ConvoutionNd 层（Convolution 层）

- **括号中的层名和参数名适用于 TensorRT8 及之前版本，TensorRT9 及之后被废弃**
- 初始示例代码
- num_output_maps & kernel_size_nd (kernel_size) & kernel & bias
- stride_nd (stride)
- padding_nd (padding)
- pre_padding
- post_padding
- padding_mode
- dilation_nd (dilation)
- num_groups
- 三维卷积的示例
- set_input 用法

初始示例代码

```python
import numpy as np
from cuda import cudart
import tensorrt as trt

nIn, cIn, hIn, wIn = 1, 1, 6, 9  # 输入张量 NCHW
cOut, hW, wW = 1, 3, 3  # 卷积权重的输出通道数、高度和宽度
data = np.tile(np.arange(1, 1 + hW * wW, dtype=np.float32).reshape(hW, wW), (cIn, hIn // hW, wIn // wW)).reshape(1, cIn, hIn, wIn)  # 输入数据
weight = np.power(10, range(4, -5, -1), dtype=np.float32).reshape(cOut, hW, wW)  # 卷积权重
bias = np.zeros(cOut, dtype=np.float32)  # 卷积偏置

np.set_printoptions(precision=8, linewidth=200, suppress=True)
cudart.cudaDeviceSynchronize()

logger = trt.Logger(trt.Logger.ERROR)
builder = trt.Builder(logger)
network = builder.create_network(1 << int(trt.NetworkDefinitionCreationFlag.EXPLICIT_BATCH))
config = builder.create_builder_config()
inputT0 = network.add_input('inputT0', trt.DataType.FLOAT, (nIn, cIn, hIn, wIn))
#---------------------------------------------------------- --------------------# 替换部分
convolutionLayer = network.add_convolution_nd(inputT0, cOut, (hW, wW), weight, bias)
#---------------------------------------------------------- --------------------# 替换部分
network.mark_output(convolutionLayer.get_output(0))
engineString = builder.build_serialized_network(network, config)
engine = trt.Runtime(logger).deserialize_cuda_engine(engineString)
context = engine.create_execution_context()
_, stream = cudart.cudaStreamCreate()

inputH0 = np.ascontiguousarray(data.reshape(-1))
outputH0 = np.empty(context.get_binding_shape(1), dtype=trt.nptype(engine.get_binding_dtype(1)))
_, inputD0 = cudart.cudaMallocAsync(inputH0.nbytes, stream)
_, outputD0 = cudart.cudaMallocAsync(outputH0.nbytes, stream)

cudart.cudaMemcpyAsync(inputD0, inputH0.ctypes.data, inputH0.nbytes, cudart.cudaMemcpyKind.cudaMemcpyHostToDevice, stream)
context.execute_async_v2([int(inputD0), int(outputD0)], stream)
```

```
cudart.cudaMemcpyAsync(outputH0.ctypes.data, outputD0, outputH0.nbytes,
cudart.cudaMemcpyKind.cudaMemcpyDeviceToHost, stream)
cudart.cudaStreamSynchronize(stream)

print("inputH0 :", data.shape)
print(data)
print("outputH0:", outputH0.shape)
print(outputH0)

cudart.cudaStreamDestroy(stream)
cudart.cudaFree(inputD0)
cudart.cudaFree(outputD0)
```

- 输入张量形状 (1,1,6,9)

$$\left[\left[\left[\begin{matrix} 1. & 2. & 3. & 1. & 2. & 3. & 1. & 2. & 3. \\ 4. & 5. & 6. & 4. & 5. & 6. & 4. & 5. & 6. \\ 7. & 8. & 9. & 7. & 8. & 9. & 7. & 8. & 9. \\ 1. & 2. & 3. & 1. & 2. & 3. & 1. & 2. & 3. \\ 4. & 5. & 6. & 4. & 5. & 6. & 4. & 5. & 6. \\ 7. & 8. & 9. & 7. & 8. & 9. & 7. & 8. & 9. \end{matrix}\right]\right]\right]$$

- 输出张量形状 (1,1,4,7)，默认卷积步长 1，跨步 1，没有边缘填充

$$\left[\left[\left[\begin{matrix} 12345.6789 & 23156.4897 & 31264.5978 & 12345.6789 & 23156.4897 & 31264.5978 & 12345.6789 \\ 45678.9123 & 56489.7231 & 64597.8312 & 45678.9123 & 56489.7231 & 64597.8312 & 45678.9123 \\ 78912.3456 & 89723.1564 & 97831.2645 & 78912.3456 & 89723.1564 & 97831.2645 & 78912.3456 \\ 12345.6789 & 23156.4897 & 31264.5978 & 12345.6789 & 23156.4897 & 31264.5978 & 12345.6789 \end{matrix}\right]\right]\right]$$

