



Denglin Hamming™ V2

dINNE ONNX Quantization Operator

DL-DG/SW-031E-01

2023-4-30

Copyright©苏州登临科技有限公司，2019 - 2025，版权所有。

未经苏州登临科技有限公司事先书面同意，不得以任何形式或方式复制或传播本文件的任何部分。

商标和许可



和其它苏州登临科技有限公司的其它登临科技的图标为苏州登临科技有限公司的商标。本手册中提及的所有其他商标均为其各自所有者的财产。

通知

所购买的产品、服务和特性由苏州登临科技有限公司与客户签订的合同规定。本文件中描述的所有或部分产品、服务和特性可能不在采购范围或使用范围内。除非合同中另有规定，本文件中的所有声明、信息和建议均按“原样”提供，无任何明示或暗示的保证或陈述。

本手册中的信息如有更改，恕不另行通知。本文件在编制过程中已尽一切努力确保内容的准确性，本文件中的所有声明、信息和建议不构成任何明示或暗示的保证。

苏州登临科技有限公司

苏州工业园区扬富路11号南岸新地一期商务楼5号1101室，江苏，中国

<http://www.denglin.ai>

Email : support@denglin.ai

Change History

Version	Change description
01	Initial version

Table of Contents

Table of Contents

1 Introduction

2 Operators

[quantized_matmul](#)

[quantized_conv2d](#)

[quantized_conv2d_backprop_input](#)

[quantized_add](#)

[quantized_batch_matmul](#)

[quantized_conv3d](#)

[quantized_maxpool2d](#)

[quantized_avgpool2d](#)

[quantized_concatenate](#)

[quantized_relu](#)

[quantized_embedding_bag](#)

1 Introduction

dINNE ONNX quantization operator is an umbrella operator which groups a set of quantization operators distinguished by attribute `operator`.

2 Operators

8-bit quantization approximates floating point value using the following formula.

$$\text{real_value} = (\text{int_value} - \text{zero_point}) * \text{scale}$$

All `scale` and `zero_point` in following operators follow this formula.

`quantized_matmul`

Attributes

`operator`: "quantized_matmul", string

`out_dtype`: string

`scales_a`: list of floats

`scales_b`: list of floats

`scales_product`: list of floats

`zero_points_a`: list of ints

`zero_points_b`: list of ints

`zero_points_product`: list of ints

Inputs

`a`

`b`

`bias` (optional)

Output

`y`

Example

```

attr = {
    "operator": "quantized_matmul",
    "out_dtype": out_dtype,
    "scales_a": (1.0),
    "scales_b": (1.0),
    "scales_product": (1.0),
    "zero_points_a": (1),
    "zero_points_b": (1),
    "zero_points_product": (1),
}
mul_node = helper.make_node("DLQuantization", ["a", "b"], ["out"], **attr,
domain="com.denglin")

```

quantized_conv2d

Attributes

operator: "quantized_conv2d", string

strides: list of ints

pad_width: list of ints.

Padding for the beginning and ending along each spatial axis. Each value of **pad_width** can take any integer greater than or equal to 0. The value represents the number of pixels added to the beginning and ending parts of the corresponding axis. The format of **pad_width** should be `[x1_begin, x2_begin...x1_end, x2_end,...]`, wherein `xi_begin` is the number of pixels added at the beginning of axis `i`, and `xi_end` is the number of pixels added at the end of axis `i`. **pad_width** must contain padding information for all axes.

dilation: list of ints

groups: int

channels: int

kernel_size: list of ints

data_layout: list of ints

kernel_layout: list of ints

out_dtype: string

data_scales: list of floats

data_zero_points: list of ints

weight_scales: list of floats

weight_zero_points: list of ints

scales: list of floats

zero_points: list of ints

Inputs

a

b

bias (optional)

Output

y

Example

```

attr = {
    "operator": "quantized_conv2d",
    "strides": (1, 1),
    "pad_width": (1, 1, 1, 1, 2, 2, 2, 2),
    "dilation": (1, 1),
    "groups": 1,
    "channels": 3,
    "kernel_size": (3, 3),
    "data_layout": "NCHW",
    "kernel_layout": "OIHW",
    "out_dtype": "uint8",
    "data_scales": (1.0),
    "weight_scales": (1.0),
    "scales": (1.0),
    "data_zero_points": (1),
    "weight_zero_points": (1),
    "zero_points": (1),
}
node = helper.make_node("DLQuantization", ["a", "b"], ["out"], **attr,
    domain="com.denglin")

