


Branch: master ▾

pytorch / torch / nn / modules / conv.py / Jump to ▾


Find file

Copy path

 driazati Remove weak script (#22212)


10c4b98 12 days ago

40 contributors

 and others

934 lines (758 sloc) 45.1 KB

RawBlameHistory



Jump to definition is available! Beta

Navigate your code with ease. In select public repositories, you can now click on function and method calls to jump to their definitions in the same repository. [Learn more](#)

You're using jump to definition to discover and navigate code. [Opt out](#) Beta

[Learn more](#) or [give us feedback](#)

1# coding=utf-8

2import math

3import torch

4from torch.nn.parameter import Parameter

5from .. import functional as F

6from .. import init

7from .module import Module

8from .utils import _single, _pair, _triple

9from ..._jit_internal import List

10

11

12class _ConvNd(Module):

13

14 __constants__ = ['stride', 'padding', 'dilation', 'groups', 'bias',

15 'padding_mode', 'output_padding', 'in_channels',

16 'out_channels', 'kernel_size']

17

18 def __init__(self, in_channels, out_channels, kernel_size, stride,

19 padding, dilation, transposed, output_padding,

20 groups, bias, padding_mode):

21 super(_ConvNd, self).__init__()

22 if in_channels % groups != 0:

23 raise ValueError('in_channels must be divisible by groups')

24 if out_channels % groups != 0:

25 raise ValueError('out_channels must be divisible by groups')

26 self.in_channels = in_channels

27 self.out_channels = out_channels

28 self.kernel_size = kernel_size

```

29     self.stride = stride
30     self.padding = padding
31     self.dilation = dilation
32     self.transposed = transposed
33     self.output_padding = output_padding
34     self.groups = groups
35     self.padding_mode = padding_mode
36     if transposed:
37         self.weight = Parameter(torch.Tensor(
38             in_channels, out_channels // groups, *kernel_size))
39     else:
40         self.weight = Parameter(torch.Tensor(
41             out_channels, in_channels // groups, *kernel_size))
42     if bias:
43         self.bias = Parameter(torch.Tensor(out_channels))
44     else:
45         self.register_parameter('bias', None)
46     self.reset_parameters()
47
48     def reset_parameters(self):
49         init.kaiming_uniform_(self.weight, a=math.sqrt(5))
50         if self.bias is not None:
51             fan_in, _ = init._calculate_fan_in_and_fan_out(self.weight)
52             bound = 1 / math.sqrt(fan_in)
53             init.uniform_(self.bias, -bound, bound)
54
55     def extra_repr(self):
56         s = ('{in_channels}, {out_channels}, kernel_size={kernel_size}'
57             ', stride={stride}')
58         if self.padding != (0,) * len(self.padding):
59             s += ', padding={padding}'
60         if self.dilation != (1,) * len(self.dilation):
61             s += ', dilation={dilation}'
62         if self.output_padding != (0,) * len(self.output_padding):
63             s += ', output_padding={output_padding}'
64         if self.groups != 1:
65             s += ', groups={groups}'
66         if self.bias is None:
67             s += ', bias=False'
68         return s.format(**self.__dict__)
69
70     def __setstate__(self, state):
71         super(_ConvNd, self).__setstate__(state)
72         if not hasattr(self, 'padding_mode'):
73             self.padding_mode = 'zeros'
74
75
76     class Conv1d(_ConvNd):
77         r"""Applies a 1D convolution over an input signal composed of several input
78         planes.
79
80         In the simplest case, the output value of the layer with input size

