```python
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.utils.model_zoo as model_zoo
from torch.autograd.variable import Variable
from collections import OrderedDict
import re
import pdb

__all__ = ['DenseNet', 'densenet121', 'densenet169', 'densenet169_par', 'densenet201', 'densenet161']


model_urls = {
    'densenet121': 'https://download.pytorch.org/models/densenet121-a639ec97.pth',
    'densenet169': 'https://download.pytorch.org/models/densenet169-b2777c0a.pth',
    'densenet201': 'https://download.pytorch.org/models/densenet201-c1103571.pth',
    'densenet161': 'https://download.pytorch.org/models/densenet161-8d451a50.pth',
}


def densenet121(pretrained=False, **kwargs):
    r"""Densenet-121 model from
    `"Densely Connected Convolutional Networks" <https://arxiv.org/pdf/1608.06993.pdf>`_

    Args:
        pretrained (bool): If True, returns a model pre-trained on ImageNet
    """
    model = DenseNet(num_init_features=64, growth_rate=32, block_config=(6, 12, 24, 16),
                     **kwargs)
    if pretrained:
        # '.'s are no longer allowed in module names, but pervious _DenseLayer
        # has keys 'norm.1', 'relu.1', 'conv.1', 'norm.2', 'relu.2', 'conv.2'.
        # They are also in the checkpoints in model_urls. This pattern is used
        # to find such keys.
        pattern = re.compile(
            r'^(.*denselayer\d+\.(?:norm|relu|conv))\.((?:[12])\.(?:weight|bias|running_mean|running_var))$')
        state_dict = model_zoo.load_url(model_urls['densenet121'])
        for key in list(state_dict.keys()):
            res = pattern.match(key)
            if res:
                new_key = res.group(1) + res.group(2)
                state_dict[new_key] = state_dict[key]
                del state_dict[key]
        model.load_state_dict(state_dict)
    model.classifier = None
    features = model.features
    features.block0 = nn.Sequential(features.conv0, features.norm0, features.relu0)
```

```
48
49      features.denseblock1 = nn.Sequential(*list(features.denseblock1))
50      features.transition1 = nn.Sequential(*list(features.transition1)[:-1])
51
52      features.denseblock2 = nn.Sequential(*list(features.denseblock2))
53      features.transition2 = nn.Sequential(*list(features.transition2)[:-1])
54
55      features.denseblock3 = nn.Sequential(*list(features.denseblock3))
56      features.transition3 = nn.Sequential(*list(features.transition3)[:-1])
57
58      features.denseblock4 =
    nn.Sequential(*(list(features.denseblock4)+[features.norm5]))
59      model.features = features
60      return model
61
62
63  def densenet169_par(pretrained=False, **kwargs):
64      r"""Densenet-169 model from
65      `"Densely Connected Convolutional Networks"
    <https://arxiv.org/pdf/1608.06993.pdf>`_
66
67      Args:
68          pretrained (bool): If True, returns a model pre-trained on ImageNet
69      """
70      model = DenseNetPar(num_init_features=64, growth_rate=32,
    block_config=(6, 12, 32, 32),
71                          **kwargs)
72      if pretrained:
73          # '.'s are no longer allowed in module names, but pervious
    _DenseLayer
74          # has keys 'norm.1', 'relu.1', 'conv.1', 'norm.2', 'relu.2',
    'conv.2'.
75          # They are also in the checkpoints in model_urls. This pattern is
    used
76          # to find such keys.
