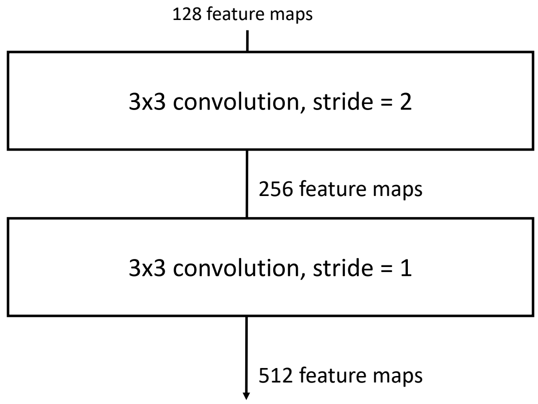
COM S 573: Machine Learning
Homework #5

1. (15 points) Given the convolutional neural network block as below



Given the input feature maps $\boldsymbol{X} \in \mathbb{R}^{64\times 64\times 128}$, all convolutional layers perform zero-padding of 1 on each side of H and W dimensions.

(a) (5 points) What is the total number of parameters in the block (you can skip bias terms)?
Sol: $3 \times 3 \times 128 \times 256 + 3 \times 3 \times 256 \times 512$

(b) (5 points) What is the total number of multi-add operations in the block?
Sol: After 1st stage the output dimension is (63-3+2)/2+1 = 32. The output dimension is 32×32. The number of multi-add operations in this stage is $3 \times 3 \times 128 \times 256 \times 32 \times 32$

After 2nd stage the output dimension is (32-3+2)/1+1 = 32. The output dimension is 32×32. The number of multi-add operations in this stage is $3 \times 3 \times 256 \times 512 \times 32 \times 32$

The total number of multi-add operations in the block are $3 \times 3 \times 128 \times 256 \times 32 \times 32 + 3 \times 3 \times 256 \times 512 \times 32 \times 32$

(c) (5 points) What is memory requirement change to store the input and output features of this block (Use percentage)?
Sol: In the first stage the memory requirement reduces to 50 percent($64 \times 64 \times 128 \implies 32 \times 32 \times 256$) and in the second stage it increases to 50 percent($32 \times 32 \times 256 \implies 32 \times 32 \times 512$). So, there is no change in memory between input and output features.

2. (20 points) Using batch normalization in neural networks requires computing the mean and variance of a tensor. Suppose a batch normalization layer takes vectors $z_1, z_2, \cdots, z_m$ as input, where m is the mini-batch size. It computes $\hat{z}_1, \hat{z}_2, \cdots, \hat{z}_m$ according to

$$\hat{z}_i = \frac{z_i - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

where

$$\mu = \frac{1}{m}\sum_{i=1}^{m} z_i, \;\; \sigma^2 = \frac{1}{m}\sum_{i=1}^{m}(z_i - \mu)^2.$$

It then applies a second transformation to obtain $\tilde{z}_1, \tilde{z}_2, \cdots, \tilde{z}_m$ using learned parameters γ and β as

$$\tilde{z}_i = \gamma \hat{z}_i + \beta.$$

In this question, you can assume that $\epsilon = 0$.

(a) (5 points) You forward-propagate a mini-batch of $m = 4$ examples in your network. Suppose you are at a batch normalization layer, where the immediately previous layer is a fully connected layer with 3 units. Therefore, the input to this batch normalization layer can be represented as the below matrix:

$$\begin{bmatrix} 12 & 14 & 14 & 12 \\ 0 & 10 & 10 & 0 \\ -5 & 5 & 5 & -5 \end{bmatrix}$$

What are $\hat{z}_i$? Please express your answer in a 3×4 matrix.

Sol:

$$\mu = \frac{1}{4}\sum_{i=1}^{4} z_i = \frac{1}{4}\begin{bmatrix} 12+14+14+12 \\ 0+10+10+0 \\ -5+5+5-5 \end{bmatrix} = \begin{bmatrix} 13 \\ 5 \\ 0 \end{bmatrix}$$

,

$$\sigma^2 = \frac{1}{4}\sum_{i=1}^{4}(z_i - \mu)^2 = \frac{1}{4}\begin{bmatrix} 1+1+1+1 \\ 25+25+25+25 \\ 25+25+25+25 \end{bmatrix} = \begin{bmatrix} 1 \\ 25 \\ 25 \end{bmatrix}$$

$$\hat{z}_i = \frac{z_i - \mu}{\sqrt{\sigma^2 + \epsilon}} = \frac{z_i - \mu}{\sigma} \implies \begin{bmatrix} -1 & 1 & 1 & -1 \\ -1 & 1 & 1 & -1 \\ -1 & 1 & 1 & -1 \end{bmatrix}$$

(b) (5 points) Continue with the above setting. Suppose $\gamma = (1, 1, 1)$, and $\beta = (0, -10, 10)$. What are $\tilde{z}_i$? Please express your answer in a 3×4 matrix.