```

```

81 :math:`(N, C_{\text{in}}, L)` and output :math:`(N, C_{\text{out}}, L_{\text{out}})` can be
82 precisely described as:
83
84 .. math::
85     \text{out}(N_i, C_{\text{out}_j}) = \text{bias}(C_{\text{out}_j}) +
86     \sum_{k=0}^{C_{\text{in}}-1} \text{weight}(C_{\text{out}_j}, k)
87     \star \text{input}(N_i, k)
88
89 where :math:`\star` is the valid `cross-correlation`_ operator,
90 :math:`N` is a batch size, :math:`C` denotes a number of channels,
91 :math:`L` is a length of signal sequence.
92
93 * :attr:`stride` controls the stride for the cross-correlation, a single
94   number or a one-element tuple.
95
96 * :attr:`padding` controls the amount of implicit zero-paddings on both sides
97   for :attr:`padding` number of points.
98
99 * :attr:`dilation` controls the spacing between the kernel points; also
100   known as the à trous algorithm. It is harder to describe, but this `link`_
101   has a nice visualization of what :attr:`dilation` does.
102
103 * :attr:`groups` controls the connections between inputs and outputs.
104   :attr:`in_channels` and :attr:`out_channels` must both be divisible by
105   :attr:`groups`. For example,
106
107   * At groups=1, all inputs are convolved to all outputs.
108   * At groups=2, the operation becomes equivalent to having two conv
109     layers side by side, each seeing half the input channels,
110     and producing half the output channels, and both subsequently
111     concatenated.
112   * At groups= :attr:`in_channels`, each input channel is convolved with
113     its own set of filters,
114     of size
115     :math:`\left\lfloor \frac{\text{out\_channels}}{\text{in\_channels}} \right\rfloor`.
116
117 .. note::
118
119     Depending of the size of your kernel, several (of the last)
120     columns of the input might be lost, because it is a valid
121     `cross-correlation`_, and not a full `cross-correlation`_.
122     It is up to the user to add proper padding.
123
124 .. note::
125
126     When `groups == in_channels` and `out_channels == K * in_channels`,
127     where `K` is a positive integer, this operation is also termed in
128     literature as depthwise convolution.
129
130     In other words, for an input of size :math:`(N, C_{\text{in}}, L_{\text{in}})`,
131     a depthwise convolution with a depthwise multiplier `K`, can be constructed by arguments
132     :math:`(C_{\text{in}}=C_{\text{in}}, C_{\text{out}}=C_{\text{in}} \times K, \dots, \text{groups}=C_{\text{in}})`.
```

```

133
134 .. include:: cudnn_deterministic.rst
135
136 Args:
137     in_channels (int): Number of channels in the input image
138     out_channels (int): Number of channels produced by the convolution
139     kernel_size (int or tuple): Size of the convolving kernel
140     stride (int or tuple, optional): Stride of the convolution. Default: 1
141     padding (int or tuple, optional): Zero-padding added to both sides of
142         the input. Default: 0
143     padding_mode (string, optional). Accepted values `zeros` and `circular` Default: `zeros`
144     dilation (int or tuple, optional): Spacing between kernel
145         elements. Default: 1
146     groups (int, optional): Number of blocked connections from input
147         channels to output channels. Default: 1
148     bias (bool, optional): If ``True``, adds a learnable bias to the output. Default: ``True``
149
150 Shape:
151     - Input:  $(N, C_{in}, L_{in})$ 
152     - Output:  $(N, C_{out}, L_{out})$  where
153
154     .. math::
155         L_{out} = \left\lfloor \frac{L_{in} + 2 \times \text{padding} - \text{dilation}}{\text{kernel\_size} - 1 - 1 \times \text{stride}} + 1 \right\rfloor
156
157
158 Attributes:
159     weight (Tensor): the learnable weights of the module of shape
160          $(\text{out\_channels}, \frac{\text{in\_channels}}{\text{groups}}, \text{kernel\_size})$ .
161         The values of these weights are sampled from
162          $\mathcal{U}(-\sqrt{k}, \sqrt{k})$  where
163          $k = \frac{1}{C_{in} * \text{kernel\_size}}$ 
164     bias (Tensor): the learnable bias of the module of shape
165         (out_channels). If :attr:`bias` is ``True``, then the values of these weights are
166         sampled from  $\mathcal{U}(-\sqrt{k}, \sqrt{k})$  where
167          $k = \frac{1}{C_{in} * \text{kernel\_size}}$ 
168
169 Examples::
170
171     >>> m = nn.Conv1d(16, 33, 3, stride=2)
172     >>> input = torch.randn(20, 16, 50)
173     >>> output = m(input)
174
175 .. _cross-correlation:
176     https://en.wikipedia.org/wiki/Cross-correlation
177
178 .. _link:
179     https://github.com/vdumoulin/conv_arithmetic/blob/master/README.md
180
181
182 def __init__(self, in_channels, out_channels, kernel_size, stride=1,
183             padding=0, dilation=1, groups=1,
184             bias=True, padding_mode='zeros'):

```

```

185     kernel_size = _single(kernel_size)
186     stride = _single(stride)
187     padding = _single(padding)
188     dilation = _single(dilation)
189     super(Conv1d, self).__init__(
190         in_channels, out_channels, kernel_size, stride, padding, dilation,
191         False, _single(0), groups, bias, padding_mode)
192
193     def forward(self, input):
194         if self.padding_mode == 'circular':
195             expanded_padding = ((self.padding[0] + 1) // 2, self.padding[0] // 2)
196             return F.conv1d(F.pad(input, expanded_padding, mode='circular'),
197                             self.weight, self.bias, self.stride,
198                             _single(0), self.dilation, self.groups)
199         return F.conv1d(input, self.weight, self.bias, self.stride,
200                         self.padding, self.dilation, self.groups)
201
202
203 class Conv2d(_ConvNd):
204     r"""Applies a 2D convolution over an input signal composed of several input
205     planes.
206
207     In the simplest case, the output value of the layer with input size
208     :math:`(N, C_{\text{in}}, H, W)` and output :math:`(N, C_{\text{out}}, H_{\text{out}}, W_{\text{out}})`
209     can be precisely described as:
210
211     .. math::
212         \text{out}(N_i, C_{\text{out}}_j) = \text{bias}(C_{\text{out}}_j) +
213         \sum_{k=0}^{C_{\text{in}} - 1} \text{weight}(C_{\text{out}}_j, k) \star \text{input}(N_i, k)
214
215     where :math:`\star` is the valid 2D `cross-correlation`_ operator,
216     :math:`N` is a batch size, :math:`C` denotes a number of channels,
217     :math:`H` is a height of input planes in pixels, and :math:`W` is
218     width in pixels.
219
220
221     * :attr:`stride` controls the stride for the cross-correlation, a single
222       number or a tuple.
223
224     * :attr:`padding` controls the amount of implicit zero-paddings on both
225       sides for :attr:`padding` number of points for each dimension.
226
227     * :attr:`dilation` controls the spacing between the kernel points; also
228       known as the à trous algorithm. It is harder to describe, but this `link`_
229       has a nice visualization of what :attr:`dilation` does.
230
231     * :attr:`groups` controls the connections between inputs and outputs.
232       :attr:`in_channels` and :attr:`out_channels` must both be divisible by
233       :attr:`groups`. For example,
234
235       * At groups=1, all inputs are convolved to all outputs.
236       * At groups=2, the operation becomes equivalent to having two conv

