WEEKLY PARTICIPATION 9: TRACKING SHAPES THROUGH CNNS

Recall that a CNN consists of several convolutional layers (each consisting of multiple channels) followed by a fully-connected layer. To determine the parameters of each layer, it is important to understand how the shape of your data transforms as it passes from layer to layer. For instance:

- The channels must always have size larger than the kernel which is applied to them. This means that if the previous layers' output consists of 3×3 channels, then the next layers' kernels cannot be 5×5.
- Once the output from the convolutional layers is flattened to become a vector before it is fed into the fully connected portion of the CNN, the size of that vector must match the size expected by the first MLP layer as input.

This participation will give you practice in determining these sizes, using the LeNet5 architecture discussed in class.

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

    self.conv1 = nn.Conv2d(in_channels = 1, out_channels = 6,
                           kernel_size = 5, stride = 1, padding = 0)
    self.conv2 = nn.Conv2d(in_channels = 6, out_channels = 16,
                           kernel_size = 5, stride = 1, padding = 0)
    self.conv3 = nn.Conv2d(in_channels = 16, out_channels = 120,
                           kernel_size = 5, stride = 1, padding = 0)
    self.linear1 = nn.Linear(120, 84)
    self.linear2 = nn.Linear(84, 10)
    self.tanh = nn.Tanh()
    self.avgpool = nn.AvgPool2d(kernel_size = 2, stride = 2)

  def forward(self, x):
    x = self.conv1(x)
    x = self.tanh(x)
    x = self.avgpool(x)
    x = self.conv2(x)
    x = self.tanh(x)
    x = self.avgpool(x)
    x = self.conv3(x)
    x = self.tanh(x)

    x = x.reshape(x.shape[0], -1)
    x = self.linear1(x)
    x = self.tanh(x)
    x = self.linear2(x)
    return x

model = LeNet()
```

Let `x` be a tensor with shape $[64,32,32]$ that consists of a minibatch of inputs returned by the Dataloader. Let's understand how the shape of the data evolves during the call `model(x)`. Give the reasoning of your answers to each of these questions. Since it is important to be able to parse the PyTorch documentation, I suggest that you read the

documentation for each of the elements used, focusing on the portions describing the shapes of the input and output tensors.

- What is the shape of the output tensor from the call `x = self.conv1(x)`?
- What is the shape of the output tensor after the first call to `avgpool`?
- What is the shape of the output tensor after the call `x = self.conv2(x)`?
- What is the shape of the output tensor after the second call to `avgpool`?
- What is the shape of the output tensor after the call `x = self.conv3(x)`?
- What is the shape of the output tensor after the call to `x = x.reshape(x.shape[0], -1)`? Read the documentation for the `Tensor.reshape` function, and explain why using these specific input arguments causes the output to have the shape it does.
- What is the shape of the output tensor after the call `x = self.linear1(x)`?
- What is the shape of the output tensor after the call `x = self.linear2(x)`?
- How many parameters are there in the convolution layers, and how many are there in the fully connected layers? Explain both your answers.
- In the example given in class, the input images are FashionMNIST, which are 28×28. Explain where exactly an error would occur if you called `model(x)` on a tensor of size $[64, 28, 28]$, and explain why/how we were able to use this architecture on 28×28 images in our in-class example.