Awesome Coding Interview

[Note: Please download this file and send it to "mdhasan.cpsd@gmail.com" after completing the task.

1. You want to implement the following architecture in PyTorch. It consists of multiple convolutional layers with different strides and padding (specified in the Model constructor below), each followed by a ReLU activation.

   Remark: All convolutional layers in PyTorch are per default in "valid" mode, which means no padding at the borders is applied (padding value of 0). If the padding value is larger than 0, padding is applied before the convolution in height and width dimension on both sides of the image. The padding value (in the constructor) defines the padding width of one side of the image.

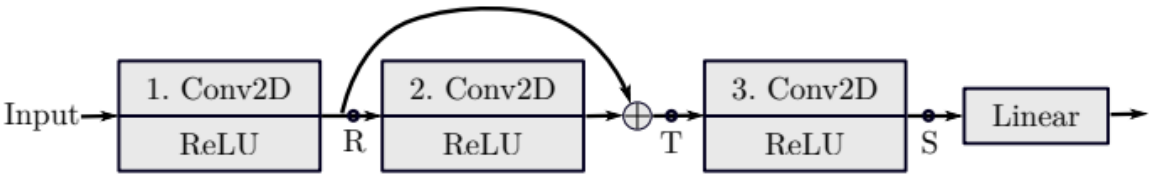

Figure 2: CNN Architecture

Define the model below,

```
import torch
from torch import nn
from torch.nn.functional import relu

class Net(nn.Module):

    def __init__(self):
        super(Net, self).__init__()
        # 3 input image channel, 6 output channels, 5x5 square convolution
        # kernel
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.conv2 = nn.Conv2d(6, 16, 5, 1,1) # Stride =1 , padding =1
        self.conv3 = nn.Conv2d(16, 20, 5, 1, 1) # Stride = 1, Padding = 1
        # an affine operation: y = Wx + b
        self.fc1 = nn.Linear(20 * 5 * 5, 120)  # 5*5 from image dimension
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)
```

```
        self.fc4 = nn.Linear(10, 2)

    def forward(self, x):
        # Max pooling over a (2, 2) window
        x = F.max_pool2d(F.relu(self.conv1(x)), (2, 2))
        # If the size is a square, you can specify with a single number
        x = F.max_pool2d(F.relu(self.conv2(x)), 2)
        x = torch.flatten(x, 1) # flatten all dimensions except the batch dimension
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = F.relu(self.fc3(x))
        x = self.fc4(x)
        return x


net = Net()
print(net)
```

2. To check the correct handling of the image sizes in your network, fill in the following tables. First, find the correct order of the convolution objects defined in the constructor. Then, enter the shape of the weights for the above-defined convolutions (bias can be neglected) and the size of your images at the locations **R, T, S** for the defined input size. The weights should be defined as ***(number of kernels, channel-dimension, x-dimension, y-dimension)*** while the image sizes should have the shape ***(batchsize, number of channels, x-dimension, y-dimension)***.

   in channels is set to 3 and hidden channels is set to 4.

   Table-1:

convolution operation	**variable name**	**weight shape**
Conv2D	conv1	(5, 6, 3, 3)
Conv2D	conv2	(5, 16, 6, 6)
Conv2D	conv3	(5, 20, 16, 16)

Table-2:

```
conv = nn.Conv2d(in_channels=3,  # number of input channels
                 out_channels=6, # number of output channels
                 kernel_size=5)
x = torch.randn(10, 3, 32, 32)
y = conv(x)
y.shape
conv2 = nn.Conv2d(in_channels=6,  # number of input channels
                 out_channels=16, # number of output channels
                 kernel_size=5,
                  stride= 1,
                  padding=1)
x = torch.randn(10, 6, 28, 28)
y = conv2(x)
y.shape
conv3 = nn.Conv2d(in_channels=16,  # number of input channels
                 out_channels=20, # number of output channels
                 kernel_size=5,
                  stride= 1,
                  padding=1)
x = torch.randn(10, 16, 26, 26)
y = conv3(x)
y.shape
```

result at position	**image shape**
Input	(10, 3, 32, 32)
R	([10, 6, 28, 28])
T	([10, 16, 26, 26])
S	([10,20, 24, 24])

3. Implement the forward pass for the given architecture in python using the PyTorch library. The input x is located on the CPU.

```
def forward(self , x):
   print(x.shape) # >> ( 1 0 , 3 , 3 2 , 3 2 )
return y
```

```
import torch
import torch.nn as nn
import torch.nn.functional as F


class Net(nn.Module):

    def __init__(self):
        super(Net, self).__init__()
        # 1 input image channel, 6 output channels, 5x5
square convolution
        # kernel
        self.conv1 = nn.Conv2d(10, 3, 32)
        self.conv2 = nn.Conv2d(3, 16, 32)
        # an affine operation: y = Wx + b
        self.fc1 = nn.Linear(16 * 5 * 5, 120)  # 5*5 from
image dimension
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        # Max pooling over a (2, 2) window
        x = F.max_pool2d(F.relu(self.conv1(x)), (2, 2))
        # If the size is a square, you can specify with a
single number
        x = F.max_pool2d(F.relu(self.conv2(x)), 2)
        x = torch.flatten(x, 1) # flatten all dimensions
except the batch dimension
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x


net = Net()
print(net)
```