君正®

Pytorch 训练量化算子详解

Date: Feb. 2022

君正®

Pytorch训练量化算子详解

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 rightsis granted. Ingenic assumes no liability, provides no warranty either expressed or implied relatingto 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 orsupporting equipment.

All information in this document should be treated as preliminary. Ingenic may make changes tothis document without notice. Anyone relying on this documentation should contact Ingenic forthe 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

目录

算子列表

算子接口均在 ops.py 中，根据我们提供的训练插件使用指南安装完的训练插件 whl 之后在当前环境中会多一个 ingenic_magik_trainingkit(同 torch 目录)，ops.py 在 ingenic_magik_trainingkitQuantizationTrainingPlugin/python 下。

Preprocess

功能描述：

标识网络输入数据的预处理均值和方差参数。方便后期模型的转化。

接口定义：

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

参数详解：

-mean -数据预处理的均值，默认值 0。
-var -数据预处理的方差，默认值 255。
-target_device -目标设备，模型运行的目标硬件设备，目前有 T40、T02、Txx、Xs1、Xs2 和 T41 六个选项。

例子：

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

返回值：

该算子并未做实际运算，传入的均值和方差为前面预处理用过的实际值，返回数据格式为 python 元组数据类型，其中第一个元素为输入的 tensor，第二个是输入的 bit 位，固定为 8, 第三个和第四个是预处理用到的均值和方差，最后一个是最小值，若均值为 0 该值为 0，否则为-1。

注意事项：

1. 数据输入网络之前，必须经过 ops.Preprocess 的处理，否则将会报错。
2. 传入的均值和方差必须为前面预处理用过的实际值，否则转换出来的模型会有问题。

Conv2d

功能描述：

封装的卷积算子，包含卷积、权重量化、Feature 量化、BatchNorm、BiasAdd 和 Relu(Relu6)激活等操作，每个操作可根据输入的参数使能或者关闭。

接口定义：

```
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)
```

参数详解：

-in_channels - 卷积的输入通道数。
-out_channels - 卷积的输出通道数。
-stride - 卷积步长，默认为 1。
-kernel_h - 卷积核高的尺寸，默认为 3。
-kernel_w - 卷积核宽的尺寸，默认为 3。
-enable_batch_norm - 卷积计算后是否进行 BatchNorm2d 操作，如果使用 BatchNorm2d，设置为 True。

-enable_bias - 是否使能 BiasAdd 操作，若为 True 则在卷积后添加 BiasAdd 操作。
-activation_fn - 最终输出经过的激活函数，目前支持的激活函数：ops.ReLU()，ops.ReLU6()，ops.Linear()，ops.LeakyReLu()，ops.PReLu()，ops.Hsigmoid()。
-padding - 卷积的 Pad 方式，0 表示不填充，1 表示 Feature 计算填充。
-groups - group 参数的作用是控制分组卷积，默认不分组，为 1 组。
-dilation - 空洞卷积的参数，默认为 1。
-quantize - 是否执行量化操作(量化权重和 Feature)，设为 True 则执行量化操作。
-quantize_last_feature - 当该卷积为网络最后一层时，设为 True 则对输出 Feature 执行量化操作，否则不对输出 Feature 执行量化，当该卷积不是网络最后一层时该值无效。
-first_layer - 第一层标识，如果为整个网络输入的第一个卷积，first_layer 需要设置为 True。
-last_layer - 最后一层标识，如果为最终输出层，last_layer 需要设置为 Ture。
-weight_bitwidth - 量化权重位宽，当 quantize 为 True 时有效，否则无效。
-input_bitwidth 输入 Feature 的量化位宽，当 quantize 为 True 时有效，否则无效。
-output_bitwidth - 卷积输出 Feature 的量化位宽。
-clip_max_value - quantize 的参数，表示 Feature 的裁剪阈值，当 quantize 为 True 时有效，否则无效，建议训练 8bit 设置成 6.0，训练 4bit 设置成 4.0。
-weight_factor - quantize 的参数，表示对权重的剪枝因子，该值越小对量化位宽的利用率越高，同时保留的有效权重会越少，当 quantize 为 True 时有效，否则无效。
-target_device - 目标设备，模型运行的目标硬件设备，目前有 T40、T02、Txx、Xs1、Xs2 和 T41 六个选项。
-auto_bitwidth - 是否自动计算位宽，target_device 为 Txx 或 Xs1 或 Xs2 并且位宽在 8bit 以下且该值为 True 时有效。
-is_focus - 如果卷积前存在 focus 操作，设置为 Ture，默认为 False。
-auto_threshold - 是否自动计算裁剪阈值，为 True 时有效。
-pre_lstm_layer - 如果为 lstm 算子的前一层，请设置为 Ture。