- 计算过程：卷积结果中各元素的**个位**代表得出该值时卷积窗口的中心位置，其他各位代表参与计算的周围元素。受限于 float32 精度，运行结果无法完整展示 9 位有效数字，以上结果矩阵手工调整了这部分显示，以展示理想运行结果。后续各参数讨论中的输出矩阵不再作调整，而是显示再有舍入误差的原始结果。

$$\left[\begin{array}{|ccc|cc} \hline 1 & 2 & 3 & 1 & \cdots \\ 4 & 5 & 6 & 4 & \\ 7 & 8 & 9 & 7 & \\ \hline 1 & 2 & 3 & 1 & \cdots \\ \vdots & & & \vdots & \end{array}\right] \odot \begin{array}{|ccc|} \hline 10^4 & 10^3 & 10^2 \\ 10^1 & 1 & 10^{-1} \\ 10^{-2} & 10^{-3} & 10^{-4} \\ \hline \end{array} = 12345.6789,$$

$$\left[\begin{array}{c|ccc|c} & \hline 2 & 3 & 1 & \cdots \\ 1 & 2 & 3 & 1 & \cdots \\ 4 & 5 & 6 & 4 & \\ 7 & 8 & 9 & 7 & \\ \hline 1 & 2 & 3 & 1 & \cdots \\ \vdots & & & \vdots & \end{array}\right] \odot \begin{array}{|ccc|} \hline 10^4 & 10^3 & 10^2 \\ 10^1 & 1 & 10^{-1} \\ 10^{-2} & 10^{-3} & 10^{-4} \\ \hline \end{array} = 23156.4897$$

- 使用旧版 API `add_convolution` 会收到警告：

```
DeprecationWarning: Use add_convolution_nd instead.
```

num_output_maps & kernel_size_nd (kernel_size) & kernel & bias

```
placeHolder = np.zeros(1, dtype=np.float32)
convolutionLayer = network.add_convolution_nd(inputT0, 1, (1, 1), placeHolder)  # 先填入一些参数，bias 为可选参数，默认值 None
convolutionLayer.num_output_maps = cOut  # 重设卷积输出通道数
convolutionLayer.kernel_size_nd = (hW, wW)  # 重设卷积窗口尺寸
convolutionLayer.kernel = weight  # 重设卷积权值
convolutionLayer.bias = bias  # 重设卷积偏置
```

- 输出张量形状 (1,1,4,7)，结果与初始示例代码相同
- 使用旧版 API `kernel_size` 会收到警告

```
DeprecationWarning: Use kernel_size_nd instead.
```

stride_nd (stride)

```
hS = wS = 2
conv = network.add_convolution_nd(inputTensor, cOut, (hW, wW), window, bias)
conv.stride_nd = (hS,wS)  # 卷积步长，默认值 (1,1)
```

- 指定 stride_nd=(2,2)（HW 维跨步均为 2），输出张量形状 (1,1,2,4)

$$\left[\left[\begin{bmatrix} 12345.679 & 31264.598 & 23156.49 & 12345.679 \\ 78912.34 & 97831.27 & 89723.16 & 78912.34 \end{bmatrix}\right]\right]$$

- 指定 stride_nd=(2,1)（H 维跨步 2），输出张量形状 (1,2,7)

$$\left[\left[\begin{bmatrix} 12345.679 & 23156.49 & 31264.598 & 12345.679 & 23156.49 & 31264.598 & 12345.679 \\ 78912.34 & 89723.16 & 97831.27 & 78912.34 & 89723.16 & 97831.27 & 78912.34 \end{bmatrix}\right]\right]$$

- 指定 stride_nd=(1,2)（W 维跨步 2），输出张量形状 (1,4,4)

$$\left[\left[\begin{bmatrix} 12345.679 & 31264.598 & 23156.49 & 12345.679 \\ 45678.914 & 64597.832 & 56489.723 & 45678.914 \\ 78912.34 & 97831.27 & 89723.16 & 78912.34 \\ 12345.679 & 31264.598 & 23156.49 & 12345.679 \end{bmatrix}\right]\right]$$