77          pattern = re.compile(
78
    r'^(.*denselayer\d+\.(?:norm|relu|conv))\.((?:[12])\.(?:weight|bias|running
    _mean|running_var))$')
79          state_dict = model_zoo.load_url(model_urls['densenet169'])
80          for key in list(state_dict.keys()):
81              res = pattern.match(key)
82              if res:
83                  new_key = res.group(1) + res.group(2)
84                  state_dict[new_key] = state_dict[key]
85                  del state_dict[key]
86          model.load_state_dict(state_dict)
87      model.classifier = None
88      features = model.features
89      features.block0 = nn.Sequential(features.conv0, features.norm0,
    features.relu0)
90
91      features.denseblock1 = nn.Sequential(*list(features.denseblock1))
92      features.transition1 = nn.Sequential(*list(features.transition1)[:-1])
93
94      features.denseblock2 = nn.Sequential(*list(features.denseblock2))
95      features.transition2 = nn.Sequential(*list(features.transition2)[:-1])
96
97      features.denseblock3 = nn.Sequential(*list(features.denseblock3))
98      features.transition3 = nn.Sequential(*list(features.transition3)[:-1])
```

```
99
100     features.denseblock4 = nn.Sequential(*(list(features.denseblock4)))
101     model.features = features
102     return model
103
104
105 def densenet169(pretrained=False, **kwargs):
106     r"""Densenet-169 model from
107     `"Densely Connected Convolutional Networks"
    <https://arxiv.org/pdf/1608.06993.pdf>`_
108
109     Args:
110         pretrained (bool): If True, returns a model pre-trained on ImageNet
111     """
112     model = DenseNet(num_init_features=64, growth_rate=32,
    block_config=(6, 12, 32, 32),
113                      **kwargs)
114     if pretrained:
115         # '.'s are no longer allowed in module names, but pervious
    _DenseLayer
116         # has keys 'norm.1', 'relu.1', 'conv.1', 'norm.2', 'relu.2',
    'conv.2'.
117         # They are also in the checkpoints in model_urls. This pattern is
    used
118         # to find such keys.
119         pattern = re.compile(
120
    r'^(.*denselayer\d+\.(?:norm|relu|conv))\.((?:[12])\.(?:weight|bias|running
    _mean|running_var))$')
121         state_dict = model_zoo.load_url(model_urls['densenet169'])
122         for key in list(state_dict.keys()):
123             res = pattern.match(key)
124             if res:
125                 new_key = res.group(1) + res.group(2)
126                 state_dict[new_key] = state_dict[key]
127                 del state_dict[key]
128         model.load_state_dict(state_dict)
129     model.classifier = None
130     features = model.features
131     features.block0 = nn.Sequential(features.conv0, features.norm0,
    features.relu0, features.pool0)
132
133     features.denseblock1 = nn.Sequential(*list(features.denseblock1))
134     features.transition1 = nn.Sequential(*list(features.transition1))
135
136     features.denseblock2 = nn.Sequential(*list(features.denseblock2))
137     features.transition2 = nn.Sequential(*list(features.transition2))
138
139     features.denseblock3 = nn.Sequential(*list(features.denseblock3))
140     features.transition3 = nn.Sequential(*list(features.transition3))
141
142     features.denseblock4 = nn.Sequential(*(list(features.denseblock4) +
    [features.norm5]))
143     model.features = features
144     return model
145
146
147 def densenet201(pretrained=False, **kwargs):
148     r"""Densenet-201 model from
149     `"Densely Connected Convolutional Networks"
```

```
    <https://arxiv.org/pdf/1608.06993.pdf>`_
150
151     Args:
152         pretrained (bool): If True, returns a model pre-trained on ImageNet
153     """
154     model = DenseNet(num_init_features=64, growth_rate=32,
    block_config=(6, 12, 48, 32),
155                      **kwargs)
156     if pretrained:
157         # '.'s are no longer allowed in module names, but pervious
    _DenseLayer
158         # has keys 'norm.1', 'relu.1', 'conv.1', 'norm.2', 'relu.2',
    'conv.2'.
159         # They are also in the checkpoints in model_urls. This pattern is
    used
160         # to find such keys.