例子：

```
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.))
```

返回值：

返回数据格式为 python 元组数据类型，其中元组中的第一个元素，为 NCHW 的 4D tensor 数据，表示卷积的计算结果。其他元素为量化训练中需要传递的参数。

注意事项：

1. 当 quantize 为 False 时执行全精度训练，当 quantize 为 True 时执行量化训练。
2. 当该卷积为网络最后一层时，如果 quantize 为 True，且 quantize_last_feature 为 True，则对权重和 Feature 均执行量化操作，否则只对权重执行量化操作。如果为最后一层，并且没有激活函数，请将 activation_fn 设置为 None。
3. 若 input_channels==out_channels==groups 即使用分组卷积，调用专门的分组卷积算子 ops.DepthwiseConv2D()。

ConvTranspose2D

功能描述：

封装的组合反卷积算子，包含深度卷积、点卷积、权重量化、Feature 量化、BatchNorm、BiasAdd 和 Relu(Relu6)激活等操作，每个操作可根据输入的参数使能或者关闭。

接口定义：

```
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)
```

参数详解：

-in_channels - 卷积的输入通道数。
-out_channels - 卷积的输出通道数。
-enable_batch_norm - 卷积计算后是否进行 BatchNorm2d 操作，如果使用 BatchNorm2d，请设置为 True。
-enable_bias - 是否使能 BiasAdd 操作，若为 True 则在卷积后添加 BiasAdd 操作。
-activation_fn - 最终输出经过的激活函数，目前支持的激活函数：ops.ReLU(),ops.ReLU6(),ops.Linear(),ops.LeakyReLu(),ops.PreLu(),ops.Hsigmoid()。
-data_format - 输入输出数据的格式，默认“NCHW”。
-quantize - 是否执行量化操作(量化权重和 Feature)，设为 True 则执行量化操作。
-quantize_last_feature - 当该卷积为网络最后一层时，设为 True 则对输出 Feature 执行量化操作，否则不对输出 Feature 执行量化，当该卷积不是网络最后一层时该值无效。
-last_layer - 最后一层标识，如果为最终输出层，last_layer 需要设置为 Ture。
-weight_bitwidth - 量化权重位宽，当 quantize 为 True 时有效，否则无效。
-input_bitwidth - 输入 Feature 的量化位宽，当 quantize 为 True 时有效，否则无效。
-output_bitwidth - 卷积输出 Feature 的量化位宽。
-clip_max_value - quantize 的参数，表示 Feature 的裁剪阈值，当 quantize 为 True 时有效，否则无效，建议训练 8bit 设置成 6.0，训练 4bit 设置成 4.0。
-weight_factor - quantize 的参数，表示对权重的剪枝因子，该值越小对量化位宽的利用率越高，同时保留的有效权重会越少，当 quantize 为 True 时有效，否则无效。
-target_device - 目标设备，模型运行的目标硬件设备，目前有 T40、T02、Txx、Xs1、Xs2 和 T41 六个选项。
-auto_bitwidth - 是否自动计算位宽，target_device 为 Txx 或 Xs1 或 Xs2 并且位宽在 8bit 以下且该值为 True 时有效。
-auto_threshold - 是否自动计算裁剪阈值，为 True 时有效。
-pre_lstm_layer - 如果为 lstm 算子的前一层，设置为 Ture。

例子：

```
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.))
```

返回值：

返回数据格式为 python 元组数据类型，其中元组中的第一个元素，为 NCHW 的 4D tensor 数据，表示组合反卷积的计算结果。其他元素为量化训练中需要传递的参数。

注意事项：

该算子为拟合的反卷积算子，Txx、Xs1、Xs2 时为 unpool+depthwise+conv，T40、T41 时为 unpool+conv，可以代替 torch.nn.ConvTranspose2d 操作。

DepthwiseConv2D

功能描述

封装的深度可分离卷积算子，包含 depthwise 卷积、权重量化、Feature 量化、BatchNorm、BiasAdd 和 Relu(Relu6)激活等操作，每个操作可根据输入的参数使能或者关闭。