```

```

237     layers side by side, each seeing half the input channels,
238     and producing half the output channels, and both subsequently
239     concatenated.
240     * At groups= :attr:`in_channels`, each input channel is convolved with
241     its own set of filters, of size:
242     :math:`\left\lfloor \frac{\text{out\_channels}}{\text{in\_channels}} \right\rfloor`.
243
244     The parameters :attr:`kernel_size`, :attr:`stride`, :attr:`padding`, :attr:`dilation` can either be:
245
246     - a single ``int`` -- in which case the same value is used for the height and width dimension
247     - a ``tuple`` of two ints -- in which case, the first ``int`` is used for the height dimension,
248       and the second ``int`` for the width dimension
249
250     .. note::
251
252         Depending of the size of your kernel, several (of the last)
253         columns of the input might be lost, because it is a valid `cross-correlation`,
254         and not a full `cross-correlation`.
255         It is up to the user to add proper padding.
256
257     .. note::
258
259         When `groups == in_channels` and `out_channels == K * in_channels`,
260         where `K` is a positive integer, this operation is also termed in
261         literature as depthwise convolution.
262
263         In other words, for an input of size :math:`(N, C_{in}, H_{in}, W_{in})`,
264         a depthwise convolution with a depthwise multiplier `K`, can be constructed by arguments
265         :math:`(in\_channels=C_{in}, out\_channels=C_{in} \times K, ..., groups=C_{in})`.
266
267     .. include:: cudnn_deterministic.rst
268
269     Args:
270         in_channels (int): Number of channels in the input image
271         out_channels (int): Number of channels produced by the convolution
272         kernel_size (int or tuple): Size of the convolving kernel
273         stride (int or tuple, optional): Stride of the convolution. Default: 1
274         padding (int or tuple, optional): Zero-padding added to both sides of the input. Default: 0
275         padding_mode (string, optional). Accepted values `zeros` and `circular` Default: `zeros`
276         dilation (int or tuple, optional): Spacing between kernel elements. Default: 1
277         groups (int, optional): Number of blocked connections from input channels to output channels. Default: 1
278         bias (bool, optional): If ``True``, adds a learnable bias to the output. Default: ``True``
279
280     Shape:
281     - Input: :math:`(N, C_{in}, H_{in}, W_{in})`
282     - Output: :math:`(N, C_{out}, H_{out}, W_{out})` where
283
284         .. math::
285             H_{out} = \left\lfloor \frac{H_{in} + 2 \times \text{padding}[0] - \text{dilation}[0]}{\text{kernel\_size}[0] - 1} - 1 \right\rfloor \times \text{stride}[0] + 1
286
287         .. math::
288

```

```

289         W_{out} = \left\lfloor\frac{W_{in} + 2 \times \text{padding}[1] - \text{dilation}[1]}
290             \times (\text{kernel\_size}[1] - 1) - 1\right\rfloor \times \text{stride}[1] + 1\right\rfloor
291
292     Attributes:
293         weight (Tensor): the learnable weights of the module of shape
294             :math:`(\text{out\_channels}, \frac{\text{in\_channels}}{\text{groups}},`
295             :math:`\text{kernel\_size}[0], \text{kernel\_size}[1])` .
296             The values of these weights are sampled from
297             :math:`\mathcal{U}(-\sqrt{k}, \sqrt{k})` where
298             :math:`k = \frac{1}{C_{\text{in}}} * \prod_{i=0}^1 \text{kernel\_size}[i]`
299         bias (Tensor): the learnable bias of the module of shape (out_channels). If :attr:`bias` is ``True``,
300             then the values of these weights are
301             sampled from :math:`\mathcal{U}(-\sqrt{k}, \sqrt{k})` where
302             :math:`k = \frac{1}{C_{\text{in}}} * \prod_{i=0}^1 \text{kernel\_size}[i]`
303
304     Examples::
305
306         >>> # With square kernels and equal stride
307         >>> m = nn.Conv2d(16, 33, 3, stride=2)
308         >>> # non-square kernels and unequal stride and with padding
309         >>> m = nn.Conv2d(16, 33, (3, 5), stride=(2, 1), padding=(4, 2))
310         >>> # non-square kernels and unequal stride and with padding and dilation
311         >>> m = nn.Conv2d(16, 33, (3, 5), stride=(2, 1), padding=(4, 2), dilation=(3, 1))
312         >>> input = torch.randn(20, 16, 50, 100)
313         >>> output = m(input)
314
315     .. _cross-correlation:
316         https://en.wikipedia.org/wiki/Cross-correlation
317
318     .. _link:
319         https://github.com/vdumoulin/conv\_arithmetic/blob/master/README.md
320     """
321     def __init__(self, in_channels, out_channels, kernel_size, stride=1,
322                 padding=0, dilation=1, groups=1,
323                 bias=True, padding_mode='zeros'):
324         kernel_size = _pair(kernel_size)
325         stride = _pair(stride)
326         padding = _pair(padding)
327         dilation = _pair(dilation)
328         super(Conv2d, self).__init__(
329             in_channels, out_channels, kernel_size, stride, padding, dilation,
330             False, _pair(0), groups, bias, padding_mode)
331
332     def forward(self, input):
333         if self.padding_mode == 'circular':
334             expanded_padding = ((self.padding[1] + 1) // 2, self.padding[1] // 2,
335                                (self.padding[0] + 1) // 2, self.padding[0] // 2)
336             return F.conv2d(F.pad(input, expanded_padding, mode='circular'),
337                             self.weight, self.bias, self.stride,
338                             _pair(0), self.dilation, self.groups)
339         return F.conv2d(input, self.weight, self.bias, self.stride,
340                         self.padding, self.dilation, self.groups)