Sol:

$$\tilde{z}_i = \gamma\hat{z}_i + \beta = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} .* \begin{bmatrix} -1 & 1 & 1 & -1 \\ -1 & 1 & 1 & -1 \\ -1 & 1 & 1 & -1 \end{bmatrix} + \begin{bmatrix} 0 \\ -10 \\ 10 \end{bmatrix} \implies \begin{bmatrix} -1 & 1 & 1 & -1 \\ -11 & -9 & -9 & -11 \\ 9 & 11 & 11 & 9 \end{bmatrix}$$

(c) (5 points) Describe the differences of computations required for batch normalization during training and testing.

Sol: We use the γ andβ from training and use the running average/population mean on the test data. population mean i.e mean of $\mu_1, mu_2, ..., \mu_n$ where μ_i is the mean of ith batch, running average take the average of the adding examples and take the over all mean of the means. where as in the training we transform the minibatches to gaussian (0,1) and transform to $y_i =$ gamma*x_i+beta.

(d) (5 points) Describe how the batch size during testing affect testing results.

Sol: The batch size of testing does not effect the performance because we consider the population mean i.e mean of $\mu_1, mu_2, ..., \mu_n$ where μ_i is the mean of ith batch.

3. (20 points) We investigate the back-propagation of the convolution using a simple example. In this problem, we focus on the convolution operation without any normalization and activation function. For simplicity, we consider the convolution in 1D cases. Given 1D inputs with a spatial size of 4 and 2 channels, *i.e.*,

$$X = \begin{bmatrix} x_{11} & x_{12} & x_{13} & x_{14} \\ x_{21} & x_{22} & x_{23} & x_{24} \end{bmatrix} \in \mathbb{R}^{2\times 4}, \tag{1}$$

we perform a 1D convolution with a kernel size of 3 to produce output Y with 2 channels. No padding is involved. It is easy to see

$$Y = \begin{bmatrix} y_{11} & y_{12} \\ y_{21} & y_{22} \end{bmatrix} \in \mathbb{R}^{2\times 2}, \tag{2}$$

where each row corresponds to a channel. There are 12 training parameters involved in this convolution, forming 4 different kernels of size 3:

$$W^{ij} = [w_1^{ij}, w_2^{ij}, w_3^{ij}], i = 1, 2, j = 1, 2, \tag{3}$$

where W^{ij} scans the i-th channel of inputs and contributes to the j-th channel of outputs.

(a) (5 points) Now we flatten X and Y to vectors as

$$\tilde{X} = [x_{11}, x_{12}, x_{13}, x_{14}, x_{21}, x_{22}, x_{23}, x_{24}]^T$$
$$\tilde{Y} = [y_{11}, y_{12}, y_{21}, y_{22}]^T$$

Please write the convolution in the form of fully connected layer as $\tilde{Y} = A\tilde{X}$ using the notations above. You can assume there is no bias term.
Hint: Note that we discussed how to view convolution layers as fully connected layers in the case of single input and output feature maps. This example asks you to extend that to the case of multiple input and output feature maps.
Sol:

$$W^{i1} = \begin{bmatrix} w_1^{11} & w_2^{11} & w_3^{11} \\ w_1^{21} & w_2^{21} & w_3^{21} \end{bmatrix} \in \mathbb{R}^{2\times3},$$

$$W^{i2} = \begin{bmatrix} w_1^{12} & w_2^{12} & w_3^{12} \\ w_1^{22} & w_2^{22} & w_3^{22} \end{bmatrix} \in \mathbb{R}^{2\times3},$$

$$\begin{bmatrix} y_{11} \\ y_{12} \\ y_{13} \\ y_{14} \end{bmatrix} = \begin{bmatrix} w_1^{11} & w_2^{11} & w_3^{11} & 0 & w_1^{21} & w_2^{21} & w_3^{21} & 0 \\ 0 & w_1^{11} & w_2^{11} & w_3^{11} & 0 & w_1^{21} & w_2^{21} & w_3^{21} \\ w_1^{12} & w_2^{12} & w_3^{12} & 0 & w_1^{22} & w_2^{22} & w_3^{22} & 0 \\ 0 & w_1^{12} & w_2^{12} & w_3^{12} & 0 & w_1^{22} & w_2^{22} & w_3^{22} \end{bmatrix} \begin{bmatrix} x_{11} \\ x_{12} \\ x_{13} \\ x_{14} \\ x_{21} \\ x_{22} \\ x_{23} \\ x_{24} \end{bmatrix}$$

