HW Assignment 4 (Due by 10:00am on November 22)

1 Implementation (100 points)

Download the skeleton code from Cougar Course. Implement the ConvNet for digit classification, as explained in the 4 steps below, using PyTorch. Make sure that you organize your code in folders as shown in the table below.

```
hw04/
  pytorch/
    train_cnn.py
    cnnExercise.py
  data/
    mnist/
```

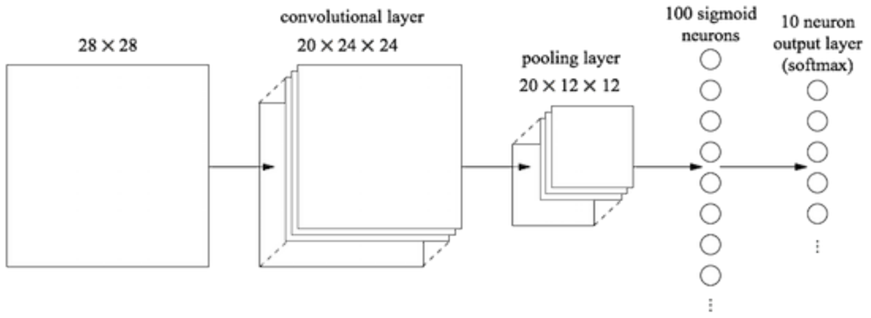
Write code only in train cnn.py. In each step you will train the corresponding architecture and report the accuracy on the test data. Hyper-parameters are kept unchanged from one step to the next, unless changes are explicitly mentioned. By default, the code is written to run on the CPU, with the expected accuracy numbers shown in cnnExercise.py. If your PyTorch installation and architecture supports GPUs, you can set self.use_gpu = False inside train cnn.py which can make the code run twice as fast. It is recommended that you read and understand the code in cnnExercise.py.

Most of the functionality needed for completing this assignment is implemented in the package torch.nn. One approach is to create a Sequential model, to which you add modules in order using the add_module() method. All types of layers and processing you need are already implemented in torch.nn, by subclassing from torch.nn.Module. Alternative, but very similar approaches for coding CNNs can also be seen in the PyTorch tutorials, e.g. the Training an Image Classifier tutorial. Relevant classes and functions: *nn.Sequential, nn.Linear, nn.Sigmoid, nn.ReLU, nn.LogSoftmax, nn.Conv2D, nn.MaxPool2d, nn.Dropout*.

The output of a convolutional layer for one example is a 3D tensor. To be used as input for a fully connected layer, it needs to be transformed into a 1D tensor. For your convenience, a generic shape transformation is implemented as a PyTorch module in the ViewOP class. You will need to use this between the last convolutional layer and the first fully connected layer.

Step 1: Baseline with one hidden layer: Create a shallow architecture using a single hidden layer, fully connected, containing 100 neurons, with sigmoid activation function. Train for 30 epochs, using SGD with a learning rate of 0.1, a mini-batch size of 10, and no regularization.

Step 2: One convolutional + one hidden layer: Insert a convolutional layer at the beginning of the network, followed by a max-pooling layer and the fully connected layer from step 1. Use 5x5 local receptive fields, a stride of 1, and 20 kernels. The max-pooling layer should combine features using 2x2 pooling windows. The overall architecture should look as in the figure below.



Step 3: Two convolutional + one hidden layer: Insert a second convolutional-pooling layer between the existing convolutional-pooling layer and the fully-connected hidden layer. Use a 5x5 local receptive field for 40 kernels and pool over 2x2 regions.

Step 4: Two convolutional + one hidden layer, ReLU: Replace sigmoid with ReLU as activation function in the entire network from step 3. Train the model using a new learning rate of 0.03.

2 Submission

Turn in a hard copy of your homework report at the beginning of class on the due date. Electronically submit on Cougar Course a hw04.zip file that contains the hw04 folder in which you change code **only in the required files**.

On a Linux system, creating the archive can be done using the command:

```
> zip -r hw04.zip hw04.
```

Please observe the following when handing in homework:

1. Structure, indent, and format your code well.
2. Use adequate comments, both block and in-line to document your code.
3. On the theory assignment, clear and complete explanations and proofs of your results are as important as getting the right answer.
4. Make sure your code runs correctly when used in the directory structure shown above.