DeonvoutionNd 层（Deconvolution 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

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

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

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()
config.max_workspace_size = 1 << 30  # 设置空间给 TensorRT 尝试优化，单位 Byte
inputT0 = network.add_input('inputT0', trt.DataType.FLOAT, (nIn, cIn, hIn, wIn))
#---------------------------------------------------------- --------------------# 替换部分
deconvolutionLayer = network.add_deconvolution_nd(inputT0, cOut, (hW, wW), weight, bias)
#---------------------------------------------------------- --------------------# 替换部分
network.mark_output(deconvolutionLayer.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)
```

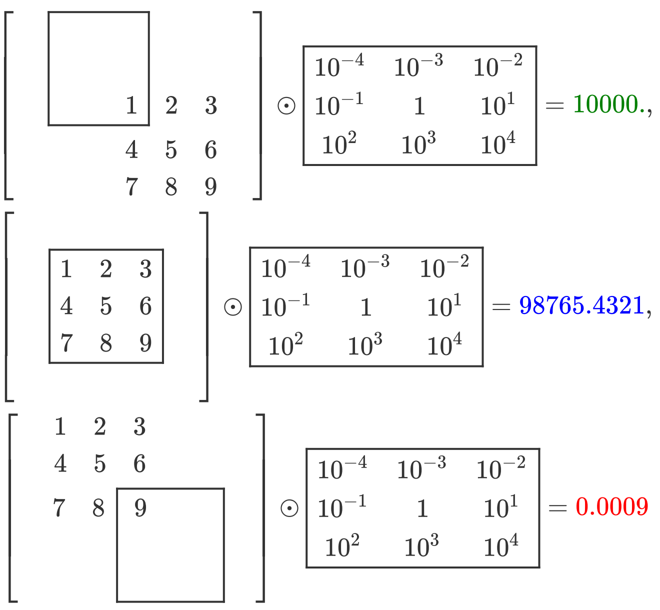
- 输入张量形状 (1,1,3,3)

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

- 输出张量形状 (1,1,5,5)，默认反卷积步长 1，跨步 1，没有边缘填充

$$\left[\left[\begin{bmatrix} 10000.0000 & 21000.0000 & 32100.0000 & 3200.0000 & 300.0000 \\ 40010.0000 & 54021.0000 & 65432.1000 & 6503.2 & 600.3000 \\ 70040.0100 & 87054.0210 & 98765.4321 & 9806.5032 & 900.6003 \\ 70.0400 & 87.0540 & 98.76540 & 9.8065 & 0.9006 \\ 0.0700 & 0.0870 & 0.09867 & 0.0098 & 0.0009 \end{bmatrix}\right]\right]$$

- 计算过程：反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，**注意反卷积核是倒序的**。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。



- 使用旧版 API `add_deconvolution` 会收到警告

```
DeprecationWarning: Use add_deconvolution_nd instead.
```

- 不设置 config.max_workspace_size 会收到报错：

```
[TensorRT] INFO: Some tactics do not have sufficient workspace memory to run. Increasing workspace size
may increase performance, please check verbose output.
[TensorRT] ERROR: 10: [optimizer.cpp::computeCosts::1855] Error Code 10: Internal Error (Could not find
any implementation for node (Unnamed Layer* 0) [Deconvolution].)
```

---

num_output_maps & kernel_size_nd (kernel_size) & kernel & bias

```python
placeHolder = np.zeros(1, dtype=np.float32)
deconvolutionLayer = network.add_deconvolution_nd(inputT0, 1, (1, 1), placeHolder)  # 先填入一些参数，bias 为可选参数，默认值 None
deconvolutionLayer.num_output_maps = cOut  # 重设反卷积输出通道数，最大值 8192
deconvolutionLayer.kernel_size_nd = (hW, wW)  # 重设反卷积窗口尺寸
deconvolutionLayer.kernel = weight  # 重设反卷积权值
deconvolutionLayer.bias = bias  # 重设反卷积偏置
```

- 输出张量形状 (1,1,5,5)，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)

