

# HW Assignment 8, Theory and Implementation

Problems marked with a (\*) are mandatory only for ITCS 8156 students. Bonus problems are optional, solving them will result in extra points.

## 1 Theory (25 points)

1. [CNN Complexity] Consider the final convolutional network from step 7 in the implementation part below. What is the total number of parameters? Show your work, i.e. a breakdown of the total number of parameters, layer by layer. Describe at least three different ways of reducing the number of parameters in this network, and discuss their relative advantages/disadvantages.

## 2 Implementation (100 points)

Implement the ConvNet for digit classification, as explained in the 7 steps below, using PYTORCH. Make sure that you organize your code in folders as shown in the table below.

```
itcs6156/  
  hw08/  
    pytorch/  
      train_cnn.py  
      cnnExercise.py  
      expand_mnist.py  
      output.txt  
      report.pdf  
      theory.pdf  
    data/  
    mnist/
```

Write code only in `train_cnn.py`. The traces from running `python3 cnnExercise.py <mode>` for each of the 7 steps below should be saved in `output.txt`. In each step you will train the corresponding architecture and report the accuracy on the test data. Hyperparameters 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 = True` 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` and `expand_mnist.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`.

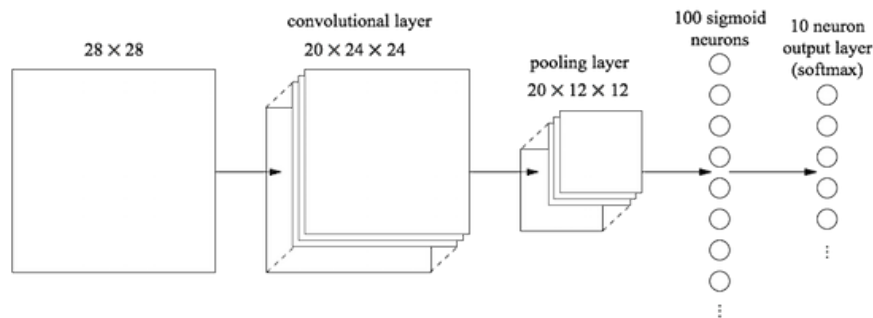
Note that the `nn.Loss` function used to compute the loss as given in the skeleton code requires as input the logarithm of the softmax probabilities, which can be done using an `nn.LogSoftmax` layer.

Also note that each layer needs to be non-linear. As such, make sure you use a non-linear activation after each pooling layer (sigmoid for steps 1 to 3, ReLU afterwards).

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 need to add this between the last convolutional layer and the first fully connected layer, using the `add_module()` function as with all other computational layers. An example is shown in the code below, where `input_size` is the total number of values (neurons) computed by the convolutional layer: `model.add_module("ChangeView", ViewOP(-1, input_size))`.

**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.

**Step 5: Two convolutional + one hidden layer, ReLU, data augmentation:** Augment the training data by displacing each training image by a single pixel: up one pixel, down one pixel, left one pixel, or right one pixel. Do this by running the program `expand_mnist.py` in the `mnist/data/` folder. Retrain the model from step 4 on the augmented dataset.

**Step 6: Two convolutional + two hidden layers, ReLU, data augmentation:** Add a second fully-connected layer with 100 neurons to the model from step 5.

**Step 7: Two convolutional + two large hidden layers, ReLU, data augmentation, Dropout:** Increase the number of neurons on the fully connected layers to 200. Do regularization by applying Dropout on the activations of the two fully-connected layers, using a dropout rate of 0.5.

## 2.1 Bonus (25 points)

In `train_cnn.py`, we have manually split the dataset into minibatches. Write code to do this using the `torch.utils.data.DataLoader` class. You may find this Data Loading and Processing Tutorial useful.

## 2.2 Bonus (50 + 50 points)

Implement all the above using NUMPY. This requires updating backpropagation (50 points) to go through convolutional and pooling layers. How to do this is explained in the UFLDL Tutorial. Alternatively, for 50 bonus points, you can implement only forward propagation and the cost in NUMPY and use `autograd` for the gradient computation.

# 3 Submission

Electronically submit on Canvas a `hw08.zip` file that contains the `hw08` folder in which your code is in the required files, as well as the `report.pdf` containing your analysis of results and the `theory.pdf` with the theory solutions.

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

```
> zip -r hw08.zip hw08.
```

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. Make sure your code runs correctly when used in the directory structure shown above. We will not debug your code.
4. For your report, it is recommended to use an editor such as Latex or Word or Jupyter-Notebook that allows editing and proper formatting of equations, plots, and tables with results.
5. Make sure your your Canvas submission contains the correct files by downloading and unzipping it after posting on Canvas.