Lab 4: Build a CNN

11210IPT 553000

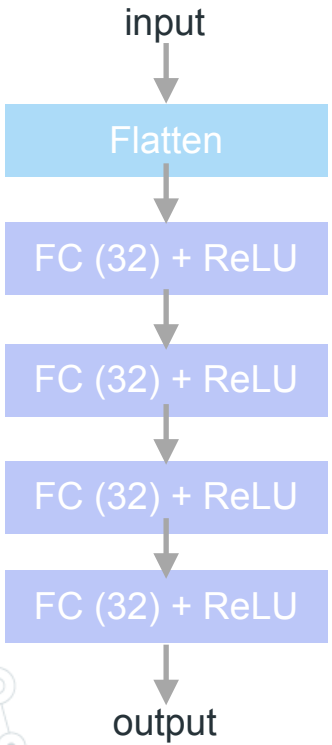
Deep Learning in Biomedical Optical Imaging

2023/10/16

Outlines

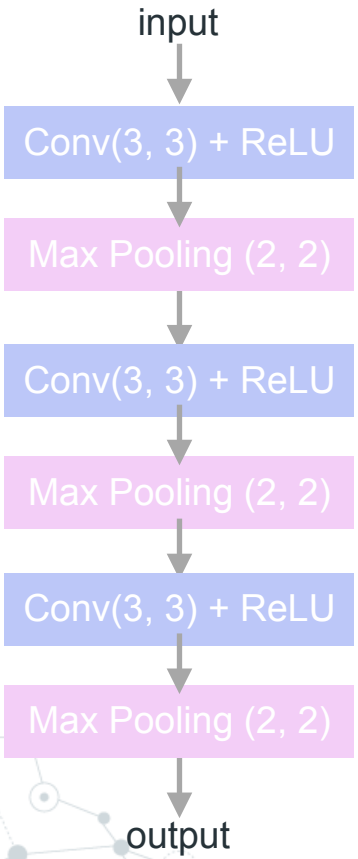
- Build a Model
- PyTorch Layers for CNN
- Global Average Pooling
- Homework 3

Build a Model - ANN




```python
class LinearModel(nn.Module):
    def __init__(self):
        super().__init__()
        self.flatten = nn.Flatten()
        self.fc1 = nn.Linear(256*256*1, 32)
        self.fc2 = nn.Linear(32, 32)
        self.fc3 = nn.Linear(32, 32)
        self.fc4 = nn.Linear(32, 1)

    def forward(self, x):
        x = self.flatten(x)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = F.relu(self.fc3(x))
        return self.fc4(x)
```

Build a Model - CNN



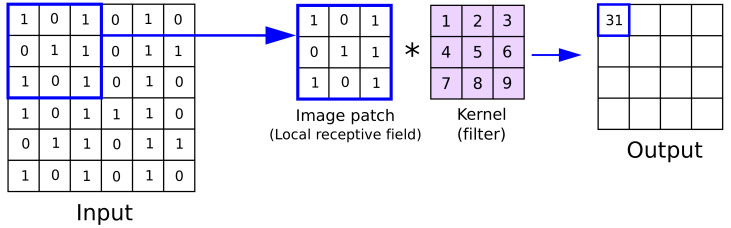

```python
class ConvModel(nn.Module):
    def __init__(self):
        super().__init__()

        # 1 channel, and using 3x3 kernels for simplicity, 256*256
        self.conv1 = nn.Conv2d(1, 32, kernel_size=3, stride=1, padding='same')
        self.pool1 = nn.MaxPool2d(kernel_size=2, stride=2) # 128*128

        self.conv2 = nn.Conv2d(32, 32, kernel_size=3, stride=1, padding='same') # 128*128
        self.pool2 = nn.MaxPool2d(kernel_size=2, stride=2) # 64*64

        self.conv3 = nn.Conv2d(32, 32, kernel_size=3, stride=1, padding='same') # 64*64
        self.pool3 = nn.MaxPool2d(kernel_size=2, stride=2) # 32*32

        # Adjust flattened dimensions based on the output size of your last pooling layer
        flattened_dim = 32 * 32 * 32

        self.fc1 = nn.Linear(flattened_dim, 32)
        self.fc2 = nn.Linear(32, 1)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = self.pool1(x)

        x = F.relu(self.conv2(x))
        x = self.pool2(x)

        x = F.relu(self.conv3(x))
        x = self.pool3(x)

        # Flatten the output for the fully connected layers
        x = x.reshape(x.size(0), -1)  # x.size(0) is the batch size

        x = F.relu(self.fc1(x))
        return self.fc2(x)
```