```python
hS = wS = 2
deconvolutionLayer = network.add_deconvolution_nd(inputT0, cOut, (hW, wW), weight, bias)
deconvolutionLayer.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 (1,1,7,7)

$$\begin{bmatrix}\begin{bmatrix}
10000. & 1000. & 20100. & 2000. & 30200. & 3000. & 300. \\
10. & 1. & 20.1 & 2. & 30.2 & 3. & 0.3 \\
40000.01 & 4000.001 & 50400.02 & 5000.002 & 60500.03 & 6000.003 & 600.0003 \\
40. & 4. & 50.4 & 5. & 60.5 & 6. & 0.6 \\
70000.04 & 7000.004 & 80700.05 & 8000.005 & 90800.06 & 9000.006 & 900.0006 \\
70. & 7. & 80.7 & 8. & 90.8 & 9. & 0.90000004 \\
0.07 & 0.007 & 0.0807 & 0.008 & 0.09079999 & 0.009 & 0.0009
\end{bmatrix}\end{bmatrix}$$

- 指定 stride_nd=(2,1)（H 维跨步为 2），输出张量形状 (1,1,7,5)

$$\begin{bmatrix}\begin{bmatrix}
10000. & 21000. & 32100. & 3200. & 300. \\
10. & 21. & 32.1 & 3.2 & 0.3 \\
40000.01 & 54000.02 & 65400.035 & 6500.0034 & 600.0003 \\
40. & 54.65.4 & 6.5 & 0.6 & \\
70000.04 & 87000.055 & 98700.07 & 9800.007 & 900.0006 \\
70. & 87. & 98.7 & 9.8 & 0.90000004 \\
0.07 & 0.087 & 0.09869999 & 0.0098 & 0.0009
\end{bmatrix}\end{bmatrix}$$

- 指定 stride_nd=(1,2)（H 维跨步为 2），输出张量形状 (1,1,5,7)

$$\begin{bmatrix}\begin{bmatrix}
10000. & 1000. & 20100. & 2000. & 30200. & 3000. & 300. \\
40010. & 4001. & 50420.1 & 5002. & 60530.2 & 6003. & 600.3 \\
70040.01 & 7004.001 & 80750.42 & 8005.002 & 90860.53 & 9006.003 & 900.6003 \\
70.04 & 7.004 & 80.7504 & 8.005 & 90.860504 & 9.006 & 0.9006 \\
0.07 & 0.007 & 0.0807 & 0.008 & 0.09079999 & 0.009 & 0.0009
\end{bmatrix}\end{bmatrix}$$

- 使用旧版 API `stride` 会收到警告

```
DeprecationWarning: Use stride_nd instead
```

---

padding_nd (padding)

```
hP = wP = 1
deconvolutionLayer = network.add_deconvolution_nd(inputT0, cOut, (hW, wW), weight, bias)
deconvolutionLayer.padding_nd = (hP, wP)  # 四周减少填充 0 层数，默认值 (0,0)
```

- 指定 padding_nd=(1,1)（HW 维均减少填充 1 层 0），输出张量形状 (1,1,3,3)
- 含义是给输入张量 HW 维均填充一层 0（[3,3] ->[5,5]）后做反卷积

$$\left[\left[\left[\begin{matrix} 54021. & 65432.1 & 6503.2 \\ 87054.02 & 98765.43 & 9806.503 \\ 87.054 & 98.765396 & 9.8064995 \end{matrix}\right]\right]\right]$$

- 指定 padding_nd=(1,0)（H 维减少填充 1 层 0），输出张量形状 (1,1,3,5)

$$\left[\left[\left[\begin{matrix} 40010. & 54021. & 65432.1 & 6503.2 & 600.3 \\ 70040.01 & 87054.02 & 98765.43 & 9806.503 & 900.6003 \\ 70.04 & 87.054 & 98.765396 & 9.8064995 & 0.9006 \end{matrix}\right]\right]\right]$$

