Table of Contents

CONV2D

CLASS `torch.nn.Conv2d(in_channels: int, out_channels: int, kernel_size: Union[T, Tuple[T, T]], stride: Union[T, Tuple[T, T]] = 1, padding: Union[T, Tuple[T, T]] = 0, dilation: Union[T, Tuple[T, T]] = 1, groups: int = 1, bias: bool = True, padding_mode: str = 'zeros')` [SOURCE]

Applies a 2D convolution over an input signal composed of several input planes.

In the simplest case, the output value of the layer with input size (N, C_{in}, H, W) and output $(N, C_{\text{out}}, H_{\text{out}}, W_{\text{out}})$ can be precisely described as:

$$\text{out}(N_i, C_{\text{out}_j}) = \text{bias}(C_{\text{out}_j}) + \sum_{k=0}^{C_{\text{in}}-1} \text{weight}(C_{\text{out}_j}, k) \star \text{input}(N_i, k)$$

where $\star$ is the valid 2D cross-correlation operator, N is a batch size, C denotes a number of channels, H is a height of input planes in pixels, and W is width in pixels.

This module supports TensorFloat32.

- `stride` controls the stride for the cross-correlation, a single number or a tuple.
- `padding` controls the amount of implicit zero-paddings on both sides for `padding` number of points for each dimension.
- `dilation` controls the spacing between the kernel points; also known as the à trous algorithm. It is harder to describe, but this link has a nice visualization of what `dilation` does.
- `groups` controls the connections between inputs and outputs. `in_channels` and `out_channels` must both be divisible by `groups`. For example,
  - At groups=1, all inputs are convolved to all outputs.
  - At groups=2, the operation becomes equivalent to having two conv layers side by side, each seeing half the input channels, and producing half the output channels, and both subsequently concatenated.
  - At groups= `in_channels`, each input channel is convolved with its own set of filters, of size: $\left\lfloor\frac{out\_channels}{in\_channels}\right\rfloor$.

The parameters `kernel_size`, `stride`, `padding`, `dilation` can either be:

- a single `int` – in which case the same value is used for the height and width dimension
- a `tuple` of two ints – in which case, the first *int* is used for the height dimension, and the second *int* for the width dimension

• NOTE

Depending of the size of your kernel, several (of the last) columns of the input might be lost, because it is a valid cross-correlation, and not a full cross-correlation. It is up to the user to add proper padding.

• NOTE

When *groups == in_channels* and *out_channels == K * in_channels*, where *K* is a positive integer, this operation is also termed in literature as depthwise convolution.

In other words, for an input of size $(N, C_{in}, H_{in}, W_{in})$, a depthwise convolution with a depthwise multiplier *K*, can be constructed by arguments $(in\_channels = C_{in}, out\_channels = C_{in} \times K, ..., groups = C_{in})$.

• NOTE

In some circumstances when using the CUDA backend with CuDNN, this operator may select a nondeterministic algorithm to increase performance. If this is undesirable, you can try to make the operation deterministic (potentially at a performance cost) by setting `torch.backends.cudnn.deterministic = True`. Please see the notes on Reproducibility for background.

Parameters

- **in_channels** (*int*) – Number of channels in the input image
- **out_channels** (*int*) – Number of channels produced by the convolution
- **kernel_size** (*int or tuple*) – Size of the convolving kernel
- **stride** (*int or tuple, optional*) – Stride of the convolution. Default: 1

- **padding** (*int* or *tuple*, *optional*) – Zero-padding added to both sides of the input. Default: 0
- **padding_mode** (*string*, *optional*) – `'zeros'`, `'reflect'`, `'replicate'` or `'circular'`. Default: `'zeros'`
- **dilation** (*int* or *tuple*, *optional*) – Spacing between kernel elements. Default: 1
- **groups** (*int*, *optional*) – Number of blocked connections from input channels to output channels. Default: 1
- **bias** (*bool*, *optional*) – If `True`, adds a learnable bias to the output. Default: `True`

Shape:

- Input: $(N, C_{in}, H_{in}, W_{in})$
- Output: $(N, C_{out}, H_{out}, W_{out})$ where

$$H_{out} = \left\lfloor \frac{H_{in} + 2 \times \text{padding}[0] - \text{dilation}[0] \times (\text{kernel\_size}[0] - 1) - 1}{\text{stride}[0]} + 1 \right\rfloor$$

$$W_{out} = \left\lfloor \frac{W_{in} + 2 \times \text{padding}[1] - \text{dilation}[1] \times (\text{kernel\_size}[1] - 1) - 1}{\text{stride}[1]} + 1 \right\rfloor$$

Variables

- **~Conv2d.weight** (*Tensor*) – the learnable weights of the module of shape $(\text{out\_channels}, \frac{\text{in\_channels}}{\text{groups}}, \text{kernel\_size}[0], \text{kernel\_size}[1])$. The values of these weights are sampled from $\mathcal{U}(-\sqrt{k}, \sqrt{k})$ where $k = \frac{groups}{C_{\text{in}} * \prod_{i=0}^{1} \text{kernel\_size}[i]}$
- **~Conv2d.bias** (*Tensor*) – the learnable bias of the module of shape (out_channels). If `bias` is `True`, then the values of these weights are sampled from $\mathcal{U}(-\sqrt{k}, \sqrt{k})$ where $k = \frac{groups}{C_{\text{in}} * \prod_{i=0}^{1} \text{kernel\_size}[i]}$

Examples

```
>>> # With square kernels and equal stride
>>> m = nn.Conv2d(16, 33, 3, stride=2)
>>> # non-square kernels and unequal stride and with padding
>>> m = nn.Conv2d(16, 33, (3, 5), stride=(2, 1), padding=(4, 2))
>>> # non-square kernels and unequal stride and with padding and dilation
>>> m = nn.Conv2d(16, 33, (3, 5), stride=(2, 1), padding=(4, 2), dilation=(3, 1))
>>> input = torch.randn(20, 16, 50, 100)
>>> output = m(input)
```

< Previous Next >