```

quantized_conv2d_backprop_input**Attributes**

operator: "quantized_conv2d_backprop_input", string

dilation: list of ints

strides: list of ints

input_sizes: list of ints

padding: string

data_format: string

out_dtype: string

out_backprop_scales: list of floats

out_backprop_zero_points: list of ints

filter_scales: list of floats

`filter_zero_points`: list of ints`scales`: list of floats`zero_points`: list of ints**Inputs**`a``b``bias` (optional)**Output**`y`**Example**

```

attr = {
    "operator": "quantized_conv2d_backprop_input",
    "dilation": dilation,
    "strides": strides,
    "padding": padding,
    "input_sizes": input_sizes,
    "kernel_size": (3, 3),
    "data_format": data_format,
    "out_dtype": out_dtype,
    "out_backprop_scales": out_backprop_scales,
    "filter_scales": filter_scales,
    "scales": scales,
    "out_backprop_zero_points": out_backprop_zero_points,
    "filter_zero_points": filter_zero_points,
    "zero_points": zero_points,
}
node = helper.make_node("DLQuantization", ["a", "b"], ["out"], **attr,
    domain="com.denglin")

```

quantized_add**Attributes**`operator`: "quantized_add", string`out_dtype`: string`lhs_scales`: list of floats`rhs_scales`: list of floats`scales`: list of floats`lhs_zero_points`: list of ints`rhs_zero_points`: list of ints

`zero_points`: list of ints`with_relu`: int**Inputs**`a``b`**Output**`y`**Example**

```

attr = {
    "operator": "quantized_add",
    "out_dtype": out_dtype,
    "lhs_scales": a_scales,
    "rhs_scales": b_scales,
    "scales": y_scales,
    "lhs_zero_points": a_zero_points,
    "rhs_zero_points": b_zero_points,
    "zero_points": y_zero_points,
    "with_relu": with_relu,
}
node = helper.make_node("DLQuantization", ["a", "b"], ["out"], **attr,
    domain="com.denglin")

```

quantized_batch_matmul**Attributes**`operator`: "quantized_batch_matmul", string`out_dtype`: string`scales_a`: list of floats`scales_b`: list of floats`scales_product`: list of floats`zero_points_a`: list of ints`zero_points_b`: list of ints`zero_points_product`: list of ints**Inputs**`a``b``bias` (optional)

Output

y

Example

```

attr = {
    "operator": "quantized_batch_matmul",
    "out_dtype": out_dtype,
    "scales_a": a_scales,
    "scales_b": b_scales,
    "scales_product": y_scales,
    "zero_points_a": a_zero_points,
    "zero_points_b": b_zero_points,
    "zero_points_product": y_zero_points,
}
mul_node = helper.make_node("DLQuantization", ["a", "b"], ["out"], **attr,
domain="com.denglin")

```

quantized_conv3d

Attributes

operator: "quantized_conv3d", string

data_zero_points: list of ints

weight_zero_points: list of ints

data_scales: list of floats

weight_scales: list of floats

scales: list of floats

zero_points: list of ints

dilation: list of ints

strides: list of ints

padding: string

data_layout: list of ints

out_dtype: string

Inputs

data

weight

bias (optional)

Output

y

quantized_maxpool2d**Attributes**

operator: "quantized_maxpool2d", string

pool_size: list of ints

strides: list of ints

padding: string

data_format: string

scales: list of floats

zero_points: list of ints

paddings: list of ints

Inputs

a

Output

y

Example

```

attr = {
    "operator": "quantized_conv3d",
    "strides": (1, 1),
    "padding": padding,
    "dilation": (1, 1),
    "data_layout": "NCHW",
    "out_dtype": out_dtype,
    "data_scales": a_scales,
    "weight_scales": b_scales,
    "scales": y_scales,
    "data_zero_points": a_zero_points,
    "weight_zero_points": b_zero_points,
    "zero_points": y_zero_points,
}
node = helper.make_node("DLQuantization", ["a", "b"], ["out"], **attr,
domain="com.denglin")