- 指定 padding_nd=(0,1)（W 维减少填充 1 层 0），输出张量形状 (1,1,5,3)

$$\left[\left[\left[\begin{matrix} 21000. & 32100. & 3200. \\ 54021. & 65432.1 & 6503.2 \\ 87054.02 & 98765.43 & 9806.503 \\ 87.054 & 98.765396 & 9.8064995 \\ 0.087 & 0.09869999 & 0.0098 \end{matrix}\right]\right]\right]$$

- 指定 padding_nd=(2,2)（HW 维均减少填充 2 层 0），输出张量形状 (1,1,1,1)

$$[[[[98765.43]]]]$$

- 使用旧版 API `padding` 会收到警告

```
DeprecationWarning: Use padding_nd instead
```

---

pre_padding

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

- 指定 pre_padding=(1,1)（HW 维头部均减少填充 1 层 0），输出张量形状 (1,1,4,4)

$$\left[\left[\left[\begin{matrix} 54021. & 65432.1 & 6503.2 & 600.3 \\ 87054.02 & 98765.43 & 9806.503 & 900.6003 \\ 87.054 & 98.765396 & 9.8064995 & 0.9006 \\ 0.087 & 0.09869999 & 0.0098 & 0.0009 \end{matrix}\right]\right]\right]$$

- 指定 pre_padding=(1,0)（H 维头部减少填充 1 层 0），输出张量形状 (1,1,4,5)

$$\left[\left[\left[\begin{matrix} 40010. & 54021. & 65432.1 & 6503.2 & 600.3 \\ 70040.01 & 87054.02 & 98765.43 & 9806.503 & 900.6003 \\ 70.04 & 87.054 & 98.765396 & 9.8064995 & 0.9006 \\ 0.07 & 0.087 & 0.09869999 & 0.0098 & 0.0009 \end{matrix}\right]\right]\right]$$

- 指定 pre_padding=(0,1)（w 维头部减少填充 1 层 0），输出张量形状 (1,1,5,4)

$$\left[\left[\left[\begin{matrix} 21000. & 32100. & 3200. & 300. \\ 54021. & 65432.1 & 6503.2 & 600.3 \\ 87054.02 & 98765.43 & 9806.503 & 900.6003 \\ 87.054 & 98.765396 & 9.8064995 & 0.9006 \\ 0.087 & 0.09869999 & 0.0098 & 0.0009 \end{matrix}\right]\right]\right]$$

post_padding

```
hPost = wPost = 1
deconvolutionLayer = network.add_deconvolution_nd(inputT0, cOut, (hW, wW), weight, bias)
deconvolutionLayer.post_padding = (hPost, wPost)  # 尾部减少填充 0 层数，默认值 (0,0)
```

- 指定 post_padding=(1,1)（HW 维尾部均减少填充 1 层 0），输出张量形状 (1,1,4,4)

$$\left[\left[\begin{matrix} 10000. & 21000. & 32100. & 3200. \\ 40010. & 54021. & 65432.1 & 6503.2 \\ 70040.01 & 87054.02 & 98765.43 & 9806.503 \\ 70.04 & 87.054 & 98.765396 & 9.8064995 \end{matrix}\right]\right]$$

- 指定 post_padding=(1,0)（H 维尾部减少填充 1 层 0），输出张量形状 (1,1,4,5)

$$\left[\left[\left[\begin{matrix} 10000. & 21000. & 32100. & 3200. & 300. \\ 40010. & 54021. & 65432.1 & 6503.2 & 600.3 \\ 70040.01 & 87054.02 & 98765.43 & 9806.503 & 900.6003 \\ 70.04 & 87.054 & 98.765396 & 9.8064995 & 0.9006 \end{matrix}\right]\right]\right]$$