PyTorch Layers for CNN

Docs > torch.nn

Convolution Layers

`nn.Conv1d`	Applies a 1D convolution over an input signal composed of several input planes.
`nn.Conv2d`	Applies a 2D convolution over an input signal composed of several input planes.
`nn.Conv3d`	Applies a 3D convolution over an input signal composed of several input planes.
`nn.ConvTranspose1d`	Applies a 1D transposed convolution operator over an input image composed of several input planes.
`nn.ConvTranspose2d`	Applies a 2D transposed convolution operator over an input image composed of several input planes.
`nn.ConvTranspose3d`	Applies a 3D transposed convolution operator over an input image composed of several input planes.



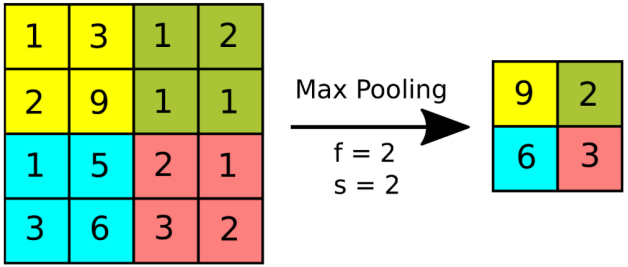
https://pytorch.org/docs/stable/nn.html#convolution-layers

PyTorch Layers for CNN

Docs > torch.nn

Pooling layers

`nn.MaxPool1d`	Applies a 1D max pooling over an input signal composed of several input planes.
`nn.MaxPool2d`	Applies a 2D max pooling over an input signal composed of several input planes.
`nn.MaxPool3d`	Applies a 3D max pooling over an input signal composed of several input planes.
`nn.MaxUnpool1d`	Computes a partial inverse of `MaxPool1d`.
`nn.MaxUnpool2d`	Computes a partial inverse of `MaxPool2d`.
`nn.MaxUnpool3d`	Computes a partial inverse of `MaxPool3d`.
`nn.AvgPool1d`	Applies a 1D average pooling over an input signal composed of several input planes.



https://pytorch.org/docs/stable/nn.html#pooling-layers

Global Average Pooling

Docs > torch.nn > AdaptiveAvgPool2d

ADAPTIVEAVGPOOL2D

CLASS torch.nn.AdaptiveAvgPool2d(*output_size*) [SOURCE]

Applies a 2D adaptive average pooling over an input signal composed of several input planes.

The output is of size H x W, for any input size. The number of output features is equal to the number of input planes.

Parameters

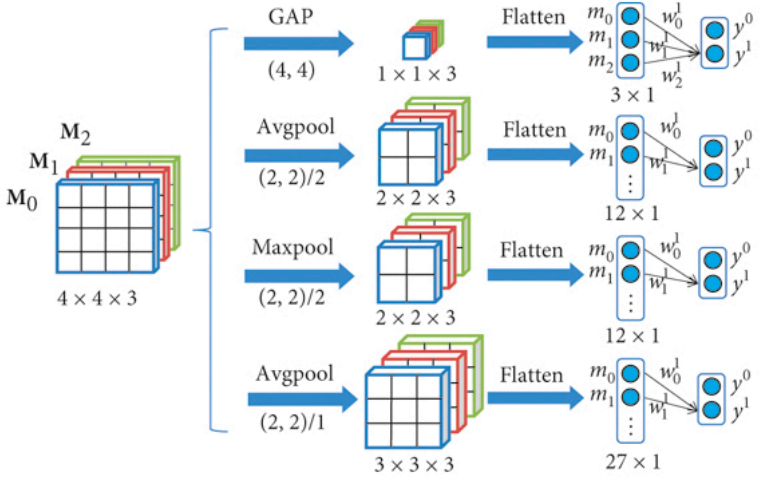
output_size (*Union*[*int*, *None*, *Tuple*[*Optional*[*int*], *Optional*[*int*]]]) – the target output size of the image of the form H x W. Can be a tuple (H, W) or a single H for a square image H x H. H and W can be either a `int`, or `None` which means the size will be the same as that of the input.