- 使用旧版 API `stride` 会收到警告

```
DeprecationWarning: Use stride_nd instead
```

padding_nd (padding)

```
hP = wP = 1
convolutionLayer = network.add_convolution_nd(inputT0, cOut, (hW, wW), weight, bias)
convolutionLayer.padding_nd = (hP, wP)  # 四周填充 0 层数，默认值 (0,0)
```

- 指定 padding_nd=(1,1)（HW 维均填充 1 层 0），输出张量形状 (1,1,6,9)

$$\left[\left[\begin{bmatrix} 1.2045 & 12.3456 & 23.1564 & 31.2645 & 12.3456 & 23.1564 & 31.2645 & 12.3456 & 23.056 \\ 1204.5078 & 12345.679 & 23156.49 & 31264.598 & 12345.679 & 23156.49 & 31264.598 & 12345.679 & 23056.09 \\ 4507.801 & 45678.914 & 56489.723 & 64597.832 & 45678.914 & 56489.723 & 64597.832 & 45678.914 & 56089.023 \\ 7801.2046 & 78912.34 & 89723.16 & 97831.27 & 78912.34 & 89723.16 & 97831.27 & 78912.34 & 89023.055 \\ 1204.5078 & 12345.679 & 23156.49 & 31264.598 & 12345.679 & 23156.49 & 31264.598 & 12345.679 & 23056.09 \\ 4507.8 & 45678.9 & 56489.7 & 64597.8 & 45678.9 & 56489.7 & 64597.8 & 45678.9 & 56089. \end{bmatrix}\right]\right]$$

- 指定 padding_nd=(1,0)（H 维填充 1 层 0），输出张量形状 (1,1,6,7)

$$\left[\left[\begin{bmatrix} 12.3456 & 23.1564 & 31.2645 & 12.3456 & 23.1564 & 31.2645 & 12.3456 \\ 12345.679 & 23156.49 & 31264.598 & 12345.679 & 23156.49 & 31264.598 & 12345.679 \\ 45678.914 & 56489.723 & 64597.832 & 45678.914 & 56489.723 & 64597.832 & 45678.914 \\ 78912.34 & 89723.16 & 97831.27 & 78912.34 & 89723.16 & 97831.27 & 78912.34 \\ 12345.679 & 23156.49 & 31264.598 & 12345.679 & 23156.49 & 31264.598 & 12345.679 \\ 45678.9 & 56489.7 & 64597.8 & 45678.9 & 56489.7 & 64597.8 & 45678.9 \end{bmatrix}\right]\right]$$

- 指定 padding_nd=(0,1)（W 维填充 1 层 0），输出张量形状 (1,1,4,9)

$$\left[\left[\begin{bmatrix} 1204.5078 & 12345.679 & 23156.49 & 31264.598 & 12345.679 & 23156.49 & 31264.598 & 12345.679 & 23056.09 \\ 4507.801 & 45678.914 & 56489.723 & 64597.832 & 45678.914 & 56489.723 & 64597.832 & 45678.914 & 56089.023 \\ 7801.2046 & 78912.34 & 89723.16 & 97831.27 & 78912.34 & 89723.16 & 97831.27 & 78912.34 & 89023.055 \\ 1204.5078 & 12345.679 & 23156.49 & 31264.598 & 12345.679 & 23156.49 & 31264.598 & 12345.679 & 23056.09 \end{bmatrix}\right]\right]$$

- 使用旧版 API `padding` 会收到警告

```
DeprecationWarning: Use padding_nd instead
```

pre_padding

```
hPre = wPre = 1
convolutionLayer = network.add_convolution_nd(inputT0, cOut, (hW, wW), weight, bias)
convolutionLayer.pre_padding = (hPre, wPre)  # 头部填充 0 层数, 默认值 (0,0)
```

- 指定 pre_padding=(1,1)（HW 维头部均填充 1 层 0），输出张量形状 (1,1,5,8)

$$\left[\left[\begin{bmatrix} 1.2045 & 12.3456 & 23.1564 & 31.2645 & 12.3456 & 23.1564 & 31.2645 & 12.3456 \\ 1204.5078 & 12345.679 & 23156.49 & 31264.598 & 12345.679 & 23156.49 & 31264.598 & 12345.679 \\ 4507.801 & 45678.914 & 56489.723 & 64597.832 & 45678.914 & 56489.723 & 64597.832 & 45678.914 \\ 7801.2046 & 78912.34 & 89723.16 & 97831.27 & 78912.34 & 89723.16 & 97831.27 & 78912.34 \\ 1204.5078 & 12345.679 & 23156.49 & 31264.598 & 12345.679 & 23156.49 & 31264.598 & 12345.679 \end{bmatrix}\right]\right]$$