161         pattern = re.compile(
162
    r'^(.*denselayer\d+\.(?:norm|relu|conv))\.((?:[12])\.(?:weight|bias|running
    _mean|running_var))$')
163         state_dict = model_zoo.load_url(model_urls['densenet201'])
164         for key in list(state_dict.keys()):
165             res = pattern.match(key)
166             if res:
167                 new_key = res.group(1) + res.group(2)
168                 state_dict[new_key] = state_dict[key]
169                 del state_dict[key]
170         model.load_state_dict(state_dict)
171     model.classifier = None
172     features = model.features
173     features.block0 = nn.Sequential(features.conv0, features.norm0,
    features.relu0, features.pool0)
174
175     features.denseblock1 = nn.Sequential(*list(features.denseblock1))
176     features.transition1 = nn.Sequential(*list(features.transition1))
177
178     features.denseblock2 = nn.Sequential(*list(features.denseblock2))
179     features.transition2 = nn.Sequential(*list(features.transition2))
180
181     features.denseblock3 = nn.Sequential(*list(features.denseblock3))
182     features.transition3 = nn.Sequential(*list(features.transition3))
183
184     features.denseblock4 = nn.Sequential(*(list(features.denseblock4) +
    [features.norm5]))
185     model.features = features
186     return model
187
188
189 def densenet161(pretrained=False, **kwargs):
190     r"""Densenet-161 model from
191     `"Densely Connected Convolutional Networks"
    <https://arxiv.org/pdf/1608.06993.pdf>`_
192
193     Args:
194         pretrained (bool): If True, returns a model pre-trained on ImageNet
195     """
196     model = DenseNet(num_init_features=96, growth_rate=48,
    block_config=(6, 12, 36, 24),
197                      **kwargs)
198     if pretrained:
```

```
199         # '.'s are no longer allowed in module names, but pervious
    _DenseLayer
200         # has keys 'norm.1', 'relu.1', 'conv.1', 'norm.2', 'relu.2',
    'conv.2'.
201         # They are also in the checkpoints in model_urls. This pattern is
    used
202         # to find such keys.
203         pattern = re.compile(
204 
    r'^(.*denselayer\d+\.(?:norm|relu|conv))\.((?:[12])\.(?:weight|bias|running
    _mean|running_var))$')
205         state_dict = model_zoo.load_url(model_urls['densenet161'])
206         for key in list(state_dict.keys()):
207             res = pattern.match(key)
208             if res:
209                 new_key = res.group(1) + res.group(2)
210                 state_dict[new_key] = state_dict[key]
211                 del state_dict[key]
212         model.load_state_dict(state_dict)
213     model.classifier = None
214     features = model.features
215     features.block0 = nn.Sequential(features.conv0, features.norm0,
    features.relu0)
216 
217     features.denseblock1 = nn.Sequential(*list(features.denseblock1))
218     features.transition1 = nn.Sequential(*list(features.transition1)[:-1])
219 
220     features.denseblock2 = nn.Sequential(*list(features.denseblock2))
221     features.transition2 = nn.Sequential(*list(features.transition2)[:-1])
222 
223     features.denseblock3 = nn.Sequential(*list(features.denseblock3))
224     features.transition3 = nn.Sequential(*list(features.transition3)[:-1])
225 
226     features.denseblock4 = nn.Sequential(*(list(features.denseblock4) +
    [features.norm5]))
227     model.features = features
228     return model
229 
230 
231 class _DenseLayer(nn.Sequential):
232     def __init__(self, num_input_features, growth_rate, bn_size,
    drop_rate, dilation):
233         super(_DenseLayer, self).__init__()
234         self.add_module('norm1', nn.BatchNorm2d(num_input_features)),
235         self.add_module('relu1', nn.ReLU(inplace=True)),
236         self.add_module('conv1', nn.Conv2d(num_input_features, bn_size *
237                         growth_rate, kernel_size=1, stride=1, bias=False)),
238         self.add_module('norm2', nn.BatchNorm2d(bn_size * growth_rate)),
239         self.add_module('relu2', nn.ReLU(inplace=True)),
240         self.add_module('conv2', nn.Conv2d(bn_size * growth_rate,
    growth_rate,
241                         kernel_size=3, stride=1, padding=dilation,
    bias=False, dilation=dilation)),
242         self.drop_rate = drop_rate
243 
244     def forward(self, x):
245         new_features = super(_DenseLayer, self).forward(x)
246         if self.drop_rate > 0:
247             new_features = F.dropout(new_features, p=self.drop_rate,
    training=self.training)
```

```
248         return torch.cat([x, new_features], 1)
249
250
251 class _DenseBlock(nn.Sequential):