Shape:

- Input: (N, C, H_{in}, W_{in}) or (C, H_{in}, W_{in}).
- Output: (N, C, S_0, S_1) or (C, S_0, S_1), where $S = \text{output\_size}$.

Examples

```
>>> # target output size of 5x7
>>> m = nn.AdaptiveAvgPool2d((5, 7))
>>> input = torch.randn(1, 64, 8, 9)
>>> output = m(input)
>>> # target output size of 7x7 (square)
>>> m = nn.AdaptiveAvgPool2d(7)
>>> input = torch.randn(1, 64, 10, 9)
```

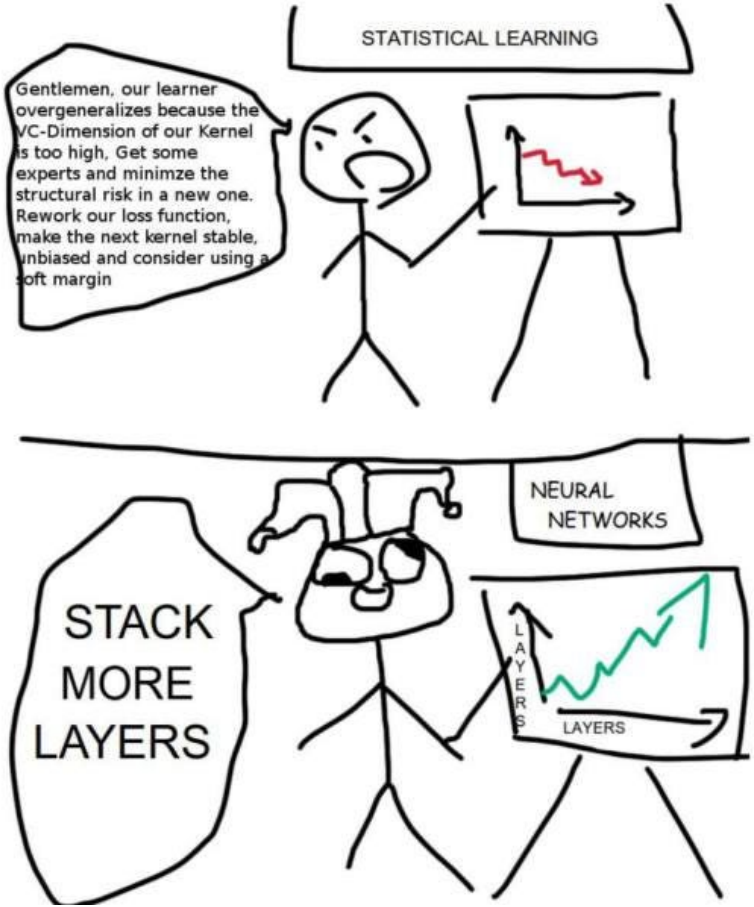


https://pytorch.org/docs/stable/generated/torch.nn.AdaptiveAvgPool2d.html#torch.nn.AdaptiveAvgPool2d

Homework 3

- **Deadline**: 23:59, 30th Oct. (GMT+8)
- We need to write a report to answer questions. Details are in hw3_description.pdf
- 🚨 **Important**: Make sure your commit is timestamped before the deadline. Late submissions might not be graded or could incur a penalty. Only the GitHub link is required on NTHU EEclass.

CODING TIME!!


STATISTICAL LEARNING
Gentlemen, our learner overgeneralizes because the VC-Dimension of our Kernel is too high, Get some experts and minimze the structural risk in a new one. Rework our loss function, make the next kernel stable, unbiased and consider using a soft margin
NEURAL NETWORKS
STACK MORE LAYERS
LAYERS
LAYERS