- 指定 pre_padding=(1,0)（H 维头部填充 1 层 0），输出张量形状 (1,1,5,7)

$$\left[\left[\begin{bmatrix} 12.3456 & 23.1564 & 31.2645 & 12.3456 & 23.1564 & 31.2645 & 12.3456 \\ 12345.679 & 23156.49 & 31264.598 & 12345.679 & 23156.49 & 31264.598 & 12345.679 \\ 45678.914 & 56489.723 & 64597.832 & 45678.914 & 56489.723 & 64597.832 & 45678.914 \\ 78912.34 & 89723.16 & 97831.27 & 78912.34 & 89723.16 & 97831.27 & 78912.34 \\ 12345.679 & 23156.49 & 31264.598 & 12345.679 & 23156.49 & 31264.598 & 12345.679 \end{bmatrix}\right]\right]$$

- 指定 pre_padding=(0,1)（W 维头部填充 1 层 0），输出张量形状 (1,1,4,8)

$$\left[\left[\begin{bmatrix} 1204.5078 & 12345.679 & 23156.49 & 31264.598 & 12345.679 & 23156.49 & 31264.598 & 12345.679 \\ 4507.801 & 45678.914 & 56489.723 & 64597.832 & 45678.914 & 56489.723 & 64597.832 & 45678.914 \\ 7801.2046 & 78912.34 & 89723.16 & 97831.27 & 78912.34 & 89723.16 & 97831.27 & 78912.34 \\ 1204.5078 & 12345.679 & 23156.49 & 31264.598 & 12345.679 & 23156.49 & 31264.598 & 12345.679 \end{bmatrix}\right]\right]$$

post_padding

```
hPost = wPost = 1
convolutionLayer = network.add_convolution_nd(inputT0, cOut, (hW, wW), weight, bias)
convolutionLayer.post_padding = (hPost, wPost)  # 尾部填充 0 层数, 默认值 (0,0)
```

- 指定 post_padding=(1,1)（HW 维尾部均填充 1 层 0），输出张量形状 (1,1,5,8)

$$\left[\left[\left[\begin{matrix} 12345.679 & 23156.49 & 31264.598 & 12345.679 & 23156.49 & 31264.598 & 12345.679 & 23056.09 \\ 45678.914 & 56489.723 & 64597.832 & 45678.914 & 56489.723 & 64597.832 & 45678.914 & 56089.023 \\ 78912.34 & 89723.16 & 97831.27 & 78912.34 & 89723.16 & 97831.27 & 78912.34 & 89023.055 \\ 12345.679 & 23156.49 & 31264.598 & 12345.679 & 23156.49 & 31264.598 & 12345.679 & 23056.09 \\ 45678.9 & 56489.7 & 64597.8 & 45678.9 & 56489.7 & 64597.8 & 45678.9 & 56089. \end{matrix}\right]\right]\right]$$

- 指定 post_padding=(1,0)（H 维尾部填充 1 层 0），输出张量形状 (1,1,5,7)

$$\left[\left[\left[\begin{matrix} 12345.679 & 23156.49 & 31264.598 & 12345.679 & 23156.49 & 31264.598 & 12345.679 \\ 45678.914 & 56489.723 & 64597.832 & 45678.914 & 56489.723 & 64597.832 & 45678.914 \\ 78912.34 & 89723.16 & 97831.27 & 78912.34 & 89723.16 & 97831.27 & 78912.34 \\ 12345.679 & 23156.49 & 31264.598 & 12345.679 & 23156.49 & 31264.598 & 12345.679 \\ 45678.9 & 56489.7 & 64597.8 & 45678.9 & 56489.7 & 64597.8 & 45678.9 \end{matrix}\right]\right]\right]$$

