Added the DecodeBox operator example.	DecodeBox on page 170
Added the NormalizedMoments operator example.	NormalizedMoments on page 208
Added the Region operator example.	Region on page 217

Table A-9: Issue I (Version 7.2)

Change	Location
Updated the Cast operator.	Cast on page 29
Updated the GroupNormalization operator.	GroupNormalization on page 75
Updated the InstanceNormalization operator.	InstanceNormalization on page 77
Updated the LayerNormalization operator.	LayerNormalization on page 83
Updated the MeanVarianceNormalization operator.	MeanVarianceNormalization on page 92
Added the FractionalPool operator.	FractionalPool on page 60
Added the L1Normalization operator.	L1Normalization on page 78
Added the L2Normalization operator.	L2Normalization on page 80
Added the RegionFuse operator.	RegionFuse on page 112
Added the FractionalPool operator example.	FractionalPool on page 180
Added the L1Normalization operator example.	L1Normalization on page 192
Added the L2Normalization operator example.	L2Normalization on page 193
Added the RegionFuse operator example.	RegionFuse on page 219

Table A-10: Issue J (Version 7.3)

Change	Location
Updated the Compress operator.	Compress on page 31
Added the BoundingBox operator.	BoundingBox on page 27
Added the EmbeddingLookupSparse operator.	EmbeddingLookupSparse on page 57
Added the SufficientStatistics operator.	SufficientStatistics on page 134
Added the BoundingBox operator example.	BoundingBox on page 156
Added the EmbeddingLookupSparse example.	EmbeddingLookupSparse on page 177
Added the SufficientStatistics example.	SufficientStatistics on page 235