```

```

341
342
343 class Conv3d(_ConvNd):
344     r"""Applies a 3D convolution over an input signal composed of several input
345     planes.
346
347     In the simplest case, the output value of the layer with input size :math:`(N, C_{in}, D, H, W)`
348     and output :math:`(N, C_{out}, D_{out}, H_{out}, W_{out})` can be precisely described as:
349
350     .. math::
351         \text{out}(N_i, C_{out\_j}) = \text{bias}(C_{out\_j}) +
352             \sum_{k=0}^{C_{in}-1} \text{weight}(C_{out\_j}, k) \star \text{input}(N_i, k)
353
354     where :math:`\star` is the valid 3D `cross-correlation`_ operator
355
356     * :attr:`stride` controls the stride for the cross-correlation.
357
358     * :attr:`padding` controls the amount of implicit zero-paddings on both
359       sides for :attr:`padding` number of points for each dimension.
360
361     * :attr:`dilation` controls the spacing between the kernel points; also known as the à trous algorithm.
362       It is harder to describe, but this `link`_ has a nice visualization of what :attr:`dilation` does.
363
364     * :attr:`groups` controls the connections between inputs and outputs.
365       :attr:`in_channels` and :attr:`out_channels` must both be divisible by
366       :attr:`groups`. For example,
367
368       * At groups=1, all inputs are convolved to all outputs.
369       * At groups=2, the operation becomes equivalent to having two conv
370         layers side by side, each seeing half the input channels,
371         and producing half the output channels, and both subsequently
372         concatenated.
373       * At groups= :attr:`in_channels`, each input channel is convolved with
374         its own set of filters, of size
375         :math:`\left\lfloor \frac{\text{out\_channels}}{\text{in\_channels}} \right\rfloor`.
376
377     The parameters :attr:`kernel_size`, :attr:`stride`, :attr:`padding`, :attr:`dilation` can either be:
378
379     - a single ``int`` -- in which case the same value is used for the depth, height and width dimension
380     - a ``tuple`` of three ints -- in which case, the first `int` is used for the depth dimension,
381       the second `int` for the height dimension and the third `int` for the width dimension
382
383     .. note::
384
385         Depending of the size of your kernel, several (of the last)
386         columns of the input might be lost, because it is a valid `cross-correlation`_,
387         and not a full `cross-correlation`_.
388         It is up to the user to add proper padding.
389
390     .. note::
391
392         When `groups == in_channels` and `out_channels == K * in_channels`,

```



```

393     where `K` is a positive integer, this operation is also termed in
394     literature as depthwise convolution.
395
396     In other words, for an input of size :math:`(N, C_{in}, D_{in}, H_{in}, W_{in})`,
397     a depthwise convolution with a depthwise multiplier `K`, can be constructed by arguments
398     :math:`(in\_channels=C_{in}, out\_channels=C_{in} \times K, ..., groups=C_{in})`.
399
400     .. include:: cudnn_deterministic.rst
401
402     Args:
403         in_channels (int): Number of channels in the input image
404         out_channels (int): Number of channels produced by the convolution
405         kernel_size (int or tuple): Size of the convolving kernel
406         stride (int or tuple, optional): Stride of the convolution. Default: 1
407         padding (int or tuple, optional): Zero-padding added to all three sides of the input. Default: 0
408         padding_mode (string, optional). Accepted values `zeros` and `circular` Default: `zeros`
409         dilation (int or tuple, optional): Spacing between kernel elements. Default: 1
410         groups (int, optional): Number of blocked connections from input channels to output channels. Default: 1
411         bias (bool, optional): If ``True``, adds a learnable bias to the output. Default: ``True``
412
413     Shape:
414     - Input: :math:`(N, C_{in}, D_{in}, H_{in}, W_{in})`
415     - Output: :math:`(N, C_{out}, D_{out}, H_{out}, W_{out})` where
416
417         .. math::
418             D_{out} = \left\lfloor \frac{D_{in} + 2 \times \text{padding}[0] - \text{dilation}[0]}{\text{kernel\_size}[0] - 1} - 1 \right\rfloor \times \text{stride}[0] + 1
419
420         .. math::
421             H_{out} = \left\lfloor \frac{H_{in} + 2 \times \text{padding}[1] - \text{dilation}[1]}{\text{kernel\_size}[1] - 1} - 1 \right\rfloor \times \text{stride}[1] + 1
422
423         .. math::
424             W_{out} = \left\lfloor \frac{W_{in} + 2 \times \text{padding}[2] - \text{dilation}[2]}{\text{kernel\_size}[2] - 1} - 1 \right\rfloor \times \text{stride}[2] + 1
425
426     Attributes:
427     weight (Tensor): the learnable weights of the module of shape
428         :math:`(\text{out\_channels}, \frac{\text{in\_channels}}{\text{groups}}, \text{kernel\_size}[0], \text{kernel\_size}[1], \text{kernel\_size}[2])`.
429         The values of these weights are sampled from
430         :math:`\mathcal{U}(-\sqrt{k}, \sqrt{k})` where
431         :math:`k = \frac{1}{C_{in}} * \prod_{i=0}^2 \text{kernel\_size}[i]`
432     bias (Tensor): the learnable bias of the module of shape (out_channels). If :attr:`bias` is ``True``,
433         then the values of these weights are
434         sampled from :math:`\mathcal{U}(-\sqrt{k}, \sqrt{k})` where
435         :math:`k = \frac{1}{C_{in}} * \prod_{i=0}^2 \text{kernel\_size}[i]`
436
437     Examples::
438
439     >>> # With square kernels and equal stride
440     >>> m = nn.Conv3d(16, 33, 3, stride=2)