接口定义：

```
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)
```

参数详解：

-in_channels - 卷积的输入通道数。
-stride - 卷积步长，默认为 1。
-kernel_h - 卷积核高的尺寸，默认为 3。
-kernel_w - 卷积核宽的尺寸，默认为 3。
-enable_batch_norm - 卷积计算后是否进行 BatchNorm2d 操作，如果使用 BatchNorm2d，请设置为 True。
-enable_bias - 是否使能 BiasAdd 操作，若为 True 则在卷积后添加 BiasAdd 操作。
-activation_fn - 最终输出经过的激活函数。
-padding - 卷积的 Pad 方式，等于 0 表示不填充，等于 1 表示 Feature 计算填充。
-quantize - 是否执行量化操作(量化权重和 Feature)，设为 True 则执行量化操作。
-quantize_last_feature - 当该卷积为网络最后一层时，设为 True 则对输出 Feature 执行量化操作，否则不对输出 Feature 执行量化，当该卷积不是网络最后一层时该值无效。
-last_layer - 最后一层标识，如果为最终输出层，last_layer 需要设置为 Ture。
-weight_bitwidth - 量化权重位宽，当 quantize 为 True 时有效，否则无效。
input_bitwidth 输入 Feature 的量化位宽，当 quantize 为 True 时有效，否则无效。
-output_bitwidth - 卷积输出 Feature 的量化位宽。
-clip_max_value -quantize 的参数，表示 Feature 的裁剪阈值，当 quantize 为 True 时有效，否则无效，建议训练 8bit 设置成 6.0，训练 4bit 设置成 3.0。
-weight_factor - quantize 的参数，表示对权重的剪枝因子，该值越小对量化位宽的利用率越高，同时保留的有效权重会越少，当 quantize 为 True 时有效，否则无效。
-target_device - 目标设备，模型运行的目标硬件设备，目前有 T40、T41、Txx、Xs1 和 Xs2 等五个选项。
-auto_bitwidth - 是否自动计算位宽，target_device 为 Txx 或 Xs1 或 Xs2 并且位宽在 8bit 以下且该值为 True 时有效。
-auto_threshold - 是否自动计算裁剪阈值，为 True 时有效。
-pre_lstm_layer - 如果为 lstm 算子的前一层，请设置为 Ture。

例子：

```
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.))
```

返回值：

返回数据格式为 python 元组数据类型，其中元组中的第一个元素，为 NCHW 的 4D tensor 数据，表示深度可分离卷积的计算结果。其他元素为量化训练中需要传递的参数。

注意事项：

1.当 quantize 为 False 时执行全精度训练，当 quantize 为 True 时执行量化训练。
2.当该卷积为网络最后一层时，如果 quantize 为 True，且 quantize_last_feature 为 True，则对权重和 Feature 均执行量化操作，否则只对权重执行量化操作。
3. T02 不支持深度可分离卷积。

FullyConnected

功能描述：

封装的组合全连接算子，包含 FullyConnected、权重量化、Feature 量化、BatchNorm、BiasAdd 和 Relu(Relu6)激活等操作，每个操作可根据输入的参数使能或者关闭。

接口定义：

```
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)
```

参数详解：

-in_channels - 卷积的输入通道数
-out_channels - 卷积的输出通道数
-enable_batch_norm - 卷积计算后是否进行 BatchNorm2d 操作，如果使用 BatchNorm2d，请设置为 True。
-enable_bias - 是否使能 BiasAdd 操作，若为 True 则在卷积后添加 BiasAdd 操作。
-activation_fn - 最终输出经过的激活函数
-quantize - 是否执行量化操作(量化权重和 Feature)，设为 True 则执行量化操作。
-quantize_last_feature - 当该卷积为网络最后一层时，设为 True 则对输出 Feature 执行量化操作，否则不对输出 Feature 执行量化，当该卷积不是网络最后一层时该值无效。
-first_layer - 第一层标识，如果为整个网络输入的第一个卷积，first_layer 需要设置为 True。
-last_layer - 最后一层标识，如果为最终输出层，last_layer 需要设置为 Ture。
-weight_bitwidth - 量化权重位宽，当 quantize 为 True 时有效，否则无效。
-input_bitwidth - 输入 Feature 的量化位宽，当 quantize 为 True 时有效，否则无效。
-output_bitwidth - 卷积输出 Feature 的量化位宽。
-clip_max_value - quantize 的参数，表示 Feature 的裁剪阈值，当 quantize 为 True 时有效，否则无效，建议训练 8bit 设置成 6.0，训练 4bit 设置成 3.0。
-weight_factor - quantize 的参数，表示对权重的剪枝因子，该值越小对量化位宽的利用率越高，同时保留的有效权重会越少，当 quantize 为 True 时有效，否则无效。
-target_device - 目标设备，模型运行的目标硬件设备，目前有 T40、T02、Txx、Xs1、Xs2、T41 六个选项。
-auto_bitwidth - 是否自动计算位宽，target_device 为 Txx 或 Xs1 或 Xs2 并且位宽在 8bit 以下且该值为 True 时有效。
-auto_threshold - 是否自动计算裁剪阈值，为 True 时有效。
-pre_lstm_layer - 如果为 lstm 算子的前一层，请设置为 Ture。

