Ingenic®

Detailed Explanation of PyTorch Quantization Operators

Date: Feb. 2022

Ingenic®

Detailed Explanation of PyTorch Quantization Operator

Release history

Date	Author	Revision	Change
Feb.2022	Allen	1.0.1	First release

Disclaimer

This documentation is provided for use with Ingenic products. No license to Ingenic property rights is granted. Ingenic assumes no liability, provides no warranty either expressed or implied relating to the usage, or intellectual property right infringement except as provided for by Ingenic Terms and Conditions of Sale.

Ingenic products are not designed for and should not be used in any medical or life sustaining or supporting equipment.

All information in this document should be treated as preliminary. Ingenic may make changes to this document without notice. Anyone relying on this documentation should contact Ingenic for the current documentation and errata.

Ingenic Semiconductor Co., Ltd.

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

Contents

I. Operator Lists

Operators' interface are defined in file ops.py. When the quantization plug-in is successfully installed, ingenic_magik_trainingkit (pytorch peer directory) will be added to the current environment , the file ops.py is located under the directory "ingenic_magik_trainingkit/QuantizationTrainingPlugin/python/".

Preprocess

Function Description:

Identify the preprocessed mean and variance parameters of network input data. Facilitate the transformation of later models.

Interface Definition:

```
Preprocess(mean=0.,
           var=255.,
           target_device="Txx")
```

Detailed Explanation of Parameters:

-mean - The average value of data preprocessing, default value is 0.
-var - Variance of data preprocessing, default value is 255.
-target_device - The target hardware device for model operation, currently has six options: T40, T02, Txx, Xs1, Xs2 or T41.

Example:

```
preprocess = ops.Preprocess(128, 128, "T40")
input = torch.rand((1,3,2,2))
input = preprocess(input)
```

Result:

The format of the return data is python tuple type, in which the first element is the input tensor, the second is the input bit, fixed as 8, the third and fourth are the mean and variance used in preprocessing respectively, and the last is the minimum value. If the mean is 0, the value is 0, otherwise it is set to -1.

Attention:

1. This method must be used before the network, otherwise an error will be reported.
2. The incoming mean and variance must be the actual values used in the previous preprocessing, otherwise the converted model will be wrong.

Conv2D

Function Description:

The encapsulated convolution operator, including convolution and weight quantization, feature quantization, batchnorm, biasadd and relu(/relu6) activate operation. Each operation can be enabled or disabled according to the corresponding parameters.

Interface Definition:

```
Conv2D(in_channels,
        out_channels,
        stride=1,
        kernel_h=3,
        kernel_w=3,
        activation_fn=ReLU6(),
        enable_batch_norm=False,
        enable_bias=True,
        padding=0,
        groups=1,
        dilation=1,
        data_format="NCHW",
        quantize=False,
        quantize_last_feature=False,
        first_layer=False,
        last_layer=False,
        weight_bitwidth=32,
        input_bitwidth=32,
        output_bitwidth=32,
        clip_max_value=6.0,
        weight_factor=3.0,
        target_device="Txx",
        auto_bitwidth=False,
        is_focus=False,
        auto_threshold=False,
        pre_lstm_layer=False)
```

Detailed Explanation of Parameters:

-in_channels - Number of input channels.
-out_channels - Number of output channels.
-stride - Stride size, default value is 1.
-kernel_h - Height of kernel , default value is 3.
-kernel_w - Width of kernel, default value is 3.

-enable_batch_norm - Enable/Disable BatchNorm2d (after convolution).
-enable_bias - Enable/Disable bias (in convolution).
-activation_fn - Activation function, please directly refer to the activation list behind the operators for details about which activation functions are supported.
-padding - Pad mode of convolution, 0 means no filling, and 1 means feature calculation filling.
-groups - The function is to control the grouping convolution, there is no grouping by default.
-dilation - The parameter of dilated convolution, it is set to 1 by default.
-data_format - Format of input and output data, it's "NCHW" by default.
-quantize - Whether to do quantization or not (weight and feature). If it is True, the quantization operation will be performed.
-quantize_last_feature - Whether to perform quantization on the last layer (Conv2d) of the network or not. If it is set to True, the quantization operation will be performed on the output feature, otherwise the quantization will not be performed on the output feature. If the current layer is not the last layer of the network, then this parameter is considered as invalid.
-first_layer - If it is the first layer of the network, it must be set to True.
-last_layer - If it is the final output layer, it must to be set to True.
-weight_bitwidth - The quantization bit of the weight. It is valid when "quantize" is set to True.
-input_bitwidth - The quantization bit width of the feature. It is valid when "quantize" is True.
-output_bitwidth - The quantization bit width of the convolution output feature. It should be set to 32 for the last layer.
-clip_max_value - Clipping threshold of the feature when "quantize" is True. It is recommended to set it to 6.0 for 8bit and 4.0 for 4bit.
-weight_factor - The pruning factor of the weight when "quantize" is True. The smaller the value, the higher the utilization of the quantization bit width, and the less effective weight will be retained.
-target_device - The target hardware device for model operation, we can support T40, T02, Txx, Xs1, Xs2 or T41 now.
-auto_bitwidth - Whether to automatically calculate the bit width when target_device is Txx or Xs1 or Xs2 and the bit width is less than 8bit .
-is_focus - If there is a focus operation before convolution, then set it to "True".
-auto_threshold - Whether to automatically calculate the clipping threshold. There is no use now.
-pre_lstm_layer - If it is the previous layer of LSTM , then set it to "True".

Example:

```
Conv1 = ops.Conv2D(3,16,
                   kernel_h = 3,
                   kernel_3 = 3,
                   Padding = 1,
```

```
                    Quantize = True,
                    enable_batch_norm =  True,
                    enable_bias = False,
                    activation_fn = ops.ReLU6(),
                    first_layer=True,
                    weight_bitwidth = 8,
                    output_bitwidth = 8,
                    clip_max_value = 4.0,
                    weight_factor = 3.0,
                    target_device = "T40")
input = torch.rand((1,3,2,2))
input_a = Conv1((input, 8, 0.))
```

Result:

The format of the return data is python tuple type, in which the first element in the tuple is the 4D tensor data of NCHW, which represents the calculation result of Conv2D, other elements are parameters to be passed in quantization training.