- 指定 post_padding=(0,1)（W 维尾部填充 1 层 0），输出张量形状 (1,1,4,8)

$$\left[\left[\left[\begin{matrix} 12345.679 & 23156.49 & 31264.598 & 12345.679 & 23156.49 & 31264.598 & 12345.679 & 23056.09 \\ 45678.914 & 56489.723 & 64597.832 & 45678.914 & 56489.723 & 64597.832 & 45678.914 & 56089.023 \\ 78912.34 & 89723.16 & 97831.27 & 78912.34 & 89723.16 & 97831.27 & 78912.34 & 89023.055 \\ 12345.679 & 23156.49 & 31264.598 & 12345.679 & 23156.49 & 31264.598 & 12345.679 & 23056.09 \end{matrix}\right]\right]\right]$$

padding_mode

```
convolutionLayer = network.add_convolution_nd(inputT0, cOut, (hW, wW), weight, bias)
convolutionLayer.stride_nd = (2, 2)  # 加上卷积步长，以便观察结果
convolutionLayer.padding_mode = trt.PaddingMode.SAME_UPPER  # 填充 0 方案，优先级高于 padding，pre_padding 和 post_padding，默认值 EXPLICIT_ROUND_DOWN
```

- 计算过程参考 TensorRT C API reference
- 指定 padding_mode = **trt.PaddingMode.SAME_UPPER**，输出张量形状 (1,1,3,5)

$$\left[\left[\left[\begin{matrix} 1204.5078 & 23156.49 & 12345.679 & 31264.598 & 23056.09 \\ 7801.2046 & 89723.16 & 78912.34 & 97831.27 & 89023.055 \\ 4507.8 & 56489.7 & 45678.9 & 64597.8 & 56089. \end{matrix}\right]\right]\right]$$

- 指定 padding_mode = **trt.PaddingMode.SAME_LOWER**，输出张量形状 (1,1,3,5)

$$\left[\left[\left[\begin{matrix} 1.2045 & 23.1564 & 12.3456 & 31.2645 & 23.056 \\ 4507.801 & 56489.723 & 45678.914 & 64597.832 & 56089.023 \\ 1204.5078 & 23156.49 & 12345.679 & 31264.598 & 23056.09 \end{matrix}\right]\right]\right]$$

- 指定 padding_mode = **trt.PaddingMode.EXPLICIT_ROUND_UP**，输出张量形状 (1,1,3,4)

$$\left[\left[\left[\begin{matrix} 12345.679 & 31264.598 & 23156.49 & 12345.679 \\ 78912.34 & 97831.27 & 89723.16 & 78912.34 \\ 45678.9 & 64597.8 & 56489.7 & 45678.9 \end{matrix}\right]\right]\right]$$

- 指定 padding_mode = **trt.PaddingMode.EXPLICIT_ROUND_DOWN**，输出张量形状 (1,1,2,4)

$$\left[\left[\left[\begin{matrix} 12345.679 & 31264.598 & 23156.49 & 12345.679 \\ 78912.34 & 97831.27 & 89723.16 & 78912.34 \end{matrix}\right]\right]\right]$$

- 指定 padding_mode = **trt.PaddingMode.CAFFE_ROUND_UP**，输出张量形状 (1,1,3,4)

$$\left[\left[\left[\begin{matrix} 12345.67931264.59823156.4912345.679 \\ 78912.3497831.2789723.1678912.34 \\ 45678.964597.856489.745678.9 \end{matrix}\right]\right]\right]$$

- 指定 padding_mode = **trt.PaddingMode.CAFFE_ROUND_DOWN**，输出张量形状 (1,1,2,4)

$$\left[\left[\begin{bmatrix} 12345.679 & 31264.598 & 23156.49 & 12345.679 \\ 78912.34 & 97831.27 & 89723.16 & 78912.34 \end{bmatrix}\right]\right]$$

---

dilation_nd (dilation)

```
hD = wD = 2
convolutionLayer = network.add_convolution_nd(inputT0, cOut, (hW, wW), weight, bias)
convolutionLayer.dilation_nd = (hD, wD)  # 卷积核扩张度，表示卷积核相邻元素在该轴上的间隔，默认值 (1,1)
```

- 指定 dilation_nd=(2,2)（卷积核在 HW 维上元素间隔均为 2），输出张量形状 (1,1,2,5)

$$\left[\left[\begin{bmatrix} 13279.847 & 21387.955 & 32198.766 & 13279.847 & 21387.955 \\ 46513.277 & 54621.387 & 65432.2 & 46513.277 & 54621.387 \end{bmatrix}\right]\right]$$