例子：

```
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.))
```

返回值：

如果 last_layer=False,返回数据格式为 python 元组数据类型，其中元组中的第一个元素，为 NCHW 的 4D tensor 数据，表示全连接的计算结果。其他元素为量化训练中需要传递的参数。
如果 last_layer=True，直接返回 tensor 类型的浮点输出。

注意事项：

1.注意输入数据维度转化， FullConnected 要求输入数据维度为二维（之前可通过 Flatten 做纬度调整）或者三维（LSTM 之后使用，注意此时将 FullConnected 的 data_format 为“NHWC”）。
2.最后一层时，如果 quantize 为 True，建议不要将 quantize_last_feature 为 True，否则对权重和 Feature 均执行量化操作。

BatchNorm

功能描述：

封装的 BatchNorm2d 组合算子，包含 BatchNorm 操作和 Feature 量化，
，主要针对网络中单个使用 Batchnorm 操作的需求。

接口定义：

```
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")
```

参数详解：

-in_channels - 卷积的输入通道数。
-activation_fn - 输出经过的激活函数，默认为 ops.Linear()（无激活函数)。
-quantize - 是否执行量化操作(量化权重和 Feature)，设为 True 则执行量化操作。
-first_layer - 第一层标识，如果为整个网络输入的第一个卷积，first_layer 需要设置为 True。
-input_bitwidth - 输入 Feature 的量化位宽，当 quantize 为 True 时有效， 否则

无效。
-output_bitwidth - 卷积输出 Feature 的量化位宽。
-clip_max_value - quantize 的参数，表示 Feature 的裁剪阈值，当 quantize 为 True 时有效，否则无效，建议训练 8bit 设置成 6.0，训练 4bit 设置成 3.0。
-target_device - 目标设备，模型运行的目标硬件设备，目前有 T40、T02、Txx、Xs1、Xs2、T41 六个选项。

例子：

```
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.))
```

返回值：

返回数据格式为 python 元组数据类型，其中元组中的第一个元素，为 NCHW 的 4D tensor 数据，表示 BatchNorm 的计算结果。其他元素为量化训练中需要传递的参数。

注意事项：

1. 若 BatchNorm 之后无激活，注意设置 activation_fn 为 ops.Linear()，而不是 None。
2. 若 BatchNorm 作为第一层使用，注意将 first_layer 置为 True。

Flatten

功能描述：

封装的 reshape 算子，在 reshape 操作前后添加节点标识，以便为后期转换。

接口定义：

```
Flatten(shape_list, target_device="Txx")
```

参数列表：

- shape_list - reshape 的维度。
- target_device - 模型运行的目标硬件设备，目前有 Txx、T02、T40、Xs1、Xs2、T41 六个选项。

例子：

```
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])
```

返回值：

返回数据格式为 python 元组数据类型，其中元组中的第一个元素，为 NCHW 的 4D tensor 数据，表示 reshape 之后的结果。其他元素为量化训练中需要传递的参数。

Route

功能描述：

封装的 concat 算子，在 concat 操作前后添加节点标识，以便为后期转换。

接口定义：

```
Route(target_device = "Txx")
```

参数详解：

-target_device - 模型运行的目标硬件设备，目前有 T40、T02、Txx、Xs1、Xs2、T41 六个选项。

例子：

```
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])
```

返回值

返回数据格式为 python 元组数据类型，其中元组中的第一个元素，为 NCHW 的 4D tensor 数据，表示 concat 之后的结果。其他元素为量化训练中需要传递的参数。