Attention:

1. Full precision training is performed when “quantize” is False, and quantization training is performed when “quantize” is true.
2. When the convolution is the last layer of the network, if “quantize” is True and “quantify_last_feature” is True, the quantization is performed on both weight and feature; otherwise, it is performed only on weight. If it is the last layer and the function is not activated, set “activation_fn” to None.
3. If “input_channels” = “out_channels” = “groups”, then use deep separable convolution, which is performed by our special grouping convolution operator “ops.DepthwiseConv2D()”.

ConvTranspose2D

Function Description:

The encapsulated combined deconvolution operator includes depth convolution, point convolution, weight quantization, feature quantization, batchnorm, biasadd and relu (relu6) activation. Each operation can be enabled or disabled according to the input parameters.

Interface Definition:

```
ConvTranspose2D(in_channels,
                    out_channels,
                    activation_fn=ReLU6(),
                    enable_batch_norm=False,
```

```
                    enable_bias=True,
                    data_format="NCHW",
                    quantize=False,
                    quantize_last_feature=False,
                    last_layer=False,
                    weight_bitwidth=32,
                    input_bitwidth=32,
                    output_bitwidth=32,
                    clip_max_value=6.0,
                    weight_factor=3.0,
                    target_device="Txx",
                    auto_bitwidth=True,
                    auto_threshold=False,
                    pre_lstm_layer=False)
```

Detailed Explanation of Parameters:

-in_channels - Number of input channels.
-out_channels - Number of output channels.
-activation_fn - Activation function, please directly refer to the activation list behind the operators for details about which activation functions are supported.
-enable_batch_norm - Enable/Disable BatchNorm2d (after transpose convolution).
-enable_bias - Enable/Disable bias (in transpose convolution).
-data_format - Format of input and output data, it's "NCHW" by default.
-quantize - Whether to do quantization(weight and feature). If it is True, the quantization operation will be performed.
-quantize_last_feature - Whether to perform quantization on the last layer (ConvTranspose2D) of the network or not. If it is set to True, the quantization operation will be performed on the output feature, otherwise the quantization will not be performed on the output feature. If the current layer is not the last layer of the network, then this parameter is considered as invalid.
-last_layer - If it is the final output layer, it must to be set to True.
-weight_bitwidth - The quantization bit of the weight . It is valid when "quantize" is True.
-input_bitwidth - The quantization bit width of the feature. It is valid when "quantize" is True.
-output_bitwidth - The quantization bit width of the convolution output feature. It should be set to 32 for the last layer.
-clip_max_value - Clipping threshold of the feature when "quantize" is True. It is recommended to set it to 6.0 for 8bit and 4.0 for 4bit.
-weight_factor - The pruning factor of the weight when "quantize" is True. The smaller the value, the higher the utilization of the quantization bit width, and the less effective weight will be retained.
-target_device - The target hardware device for model operation, we can support T40, Txx, Xs1, Xs2 or T41 now.

-auto_bitwidth - Whether to automatically calculate the bit width when target_device is Txx or Xs1 or Xs2 and the bit width is less than 8bit .
-auto_threshold - Whether to automatically calculate the clipping threshold. There is no use now.
-pre_lstm_layer - If it is the previous layer of LSTM , then set it to True.

Example:

```
Conv1_c = ops.ConvTranspose2D(3,16,
                              quantize=True,
                              enable_batch_norm = True,
                              enable_bias = False,
                              activation_fn = ops.ReLU6(),
                              weight_bitwidth = 8,
                              output_bitwidth = 8,
                              clip_max_value = 4.0,
                              weight_factor=3.0,
                              target_device="T40")
input = torch.rand((1,3,2,2))
output = Conv1_c((input_i, 8, 0.))
```

Result:

The format of the return data is python tuple type, in which the first element in the tuple is the 4D tensor data of NCHW, which represents the calculation result of ConvTranspose2D, others are parameters to be passed in quantization training.

Attention:

This operator can replace torch.nn.Convtranspose2d operation, unpool+depthwise+ conv for Txx, Xs1 or Xs2, and unpool+conv for T40 or T41.

DepthwiseConv2D

Function Description:

The encapsulated deep separable convolution operator includes operations such as depthwise convolution, weight quantization, feature quantization, batchnorm, biasadd and relu (relu6) activation. Each operation can be enabled or disabled according to the input parameters.

Interface Definition:

```
DepthwiseConv2D(in_channels,
                stride=1,
                kernel_h=3,
                kernel_w=3,
```

```
                    activation_fn=ReLU6(),
                    enable_batch_norm=False,
                    enable_bias=True,
                    padding=0,
                    data_format="NCHW",
                    quantize=False,
                    quantize_last_feature=False,
                    last_layer=False,
                    weight_bitwidth=32,
                    input_bitwidth=32,
                    output_bitwidth=32,
                    clip_max_value=6.0,
                    weight_factor=3.0,
                    target_device="Txx",
                    auto_bitwidth=True,
                    auto_threshold=False,
                    pre_lstm_layer=False)
```

Detailed Explanation of Parameters:

-in_channels - Number of input channels.
-stride - Stride size, default value is 1.
-kernel_h - Height of kernel , default value is 3.
-kernel_w - Width of kernel, default value is 3.
-activation_fn - Activation function, please directly refer to the activation list behind the operators for details about which activation functions are supported.
-enable_batch_norm - Enable/Disable BatchNorm2d (after depthwise convolution).
-enable_bias - Enable/Disable bias (in depthwise convolution).
-padding - Pad mode of convolution, 0 means no filling, and 1 means feature calculation filling.
-data_format - Format of input and output data, it's "NCHW" by default.
-quantize - Whether to do quantization(weight and feature). If it is True, the quantization operation will be performed.
-quantize_last_feature - Whether to perform quantization on the last layer (DepthwiseConv2D) of the network or not. If it is set to True, the quantization operation will be performed on the output feature, otherwise the quantization will not be performed on the output feature. If the current layer is not the last layer of the network, then this parameter is considered as invalid.
-last_layer - If it is the final output layer, it must to be set to True.
-weight_bitwidth - The quantization bit of the weight . It is valid when "quantize" is True.
-input_bitwidth - The quantization bit width of the feature. It is valid when "quantize" is True.
-output_bitwidth - The quantization bit width of convolution output feature. It should be set to 32 for the last layer.