252     def __init__(self, num_layers, num_input_features, bn_size,
    growth_rate, drop_rate, dilation=1):
253         super(_DenseBlock, self).__init__()
254         for i in range(num_layers):
255             layer = _DenseLayer(num_input_features + i * growth_rate,
    growth_rate, bn_size, drop_rate, dilation)
256             self.add_module('denselayer%d' % (i + 1), layer)
257
258
259 class _Transition(nn.Sequential):
260     def __init__(self, num_input_features, num_output_features):
261         super(_Transition, self).__init__()
262         self.add_module('norm', nn.BatchNorm2d(num_input_features))
263         self.add_module('relu', nn.ReLU(inplace=True))
264         self.add_module('conv', nn.Conv2d(num_input_features,
    num_output_features,
265                                           kernel_size=1, stride=1,
    bias=False))
266         self.add_module('pool', nn.AvgPool2d(kernel_size=2, stride=2))
267
268
269 class DenseNet(nn.Module):
270     r"""Densenet-BC model class, based on
271     `"Densely Connected Convolutional Networks"
    <https://arxiv.org/pdf/1608.06993.pdf>`_
272
273     Args:
274         growth_rate (int) - how many filters to add each layer (`k` in
    paper)
275         block_config (list of 4 ints) - how many layers in each pooling
    block
276         num_init_features (int) - the number of filters to learn in the
    first convolution layer
277         bn_size (int) - multiplicative factor for number of bottle neck
    layers
278           (i.e. bn_size * k features in the bottleneck layer)
279         drop_rate (float) - dropout rate after each dense layer
280         num_classes (int) - number of classification classes
281     """
282     def __init__(self, growth_rate=32, block_config=(6, 12, 24, 16),
283                  num_init_features=64, bn_size=4, drop_rate=0,
    num_classes=1000,
284                  is_downsamples=[True, True, True, True], dilations=[1,
    1, 1, 1, 1]):
285
286         super(DenseNet, self).__init__()
287         self.is_downsamples = is_downsamples
288         self.dilations = dilations
289         # First convolution
290         self.features = nn.Sequential(OrderedDict([
291             ('conv0', nn.Conv2d(3, num_init_features, kernel_size=7,
    stride=2, padding=3, bias=False)),
292             ('norm0', nn.BatchNorm2d(num_init_features)),
293             ('relu0', nn.ReLU(inplace=True)),
294             ('pool0', nn.MaxPool2d(kernel_size=3, stride=2, padding=1)),
295         ]))
```

```
296
297         # Each denseblock
298         num_features = num_init_features
299         for i, num_layers in enumerate(block_config):
300             block = _DenseBlock(num_layers=num_layers,
    num_input_features=num_features,
301                                 bn_size=bn_size, growth_rate=growth_rate,
    drop_rate=drop_rate, dilation=dilations[i])
302             self.features.add_module('denseblock%d' % (i + 1), block)
303             num_features = num_features + num_layers * growth_rate
304             if i != len(block_config) - 1:
305                 trans = _Transition(num_input_features=num_features,
    num_output_features=num_features // 2)
306                 self.features.add_module('transition%d' % (i + 1), trans)
307                 num_features = num_features // 2
308
309         # Final batch norm
310         self.features.add_module('norm5', nn.BatchNorm2d(num_features))
311
312         # Linear layer
313         self.classifier = nn.Linear(num_features, num_classes)
314
315         # Official init from torch repo.
316         for m in self.modules():
317             if isinstance(m, nn.Conv2d):
318                 nn.init.kaiming_normal(m.weight.data)
319             elif isinstance(m, nn.BatchNorm2d):
320                 m.weight.data.fill_(1)
321                 m.bias.data.zero_()
322             elif isinstance(m, nn.Linear):
323                 m.bias.data.zero_()
324
325     def forward(self, x):
326         x = self.features.block0(x)  # 1/2
327
328         x = self.features.denseblock1(x)
329         x = self.features.transition1(x)
330
331         x = self.features.denseblock2(x)
332         x = self.features.transition2(x)
333
334         x = self.features.denseblock3(x)
335         x = self.features.transition3(x)
336
337         x = self.features.denseblock4(x)
338         return x
339
340
341 if __name__ == "__main__":
342     net = densenet169(pretrained=True).cuda()
343     pdb.set_trace()
344     x = torch.Tensor(2, 3, 256, 256).cuda()
345     sb = net(Variable(x))
346     pdb.set_trace()
```