- 指定 post_padding=(0,1)（W 维尾部减少填充 1 层 0），输出张量形状 (1,1,5,4)

$$\left[\left[\left[\begin{matrix} 10000. & 21000. & 32100. & 3200. \\ 40010. & 54021. & 65432.1 & 6503.2 \\ 70040.01 & 87054.02 & 98765.43 & 9806.503 \\ 70.04 & 87.054 & 98.765396 & 9.8064995 \\ 0.07 & 0.087 & 0.09869999 & 0.0098 \end{matrix}\right]\right]\right]$$

padding_mode

```
deconvolutionLayer = network.add_deconvolution_nd(inputT0, cOut, (hW, wW), weight, bias)
deconvolutionLayer.stride_nd = (2, 2)  # 加上卷积步长，以便观察结果
deconvolutionLayer.padding_mode = trt.PaddingMode.SAME_UPPER
```

- 计算过程参考 TensorRT C API reference
- 指定 padding_mode = **trt.PaddingMode.SAME_UPPER**，输出张量形状 (1,1,6,6)

$$\left[\left[\begin{bmatrix} 10000. & 1000. & 20100. & 2000. & 30200. & 3000. \\ 10. & 1. & 20.1 & 2. & 30.2 & 3. \\ 40000.01 & 4000.001 & 50400.02 & 5000.002 & 60500.03 & 6000.003 \\ 40. & 4. & 50.4 & 5. & 60.5 & 6. \\ 70000.04 & 7000.004 & 80700.05 & 8000.005 & 90800.06 & 9000.006 \\ 70. & 7. & 80.7 & 8. & 90.8 & 9. \end{bmatrix}\right]\right]$$

- 指定 padding_mode = **trt.PaddingMode.SAME_LOWER**，输出张量形状 (1,1,6,6)

$$\left[\left[\begin{bmatrix} s1. & 20.1 & 2. & 30.2 & 3. & 0.3 \\ 4000.001 & 50400.02 & 5000.002 & 60500.03 & 6000.003 & 600.0003 \\ 4. & 50.4 & 5. & 60.5 & 6. & 0.6 \\ 7000.004 & 80700.05 & 8000.005 & 90800.06 & 9000.006 & 900.0006 \\ 7. & 80.7 & 8. & 90.8 & 9. & 0.90000004 \\ 0.007 & 0.0807 & 0.008 & 0.09079999 & 0.009 & 0.0009 \end{bmatrix}\right]\right]$$

- 指定 padding_mode = **trt.PaddingMode.EXPLICIT_ROUND_UP**，输出张量形状 (1,1,7,7)

$$\left[\left[\begin{bmatrix} 10000. & 1000. & 20100. & 2000. & 30200. & 3000. & 300. \\ 10. & 1. & 20.1 & 2. & 30.2 & 3. & 0.3 \\ 40000.01 & 4000.001 & 50400.02 & 5000.002 & 60500.03 & 6000.003 & 600.0003 \\ 40. & 4. & 50.4 & 5. & 60.5 & 6. & 0.6 \\ 70000.04 & 7000.004 & 80700.05 & 8000.005 & 90800.06 & 9000.006 & 900.0006 \\ 70. & 7. & 80.7 & 8. & 90.8 & 9. & 0.90000004 \\ 0.07 & 0.007 & 0.0807 & 0.008 & 0.09079999 & 0.009 & 0.0009 \end{bmatrix}\right]\right]$$