-clip_max_value - Clipping threshold of the feature when "quantize" is True. It is recommended to set it to 6.0 for 8bit and 4.0 for 4bit.
-weight_factor - The pruning factor of the weight when "quantize" is True. The smaller the value, the higher the utilization of the quantization bit width, and the less effective weight will be retained.
-target_device - The target hardware device for model operation, we can support T40, Txx, Xs1, Xs2 or T41 now.
-auto_bitwidth - Whether to automatically calculate the bit width when target_device is Txx or Xs1 or Xs2 and the bit width is less than 8bit .
-auto_threshold - Whether to automatically calculate the clipping threshold. There is no use now.
-pre_lstm_layer - If it is the previous layer of LSTM , then set it to True.

Example:

```
dw = ops.DepthwiseConv2D(3,1,
                         padding=1,
                         quantize=True,
                         enable_batch_norm = True,
                         enable_bias = False,
                         activation_fn = ops.ReLU6(),
                         weight_bitwidth = 8,
                         output_bitwidth = 8,
                         clip_max_value = 4.0,
                         weight_factor = 3.0,
                         target_device = "T40")
input = torch.rand((1,3,2,2))
output = dw((input, 8, 0.))
```

Result:

The format of the return data is python tuple type, in which the first element in the tuple is the 4D tensor data of NCHW, which represents the calculation result of DepthwiseConv2D, others are parameters to be passed in quantization training.

Attention:

1. Full precision training is performed when "quantize" is false, and quantization training is performed when "quantize" is true.
2. When the depthwise convolution is the last layer of the network, if quantize is True, and quantize_last_feature is True, Then quantization is performed on both the weight and the feature, Otherwise, only quantization is performed on the weight only.
3. “T02” does not support deep separable convolution.

FullyConnected

Function Description:

The encapsulated full connection operator includes operations such as fully connected, weight quantization, feature quantization, batchnorm, biasadd and relu (relu6) activation. Each operation can be enabled or disabled according to the input parameters..

Interface Definition:

```
FullyConnected(in_channels,
               out_channels,
               activation_fn=ReLU6(),
               enable_batch_norm=False,
               enable_bias=True,
               data_format="NCHW",
               quantize=False,
               quantize_last_feature=False,
               last_layer=False,
               weight_bitwidth=32,
               input_bitwidth=32,
               output_bitwidth=32,
               clip_max_value=6.0,
               weight_factor=3.0,
               target_device="Txx",
               auto_bitwidth=True,
               auto_threshold=False,
               pre_lstm_layer=False)
```

Detailed Explanation of Parameters:

-in_channels - Number of input channels.
-out_channels - Number of output channels.
-activation_fn - Activation function, please directly refer to the activation list behind the operators for details about which activation functions are supported.
-enable_batch_norm - Enable/Disable BatchNorm2d (after full-connected operation)
-enable_bias - Enable/Disable bias (in full-connected).
-data_format - Format of input and output data, it's "NCHW" by default.
-quantize - Whether to do quantization(weight and feature). If it is True, the quantization operation will be performed.
-quantize_last_feature - Whether to perform quantization on the last layer (FullyConnected) of the network or not. If it is set to True, the quantization operation will be performed on the output feature, otherwise the quantization will not be performed on the output feature. If the current layer is not the last layer of the network,

then this parameter is considered as invalid.
-last_layer - If it is the final output layer, it must to be set to True.
-weight_bitwidth - The quantization bit of the weight . It is valid when "quantize" is True.
-input_bitwidth - The quantization bit width of the feature. It is valid when "quantize" is True.
-output_bitwidth - The quantization bit width of convolution output feature. It should be set to 32 for the last layer.
-clip_max_value - Clipping threshold of the feature when "quantize" is True. It is recommended to set it to 6.0 for 8bit and 4.0 for 4bit.
-weight_factor - The pruning factor of the weight when "quantize" is True. The smaller the value, the higher the utilization of the quantization bit width, and the less effective weight will be retained.
-target_device - The target hardware device for model operation, we can support T40, T02, Txx, Xs1, Xs2 or T41 now.
-auto_bitwidth - Whether to automatically calculate the bit width when target_device is Txx or Xs1 or Xs2 and the bit width is less than 8bit .
-auto_threshold - Whether to automatically calculate the clipping threshold. There is no use now.
-pre_lstm_layer - If it is the previous layer of LSTM , then set it to True.

Example:

```
fc=ops.FullConnected(3,10,
                     activation_fn=None,
                     quantize=True,
                     last_layer=True,
                     weight_bitwidth=8,
                     input_bitwidth=8,
                     output_bitwidth=32,
                     target_device="T40")
input = torch.rand((1,3))
output = fc((input, 8, 0.))
```

Result:

If last_layer is False, the format of the return data is python tuple type, in which the first element in the tuple is the 4D tensor data of NCHW, which represents the calculation result of FullyConnected, other elements are parameters to be passed in quantization training.
If last_layer is True, return the floating-point output of tensor type.

Attention:

Pay attention to the conversion of input data dimension. Fullconnected requires that the input data dimension be two-dimensional (latitude adjustment can be made through flatten before it) or three-dimensional (used after LSTM. Note that the

data_format of fullconnected is "NHWC" at this time)

BatchNorm

Function Description:

The Encapsulated batchnorm 2D combination operator, including batchnorm operation and feature quantization. It is mainly aimed at the needs of single batchnorm operation in the network.