```

quantized_avgpool2d

Attributes

`operator`: "quantized_avgpool2d", string

`pool_size`: list of ints

`strides`: list of ints

`padding`: string

`data_format`: string

`scales`: list of floats

`zero_points`: list of ints

`mode`: string

`paddings`: list of ints

Inputs

`a`

Output

`y`

Example

```
attr = {
    "operator": "quantized_avgpool2d",
    "pool_size": pool_size,
    "strides": strides,
    "padding": padding,
    "data_format": data_format,
    "scales": scales,
    "zero_points": zero_points,
    "paddings": paddings,
    "mode": mode,
}
node = helper.make_node("DLQuantization", ["a"], ["out"], **attr, domain="com.denglin")
```

quantized_concatenate

Attributes

`operator`: "quantized_concatenate", string

`axis`: int

`inputs_scales`: list of floats

`inputs_zero_points`: list of ints

`scales`: list of floats`zero_points`: list of ints`out_dtype`: string`mode`: string**Inputs**`inputs` (variadic)

List of tensors for concatenation.

Output`y`**Example**

```

attr = {
    "operator": "quantized_concatenate",
    "axis": axis,
    "inputs_scales": inputs_scales,
    "inputs_zero_points": inputs_zero_points,
    "scales": output_scales,
    "zero_points": output_zero_points,
    "out_dtype": out_dtype,
    "mode": mode,
}
node = helper.make_node("DLQuantization", ["a", "b"], ["out"], **attr,
domain="com.denglin")

```

quantized_relu**Attributes**`operator`: "quantized_relu", string`scales`: list of floats`zero_points`: list of ints`mode`: string**Inputs**`a`**Output**`y`**Example**

```
attr = {
    "operator": "quantized_relu",
    "scales": y_scales,
    "zero_points": y_zero_points,
    "mode": mode,
}
node = helper.make_node("DLQuantization", ["a"], ["out"], **attr, domain="com.denglin")
```

quantized_embedding_bag

Attributes

`operator`: "quantized_embedding_bag", string

`include_last_offset`: int

`mode`: string

Inputs

`embedding_matrix`

The embedding matrix with the number of rows equal to the maximum possible index + 1, and the number of columns equal to the embedding size.

`scales`

per-channel scales of `embedding_matrix`.

`zero_points`

per-channel zero_points of `embedding_matrix`.

`indices`

Tensor containing bags of indices into the embedding matrix.

`offsets`

`offsets` determines the starting index position of each bag (sequence) in input.

Output

`embedding_bag`, `offset2bag`, `bag_size`, `max_indices`

Example

```
attr = {
    "operator": "quantized_embedding_bag",
    "include_last_offset": include_last_offset,
    "mode": mode,
}
node = helper.make_node(
    "DLQuantization",
    ["embedding_matrix", "scales", "zero_points", "indices", "offsets"],
    ["embedding_bag", "offset2bag", "bag_size", "max_indices"],
```

```

    **attr,
    domain="com.denglin",
)

graph = helper.make_graph(
    [node],
    "quantized_embedding_bag_test",
    inputs=[
        helper.make_tensor_value_info("embedding_matrix", TensorProto.UINT8,
list(a_shape)),
        helper.make_tensor_value_info("scales", TensorProto.FLOAT, list(b_shape)),
        helper.make_tensor_value_info("zero_points", TensorProto.FLOAT, list(b_shape)),
        helper.make_tensor_value_info("indices", TensorProto.INT64, list(indices_shape)),
        helper.make_tensor_value_info("offsets", TensorProto.INT64, list(offsets_shape)),
    ],
    outputs=[
        helper.make_tensor_value_info("embedding_bag", TensorProto.FLOAT, list(out_shape)),
        helper.make_tensor_value_info("offset2bag", TensorProto.FLOAT, list(out_shape)),
        helper.make_tensor_value_info("bag_size", TensorProto.FLOAT, list(out_shape)),
        helper.make_tensor_value_info("max_indices", TensorProto.FLOAT, list(out_shape)),
    ],
)

model = helper.make_model(graph, producer_name="quantized_embedding_bag_test")

```