```

```

445     >>> # non-square kernels and unequal stride and with padding
446     >>> m = nn.Conv3d(16, 33, (3, 5, 2), stride=(2, 1, 1), padding=(4, 2, 0))
447     >>> input = torch.randn(20, 16, 10, 50, 100)
448     >>> output = m(input)
449
450 .. _cross-correlation:
451     https://en.wikipedia.org/wiki/Cross-correlation
452
453 .. _link:
454     https://github.com/vdumoulin/conv_arithmetic/blob/master/README.md
455 """
456 def __init__(self, in_channels, out_channels, kernel_size, stride=1,
457              padding=0, dilation=1, groups=1,
458              bias=True, padding_mode='zeros'):
459     kernel_size = _triple(kernel_size)
460     stride = _triple(stride)
461     padding = _triple(padding)
462     dilation = _triple(dilation)
463     super(Conv3d, self).__init__(
464         in_channels, out_channels, kernel_size, stride, padding, dilation,
465         False, _triple(0), groups, bias, padding_mode)
466
467 def forward(self, input):
468     if self.padding_mode == 'circular':
469         expanded_padding = ((self.padding[2] + 1) // 2, self.padding[2] // 2,
470                             (self.padding[1] + 1) // 2, self.padding[1] // 2,
471                             (self.padding[0] + 1) // 2, self.padding[0] // 2)
472         return F.conv3d(F.pad(input, expanded_padding, mode='circular'),
473                         self.weight, self.bias, self.stride, _triple(0),
474                         self.dilation, self.groups)
475     return F.conv3d(input, self.weight, self.bias, self.stride,
476                     self.padding, self.dilation, self.groups)
477
478
479 class _ConvTransposeMixin(object):
480     def forward(self, input, output_size=None):
481         # type: (Tensor, Optional[List[int]]) -> Tensor
482         output_padding = self._output_padding(input, output_size, self.stride, self.padding, self.kernel_size)
483         func = self._backend.ConvNd(
484             self.stride, self.padding, self.dilation, self.transposed,
485             output_padding, self.groups)
486         if self.bias is None:
487             return func(input, self.weight)
488         else:
489             return func(input, self.weight, self.bias)
490
491     def _output_padding(self, input, output_size, stride, padding, kernel_size):
492         # type: (Tensor, Optional[List[int]], List[int], List[int], List[int]) -> List[int]
493         if output_size is None:
494             ret = _single(self.output_padding) # converting to list if was not already
495         else:
496             k = input.dim() - 2

```

```

497         if len(output_size) == k + 2:
498             output_size = output_size[2:]
499         if len(output_size) != k:
500             raise ValueError(
501                 "output_size must have {} or {} elements (got {})"
502                 .format(k, k + 2, len(output_size)))
503
504         min_sizes = torch.jit.annotate(List[int], [])
505         max_sizes = torch.jit.annotate(List[int], [])
506         for d in range(k):
507             dim_size = ((input.size(d + 2) - 1) * stride[d] -
508                         2 * padding[d] + kernel_size[d])
509             min_sizes.append(dim_size)
510             max_sizes.append(min_sizes[d] + stride[d] - 1)
511
512         for i in range(len(output_size)):
513             size = output_size[i]
514             min_size = min_sizes[i]
515             max_size = max_sizes[i]
516             if size < min_size or size > max_size:
517                 raise ValueError((
518                     "requested an output size of {}, but valid sizes range "
519                     "from {} to {} (for an input of {})".format(
520                         output_size, min_sizes, max_sizes, input.size()[2:]))
521
522         res = torch.jit.annotate(List[int], [])
523         for d in range(k):
524             res.append(output_size[d] - min_sizes[d])
525
526         ret = res
527     return ret
528
529
530 class ConvTranspose1d(_ConvTransposeMixin, _ConvNd):
531     r"""Applies a 1D transposed convolution operator over an input image
532     composed of several input planes.
533
534     This module can be seen as the gradient of Conv1d with respect to its input.
535     It is also known as a fractionally-strided convolution or
536     a deconvolution (although it is not an actual deconvolution operation).
537
538     * :attr:`stride` controls the stride for the cross-correlation.
539
540     * :attr:`padding` controls the amount of implicit zero-paddings on both
541       sides for ``dilation * (kernel_size - 1) - padding`` number of points. See note
542       below for details.
543
544     * :attr:`output_padding` controls the additional size added to one side
545       of the output shape. See note below for details.
546
547     * :attr:`dilation` controls the spacing between the kernel points; also known as the à trous algorithm.
548       It is harder to describe, but this `link`_ has a nice visualization of what :attr:`dilation` does.