Interface Definition:

```
BatchNorm(in_channels,
          activation_fn=Linear(),
          quantize=False,
          first_layer=False,
          input_bitwidth=32,
          output_bitwidth=32,
          clip_max_value=6.0,
          auto_threshold=True,
          target_device="Txx")
```

Detailed Explanation of Parameters:

-in_channels - Number of input channels.
-activation_fn - Activation function, please directly refer to the activation list behind the operators for details about which activation functions are supported.
-quantize - Whether to do quantization(weight and feature). If it is True, the quantization operation will be performed.
-first_layer - If it is the first layer of the network, it must be set to True.
-input_bitwidth - The quantization bit width of the feature. It is valid when "quantize" is True.
-output_bitwidth - The quantization bit width of batchnorm output feature. It should be set to 32 for the last layer.
-clip_max_value - Clipping threshold of the feature when "quantize" is True.
-auto_threshold - Whether to automatically calculate the clipping threshold. There is no use.
-target_device - The target hardware device for model operation, we can support T40, Txx, Xs1, Xs2 or T41 now.

Example:

```
batch=ops.BatchNorm(3,
                    first_layer=False,
                    quantize=True,
                    input_bitwidth=8,
```

```
                    output_bitwidth=8,
                    target_device="T40")
input = torch.rand((1,3,2,2))
output = batch((input, 8, 0.))
```

Result:

The format of the return data is python tuple type, in which the first element in the tuple is the 4D tensor data of NCHW, which represents the calculation result of BatchNorm, others are parameters to be passed in quantization training.

Attention:

1. If there is no activate function after BatchNorm in original network, please set activation_fn as ops Linear () instead of None.
2. If BatchNorm is used as the first layer, then set first_layer as true.

Flatten

Function Description:

Encapsulated reshape operator.

Interface Definition:

Flatten(shape_list, target_device="Txx")

Detailed Explanation of Parameters:

- shape_list - Dimension list.
- target_device - The target hardware device for model operation, we can support T40, T02, Txx, Xs1, Xs2 or T41 now.

Example:

```
flatten = ops.Flatten([-1,48],  target_device=”T40”)
input = torch.rand((2,3,16,1))
output = flatten((input, 8, 0.))
>>>output[0].shape
>>>torch.Size([2, 16])
```

Result:

The format of the return data is python tuple type, in which the first element in the tuple is the 4D tensor data of NCHW, which represents the calculation result of Flatten, others are parameters to be passed in quantization training.

Route

Function Description:

Encapsulated concat operator.

Interface Definition:

Route(target_device = "Txx")

Detailed Explanation of Parameters:

-target_device - The target hardware device for model operation, we can support T40, T02, Txx, Xs1, Xs2 or T41 now.

Example:

```
cat = ops.Route("T40")
input = torch.rand((1,3,2,2))
input2 = torch.rand((1,3,2,2))
output = cat([input, input2])
>>> output[0].shape
torch.Size([1, 6, 2, 2])
```

Result:

The format of the return data is python tuple type, in which the first element in the tuple is the 4D tensor data of NCHW, which represents the calculation result of Route, others are parameters to be passed in quantization training.

Attention:

When target_ Device = "T02" supports up to 8 concats.

Shortcut

Function Description:

Encapsulated shortcut combined operator, includes feature quantization, Activation , Each operation can be enabled or disabled according to the input parameters.

Interface Definition:

```
Shortcut(out_channels,
        activation_fn=ReLU6(),
        quantize=False,
```

```
            quantize_last_feature=False,
            last_layer=False,
            input_bitwidth=32,
            output_bitwidth=32,
            clip_max_value=2.0,
            target_device="Txx")
```

Detailed Explanation of Parameters:

-out_channels - Number of output channels.
-activation_fn - Activation function, please directly refer to the activation list behind the operators for details about which activation functions are supported.
-quantize - Whether to perform quantization operation (quantization weight and feature). If it is set to True, the quantization operation will be performed.
-quantize_last_feature - Whether to perform quantization on the last layer of the network or not. If it is set to True, the quantization operation will be performed on the output feature, otherwise the quantization will not be performed on the output feature. If the current layer is not the last layer of the network, then this parameter is considered as invalid.
-last_layer - If it is the final output layer, last_layer needs to be set to True.
-input_bitwidth - The quantization bit width of the feature. It is valid when "quantize" is set to true.
-output_bitwidth - The quantization bit width of shortcut output feature.
-clip_max_value - The parameter of quantify indicates the clipping threshold of the feature. It is valid when quantify is true, otherwise it is invalid.
-target_device - The target hardware device for model operation, we can support T40, Txx, Xs1, Xs2 or T41 now.

Example:

```
add=ops.Shortcut(3,
                 activation_fn=None,
                 quantize=True,
                 input_bitwidth=8,
                 output_bitwidth=8,
                 target_device="T40")
input = torch.rand((1,3,2,2))
input2 = torch.rand((1,3,2,2))
output = add([(input, 8, 0.), (input2, 8, 0.)])
```

Result:

The format of the return data is python tuple type, in which the first element in the tuple is the 4D tensor data of NCHW, which represents the calculation result of Shortcut, others are parameters to be passed in quantization training.

Attention:

1. Note that the width and height of multiple inputs are consistent.
2. At present, it supports the addition of two inputs at most, and the input bit width should be the same.

Maxpool2D

Function Description:

Encapsulated maxpool operator, in quantitative training, replaces nn.Maxpool2d() operator.

Interface Definition:

```
Maxpool2D(kernel_h=2,
          kernel_w=2,
          stride=2,
          padding=0,
          target_device="Txx")
```

Detailed Explanation of Parameters:

-kernel_h - Height of scaling kernel, default value is 2.
-kernel_w - Width of scaling kernel, default value is 2.
-stride - Stride size, default value is 2.
-padding - Pad mode, 0 means no filling, and 1 means feature calculation filling.
-target_device - The target hardware device for model operation, we can support T40, T02, Txx, Xs1, Xs2 or T41 now.

Example:

```
maxpool = ops.Maxpool2D(target_device="T40")
input = torch.rand((1,3,2,2))
output = maxpool(input)
>>> output.shape
torch.Size([1, 3, 1, 1])
```

Result:

The format of the return data is python tuple type, in which the first element in the tuple is the 4D tensor data of NCHW, which represents the calculation result of Maxpool2D, others are parameters to be passed in quantization training.

Attention:

When target_device = "T02", kernel_h= kernel_w=stirde=2, padding=0.