- 指定 padding_mode = **trt.PaddingMode.EXPLICIT_ROUND_DOWN**，输出张量形状 (1,1,7,7)

$$\left[\left[\begin{bmatrix} 10000. & 1000. & 20100. & 2000. & 30200. & 3000. & 300. \\ 10. & 1. & 20.1 & 2. & 30.2 & 3. & 0.3 \\ 40000.01 & 4000.001 & 50400.02 & 5000.002 & 60500.03 & 6000.003 & 600.0003 \\ 40. & 4. & 50.4 & 5. & 60.5 & 6. & 0.6 \\ 70000.04 & 7000.004 & 80700.05 & 8000.005 & 90800.06 & 9000.006 & 900.0006 \\ 70. & 7. & 80.7 & 8. & 90.8 & 9. & 0.90000004 \\ 0.07 & 0.007 & 0.0807 & 0.008 & 0.09079999 & 0.009 & 0.0009 \end{bmatrix}\right]\right]$$

- 指定 padding_mode = **trt.PaddingMode.CAFFE_ROUND_UP**，输出张量形状 (1,1,7,7)

$$\left[\left[\begin{bmatrix} 10000. & 1000. & 20100. & 2000. & 30200. & 3000. & 300. \\ 10. & 1. & 20.1 & 2. & 30.2 & 3. & 0.3 \\ 40000.01 & 4000.001 & 50400.02 & 5000.002 & 60500.03 & 6000.003 & 600.0003 \\ 40. & 4. & 50.4 & 5. & 60.5 & 6. & 0.6 \\ 70000.04 & 7000.004 & 80700.05 & 8000.005 & 90800.06 & 9000.006 & 900.0006 \\ 70. & 7. & 80.7 & 8. & 90.8 & 9. & 0.90000004 \\ 0.07 & 0.007 & 0.0807 & 0.008 & 0.09079999 & 0.009 & 0.0009 \end{bmatrix}\right]\right]$$

- 指定 padding_mode = **trt.PaddingMode.CAFFE_ROUND_DOWN**，输出张量形状 (1,1,7,7)

$$\left[\left[\begin{bmatrix} 10000. & 1000. & 20100. & 2000. & 30200. & 3000. & 300. \\ 10. & 1. & 20.1 & 2. & 30.2 & 3. & 0.3 \\ 40000.01 & 4000.001 & 50400.02 & 5000.002 & 60500.03 & 6000.003 & 600.0003 \\ 40. & 4. & 50.4 & 5. & 60.5 & 6. & 0.6 \\ 70000.04 & 7000.004 & 80700.05 & 8000.005 & 90800.06 & 9000.006 & 900.0006 \\ 70. & 7. & 80.7 & 8. & 90.8 & 9. & 0.90000004 \\ 0.07 & 0.007 & 0.0807 & 0.008 & 0.09079999 & 0.009 & 0.0009 \end{bmatrix}\right]\right]$$

---

num_groups

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

nIn, cIn, hIn, wIn = 1, 2, 3, 3  # 调整部分输入输出参数
nGroup = 2
cOut, hW, wW = nGroup, 3, 3
data = np.arange(1, 1 + nIn * cIn * hIn * wIn, dtype=np.float32).reshape(nIn, cIn, hIn, wIn)
data = np.concatenate([data, data], 0)  # 输入张量通道数必须能被分组数整除
weight = np.power(10, range(4, -5, -1), dtype=np.float32)
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()
config.max_workspace_size = 1 << 30
inputT0 = network.add_input('inputT0', trt.DataType.FLOAT, (nIn, cIn, hIn, wIn))
deconvolutionLayer = network.add_deconvolution_nd(inputT0, cOut, (hW, wW), weight, bias)
deconvolutionLayer.num_groups = nGroup  # 分组数，默认值 1
network.mark_output(deconvolutionLayer.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)
- 输出张量形状 (2, 4, 7)，其中指定 num_groupds=2，输入张量和反卷积核均在 C 维上被均分为 2 组，各自反卷积后再拼接到一起

