PoolingNd 层（Pooling 层）

- **参数及示例（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
- type
- blend_factor
- window_size_nd (window_size)
- stride_nd (stride)
- padding_nd (padding)
- pre_padding
- post_padding
- padding_mode
- average_count_excludes_padding
- 三维池化的示例

初始示例代码

```
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
hW, wW = 2, 2  # 池化窗口 HW
data = np.tile(np.arange(1, 1 + 9, dtype=np.float32).reshape(1, 3, 3), (nIn, cIn, hIn // 3, wIn // 3))
 # 输入数据

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
poolLayer = network.add_pooling_nd(inputT0, trt.PoolingType.MAX, (hW, wW))
#------------------------------------------------------------ --------------------# 替换部分
network.mark_output(poolLayer.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)

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

- 输出张量形状 (1,1,3,4)

$$\left[\left[\begin{bmatrix} \color{green}{5.} & \color{blue}{6.} & 6. & 5. \\ 8. & 9. & 9. & 8. \\ 8. & 9. & 9. & 8. \end{bmatrix}\right]\right]$$

- 计算过程:

$$\begin{bmatrix} \boxed{\begin{matrix} 1 & 2 \\ 4 & 5 \end{matrix}} & \begin{matrix} 3 \\ 6 \end{matrix} & \cdots \\ \begin{matrix} 7 & 8 \end{matrix} & 9 & \cdots \\ \vdots & \vdots & \end{bmatrix} = \color{green}{5.}, \begin{bmatrix} \begin{matrix} 1 \\ 4 \end{matrix} & \boxed{\begin{matrix} 2 & 3 \\ 5 & 6 \end{matrix}} & \cdots \\ 7 & \begin{matrix} 8 & 9 \end{matrix} & \cdots \\ \vdots & \vdots & \end{bmatrix} = \color{blue}{6.}$$

type

```
poolLayer = network.add_pooling_nd(inputT0, trt.PoolingType.MAX, (hW, wW))
poolLayer.type = trt.PoolingType.AVERAGE
```

- 指定平均值池化，输出张量形状 (1,1,3,4)

$$\left[\left[\begin{bmatrix} 2. & 2.5 & 3. & 2. \\ 3.5 & 4. & 4.5 & 3.5 \\ 5. & 5.5 & 6. & 5. \end{bmatrix}\right]\right]$$

- 可用的池化方式

trt.PoolingType 名	说明
AVERAGE	均值池化
MAX	最大值池化
MAX_AVERAGE_BLEND	混合池化，见下面的 blend_factor 部分

blend_factor

```
bF = 0.5
poolLayer = network.add_pooling_nd(inputT0, trt.PoolingType.MAX_AVERAGE_BLEND, (hW, wW))
poolLayer.blend_factor = bF  # 均值池化的比例，默认值 1.0
```

- 指定 blend_factor=0.5，输出张量形状 (1,1,3,4)

$$\left[\left[\left[\begin{matrix} 4. & 4.75 & 5. & 4. \\ 6.25 & 7. & 7.25 & 6.25 \\ 7. & 7.75 & 8. & 7. \end{matrix}\right]\right]\right]$$

- 计算过程：

$$bF \cdot \left[\left[\left[\begin{matrix} 3. & 3.5 & 4. & 3. \\ 4.5 & 5. & 5.5 & 4.5 \\ 6. & 6.5 & 7. & 6. \end{matrix}\right]\right]\right] + (1-bF) \cdot \left[\left[\left[\begin{matrix} 5. & 6. & 6. & 5. \\ 8. & 9. & 9. & 8. \\ 8. & 9. & 9. & 8. \end{matrix}\right]\right]\right] = \left[\left[\left[\begin{matrix} 4. & 4.75 & 5. & 4. \\ 6.25 & 7. & 7.25 & 6.25 \\ 7. & 7.75 & 8. & 7. \end{matrix}\right]\right]\right]$$

---

window_size_nd (window_size)