Avgpool2D

Function Description:

The encapsulated avgpool combination operator plays the same role of avgpool2d. It includes avgpool and feature quantization operations.

Interface Definition:

```
Avgpool2D(in_channels,
          kernel_h=2,
          kernel_w=2,
          stride=2,
          padding=0,
          quantize=False,
          input_bitwidth=32,
          output_bitwidth=32,
          target_device="Txx")
```

Detailed Explanation of Parameters:

-in_channels - Number of input channels.
-kernel_h - Height of scaling kernel, default value is 2.
-kernel_w - Width of scaling kernel, default value is 2.
-stride - Stride size, default value is 2.
-padding - Pad mode, 0 means no filling, and 1 means feature calculation filling.
-quantize - Whether to perform quantization operation (quantization weight and feature). If it is set to True, the quantization operation will be performed.
-input_bitwidth - The quantization bit width of the feature. It is valid when "quantize" is set to true.
-output_bitwidth - The quantization bit width of Avgpool2D output feature.
-target_device - The target hardware device for model operation, we can support T40, Txx, Xs1, Xs2 or T41 now.

Example:

```
avgpool=ops.Avgpool2D(3,
                      quantize=True,
                      input_bitwidth=8,
                      output_bitwidth=8,
                      target_device="T40")
input = torch.rand((1,3,2,2))
output = avgpool((input, 8, 0.))
```

Result:

The format of the return data is python tuple type, in which the first element in the

tuple is the 4D tensor data of NCHW, which represents the calculation result of Avgpool2D, other elements are parameters to be passed in quantization training.

Attention:

Avgpool2d is not supported when target_device = "T02".

AdaptiveAvgpool2D

Function Description:

Encapsulated adaptiveavgpool2D operator, including global average pooling and quantifying feature operations.

Interface Definition:

```
AdaptiveAvgpool2D(in_channels,
                  keepdim=True,
                  quantize=False,
                  input_bitwidth=32,
                  output_bitwidth=32,
                  last_layer=False,
                  target_device="Txx")
```

Detailed Explanation of Parameters:

-in_channels - Number of input channels.
-keepdim - Whether to keep the width and height dimensions of the input feature. The default value is True, if it is False, you will get a two dimensional output.
-quantize - Whether to perform quantization operation (quantization weight and feature). If it is set to True, the quantization operation will be performed.
-input_bitwidth - The quantization bit width of the feature. It is valid when "quantize" is set to true.
-output_bitwidth - The quantization bit width of AdaptiveAvgpool2D output feature.
-target_device - The target hardware device for model operation, we can support T40, T02, Txx, Xs1, Xs2 or T41 now.

Example:

```
avg=ops.AdaptiveAvgpool2D(3,
                          keepdim=True, ##Keep original dimension
                          quantize=True,
                          input_bitwidth=8,
                          output_bitwidth=8,
                          target_device="T40")
input = torch.rand((1,3,2,2))
```

```
output = avg((input, 8, 0.))
>>> output[0].shape
torch.Size([1, 3, 1, 1])

avg=ops.AdaptiveAvgpool2D(3,
                          keepdim=False, ##Do not keep original dimension
                          quantize=True,
                          input_bitwidth=8,
                          output_bitwidth=8,
                          target_device="T40")

output = avg(input)
>>> output[0].shape
torch.Size([1, 3])
```

Result:

The format of the return data is python tuple type, in which the first element in the tuple is the 4D tensor data of NCHW, which represents the calculation result of AdaptiveAvgpool2D, other elements are parameters to be passed in quantization training.

Attention:

1. The output width and height of the global average pool are 1.
2. When target_device = "T02", it only supports adding adaptive average pool at the last layer of the network.

Unpool2D

Function Description:

The encapsulated unpool2d operator supports three filling methods.

Interface Definition:

```
Unpool2D(kernel_h=2,
         kernel_w=2,
         mode="zero",
         quantize=False,
         last_layer=False,
         target_device="Txx")
```

Detailed Explanation of Parameters:

-kernel_h - Height of scaling kernel , default value is 2.

-kernel_w - Width of scaling kernel, default value is 2.
-mode - The filling method for sampling is "zero" by default, which means 0 is filled, "nearest" and "bilinear" are optional.
-quantize - Whether to perform quantization operation (feature). If it is set to True, the quantization operation will be performed.
-last_layer - If it is the final output layer, last_layer must be set to True.
-target_device - The target hardware device for model operation, we can support T40, Txx, Xs1, Xs2 or T41 now.

Example:

```
up = ops.Unpool2D(target_device="T40")
Input = torch.rand((1,3,2,2))
Output = up(input)
>>> output.shape
torch.Size([1, 3, 4, 4])
```

Result:

The format of the return data is python tuple type, in which the first element in the tuple is the 4D tensor data of NCHW, which represents the calculation result of Unpool2D, other elements are parameters to be passed in quantization training.

Se_Block

Function Description:

Encapsulated Se_Block operator, which includes adaptive average pool, convolution, multiply, weight quantization, feature quantization, batchnorm, biasadd and other operations. Each operation can be enabled or disabled according to the input parameters.
You can get the meaning and usage of parameters from the implementation of functions in ops.py.

Interface Definition:

```
Se_Block(in_channels,
         rate,
         quantize=False,
         quantize_last_feature=False,
         last_layer=False,
         enable_bias=True,
         enable_batch_norm=False,
         weight_bitwidth=32,
         input_bitwidth=32,
         output_bitwidth=32,
```

```
                weight_factor=3.0,
                clip_max_value=6.0,
                data_format="NCHW",
                target_device="Txx")
```

Detailed Explanation of Parameters:

-in_channels - Number of input channels.
-rate - Scaling rate of channel.
-quantize - Whether to do quantization(weight and feature). If it is True, the quantization operation will be performed.
-quantize_last_feature - Whether to perform quantization on the last layer of the network or not. If it is set to True, the quantization operation will be performed on the output feature, otherwise the quantization will not be performed on the output feature. If the current layer is not the last layer of the network, then this parameter is considered as invalid.
-last_layer - If it is the final output layer, it must to be set to True.
-enable_bias - Enable/Disable bias (in convolution in se_block).
-enable_batch_norm - Enable/Disable BatchNorm2d (after convolution in se_block).
-weight_bitwidth - The quantization bit of the weight . It is valid when "quantize" is True.
-input_bitwidth - The quantization bit width of the feature. It is valid when "quantize" is True.
-output_bitwidth - The quantization bit width of convolution output feature. It should be set to 32 for the last layer.
-weight_factor - The pruning factor of the weight when "quantize" is True. The smaller the value, the higher the utilization of the quantization bit width, and the less effective weight will be retained.
-clip_max_value - Clipping threshold of the feature when "quantize" is True. It is recommended to set it to 6.0 for 8bit and 4.0 for 4bit.
-data_format - Format of input and output data, it's "NCHW" by default.
-target_device - The target hardware device for model operation, we can support T40, Txx, Xs1, Xs2 or T41 now.