```

```

549
550 * :attr:`groups` controls the connections between inputs and outputs.
551 :attr:`in_channels` and :attr:`out_channels` must both be divisible by
552 :attr:`groups`. For example,
553
554 * At groups=1, all inputs are convolved to all outputs.
555 * At groups=2, the operation becomes equivalent to having two conv
556   layers side by side, each seeing half the input channels,
557   and producing half the output channels, and both subsequently
558   concatenated.
559 * At groups= :attr:`in_channels`, each input channel is convolved with
560   its own set of filters (of size
561    $\left\lfloor \frac{\text{out\_channels}}{\text{in\_channels}} \right\rfloor$ ).
562
563 .. note::
564
565     Depending of the size of your kernel, several (of the last)
566     columns of the input might be lost, because it is a valid `cross-correlation`,
567     and not a full `cross-correlation`.
568     It is up to the user to add proper padding.
569
570 .. note::
571     The :attr:`padding` argument effectively adds ``dilation * (kernel_size - 1) - padding``
572     amount of zero padding to both sizes of the input. This is set so that
573     when a :class:`~torch.nn.Conv1d` and a :class:`~torch.nn.ConvTranspose1d`
574     are initialized with same parameters, they are inverses of each other in
575     regard to the input and output shapes. However, when ``stride > 1``,
576     :class:`~torch.nn.Conv1d` maps multiple input shapes to the same output
577     shape. :attr:`output_padding` is provided to resolve this ambiguity by
578     effectively increasing the calculated output shape on one side. Note
579     that :attr:`output_padding` is only used to find output shape, but does
580     not actually add zero-padding to output.
581
582 .. include:: cudnn_deterministic.rst
583
584 Args:
585     in_channels (int): Number of channels in the input image
586     out_channels (int): Number of channels produced by the convolution
587     kernel_size (int or tuple): Size of the convolving kernel
588     stride (int or tuple, optional): Stride of the convolution. Default: 1
589     padding (int or tuple, optional): ``dilation * (kernel_size - 1) - padding`` zero-padding
590       will be added to both sides of the input. Default: 0
591     output_padding (int or tuple, optional): Additional size added to one side
592       of the output shape. Default: 0
593     groups (int, optional): Number of blocked connections from input channels to output channels. Default: 1
594     bias (bool, optional): If ``True``, adds a learnable bias to the output. Default: ``True``
595     dilation (int or tuple, optional): Spacing between kernel elements. Default: 1
596
597 Shape:
598   - Input:  $(N, C_{\text{in}}, L_{\text{in}})$ 
599   - Output:  $(N, C_{\text{out}}, L_{\text{out}})$  where
600
```

```

601         .. math::
602             L_{out} = (L_{in} - 1) \times \text{stride} - 2 \times \text{padding} + \text{dilation}
603                     \times (\text{kernel\_size} - 1) + \text{output\_padding} + 1
604
605     Attributes:
606         weight (Tensor): the learnable weights of the module of shape
607             :math:`(\text{in\_channels}, \frac{\text{out\_channels}}{\text{groups}})`
608             :math:`\text{kernel\_size}`.
609             The values of these weights are sampled from
610             :math:`\mathcal{U}(-\sqrt{k}, \sqrt{k})` where
611             :math:`k = \frac{1}{C_{in}} * \text{kernel\_size}`
612         bias (Tensor): the learnable bias of the module of shape (out_channels).
613             If :attr:`bias` is ``True``, then the values of these weights are
614             sampled from :math:`\mathcal{U}(-\sqrt{k}, \sqrt{k})` where
615             :math:`k = \frac{1}{C_{in}} * \text{kernel\_size}`
616     """
617
618     def __init__(self, in_channels, out_channels, kernel_size, stride=1,
619                  padding=0, output_padding=0, groups=1, bias=True,
620                  dilation=1, padding_mode='zeros'):
621         kernel_size = _single(kernel_size)
622         stride = _single(stride)
623         padding = _single(padding)
624         dilation = _single(dilation)
625         output_padding = _single(output_padding)
626         super(ConvTranspose1d, self).__init__(
627             in_channels, out_channels, kernel_size, stride, padding, dilation,
628             True, output_padding, groups, bias, padding_mode)
629
630     def forward(self, input, output_size=None):
631         # type: (Tensor, Optional[List[int]]) -> Tensor
632         if self.padding_mode != 'zeros':
633             raise ValueError('Only `zeros` padding mode is supported for ConvTranspose1d')
634
635         output_padding = self._output_padding(input, output_size, self.stride, self.padding, self.kernel_size)
636         return F.conv_transpose1d(
637             input, self.weight, self.bias, self.stride, self.padding,
638             output_padding, self.groups, self.dilation)
639
640
641     class ConvTranspose2d(_ConvTransposeMixin, _ConvNd):
642         r"""Applies a 2D transposed convolution operator over an input image
643         composed of several input planes.
644
645         This module can be seen as the gradient of Conv2d with respect to its input.
646         It is also known as a fractionally-strided convolution or
647         a deconvolution (although it is not an actual deconvolution operation).
648
649         * :attr:`stride` controls the stride for the cross-correlation.
650
651         * :attr:`padding` controls the amount of implicit zero-paddings on both
652         sides for ``dilation * (kernel_size - 1) - padding`` number of points. See note

```

```
653     below for details.
654
655     * :attr:`output_padding` controls the additional size added to one side
656       of the output shape. See note below for details.
657
658     * :attr:`dilation` controls the spacing between the kernel points; also known as the à trous algorithm.
659       It is harder to describe, but this `link`_ has a nice visualization of what :attr:`dilation` does.
660
661     * :attr:`groups` controls the connections between inputs and outputs.
662       :attr:`in_channels` and :attr:`out_channels` must both be divisible by
663       :attr:`groups`. For example,
664
665       * At groups=1, all inputs are convolved to all outputs.
666       * At groups=2, the operation becomes equivalent to having two conv
667         layers side by side, each seeing half the input channels,
668         and producing half the output channels, and both subsequently
669         concatenated.
670       * At groups= :attr:`in_channels`, each input channel is convolved with
671         its own set of filters (of size
672         :math:\left\lfloor \frac{\text{out\_channels}}{\text{in\_channels}} \right\rfloor).
673
674     The parameters :attr:`kernel_size`, :attr:`stride`, :attr:`padding`, :attr:`output_padding`
675     can either be:
676
677     - a single ``int`` -- in which case the same value is used for the height and width dimensions
678     - a ``tuple`` of two ints -- in which case, the first ``int`` is used for the height dimension,
679       and the second ``int`` for the width dimension
680
681     .. note::
682
683         Depending of the size of your kernel, several (of the last)
684         columns of the input might be lost, because it is a valid `cross-correlation`_,
685         and not a full `cross-correlation`_.
686         It is up to the user to add proper padding.
687
688     .. note::
689
690         The :attr:`padding` argument effectively adds ``dilation * (kernel_size - 1) - padding``
691         amount of zero padding to both sizes of the input. This is set so that
692         when a :class:`~torch.nn.Conv2d` and a :class:`~torch.nn.ConvTranspose2d`
693         are initialized with same parameters, they are inverses of each other in
694         regard to the input and output shapes. However, when ``stride > 1``,
695         :class:`~torch.nn.Conv2d` maps multiple input shapes to the same output
696         shape. :attr:`output_padding` is provided to resolve this ambiguity by
697         effectively increasing the calculated output shape on one side. Note
698         that :attr:`output_padding` is only used to find output shape, but does
699         not actually add zero-padding to output.
700
701     .. include:: cudnn_deterministic.rst
702
703     Args:
704         in_channels (int): Number of channels in the input image
705         out_channels (int): Number of channels produced by the convolution
```

```

705 kernel_size (int or tuple): Size of the convolving kernel
706 stride (int or tuple, optional): Stride of the convolution. Default: 1
707 padding (int or tuple, optional): ``dilation * (kernel_size - 1) - padding`` zero-padding
708     will be added to both sides of each dimension in the input. Default: 0
709 output_padding (int or tuple, optional): Additional size added to one side
710     of each dimension in the output shape. Default: 0
711 groups (int, optional): Number of blocked connections from input channels to output channels. Default: 1
712 bias (bool, optional): If ``True``, adds a learnable bias to the output. Default: ``True``
713 dilation (int or tuple, optional): Spacing between kernel elements. Default: 1
714
715 Shape:
716 - Input: :math:`(N, C_{in}, H_{in}, W_{in})`
717 - Output: :math:`(N, C_{out}, H_{out}, W_{out})` where
718
719 .. math::
720     H_{out} = (H_{in} - 1) \times \text{stride}[0] - 2 \times \text{padding}[0] + \text{dilation}[0]
721             \times (\text{kernel\_size}[0] - 1) + \text{output\_padding}[0] + 1
722 .. math::
723     W_{out} = (W_{in} - 1) \times \text{stride}[1] - 2 \times \text{padding}[1] + \text{dilation}[1]
724             \times (\text{kernel\_size}[1] - 1) + \text{output\_padding}[1] + 1
725
726 Attributes:
727 weight (Tensor): the learnable weights of the module of shape
728     :math:`(\text{in\_channels}, \frac{\text{out\_channels}}{\text{groups}},`
729     :math:`\text{kernel\_size}[0], \text{kernel\_size}[1])` .
730     The values of these weights are sampled from
731     :math:`\mathcal{U}(-\sqrt{k}, \sqrt{k})` where
732     :math:`k = \frac{1}{C_{in}} * \prod_{i=0}^1 \text{kernel\_size}[i]`
733 bias (Tensor): the learnable bias of the module of shape (out_channels)
734     If :attr:`bias` is ``True``, then the values of these weights are
735     sampled from :math:`\mathcal{U}(-\sqrt{k}, \sqrt{k})` where
736     :math:`k = \frac{1}{C_{in}} * \prod_{i=0}^1 \text{kernel\_size}[i]`
737
738 Examples::
739
740 >>> # With square kernels and equal stride
741 >>> m = nn.ConvTranspose2d(16, 33, 3, stride=2)
742 >>> # non-square kernels and unequal stride and with padding
743 >>> m = nn.ConvTranspose2d(16, 33, (3, 5), stride=(2, 1), padding=(4, 2))
744 >>> input = torch.randn(20, 16, 50, 100)
745 >>> output = m(input)
746 >>> # exact output size can be also specified as an argument
747 >>> input = torch.randn(1, 16, 12, 12)
748 >>> downsample = nn.Conv2d(16, 16, 3, stride=2, padding=1)
749 >>> upsample = nn.ConvTranspose2d(16, 16, 3, stride=2, padding=1)
750 >>> h = downsample(input)
751 >>> h.size()
752 torch.Size([1, 16, 6, 6])
753 >>> output = upsample(h, output_size=input.size())
754 >>> output.size()
755 torch.Size([1, 16, 12, 12])
756