```
poolLayer = network.add_pooling_nd(inputT0, trt.PoolingType.MAX, (1, 1))  # 默认池化窗口大小和步长都设为给定参数
poolLayer.window_size_nd = (hW, wW)  # 池化窗口大小
```

- 输出张量形状 (1,1,5,8)，池化窗口尺寸不变，但是步长会保持 (1,1)

$$\left[\left[\left[\begin{matrix} 4. & 5. & 5. & 4. & 5. & 5. & 4. & 5. \\ 7. & 8. & 8. & 7. & 8. & 8. & 7. & 8. \\ 7. & 8. & 8. & 7. & 8. & 8. & 7. & 8. \\ 4. & 5. & 5. & 4. & 5. & 5. & 4. & 5. \\ 7. & 8. & 8. & 7. & 8. & 8. & 7. & 8. \end{matrix}\right]\right]\right]$$

- 使用旧版 API `window_size` 会收到警告

```
DeprecationWarning: Use window_size_nd instead.
```

---

stride_nd (stride)

```
hS = wS = 1
poolLayer = network.add_pooling_nd(inputT0, trt.PoolingType.MAX, (hW, wW))
poolLayer.stride_nd = (hS, wS)  # 池化窗口步长
```

- 指定 stride=(hS,wS)，输出张量形状 (1,1,5,8)，

$$\left[\left[\left[\begin{matrix} 5. & 6. & 6. & 5. & 6. & 6. & 5. & 6. \\ 8. & 9. & 9. & 8. & 9. & 9. & 8. & 9. \\ 8. & 9. & 9. & 8. & 9. & 9. & 8. & 9. \\ 5. & 6. & 6. & 5. & 6. & 6. & 5. & 6. \\ 8. & 9. & 9. & 8. & 9. & 9. & 8. & 9. \end{matrix}\right]\right]\right]$$

- 使用旧版 API `stride` 会收到警告

```
DeprecationWarning: Use stride_nd instead.
```

---

padding_nd (padding)

```
hP = wP = 1
poolLayer = network.add_pooling_nd(inputT0, trt.PoolingType.MAX, (hW, wW))
poolLayer.padding_nd = (hP, wP)  # 四周填充 0 层数，默认值 (0,0)
```

- 指定 padding=(1,1)(HW 维均填充 1 层 0），输出张量形状 (1,1,4,5)

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

- 指定 padding=(1,0)(H 维填充 1 层 0），输出张量形状 (1,1,4,4)

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

- 指定 padding=(0,1)(W 维填充 1 层 0），输出张量形状 (1,1,3,5)

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

- 使用旧版 API `padding` 会收到警告

```
DeprecationWarning: Use padding_nd instead.
```

---

pre_padding

```
hPre = wPre = 1
poolLayer = network.add_pooling_nd(inputT0, trt.PoolingType.MAX, (hW, wW))
poolLayer.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 (1,1,3,5)

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

- 指定 pre_padding=(1,0)（H 维头部填充 1 层 0），输出张量形状 (1,1,3,4)

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

- 指定 pre_padding=(0,1)（W 维头部填充 1 层 0），输出张量形状 (1,1,3,5)

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

---

post_padding

```
hPost = wPost = 1
poolLayer = network.add_pooling_nd(inputT0, trt.PoolingType.MAX, (hW, wW))
poolLayer.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 (1,1,3,5)

$$\left[\left[\begin{bmatrix} 5. & 6. & 6. & 5. & 6. \\ 8. & 9. & 9. & 8. & 9. \\ 8. & 9. & 9. & 8. & 9. \end{bmatrix}\right]\right]$$

- 指定 post_padding=(1,0)（H 维尾部填充 1 层 0），输出张量形状 (1,1,3,4)

$$\left[\left[\begin{bmatrix} 5. & 6. & 6. & 5. \\ 8. & 9. & 9. & 8. \\ 8. & 9. & 9. & 8. \end{bmatrix}\right]\right]$$

- 指定 post_padding=(0,1)（W 维尾部填充 1 层 0），输出张量形状 (1,1,3,5)

$$\left[\left[\begin{bmatrix} 5. & 6. & 6. & 5. & 6. \\ 8. & 9. & 9. & 8. & 9. \\ 8. & 9. & 9. & 8. & 9. \end{bmatrix}\right]\right]$$

---

padding_mode

```
inputT0 = network.add_input('inputT0', trt.DataType.FLOAT, (nIn, cIn, hIn - 1, wIn))  # 去除输入张量的最后一行，以便观察结果
poolLayer = network.add_pooling_nd(inputT0, trt.PoolingType.MAX, (hW, wW))
poolLayer.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 (1,1,3,5)

$$\left[\left[\begin{bmatrix} 5. & 6. & 6. & 5. & 6. \\ 8. & 9. & 9. & 8. & 9. \\ 5. & 6. & 6. & 5. & 6. \end{bmatrix}\right]\right]$$