Example:

```
se_block = ops.Se_Block(16,
                        2,
                        quantize=True,
                        weight_bitwidth=8,
                        input_bitwidth=8,
                        output_bitwidth=8,
                        target_device="T40")
input = torch.rand((1,16,2,2))
output = se_block((input, 8, 0.))
```

Result:

The format of the return data is python tuple type, in which the first element in the tuple is the 4D tensor data of NCHW, which represents the calculation result of Se_Block, others are parameters to be passed in quantization training.

Mul

Function Description:

Encapsulated mul operator, including feature quantization operation.

Interface Definition:

```
Mul(out_channels,
      quantize=False,
      quantize_last_feature=False,
      last_layer=False,
      input_bitwidth=32,
      output_bitwidth=32,
      target_device="Txx")
```

Detailed Explanation of Parameters:

-out_channels - Channels of output.
-quantize - Whether to perform quantization operation (quantization weight and feature). If it is set to True, the quantization operation will be performed.
-quantize_last_feature - Whether to perform quantization on the last layer of the network or not. If it is set to True, the quantization operation will be performed on the output feature, otherwise the quantization will not be performed on the output feature. If the current layer is not the last layer of the network, then this parameter is considered as invalid.
-last_layer - If it is the final output layer, last_layer needs to be set to True.
-input_bitwidth - The quantization bit width of the feature. It is valid when "quantize" is set to true.
-target_device - The target hardware device for model operation, we can support T40, Txx, Xs1, Xs2 or T41 now.

Example:

```
mul=ops.Mul(3,
                  quantize=True,
                  input_bitwidth=8,
                  output_bitwidth=8,
                  target_device="T40")
```

```
input = torch.rand((1,16,2,2))
input2 = torch.rand((1,16,2,2))
output = mul([(input, 8, 0.), (input2, 8, 0.)])
```

Result:

The format of the return data is python tuple type, in which the first element in the tuple is the 4D tensor data of NCHW, which represents the calculation result of Mul, others are parameters to be passed in quantization training.

Attention:

At present, the multiplication operation of two input data is supported.

Channel_Split

Function Description:

Encapsulated torch Split operation.

Interface Definition:

Channel_Split(split, target_device="Txx")

Detailed Explanation of Parameters:

-split-channel -- List of split.
-target_device -- The target hardware device for model operation, we can support Txx, Xs1 or Xs2 now.

Example:

```
channel_split = ops.Channel_Split([4,4],
                                  target_device="Txx")
input = torch.rand((1,8,2,2))
output = channel_split(input)
>>> output[0].shape
torch.Size([1, 4, 2, 2])
>>> output[1].shape
torch.Size([1, 4, 2, 2])
```

Result:

The format of the return data is python tuple type, in which the first element in the tuple is the 4D tensor data of NCHW, which represents the calculation result of Channel_Split, others are parameters to be passed in quantization training.

Attention:

1.Parameter split-channel must be list.
2.The number of Txx split channels must be a multiple of 16.
eg:ops.Channel_split([16,48]).

Pixel_Shuffle

Function Description:

Encapsulated up sampling pixel shuffle operator. In quantization training, it can replace nn.PixelShuffle operation.

Interface Definition:

```
Pixel_Shuffle(upscale_factor,
                target_device="T40")
```

Detailed Explanation of Parameters:

-upscale_factor - Scale up factor.
-target_device - The target hardware device of the model only supports T40 or T41.

Example:

```
import torch
pixel_shuffle = ops.Pixel_Shuffle(2)
input = torch.rand((1,128,2,2))
output = pixel_shuffle(input)
>>> output.shape
torch.Size([1, 32, 4, 4])
```

Result:

The format of the return data is python tuple type, in which the first element in the tuple is the 4D tensor data of NCHW, which represents the calculation result of Pixel_Shuffle, others are parameters to be passed in quantization training.

Attention:

Note that the output channel of pixel shuffle must be a multiple of 32.

LSTM

Function Description:

Encapsulated LSTM combination operator; In quantization training, it replaces torch.nn. LSTM operator.

Interface Definition:

```
LSTM(input_size,
        hidden_size,
        num_layers=1,
        bias=True,
        bidirectional=False,
        initial_state_fw=None,
        initial_state_bw=None,
        return_output_states=False,
        batch_first=True,
        use_squeeze=False,
        quantize=False,
        input_bitwidth=32,
        weight_bitwidth=32,
        output_bitwidth=32,
        weight_factor=3.0,
        target_device="Txx")
```

Detailed Explanation of Parameters:

-input_size - Size of input.
-hidden_size - Size of hidden layer .
-num_layers lstm - The number of stacking layers of LSTM, we only support 1 now.
-bias - Enable/Disable bias.
-bidirectional - Bidirectional switch.
-initial_state_fw - Initialize the forward LSTM state, and initially assign all values to the 0 state.
-initial_state_bw - Initialize the reverse LSTM state, and assign all values to the 0 state at the beginning.
-return_output_states - Return to output status.
-batch_first - The input data dimension batch takes precedence. At present, only this method is supported.
-use_squeeze - Remove the flag bit with dimension 1 and dimension change.
-quantize - Whether to perform quantization operation (quantization weight and feature). If it is set to True, the quantization operation will be performed.
-weight_bitwidth - The quantization bit of weight . It is valid when "quantize" is True.
-input_bitwidth - The quantization bit width of the feature. It is valid when "quantize" is true.
-output_bitwidth - The quantization bit width of convolution output feature.
-weight_factor - The parameter of quantify indicates the pruning factor of the weight. The smaller the value, the higher the utilization of the quantization bit width, and the

less effective weight will be retained. It is effective when "quantize" is True, otherwise it is invalid.