- 指定 dilation_nd=(2,1)（卷积核在 H 维上元素间隔为 2），输出张量形状 (1,1,2,7)

$$\left[\left[\begin{bmatrix} 12378.946 & 23189.756 & 31297.865 & 12378.946 & 23189.756 & 31297.865 & 12378.946 \\ 45612.38 & 56423.188 & 64531.297 & 45612.38 & 56423.188 & 64531.297 & 45612.38 \end{bmatrix}\right]\right]$$

- 指定 dilation_nd=(1,2)（卷积核在 W 维上元素间隔为 2），输出张量形状 (1,1,4,5)

$$\left[\left[\begin{bmatrix} 13246.58 & 21354.688 & 32165.498 & 13246.58 & 21354.688 \\ 46579.816 & 54687.918 & 65498.734 & 46579.816 & 54687.918 \\ 79813.25 & 87921.35 & 98732.17 & 79813.25 & 87921.35 \\ 13246.58 & 21354.688 & 32165.498 & 13246.58 & 21354.688 \end{bmatrix}\right]\right]$$

- 使用旧版 API `dilation` 会收到警告

```
DeprecationWarning: Use dilation_nd instead
```

---

num_groups

```
import numpy as np
from cuda import cudart
import tensorrt as trt

nIn, cIn, hIn, wIn = 1, 2, 6, 9  # 调整部分输入输出参数
nGroup = 2
cOut, hW, wW = nGroup, 3, 3
data = np.tile(np.arange(1, 1 + hW * wW, dtype=np.float32).reshape(hW, wW), (cIn, hIn // hW, wIn //
wW)).reshape(cIn, hIn, wIn)
weight = np.power(10, range(4, -5, -1), dtype=np.float32)
weight = np.concatenate([weight, -weight], 0)  # 输入张量通道数必须能被分组数整除
bias = np.zeros(cOut, dtype=np.float32)

np.set_printoptions(precision=8, linewidth=200, suppress=True)
cudart.cudaDeviceSynchronize()

logger = trt.Logger(trt.Logger.ERROR)
builder = trt.Builder(logger)
network = builder.create_network(1 << int(trt.NetworkDefinitionCreationFlag.EXPLICIT_BATCH))
config = builder.create_builder_config()
inputT0 = network.add_input('inputT0', trt.DataType.FLOAT, (nIn, cIn, hIn, wIn))
convolutionLayer = network.add_convolution_nd(inputT0, cOut, (hW, wW), weight, bias)
```

```
convolutionLayer.num_groups = nGroup  # 分组数，默认值 1

network.mark_output(convolutionLayer.get_output(0))
engineString = builder.build_serialized_network(network, config)
engine = trt.Runtime(logger).deserialize_cuda_engine(engineString)
context = engine.create_execution_context()
_, stream = cudart.cudaStreamCreate()

inputH0 = np.ascontiguousarray(data.reshape(-1))
outputH0 = np.empty(context.get_binding_shape(1), dtype=trt.nptype(engine.get_binding_dtype(1)))
_, inputD0 = cudart.cudaMallocAsync(inputH0.nbytes, stream)
_, outputD0 = cudart.cudaMallocAsync(outputH0.nbytes, stream)

cudart.cudaMemcpyAsync(inputD0, inputH0.ctypes.data, inputH0.nbytes,
cudart.cudaMemcpyKind.cudaMemcpyHostToDevice, stream)
context.execute_async_v2([int(inputD0), int(outputD0)], stream)
cudart.cudaMemcpyAsync(outputH0.ctypes.data, outputD0, outputH0.nbytes,
cudart.cudaMemcpyKind.cudaMemcpyDeviceToHost, stream)
cudart.cudaStreamSynchronize(stream)

print("inputH0 :", data.shape)
print(data)
print("outputH0:", outputH0.shape)
print(outputH0)

cudart.cudaStreamDestroy(stream)
cudart.cudaFree(inputD0)
cudart.cudaFree(outputD0)
```

