Introduction to Deep Learning (CS474)

Lecture 17

Rupam

Outline

- **Module 2**
  - Discussion on building a simple ConvNet using PyTorch

Rupam

Introduction

- I hope that you remember our discussion on CIFAR-10 dataset!
- With the building blocks which was discussed in our earlier lectures, we can now proceed to build our convolutional neural network in PyTorch for detecting birds and airplanes.
- Look into the following code snippet for *model* building!

```
nn.Conv2d(3, 16, kernel_size=3, padding=1),
nn.Tanh(),
nn.MaxPool2d(2),
nn.Conv2d(16, 8, kernel_size=3, padding=1),
nn.Tanh(),
nn.MaxPool2d(2)
```

Slide credit: E. STEVENS, L. ANTIGA, and T. VIEHMANN

Rupam

Putting it all together

- The first **convolution** takes us from 3 RGB channels to 16, thereby giving the network a chance to generate 16 independent features that operate to (hopefully) *discriminate* **low-level features** of birds and airplanes.
- Then we apply the **`Tanh` activation function**.
- The resulting 16-channel 32 × 32 image is **pooled** to a 16-channel 16 × 16 image by the first MaxPool .
- At this point, the downsampled image undergoes another convolution that generates an 8-channel 16 × 16 output.
- With any luck, this output will consist of **higher-level features**.
- Again, we apply a **`Tanh`** activation and then **pool** to an 8-channel 8 × 8 output.

Slide credit: E. STEVENS, L. ANTIGA, and T. VIEHMANN

Rupam

Putting it all together

- Where does this end?

- After the input image has been reduced to a set of 8 × 8 features, we expect to be able to output some probabilities from the network.

- However, probabilities are a pair of numbers in a 1D vector (one for airplane, one for bird), but here we're still dealing with multi-channel 2D features.

- we already know what we need to do: turn the 8-channel 8 × 8 image into a 1D vector and complete our network with a set of fully connected layers.

Slide credit: E. STEVENS, L. ANTIGA, and T. VIEHMANN

Rupam

Putting it all together

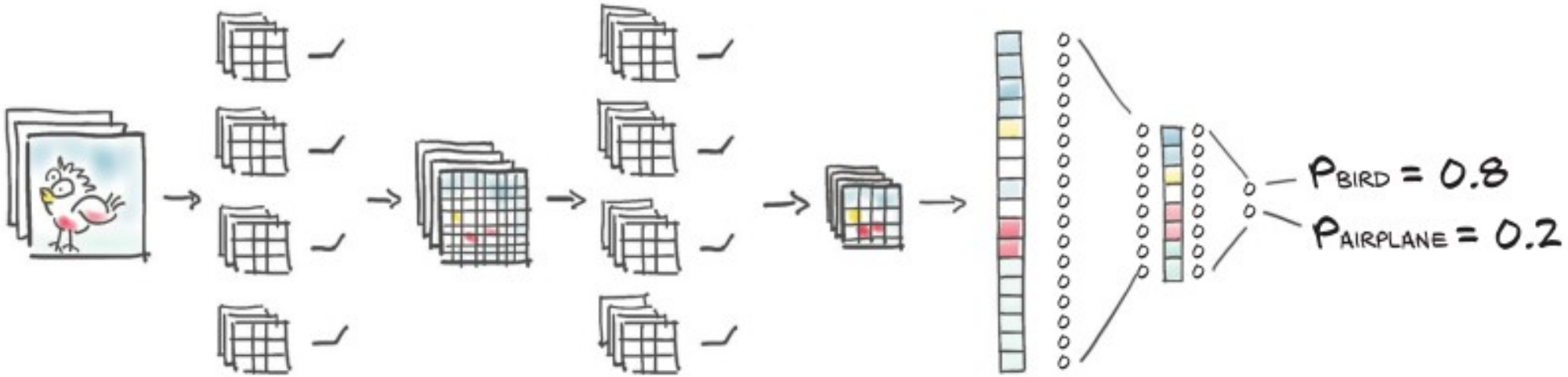


Image credit: E. STEVENS, L. ANTIGA, and T. VIEHMANN

Rupam

Subclassing nn.Module

- At some point in developing neural networks, we will find ourselves in a situation where we want to compute something that the *premade* modules do not cover..

- we learn how to make our own **`nn.Module`** subclasses that we can then use just like the prebuilt ones or nn.Sequential.

- When we want to build models that do more complex things than just applying one layer after another, we need to leave nn.Sequential for something that gives us added flexibility.

- **PyTorch** allows us to use any computation in our model by subclassing nn.Module.

Slide credit: E. STEVENS, L. ANTIGA, and T. VIEHMANN

Rupam

Subclassing nn.Module

- In order to subclass nn.Module, at a minimum we need to define a **forward** function that takes the inputs to the module and returns the output.

- With **PyTorch**, if we use standard torch operations, *autograd* will take care of the backward pass automatically; and indeed, an `nn.Module` never comes with a backward .

- Typically, our computation will use other modules–**premade** like convolutions or customized.

- To include these submodules, we typically define them in the constructor **`__init__`** and assign them to self for use in the forward function.

Slide credit: E. STEVENS, L. ANTIGA, and T. VIEHMANN

Rupam

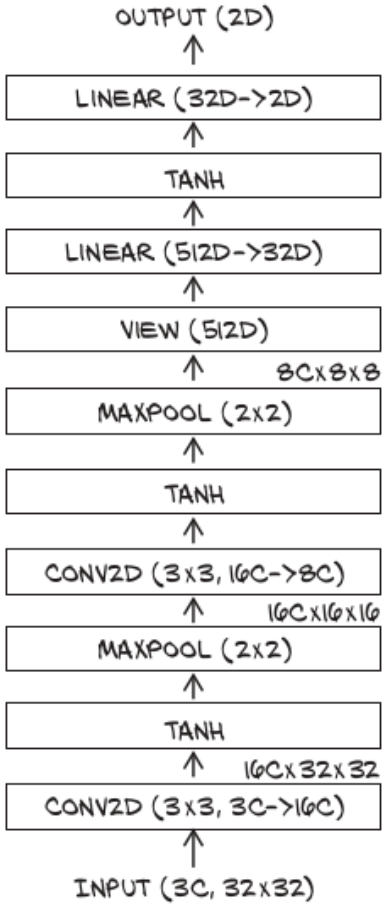
Subclassing nn.Module (example)

```
import torch
import torch.nn as nn

class Net(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 16, kernel_size=3, padding=1)
        self.act1 = nn.Tanh()
        self.pool1 = nn.MaxPool2d(2)
        self.conv2 = nn.Conv2d(16, 8, kernel_size=3, padding=1)
        self.act2 = nn.Tanh()
        self.pool2 = nn.MaxPool2d(2)
        self.fc1 = nn.Linear(8 * 8 * 8, 32)
        self.act3 = nn.Tanh()
        self.fc2 = nn.Linear(32, 2)

    def forward(self, x):
        out = self.pool1(self.act1(self.conv1(x)))
        out = self.pool2(self.act2(self.conv2(out)))
        out = out.view(-1, 8 * 8 * 8) # <1>
        out = self.act3(self.fc1(out))
        out = self.fc2(out)
        return out
```

Slide credit: E. STEVENS, L. ANTIGA, and T. VIEHMANN

Rupam

Subclassing nn.Module (example)



Slide credit: E. STEVENS, L. ANTIGA, and T. VIEHMANN

References

- All the contents present in the slides are taken from various online resources. Due credit is given in the respective slides.These slides are used for *academic* purposes only.