-target_device - The target hardware device for model operation, we can support T40, T02, Txx, Xs1, Xs2 or T41 now.

Example:

```
lstm = ops.LSTM(input_size = 16,
                hidden_size=48,
                bias=False,
                use_squeeze=True
                quantize=True,
                input_bitwidth=8,
                weight_bitwidth=8,
                output_bitwidth=8).cuda()
input = torch.rand((1,16,1,48)).cuda()
output = lstm(input)
>>> output[0][0].shape
torch.Size([1, 48, 48])
```

Result:

The format of the return data is python tuple type, in which the first element in the tuple is the 4D tensor data of NCHW, which represents the calculation result of LSTM, others are parameters to be passed in quantization training.

Attention:

The convolution, full connection or other operator in front of LSTM need to set parameter “pre_lstm_layer” to True.

Embedding

Function Description:

Encapsulated embedding operator is used to replace nn.Embedding operator.

Interface Definition:

```
Embedding(num_embeddings,
          embedding_dim,
          padding_idx=None,
          first_layer=False,
          quantize=False,
          input_bitwidth=32,
          output_bitwidth=32,
          multi_inputs=True,
```

```
                target_device="Txx",
                pre_lstm_layer=False)
```

Detailed Explanation of Parameters:

-num_embeddings - Number of words in the dictionary.
-embedding_dim - Word vector dimension.
-padding_idx - Populated index value.
-first_layer - If it is the first layer, this must be set to True.
-quantize - Whether to perform quantization operation (quantization weight and feature). If it is set to true, the quantization operation will be performed..
-input_bitwidth - Quantization bit width of input feature.
-output_bitwidth - Quantization bit width of the output feature.
-multi_inputs - Identifies whether there are 2 inputs. The default value is true.
-target_device - The target hardware equipment for model operation currently has five options: T40, Txx, Xs1, Xs2 or T41.
-pre_lstm_layer - If it is the previous layer of LSTM operator, set it to True.

Example:

```
embedding = ops.Embedding(num_embeddings=100,
                          embedding_dim=128,
                          padding_idx=2,
                          quantize=True,
                          input_bitwidth=8,
                          output_bitwidth=8,
                          multi_inputs=False,
                          target_device="T40").cuda()
input = (torch.rand(2,4)*5).cuda()
output = embedding(input)
>>>output.shape
torch.Size([2, 4, 128])
```

Result:

The format of the return data is python tuple type, in which the first element in the tuple is the 4D tensor data of NCHW, which represents the calculation result of Embedding, others are parameters to be passed in quantization training.

Attention:

Embedding input parameter multi_inputs defaults is True. If there are not multiple inputs, set multi_inputs to False.

Focus

Function Description:

Encapsulated focus operator is used to replace the focus operation in the yolov5 series.

Interface Definition:

Focus(target_device="Txx")

Detailed Explanation of Parameters:

-target_device - The target hardware device for model operation, we can support T40, Txx, Xs1, Xs2 or T41 now.

Example:

```
focus = ops.Focus(target_device="T40")
input = torch.rand((1,3,16,16))
output = focus(input)
>>> output.shape
torch.Size([1, 12, 8, 8])
```

Result:

The format of the return data is python tuple type, in which the first element in the tuple is the 4D tensor data of NCHW, which represents the calculation result of Focus, others are parameters to be passed in quantization training.

Attention:

1. The focus operator only supports the use before the first layer convolution.
2. Focus is currently encapsulated in conv2d and used through is_ Focus is on or off, and cannot be used alone.

Max

Function Description:

Returns the largest of the two input features to replace torch Max() operation.

Interface Definition:

Max(target_device="T40")

Detailed Explanation of Parameters:

-target_device - There are only T40 or T41 devices for the target hardware devices

of the model.

Example:

```
Input = torch.rand((1,3,2,2))
input2 = torch.rand((1,3,2,2))
output = ops.Max([input,input2])
max = ops.Max(target_device="T40")
output = max([(input, 8, 0.), (input2, 8, 0.)])
```

Result:

The format of the return data is python tuple type, in which the first element in the tuple is the 4D tensor data of NCHW, which represents the calculation result of Max, others are parameters to be passed in quantization training.

Activation Lists

Attention: As activation_fn the parameter is passed in for use and is not called separately.

ReLU6

Function Description:

Encapsulated ReLU6 function.

Interface Definition:

ReLU6(target_device = "Txx")

Result:

Result of activation function.

ReLU

Function Description:

Encapsulated Relu function.

Interface Definition:

ReLU(target_device = "Txx")

Result:

Result of activation function.

Linear

Function Description:

Do nothing,.

Interface Definition:

Linear(target_device = "Txx")

Result:

Return the input value without any operation.

PReLU

Function Description:

Encapsulated Prelu activation function.

Interface Definition:

PReLU(in_channels, target_device="Txx")

Detailed Explanation of Parameters:

-in_channels - input channels.

Result:

Result of activation function.

Hsigmoid

Function Description:

The Encapsulated hsigmoid operator. The Hsigmoid activation function is the approximation of sigmoid, which should be fitted as much as possible.

Interface Definition:

```
Hsigmoid(in_channels,
        offset_value = 3.0,
        clip_max_value = 6.0,
        target_device = "Txx")
```

Result:

Result of activation function.

Hswish

Function Description:

Encapsulated hswish operator.

Interface Definition:

```
Hswish(in_channels,
      target_device = "Txx")
```

Result:

Result of activation function.

LeakyReLU

Function Description:

Encapsulated LeakyReLU operator.

Interface Definition:

```
LeakyReLU(target_device = "Txx")
```

Result:

Result of activation function.