```



```

757 .. _cross-correlation:
758     https://en.wikipedia.org/wiki/Cross-correlation
759
760 .. _link:
761     https://github.com/vdumoulin/conv_arithmetic/blob/master/README.md
762 """
763
764 def __init__(self, in_channels, out_channels, kernel_size, stride=1,
765              padding=0, output_padding=0, groups=1, bias=True,
766              dilation=1, padding_mode='zeros'):
767     kernel_size = _pair(kernel_size)
768     stride = _pair(stride)
769     padding = _pair(padding)
770     dilation = _pair(dilation)
771     output_padding = _pair(output_padding)
772     super(ConvTranspose2d, self).__init__(
773         in_channels, out_channels, kernel_size, stride, padding, dilation,
774         True, output_padding, groups, bias, padding_mode)
775
776 def forward(self, input, output_size=None):
777     # type: (Tensor, Optional[List[int]]) -> Tensor
778     if self.padding_mode != 'zeros':
779         raise ValueError('Only `zeros` padding mode is supported for ConvTranspose2d')
780
781     output_padding = self._output_padding(input, output_size, self.stride, self.padding, self.kernel_size)
782
783     return F.conv_transpose2d(
784         input, self.weight, self.bias, self.stride, self.padding,
785         output_padding, self.groups, self.dilation)
786
787
788 class ConvTranspose3d(_ConvTransposeMixin, _ConvNd):
789     r"""Applies a 3D transposed convolution operator over an input image composed of several input
790     planes.
791
792     The transposed convolution operator multiplies each input value element-wise by a learnable kernel,
793     and sums over the outputs from all input feature planes.
794
795     This module can be seen as the gradient of Conv3d with respect to its input.
796     It is also known as a fractionally-strided convolution or
797     a deconvolution (although it is not an actual deconvolution operation).
798
799     * :attr:`stride` controls the stride for the cross-correlation.
800
801     * :attr:`padding` controls the amount of implicit zero-paddings on both
802     sides for ``dilation * (kernel_size - 1) - padding`` number of points. See note
803     below for details.
804
805     * :attr:`output_padding` controls the additional size added to one side
806     of the output shape. See note below for details.
807
808     * :attr:`dilation` controls the spacing between the kernel points; also known as the à trous algorithm.
809     It is harder to describe, but this `link`_ has a nice visualization of what :attr:`dilation` does.