- 指定 num_groupds=2，输入张量和卷积核均在 C 维上被均分为 2 组，各自卷积后再拼接到一起，输出张量形状 (1,2,4,7)

$$\begin{bmatrix}\begin{bmatrix}\begin{bmatrix} 12345.679 & 23156.49 & 31264.598 & 12345.679 & 23156.49 & 31264.598 & 12345.679 \\ 45678.914 & 56489.723 & 64597.832 & 45678.914 & 56489.723 & 64597.832 & 45678.914 \\ 78912.34 & 89723.16 & 97831.27 & 78912.34 & 89723.16 & 97831.27 & 78912.34 \\ 12345.679 & 23156.49 & 31264.598 & 12345.679 & 23156.49 & 31264.598 & 12345.679 \end{bmatrix} \\ \begin{bmatrix} -12345.679 & -23156.49 & -31264.598 & -12345.679 & -23156.49 & -31264.598 & -12345.679 \\ -45678.914 & -56489.723 & -64597.832 & -45678.914 & -56489.723 & -64597.832 & -45678.914 \\ -78912.34 & -89723.16 & -97831.27 & -78912.34 & -89723.16 & -97831.27 & -78912.34 \\ -12345.679 & -23156.49 & -31264.598 & -12345.679 & -23156.49 & -31264.598 & -12345.679 \end{bmatrix}\end{bmatrix}\end{bmatrix}$$

- int8 模式中，每组的尺寸（cIn/nGroup 和 cOut/nGroup）必须是 4 的倍数

三维卷积的示例

```
import numpy as np
from cuda import cudart
import tensorrt as trt

nIn, cIn, hIn, wIn = 1, 2, 6, 9  # 调整部分输入输出参数
cOut, hW, wW = 1, 3, 3
data = np.tile(np.arange(1, 1 + hW * wW, dtype=np.float32).reshape(hW, wW), (cIn, hIn // hW, wIn //
wW)).reshape(cIn, hIn, wIn)
weight = np.power(10, range(4, -5, -1), dtype=np.float32).reshape(cOut, hW, wW)
weight = np.concatenate([weight, -weight], 0)
bias = np.zeros(cOut, dtype=np.float32)

np.set_printoptions(precision=8, linewidth=200, suppress=True)
```

```
cudart.cudaDeviceSynchronize()

logger = trt.Logger(trt.Logger.ERROR)
builder = trt.Builder(logger)
network = builder.create_network(1 << int(trt.NetworkDefinitionCreationFlag.EXPLICIT_BATCH))
config = builder.create_builder_config()
inputT0 = network.add_input('inputT0', trt.DataType.FLOAT, (nIn, 1, cIn, hIn, wIn))  # 要求输入至少为 5 维
convolutionLayer = network.add_convolution_nd(inputT0, cOut, weight.shape, weight, bias)  # 卷积核是 3 维的
network.mark_output(convolutionLayer.get_output(0))
engineString = builder.build_serialized_network(network, config)
engine = trt.Runtime(logger).deserialize_cuda_engine(engineString)
context = engine.create_execution_context()
_, stream = cudart.cudaStreamCreate()

inputH0 = np.ascontiguousarray(data.reshape(-1))
outputH0 = np.empty(context.get_binding_shape(1), dtype=trt.nptype(engine.get_binding_dtype(1)))
_, inputD0 = cudart.cudaMallocAsync(inputH0.nbytes, stream)
_, outputD0 = cudart.cudaMallocAsync(outputH0.nbytes, stream)

cudart.cudaMemcpyAsync(inputD0, inputH0.ctypes.data, inputH0.nbytes,
cudart.cudaMemcpyKind.cudaMemcpyHostToDevice, stream)
context.execute_async_v2([int(inputD0), int(outputD0)], stream)
cudart.cudaMemcpyAsync(outputH0.ctypes.data, outputD0, outputH0.nbytes,
cudart.cudaMemcpyKind.cudaMemcpyDeviceToHost, stream)
cudart.cudaStreamSynchronize(stream)

print("inputH0 :", data.shape)
print(data)
print("outputH0:", outputH0.shape)
print(outputH0)

cudart.cudaStreamDestroy(stream)
cudart.cudaFree(inputD0)
cudart.cudaFree(outputD0)
```

- 输出张量形状 (1,1,1,4,7)，相当于把前面 num_groups 例子中结果的两个通道加在一起，得到了全部元素均为 0 的结果

$$\left[\left[\left[\begin{bmatrix} -0.00018907 & 0.00053437 & -0.00014376 & -0.00018907 & 0.00053437 & -0.00014376 & -0.00018907 \\ 0.00176249 & -0.00044376 & 0.00083124 & 0.00176249 & -0.00044376 & 0.00083124 & 0.00176249 \\ -0.00185 & -0.00015 & 0.0089375 & -0.00185 & -0.00015 & 0.0089375 & -0.00185 \\ -0.00018907 & 0.00053437 & -0.00014376 & -0.00018907 & 0.00053437 & -0.00014376 & -0.00018907 \end{bmatrix}\right]\right]\right]$$