II. Network Example

The following is an example of our open source T40 classification network, the file is in magik-toolkit/TrainingKit/pytorch/sample/network.py.

```
class ConvBlock_T40(nn.Module):
    def __init__(self, in_channels, out_channels, kernel_size, stride, padding,
```

```
                    is_quantize, first_layer, input_bitwidth, BITW, BITA, auto=True):
        super(ConvBlock_T40, self).__init__()
        self.conv = ops.Conv2D(in_channels, out_channels,
                               stride=stride,
                               kernel_h=kernel_size,
                               kernel_w=kernel_size,
                               activation_fn=None,
                               enable_batch_norm=True,
                               enable_bias=False,
                               quantize=is_quantize,
                               padding=padding,
                               first_layer=first_layer,
                               weight_bitwidth = BITW,
                               input_bitwidth=input_bitwidth,
                               output_bitwidth=BITA,
                               clip_max_value = 6.0,
                               target_device="T40",
                               auto_bitwidth=False)
    def forward(self, input):
        out = self.conv(input)
        return out

class Network_T40(nn.Module):
    def __init__(self, is_quantize=False, BITA = 32, BITW = 32):
        super(Network_T40, self).__init__()
        self.preprocess = ops.Preprocess(0, 255., target_device="T40")
        #Call preprocess in front of the network.

        self.bn = ops.BatchNorm(3, None, is_quantize, first_layer=True, 8, BITA,
                                target_device="T40")
        #It's the first layer of the network,  please set first_layer=True.

        self.conv0 = ConvBlock_T40(3, 32, 3, 1, 1, is_quantize, False, 8, BITW, BITA)
        self.maxpool = ops.Maxpool2D(target_device="T40")
        self.conv1 = ConvBlock_T40(32, 32, 3, 1, 1, is_quantize, False, BITA, BITW,
BITA)
        self.route = ops.Route(target_device="T40")
        self.conv2 = ConvBlock_T40(64, 32, 3, 2, 1, is_quantize, False, BITA, BITW,
BITA)
        self.conv3 = ConvBlock_T40(32, 64, 3, 1, 1, is_quantize, False, BITA, BITW,
BITA)
        self.unpool=ops.Unpool2D(2,2,target_device="T40",quantize=is_quantize)
        self.conv4 = ConvBlock_T40(64, 64, 3, 1, 1, is_quantize, False, BITA, BITW,
BITA)
```

```
self.conv5 = ConvBlock_T40(64, 64, 3, 1, 1, is_quantize, False, BITA, BITW, BITA)
self.shortcut= ops.Shortcut(64, quantize=is_quantize, input_bitwidth=BITA, output_bitwidth = BITA, clip_max_value = 2.0, target_device="T40")
#You cannot use “+” or “torch.add” in the forward of the network, and you need to use our encapsulated shortcut.
self.bn1 = ops.BatchNorm(64, ops.PReLU(64), is_quantize, False, BITA, BITA, target_device="T40")
#A single batchnorm operator that calls PReLU activation is given.
self.conv6 = ConvBlock_T40(64, 128, 3, 2, 1, is_quantize, False, BITA, BITW, BITA)
self.conv7 = ConvBlock_T40(128, 128, 3, 1, 1, is_quantize, False, BITA, BITW, BITA)
self.conv8 = ConvBlock_T40(128, 256, 3, 2, 1, is_quantize, False, BITA, BITW, BITA)
self.conv9 = ConvBlock_T40(256, 256, 3, 1, 1, is_quantize, False, BITA, BITW, BITA)
self.conv10 = ConvBlock_T40(256, 256, 3, 2, 1, is_quantize, False, BITA, BITW, BITA)
self.rnn=ops.LSTM(256, 256, bidirectional=True, batch_first=True,
                  use_squeeze=True, quantize=is_quantize,
                  input_bitwidth=BITA, output_bitwidth=BITA,
                  weight_bitwidth=BITW, target_device="T40")
#Please set use_squeeze to true , so you do not need to worry about the dimension transformation of LSTM input feature.

self.flatten = ops.Flatten([-1, 256*2], target_device="T40")
#Use ops.Flatten replacement torch.reshape, shape_list size settings as needed, -1 is used for batch uncertainty, remaining dimensions must be given definite values.
self.fc1 = ops.FullConnected(256*2, 128, enable_batch_norm=True,
                             quantize=is_quantize, activation_fn=None,
                             last_layer=False, weight_bitwidth = BITW,
                             input_bitwidth=BITA, output_bitwidth=BITA,
                             clip_max_value=6.0, target_device="T40")
self.fc2 = ops.FullConnected(128, 10, enable_bias=True, quantize=is_quantize,
                             activation_fn=None, last_layer=True,
                             weight_bitwidth = BITW, input_bitwidth=BITA,
                             output_bitwidth=32, clip_max_value=6.0,
                             target_device="T40")
#The last layer, pay attention to the setting of last_layer=True and output_ bitwidth = 32. If there is activation later, it is recommended to do it alone after this layer. At this time, you can call the activation operator of the original torch since it is floating-point output.
self.sigmoid = torch.nn.Sigmoid()
```

```
def forward(self, input):
    pinput = self.preprocess(input)
    pbn = self.bn(pinput)
    conv0 = self.conv0(pbn)
    max1 = self.maxpool(conv0)
    conv1 = self.conv1(max1)
    concat1 = self.route([max1, conv1])
    conv2 = self.conv2(concat1)
    conv3 = self.conv3(conv2)
    unpool1 = self.unpool(conv3)
    conv4 = self.conv4(unpool1)
    max2 = self.maxpool(conv4)
    conv5 = self.conv5(max2)
    concat2 = self.shortcut([max2, conv5])
    concat2 = self.bn1(concat2)
    conv6 = self.conv6(concat2)
    conv7 = self.conv7(conv6)
    conv8 = self.conv8(conv7)
    conv9 = self.conv9(conv8)
    conv10 = self.conv10(conv9)
    rnn, _ = self.rnn(conv10)
    res3 = self.flatten(rnn)
    fc1 = self.fc1(res3)
    fc2 = self.fc2(fc1)
    fc = self.sigmoid(fc2)
    return fc
```