```



```

809
810 * :attr:`groups` controls the connections between inputs and outputs.
811 :attr:`in_channels` and :attr:`out_channels` must both be divisible by
812 :attr:`groups`. For example,
813
814 * At groups=1, all inputs are convolved to all outputs.
815 * At groups=2, the operation becomes equivalent to having two conv
816 layers side by side, each seeing half the input channels,
817 and producing half the output channels, and both subsequently
818 concatenated.
819 * At groups= :attr:`in_channels`, each input channel is convolved with
820 its own set of filters (of size
821  $\left\lfloor \frac{\text{out\_channels}}{\text{in\_channels}} \right\rfloor$ ).
822
823 The parameters :attr:`kernel_size`, :attr:`stride`, :attr:`padding`, :attr:`output_padding`
824 can either be:
825
826 - a single ``int`` -- in which case the same value is used for the depth, height and width dimensions
827 - a ``tuple`` of three ints -- in which case, the first ``int`` is used for the depth dimension,
828 the second ``int`` for the height dimension and the third ``int`` for the width dimension
829
830 .. note::
831
832 Depending of the size of your kernel, several (of the last)
833 columns of the input might be lost, because it is a valid `cross-correlation`,
834 and not a full `cross-correlation`.
835 It is up to the user to add proper padding.
836
837 .. note::
838 The :attr:`padding` argument effectively adds ``dilation * (kernel_size - 1) - padding``
839 amount of zero padding to both sizes of the input. This is set so that
840 when a :class:`~torch.nn.Conv3d` and a :class:`~torch.nn.ConvTranspose3d`
841 are initialized with same parameters, they are inverses of each other in
842 regard to the input and output shapes. However, when ``stride > 1``,
843 :class:`~torch.nn.Conv3d` maps multiple input shapes to the same output
844 shape. :attr:`output_padding` is provided to resolve this ambiguity by
845 effectively increasing the calculated output shape on one side. Note
846 that :attr:`output_padding` is only used to find output shape, but does
847 not actually add zero-padding to output.
848
849 .. include:: cudnn_deterministic.rst
850
851 Args:
852 in_channels (int): Number of channels in the input image
853 out_channels (int): Number of channels produced by the convolution
854 kernel_size (int or tuple): Size of the convolving kernel
855 stride (int or tuple, optional): Stride of the convolution. Default: 1
856 padding (int or tuple, optional): ``dilation * (kernel_size - 1) - padding`` zero-padding
857 will be added to both sides of each dimension in the input. Default: 0
858 output_padding (int or tuple, optional): Additional size added to one side
859 of each dimension in the output shape. Default: 0
860 groups (int, optional): Number of blocked connections from input channels to output channels. Default: 1

```

```

861     bias (bool, optional): If ``True``, adds a learnable bias to the output. Default: ``True``
862     dilation (int or tuple, optional): Spacing between kernel elements. Default: 1
863
864     Shape:
865     - Input: :math:(N, C_{in}, D_{in}, H_{in}, W_{in})`
866     - Output: :math:(N, C_{out}, D_{out}, H_{out}, W_{out})` where
867
868     .. math::
869         D_{out} = (D_{in} - 1) \times \text{stride}[0] - 2 \times \text{padding}[0] + \text{dilation}[0]
870             \times (\text{kernel\_size}[0] - 1) + \text{output\_padding}[0] + 1
871     .. math::
872         H_{out} = (H_{in} - 1) \times \text{stride}[1] - 2 \times \text{padding}[1] + \text{dilation}[1]
873             \times (\text{kernel\_size}[1] - 1) + \text{output\_padding}[1] + 1
874     .. math::
875         W_{out} = (W_{in} - 1) \times \text{stride}[2] - 2 \times \text{padding}[2] + \text{dilation}[2]
876             \times (\text{kernel\_size}[2] - 1) + \text{output\_padding}[2] + 1
877
878     Attributes:
879     weight (Tensor): the learnable weights of the module of shape
880         :math:(\text{in\_channels}, \frac{\text{out\_channels}}{\text{groups}},`
881         :math:\text{kernel\_size}[0], \text{kernel\_size}[1], \text{kernel\_size}[2])`.
882         The values of these weights are sampled from
883         :math:\mathcal{U}(-\sqrt{k}, \sqrt{k})` where
884         :math:k = \frac{1}{C_{in}} * \prod_{i=0}^2 \text{kernel\_size}[i]`
885     bias (Tensor): the learnable bias of the module of shape (out_channels)
886         If :attr:`bias` is ``True``, then the values of these weights are
887         sampled from :math:\mathcal{U}(-\sqrt{k}, \sqrt{k})` where
888         :math:k = \frac{1}{C_{in}} * \prod_{i=0}^2 \text{kernel\_size}[i]`
889
890     Examples::
891
892     >>> # With square kernels and equal stride
893     >>> m = nn.ConvTranspose3d(16, 33, 3, stride=2)
894     >>> # non-square kernels and unequal stride and with padding
895     >>> m = nn.ConvTranspose3d(16, 33, (3, 5, 2), stride=(2, 1, 1), padding=(0, 4, 2))
896     >>> input = torch.randn(20, 16, 10, 50, 100)
897     >>> output = m(input)
898
899     .. _cross-correlation:
900         https://en.wikipedia.org/wiki/Cross-correlation
901
902     .. _link:
903         https://github.com/vdumoulin/conv\_arithmetic/blob/master/README.md
904     """
905
906     def __init__(self, in_channels, out_channels, kernel_size, stride=1,
907                  padding=0, output_padding=0, groups=1, bias=True,
908                  dilation=1, padding_mode='zeros'):
909         kernel_size = _triple(kernel_size)
910         stride = _triple(stride)
911         padding = _triple(padding)

```

```
913     dilation = _triple(dilation)
914     output_padding = _triple(output_padding)
915     super(ConvTranspose3d, self).__init__(
916         in_channels, out_channels, kernel_size, stride, padding, dilation,
917         True, output_padding, groups, bias, padding_mode)
918
919     def forward(self, input, output_size=None):
920         # type: (Tensor, Optional[List[int]]) -> Tensor
921         if self.padding_mode != 'zeros':
922             raise ValueError('Only `zeros` padding mode is supported for ConvTranspose3d')
923
924         output_padding = self._output_padding(input, output_size, self.stride, self.padding, self.kernel_size)
925
926         return F.conv_transpose3d(
927             input, self.weight, self.bias, self.stride, self.padding,
928             output_padding, self.groups, self.dilation)
929
930
931 # TODO: Conv2dLocal
932 # TODO: Conv2dMap
933 # TODO: ConvTranspose2dMap
```