注意事项

当为 target_device = "T02"时最多支持 8 路 concat。

Shortcut

功能描述：

封装的 shortcut 组合算子，Feature 量化、激活等操作，每个操作可根据输入的参数使能或者关闭。

接口定义：

```
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")
```

参数详解：

-out_channels - 输出 Feature 的通道数。
-activation_fn - 输出经过的激活函数。
-quantize - 是否执行量化操作(量化权重和 Feature)，设为 True 则执行量化操作。
-quantize_last_feature - 当该卷积为网络最后一层时，设为 True 则对输出 Feature 执行量化操作，否则不对输出 Feature 执行量化，当该卷积不是网络最后一层时该值无效。
-last_layer - 表示当前层是否为网络的最后一层（对于量化的网络只有最后一层可以选择是否只量化权重，不量化 Feature，其他层的权重和 Feature 均需量化）。
-input_bitwidth - 输入 Feature 的量化位宽，当 quantize 为 True 时有效， 否则无效。
-output_bitwidth - 输出 Feature 的量化位宽。
-clip_max_value - quantize 的参数，表示 Feature 的裁剪阈值，当 quantize 为 True 时有效，否则无效，训练 8bit 设置成 2.0，4bit 精度差的时候可以适当调低至 1.5。
-target_device - 模型运行的目标硬件设备，目前有 Txx、T40、Xs1 和 Xs2 四个选项。

例子：

```
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.)])
```

返回值

返回数据格式为 python 元组数据类型，其中元组中的第一个元素，为 NCHW 的 4D tensor 数据，表示 shortcut 的计算结果。其他元素为量化训练中需要传递的参数。

注意事项

1. 注意 shortcut 的多个输入的 feature 的宽高维度一致。
2. 目前最多支持两个输入相加，且输入位宽要一致。

Maxpool2D

功能描述：

封装的 maxpool 算子，量化训练中，替换 nn.Maxpool2d 算子

接口定义：

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

参数详解：

-kernel_h - 核的高。
-kernel_w - 核的宽。
-stride - 最大池化的步长。
-padding - 最大池化的 Pad 方式，等于 0 表示不填充，等于 1 表示 Feature 计算填充。
-target_device 模型运行的目标硬件设备，目前有 T40、T02、Txx、Xs1 和 Xs2、T41 六个选项。

例子：

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

返回值：

返回数据格式为 python 元组数据类型，其中元组中的第一个元素，为 NCHW 的 4D tensor 数据，表示 maxpool 的计算结果。其他元素为量化训练中需要传递的参数。

注意事项：

当 target_device = "T02"时，kernel_h= kernel_w=stirde=2，padding=0。

Avgpool2D

功能描述：

封装的 avgpool 组合算子，量化训练中，起同等替换 nn.AvgPool2d 的作用。其中包含 avgpool，Feature 量化操作。

接口定义：

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

参数详解：

-in_channels - 输入 Feature 的通道数。
-kernel_h - 核的高。
-kernel_w - 核的宽。
-stride - 平均池化的 stride。
-padding - 平均池化的 Pad 方式，等于 0 表示不填充，等于 1 表示 Feature 计算填充。
-quantize -是否执行量化操作(量化权重和 Feature)，设为 True 则执行量化操作。
-input_bitwidth 输入 Feature 的量化位宽，当 quantize 为 True 时有效， 否则无效。
-output_bitwidth 输出 Feature 的量化位宽。
-target_device 模型运行的目标硬件设备，目前有 Txx、T40、Xs1 和 Xs2、T41 五个选项。

例子：

```
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.))
```

返回值：

返回数据格式为 python 元组数据类型，其中元组中的第一个元素，为 NCHW 的 4D tensor 数据，表示 avgpool2d 的计算结果。其他元素为量化训练中需要传递的参数。

注意事项：

1.当 target_device = "T02"时，不支持 avgpool2d。

AdaptiveAvgpool2D

功能描述：

封装的 adaptiveavgpool 组合算子，包含全局平均池化，量化 feature 操作。

接口定义：

```
AdapyiveAvgpool2D(in_channels,
                  keepdim=True,
                  quantize=False,
                  input_bitwidth=32,
                  output_bitwidth=32,
                  last_layer=False,
                  target_device="Txx")
```

参数详解：

-in_channels - 输入 Feature 的通道数。
-keepdim - 是否保留输入 feature 的宽高维度，默认为 True。
-quantize - 是否执行量化操作(量化权重和 Feature)，设为 True 则执行量化操作。
-input_bitwidth - 输入 Feature 的量化位宽，当 quantize 为 True 时有效， 否则无效。
-output_bitwidth - 输出 Feature 的量化位宽。
-target_device - 模型运行的目标硬件设备，目前有 T02、Txx、T40、Xs1 和 Xs2、T41 六个选项。