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

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

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

$$\left[\left[\begin{bmatrix} 5. & 6. & 6. & 5. & 6. \\ 8. & 9. & 9. & 8. & 9. \\ 5. & 6. & 6. & 5. & 6. \end{bmatrix}\right]\right]$$

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

$$\left[\left[\begin{bmatrix} 5. & 6. & 6. & 5. \\ 8. & 9. & 9. & 8. \end{bmatrix}\right]\right]$$

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

$$\left[\left[\begin{bmatrix} 5. & 6. & 6. & 5. & 6. \\ 8. & 9. & 9. & 8. & 9. \\ 5. & 6. & 6. & 5. & 6. \end{bmatrix}\right]\right]$$

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

$$\left[\left[\begin{bmatrix}5. & 6. & 6. & 5.\\ 8. & 9. & 9. & 8.\end{bmatrix}\right]\right]$$

---

average_count_excludes_padding

```python
poolLayer = network.add_pooling_nd(inputT0, trt.PoolingType.AVERAGE, (hW, wW))
poolLayer.padding_nd = (1, 1)  # 不支持 pre_padding 或 post_padding
poolLayer.average_count_excludes_padding = False  # 是否排除分母中填充 0 的计数，默认值 True
```

- 指定 average_count_excludes_padding=False，均值池化连着填充的 0 一起计算（默认填充的 0 不计入分母），输出张量形状 (1,1,4,5)

$$\left[\left[\begin{bmatrix}0.25 & 1.25 & 0.75 & 1. & 1.25\\ 2.75 & 7. & 6. & 6.5 & 7.\\ 1.25 & 4. & 3. & 3.5 & 4.\\ 1.75 & 4.25 & 3.75 & 4. & 4.25\end{bmatrix}\right]\right]$$

三维池化的示例

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

nIn, cIn, hIn, wIn = 1, 2, 6, 9  # 输入张量 NCHW
cW, hW, wW = 2, 2, 2  # 池化窗口 HW
data = np.tile(np.arange(1, 1 + 9, dtype=np.float32).reshape(3, 3), (2, 2, 3)).reshape(nIn, 1, cIn, hIn, wIn)
data[0, 0, 1] *= 10

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))
#---------------------------------------------------------- --------------------# 替换部分
poolLayer = network.add_pooling_nd(inputT0, trt.PoolingType.MAX, (cW, hW, wW))
#---------------------------------------------------------- --------------------# 替换部分
network.mark_output(poolLayer.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)
```

```python
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 (1,1,2,6,9)

$$\left[\left[\left[\begin{matrix}1. & 2. & 3. & 1. & 2. & 3. & 1. & 2. & 3.\\ 4. & 5. & 6. & 4. & 5. & 6. & 4. & 5. & 6.\\ 7. & 8. & 9. & 7. & 8. & 9. & 7. & 8. & 9.\\ 1. & 2. & 3. & 1. & 2. & 3. & 1. & 2. & 3.\\ 4. & 5. & 6. & 4. & 5. & 6. & 4. & 5. & 6.\\ 7. & 8. & 9. & 7. & 8. & 9. & 7. & 8. & 9.\end{matrix}\right]\left[\begin{matrix}10. & 20. & 30. & 10. & 20. & 30. & 10. & 20. & 30.\\ 40. & 50. & 60. & 40. & 50. & 60. & 40. & 50. & 60.\\ 70. & 80. & 90. & 70. & 80. & 90. & 70. & 80. & 90.\\ 10. & 20. & 30. & 10. & 20. & 30. & 10. & 20. & 30.\\ 40. & 50. & 60. & 40. & 50. & 60. & 40. & 50. & 60.\\ 70. & 80. & 90. & 70. & 80. & 90. & 70. & 80. & 90.\end{matrix}\right]\right]\right]$$

- 输出张量形状 (1,1,1,3,4)，最大元素全都来自靠后的通道

$$\left[\left[\left[\left[\begin{matrix}50. & 60. & 60. & 50.\\ 80. & 90. & 90. & 80.\\ 80. & 90. & 90. & 80.\end{matrix}\right]\right]\right]\right]$$