$$\left[\left[\begin{matrix}\begin{bmatrix} 10000. & 21000. & 32100. & 3200. & 300. \\ 40010. & 54021. & 65432.1 & 6503.2 & 600.3 \\ 70040.01 & 87054.02 & 98765.43 & 9806.503 & 900.6003 \\ 70.04 & 87.054 & 98.76539 & 9.8064995 & 0.9006 \\ 0.07 & 0.087 & 0.09869999 & 0.0098 & 0.0009 \end{bmatrix} \\ \begin{bmatrix} -10000. & -21000. & -32100. & -3200. & -300. \\ -40010. & -54021. & -65432.1 & -6503.2 & -600.3 \\ -70040.01 & -87054.02 & -98765.43 & -9806.503 & -900.6003 \\ -70.04 & -87.054 & -98.76539 & -9.8064995 & -0.9006 \\ -0.07 & -0.087 & -0.09869999 & -0.0098 & -0.0009 \end{bmatrix}\end{matrix}\right]\right]$$

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

三维反卷积的示例

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

nIn, cIn, hIn, wIn = 1, 2, 3, 3  # 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)
weight = np.concatenate([weight, -weight], 0).reshape(cIn, 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.max_workspace_size = 1 << 30
inputT0 = network.add_input('inputT0', trt.DataType.FLOAT, (nIn, 1, cIn, hIn, wIn))  # 要求输入至少为 5 维
deconvolutionLayer = network.add_deconvolution_nd(inputT0, cOut, weight.shape, weight, bias)  # 卷积核是
3 维的
network.mark_output(deconvolutionLayer.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 (1,1,3,5,5)，C 维中间层的结果相当于两端的两个通道加在一起，得到了 0 的结果

$$
\left[\left[\left[\begin{matrix}
\begin{bmatrix}
10000. & 21000. & 32100. & 3200. & 300. \\
40010. & 54021. & 65432.1 & 6503.2 & 600.3 \\
70040.01 & 87054.02 & 98765.43 & 9806.503 & 900.6003 \\
70.04 & 87.054 & 98.76539 & 9.8064995 & 0.9006 \\
0.07 & 0.087 & 0.09869999 & 0.0098 & 0.0009
\end{bmatrix} \\
\begin{bmatrix}
0. & 0. & 0. & 0. & 0. \\
0. & 0. & 0. & 0. & 0. \\
0. & 0.00099945 & 0.0019989 & 0.00007031 & 0. \\
0. & 0.00000525 & 0. & 0.00000014 & 0. \\
0. & 0. & 0. & 0. & 0.
\end{bmatrix} \\
\begin{bmatrix}
-10000. & -21000. & -32100. & -3200. & -300. \\
-40010. & -54021. & -65432.1 & -6503.2 & -600.3 \\
-70040.01 & -87054.02 & -98765.43 & -9806.503 & -900.6003 \\
-70.04 & -87.054 & -98.76539 & -9.8064995 & -0.9006 \\
-0.07 & -0.087 & -0.09869999 & -0.0098 & -0.0009
\end{bmatrix}
\end{matrix}\right]\right]\right]
$$

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, 3, 3
cOut, hW, wW = 1, 3, 3
data = np.arange(1, 1 + nIn * cIn * hIn * wIn, dtype=np.float32).reshape(nIn, cIn, hIn, wIn)
weight = np.power(10, range(4, -5, -1), dtype=np.float32)
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.max_workspace_size = 1 << 30
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

deconvolutionLayer = network.add_deconvolution_nd(dequantizeLayer0.get_output(0), cOut, (hW, wW),
trt.Weights())  # 需要把 weight 设为空权重（不能用 np.array()）
deconvolutionLayer.set_input(1, dequantizeLayer1.get_output(0))
#----------------------------------------------------------- --------------------# 替换部分
network.mark_output(deconvolutionLayer.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 (1,1,5,5)

$$\left[\left[\begin{matrix} 127. & 381. & 735. & 581. & 300. \\ 518. & 1164. & 1829. & 1265. & 600. \\ 929. & 1959. & 2924. & 1949. & 900. \\ 0. & 0. & 0. & 0. & 0. \end{matrix}\right]\right]$$