Layer Calculations

Bert Gollnick

Layer Calculations

Problem

```python
model = nn.Sequential(OrderedDict([
    ('conv1', nn.Conv2d(3, 8, 3)), # out: (BS, 8, 30, 30)
    ('relu1', nn.ReLU()),
    ('pool', nn.MaxPool2d(2, 2)), # out: (BS, 8, 15, 15)
    ('conv2', nn.Conv2d(8, 16, 3)), # out: (BS, 16, 13, 13)
    ('relu2', nn.ReLU()),
    ('pool2', nn.MaxPool2d(2, 2)), # out: (BS, 16, 6, 6)
    ('flatten', nn.Flatten()),  # shape: (3, 16*6*6)
    ('fc1', nn.Linear(16 * 6 * 6, 127)),
    ('relu3', nn.ReLU()),
    ('fc2', nn.Linear(128, 64)),
    ('relu4', nn.ReLU()),
    ('fc3', nn.Linear(64, 1)),
    ('sigmoid', nn.Sigmoid())
]))
```

```
    140     for module in self:
--> 141         input = module(input)
    142     return input

File c:\anaconda\envs\pytorch\lib\site-packages\torch\nn\modules\module.py:1102, in Module._call_impl(self, *input, **kwargs)
   1098 # If we don't have any hooks, we want to skip the rest of the logic in
   1099 # this function, and just call forward.
   1100 if not (self._backward_hooks or self._forward_hooks or self._forward_pre_hooks or _global_backward_hooks
   1101         or _global_forward_hooks or _global_forward_pre_hooks):
-> 1102     return forward_call(*input, **kwargs)
   1103 # Do not call functions when jit is used
   1104 full_backward_hooks, non_full_backward_hooks = [], []

File c:\anaconda\envs\pytorch\lib\site-packages\torch\nn\modules\linear.py:103, in Linear.forward(self, input)
    102 def forward(self, input: Tensor) -> Tensor:
--> 103     return F.linear(input, self.weight, self.bias)

File c:\anaconda\envs\pytorch\lib\site-packages\torch\nn\functional.py:1848, in linear(input, weight, bias)
   1846 if has_torch_function_variadic(input, weight, bias):
   1847     return handle_torch_function(linear, (input, weight, bias), input, weight, bias=bias)
-> 1848 return torch._C._nn.linear(input, weight, bias)

RuntimeError: mat1 and mat2 shapes cannot be multiplied (1x127 and 128x64)
```

Layer Calculations

Tensor Dimensions

Dimension	Structure	Used for	Example Size
1	batch_size	Labels/predictions	[16]
2	batch_size, features	nn.Linear()	[16, 512]
3	batch_size, channels, features	nn.Conv1d()	[16, 1, 512]
4	batch_size, channels, height, width	nn.Conv2d()	[16, 1, 224, 224]
5	batch_size, channels, depth, height, width	nn.Conv3d()	[16, 1, 5, 224, 224]

Layer Calculations

Conv2d Layers

- Requires channels: in_channels and out_channels

```
(class) Conv2d(in_channels: int, out_channels: int, kernel_size: _size_2_t,
stride: _size_2_t = 1, padding: _size_2_t | str = 0, dilation: _size_2_t = 1,
groups: int = 1, bias: bool = True, padding_mode: str = 'zeros', device: Any |
None = None, dtype: Any | None = None)
```

in_channels

Color channels: 1

Color channels: 3

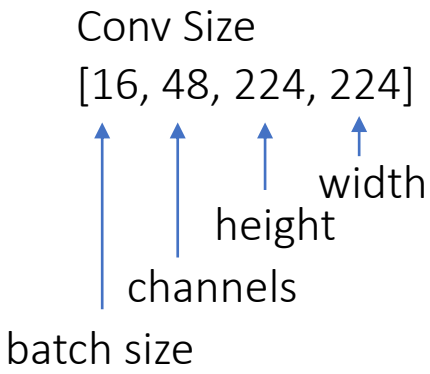
out_channels

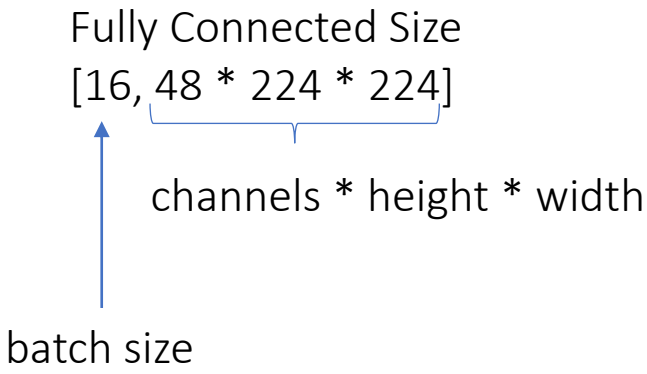
- number of channels created by convolution
- depends on kernel size
- will be in_channel size for next convolutional layer
- Best practice: conv layers get more and more channels at beginning of network

Layer Calculations

Transition from Convolutional Layer to Fully Connected

- Conv2d has a 4D shape
- Fully connected has a 2D shape
- Can be achieved with nn.Flatten()





Layer Calculations

Fully Connected Layers

- Require features NOT channels

- in_features: input sample size
- out_features: output sample size

```
(class) Linear(in_features: int, out_features: int, bias: bool = True, device:
Any | None = None, dtype: Any | None = None)
```

```
('fc1', nn.Linear(16 * 6 * 6, 128))
('relu3', nn.ReLU()),
('fc2', nn.Linear(128, 64)),
('relu4', nn.ReLU()),
('fc3', nn.Linear(64, 1)),
```