set_input 用法

- 参考 link

```
import numpy as np
from cuda import cudart
import tensorrt as trt

nIn, cIn, hIn, wIn = 1, 1, 6, 9
cOut, hW, wW = 1, 3, 3
data = np.tile(np.arange(1, 1 + hW * wW, dtype=np.float32).reshape(hW, wW), (cIn, hIn // hW, wIn // wW)).reshape(1, cIn, hIn, wIn)
weight = np.power(10, range(4, -5, -1), dtype=np.float32).reshape(cOut, hW, wW)
bias = np.zeros(cOut, dtype=np.float32)
```

```
np.set_printoptions(precision=8, linewidth=200, suppress=True)
cudart.cudaDeviceSynchronize()

logger = trt.Logger(trt.Logger.ERROR)
builder = trt.Builder(logger)
network = builder.create_network(1 << int(trt.NetworkDefinitionCreationFlag.EXPLICIT_BATCH))
config = builder.create_builder_config()
config.flags = 1 << int(trt.BuilderFlag.INT8)  # 需要打开 int8 模式
inputT0 = network.add_input('inputT0', trt.DataType.FLOAT, (nIn, cIn, hIn, wIn))
#---------------------------------------------------------- --------------------# 替换部分
constantLayer0 = network.add_constant([], np.array([1], dtype=np.float32))
constantLayer1 = network.add_constant([], np.array([1], dtype=np.float32))
weightLayer = network.add_constant([cOut, cIn, hW, wW], weight)

quantizeLayer0 = network.add_quantize(inputT0, constantLayer0.get_output(0))
quantizeLayer0.axis = 0
dequantizeLayer0 = network.add_dequantize(quantizeLayer0.get_output(0), constantLayer1.get_output(0))
dequantizeLayer0.axis = 0
quantizeLayer1 = network.add_quantize(weightLayer.get_output(0), constantLayer0.get_output(0))
quantizeLayer1.axis = 0
dequantizeLayer1 = network.add_dequantize(quantizeLayer1.get_output(0), constantLayer1.get_output(0))
dequantizeLayer1.axis = 0

convolutionLayer = network.add_convolution_nd(dequantizeLayer0.get_output(0), cOut, (hW, wW),
trt.Weights())  # 需要把 weight 设为空权重（不能用 np.array()）
convolutionLayer.set_input(1, dequantizeLayer1.get_output(0))
#---------------------------------------------------------- --------------------# 替换部分
network.mark_output(convolutionLayer.get_output(0))
engineString = builder.build_serialized_network(network, config)
engine = trt.Runtime(logger).deserialize_cuda_engine(engineString)
context = engine.create_execution_context()
_, stream = cudart.cudaStreamCreate()

inputH0 = np.ascontiguousarray(data.reshape(-1))
outputH0 = np.empty(context.get_binding_shape(1), dtype=trt.nptype(engine.get_binding_dtype(1)))
_, inputD0 = cudart.cudaMallocAsync(inputH0.nbytes, stream)
_, outputD0 = cudart.cudaMallocAsync(outputH0.nbytes, stream)

cudart.cudaMemcpyAsync(inputD0, inputH0.ctypes.data, inputH0.nbytes,
cudart.cudaMemcpyKind.cudaMemcpyHostToDevice, stream)
context.execute_async_v2([int(inputD0), int(outputD0)], stream)
cudart.cudaMemcpyAsync(outputH0.ctypes.data, outputD0, outputH0.nbytes,
cudart.cudaMemcpyKind.cudaMemcpyDeviceToHost, stream)
cudart.cudaStreamSynchronize(stream)

print("inputH0 :", data.shape)
print(data)
print("outputH0:", outputH0.shape)
print(outputH0)

cudart.cudaStreamDestroy(stream)
cudart.cudaFree(inputD0)
cudart.cudaFree(outputD0)
```

- 输出张量形状 (1,1,4,7)

$$\left[\left[\begin{bmatrix} 726. & 791. & 772. & 726. & 791. & 772. & 726. \\ 1821. & 1886. & 1867. & 1821. & 1886. & 1867. & 1821. \\ 2817. & 2882. & 2863. & 2817. & 2882. & 2863. & 2817. \\ 726. & 791. & 772. & 726. & 791. & 772. & 726. \end{bmatrix}\right]\right]$$