例子：

```
avg=ops.AdaptiveAvgpool2D(3,
                          keepdim=True, ##保留原始维度
```

```
                                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, ##保留原始维度
                                quantize=True,
                                input_bitwidth=8,
                                output_bitwidth=8,
                                target_device="T40")

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

返回值：

返回数据格式为 python 元组数据类型，其中元组中的第一个元素，为 NCHW 的 4D tensor 数据，表示 adaptiveavgpool 的计算结果。其他元素为量化训练中需要传递的参数。

注意事项：

1. 全局平均池化的输出宽、高均为 1。
2. 当 target_device = "T02"时，只支持在网络的最后一层添加 adaptiveavgpool。

Unpool2D

功能描述：

封装的 unpool2d 算子，支持三种填充方式。

接口定义：

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

参数详解：

-kernel_h - 核的高。
-kernel_w - 核的宽。
-mode - 上采样的填充方式，默认“zero”，表示填 0，可选“nearest”，采用最近邻方式填充，“bilinear”，采用线性插值。
-quantize - 是否执行量化操作(量化权重和 Feature)，设为 True 则执行量化操作。
-last_layer - 是否为最后一层。
-target_device - 模型运行的目标硬件设备，目前有 T40、T02、Txx、Xs1 和 Xs2、T41 六个选项。

例子：

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

返回值：

返回数据格式为 python 元组数据类型，其中元组中的第一个元素，为 NCHW 的 4D tensor 数据，表示 unpool 的计算结果。其他元素为量化训练中需要传递的参数。

Se_Block

功能描述：

封装的组合 Se_Block 算子。权重量化、Feature 量化、BatchNorm、BiasAdd 等操作，每个操作可根据输入的参数使能或者关闭。

接口定义：

```
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")

参数详解：

-in_channels - 卷积的输入通道数。
-rate - 通道缩放比例。
-enable_batch_norm - 是否使能 batchnorm 操作。
-enable_bias - 是否使能 BiasAdd 操作，若为 True 则在卷积后添加 BiasAdd 操作。
-quantize - 是否执行量化操作(量化权重和 Feature)，设为 True 则执行量化操作。
-quantize_last_feature - 当该卷积为网络最后一层时，设为 True 则对输出 Feature 执行量化操作，否则不对输出 Feature 执行量化，当该卷积不是网络最后一层时该值无效。
-last_layer - 最后一层标识，如果为最终输出层，last_layer 需要设置为 Ture。
-weight_bitwidth - 量化权重位宽，当 quantize 为 True 时有效，否则无效。
-input_bitwidth - 输入 Feature 的量化位宽，当 quantize 为 True 时有效，否则无效。
-output_bitwidth - 卷积输出 Feature 的量化位宽。
-clip_max_value - quantize 的参数，表示 Feature 的裁剪阈值，当 quantize 为 True 时有效，否则无效，建议训练 8bit 设置成 6.0，训练 4bit 设置成 3.0。
-weight_factor - quantize 的参数，表示对权重的剪枝因子，该值越小对量化位宽的利用率越高，同时保留的有效权重会越少，当 quantize 为 True 时有效，否则无效。
-target_device - 目标设备，模型运行的目标硬件设备，目前有 T40、Txx、Xs1、Xs2、T41 五个选项。

例子：

```
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.))
```

返回值：

返回数据格式为 python 元组数据类型，其中元组中的第一个元素，为 NCHW 的 4D tensor 数据，表示 se_block 的计算结果。其他元素为量化训练中需要传递的参数。

Mul

功能描述：

封装的 mul 组合算子，包含 Feature 的相乘、Feature 量化操作。

接口定义：

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

参数详解：

-out_channels - 输出 Feature 的通道数。
-quantize - 是否执行量化操作(量化权重和 Feature)，设为 True 则执行量化操作。
-quantize_last_feature - 当该卷积为网络最后一层时，设为 True 则对输出 Feature 执行量化操作，否则不对输出 Feature 执行量化，当该卷积不是网络最后一层时该值无效。
-last_layer - 表示当前层是否为网络的最后一层（对于量化的网络只有最后一层可以选择是否只量化权重，不量化 Feature，其他层的权重和 Feature 均需量化）。
-input_bitwidth - 输入 Feature 的量化位宽，当 quantize 为 True 时有效， 否则无效。
-output_bitwidth - 输出 Feature 的量化位宽。
-target_device - 模型运行的目标硬件设备，目前有 Txx、T40、Xs1、Xs2、T41 五个选项。

例子：

```
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.)])
```

返回值：

返回数据格式为 python 元组数据类型，其中元组中的第一个元素，为 NCHW 的 4D tensor 数据，表示 Mul 的计算结果。其他元素为量化训练中需要传递的参

数。

注意事项：

1.目前支持两个输入数据的乘操作。

Channel_Split

功能描述：

封装的 torch.split 操作，量化训练中用来替换 torch.split 操作。

接口定义：

Channel_Split(split, target_device="Txx")

参数详解：

-split -channel 切分的列表。
-target_device 模型运行的目标硬件设备，目前有"Txx", "Xs1", "Xs2"三个选项。

例子：

```
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])
```

返回值：

返回数据格式为 python 元组数据类型，其中元组中的第一个元素，为 NCHW 的 4D tensor 数据，表示分割后的 featrue。其他元素为量化训练中需要传递的参数。

注意事项：

1.参数 split 必须是 list。
2.Txx 分割通道数必须是 16 的倍数。
eg：ops.Channel_split([16,48])。

Pixel_Shuffle

功能描述：

封装的上采样 PixelShuffle 算子，量化训练中，替换 nn.PixelShuffle 操作。

接口定义：

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

参数详解：

-upscale_factor - 向上缩放因子, upscale_factor 支持为 2
-target_device - 模型运行的目标硬件设备，目前只支持 T40、T41

例子：

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

返回值：

返回数据格式为 python 元组数据类型，其中元组中的第一个元素，为 NCHW 的 4D tensor 数据，表示 pixel_shuffel 操作后的 featrue。其他元素为量化训练中需要传递的参数。

注意事项：

1.注意经过 pixelshuffle 的输出通道必须是 32 的倍数。

LSTM

功能描述：

封装的 LSTM 组合算子，量化训练替换 torch.nn.LSTM 算子。

接口定义：

```
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")
```

参数详解：

-input_size - 输入的时间序列大小。
-hidden_size - 隐藏层大小。
-num_layers lstm - lstm 的堆叠层数，目前仅支持为 1。
-bias - 是否使能 BiasAdd 操作，若为 True 则在卷积后添加 BiasAdd 操作。
-bidirectional - 双向标识。
-initial_state_fw - 初始化正向 lstm 状态，初始时全部赋值为 0 状态。
-initial_state_bw - 初始化反向 lstm 状态，初始时全部赋值为 0 状态。
-return_output_states - 是否返回输出状态。
-batch_first - 输入数据维度 batch 优先，目前只支持这种方式。
-use_squeeze - 去除维度为 1 且做维度变化的标志位。
-quantize - 是否执行量化操作(量化权重和 Feature)，设为 True 则执行量化操作。
-input_bitwidth - 输入 Feature 的量化位宽，当 quantize 为 True 时有效， 否则无效。
-weight_bitwidth - 权重的量化位宽。
-output_bitwidth - 输出的量化位宽。
-weight_factor - quantize 的参数，表示对权重的剪枝因子。
-target_device - 模型运行的目标硬件设备，目前有 T40、T02、Txx、Xs1、Xs2、T41 六个选项。

例子：

```
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])
```

返回值：

返回数据格式为 python 元组数据类型，其中元组中的第一个元素，为 NCHW 的 4D tensor 数据，表示 LSTM 的结果。其他元素为量化训练中需要传递的参数。

注意事项：

lstm 前面的卷积或者全连接等算子需要设置 pre_lstm_layer 为 True。

Embedding

功能描述：

封装的 Embedding 算子。用于替换 nn.Embedding 算子。

接口定义：

```
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)
```

参数详解：

-num_embeddings - 字典中单词数量。
-embedding_dim - 词向量维度。
-padding_idx - 填充的索引值。
-first_layer - 第一层标识。
-quantize - 是否执行量化操作(量化权重和 Feature)，设为 True 则执行量化操作。
-input_bitwidth - 输入 Feature 的量化位宽。
-output_bitwidth - 输出 Feature 的量化位宽。
-multi_inputs - 标识是否为 2 个输入，默认值为 True。
-target_device - 模型运行的目标硬件设备，目前有 T40，Txx、Xs1、Xs2、T41 五个选项。
-pre_lstm_layer - 如果为 lstm 算子的前一层，请设置为 Ture。

例子：

```
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])
```

返回值：

返回数据格式为 python 元组数据类型，其中元组中的第一个元素，为 NCHW 的 4D tensor 数据，表示 Embedding 的结果。其他元素为量化训练中需要传递的参数。

注意事项：

embedding 输入参数 multi_inputs 默认为 True，如果非多个输入，请将 multi_inputs 设计成 False。

Focus

功能描述：

封装的 Focus 操作算子。用于代替 yolov5 系列中的 Focus 操作。

接口定义：

Focus(target_device=”Txx”)

参数详解：

-target_device -模型运行的目标硬件设备，目前有 T40，Txx、Xs1、Xs2、T41 五个选项。

例子：

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

返回值：

返回数据格式为 python 元组数据类型，其中元组中的第一个元素，为 NCHW

的 4D tensor 数据，表示 foces 处理完的 feature，其他元素为量化训练中需要传递的参数。

注意事项：

1. Focus 算子只支持第一层卷积前使用。
2. Focus 目前封装在 Conv2D 中使用，通过 is_focus 开启或关闭，不支持单独使用。

Max

功能描述：

返回俩个输入 feature 中最大的 feature,用于替换 torch.max()操作。

接口定义：

Max(target_device=”T40”)

参数详解：

-target_device -模型运行的目标硬件设备，目前只有 T40，T41 设备。

例子：

```
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.)])
```

返回值：

返回数据格式为 python 元组数据类型，其中元组中的第一个元素，表示 max 操作完的 feature，其他元素为量化训练中需要传递的参数。

封装的激活函数

注意事项：仅作为卷积等的 activation_fn 参数传入使用，不单独调用。

ReLU6

功能描述：

封装的 ReLU6 函数。

接口定义：

ReLU6(target_device = "Txx")

返回值：

返回 ReLU6 激活后的特征图。

ReLU

功能描述：

封装的 ReLU 函数。

接口定义：

ReLU(target_device = "Txx")

返回值：

返回 ReLU 激活后的特征图。

Linear

功能描述：

表示无激活函数。

接口定义：

Linear(target_device = "Txx")

返回值：

返回输入值，不做任何操作。

PReLU

功能描述：

封装的 prelu 激活函数

接口定义：

PReLU（in_channels, target_device="Txx"）

参数详解：

-in_channels 输入数据的通道数

返回值：

返回 PReLu 激活后的特征图。

Hsigmoid

功能描述：

封装的 Hsigmoid 算子。hsigmoid 激活函数是对 sigmoid 的近似，尽可能对 sigmoid 进行拟合。

接口定义：

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

返回值：

返回 hsigmoid 激活后的特征图。

Hswish

功能描述：

封装的 Hswish 算子。

接口定义：

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

返回值：

返回 Hswish 激活后的特征图。

LeakyReLU

功能描述：

封装的 LeakyReLU 算子。

接口定义：

LeakyReLU(target_device = "Txx")

返回值：

返回 LeakyReLU 激活后的特征图。

网络实例

以下是我们开源出来一个 T40 的分类网络例子，具体在 magik-toolkit/TrainingKit/pytorch/sample/network.py，具体为：

```
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")
        #网络最前面要调用 Preprocess
        self.bn = ops.BatchNorm(3, None, is_quantize, first_layer=True, 8, BITA,
                                target_device="T40")
```

```
    #网络的第一层，注意设置 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")
    #网络中间不能使用+或 torch.add，需使用封装的 Shortcut
     self.bn1 = ops.BatchNorm(64, ops.PReLU(64), is_quantize, False, BITA, BITA,
target_device="T40")
     #给了一个调用 PReLU 激活的单个 BatchNorm 算子
     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")
    #LSTM 的之前的 h 压缩及维度变换封装在 ops.LSTM 内部操作，打开
use_squeeze 即可

    self.flatten = ops.Flatten([-1, 256*2], target_device="T40")
    #不能调用原生的 reshape 算子，需用 Flatten，shape_list 按实际给，batch 不
确定用-1，其余维度得给出确定值
    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")
        #最后一层，注意设置 last_layer=True 以及 output_bitwidth=32，若之后有激
活，建议在之后单做，此时因为是浮点输出可以调用原 torch 的激活算子
        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
```