(b) (5 points) Next, for the back-propagation, assume we've already computed the gradients of loss L with respect to $\tilde{Y}$:

$$\frac{\partial L}{\partial \tilde{Y}} = \left[\frac{\partial L}{\partial y_{11}}, \frac{\partial L}{\partial y_{12}}, \frac{\partial L}{\partial y_{21}}, \frac{\partial L}{\partial y_{22}}\right]^T, \tag{4}$$

Please write the back-propagation step of the convolution in the form of $\frac{\partial L}{\partial \tilde{X}} = B\frac{\partial L}{\partial \tilde{Y}}$. Explain the relationship between A and B.
Sol:

$$\tilde{Y} = A\tilde{X}$$
$$\frac{\partial L}{\partial \tilde{X}} = \frac{\partial \tilde{Y}}{\partial \tilde{X}}\frac{\partial L}{\partial \tilde{Y}} = A^T\frac{\partial L}{\partial \tilde{Y}}$$

$\implies B = A^T$

(c) (10 points) While the forward propagation of the convolution on X to Y could be written into $\tilde{Y} = A\tilde{X}$, could you figure out whether $\frac{\partial L}{\partial \tilde{X}} = B\frac{\partial L}{\partial \tilde{Y}}$ also corresponds to a convolution on $\frac{\partial L}{\partial Y}$ to $\frac{\partial L}{\partial X}$? If yes, write down the kernels for this convolution. If no, explain why.
Sol:

$$X \implies Y$$
$$\tilde{Y} = A\tilde{X}$$

$$\frac{\partial L}{\partial \tilde{X}} = B \frac{\partial L}{\partial \tilde{Y}}$$

$$\frac{\partial L}{\partial \tilde{X}} = \begin{bmatrix} w_1^{11} & w_2^{11} & w_3^{11} & 0 & w_1^{21} & w_2^{21} & w_3^{21} & 0 \\ 0 & w_1^{11} & w_2^{11} & w_3^{11} & 0 & w_1^{21} & w_2^{21} & w_3^{21} \\ w_1^{12} & w_2^{12} & w_3^{12} & 0 & w_1^{22} & w_2^{22} & w_3^{22} & 0 \\ 0 & w_1^{12} & w_2^{12} & w_3^{12} & 0 & w_1^{22} & w_2^{22} & w_3^{22} \end{bmatrix}^T \frac{\partial L}{\partial \tilde{Y}}$$

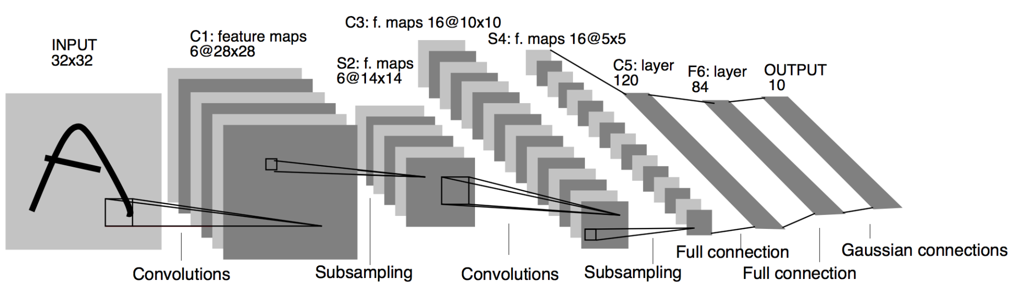
$$\begin{bmatrix} \frac{\partial L}{\partial y_{11}} & \frac{\partial L}{\partial y_{12}} \\ \frac{\partial L}{\partial y_{21}} & \frac{\partial L}{\partial y_{22}} \end{bmatrix} \Longrightarrow \begin{bmatrix} \frac{\partial L}{\partial x_{11}} & \frac{\partial L}{\partial x_{12}} & \frac{\partial L}{\partial x_{13}} & \frac{\partial L}{\partial x_{14}} \\ \frac{\partial L}{\partial x_{21}} & \frac{\partial L}{\partial x_{22}} & \frac{\partial L}{\partial x_{23}} & \frac{\partial L}{\partial x_{24}} \end{bmatrix}$$

To get $\frac{\partial L}{\partial Y}$ of size 2×2 to $\frac{\partial L}{\partial X}$ we need to perform convolution with a kernel size of 2×3 and zero padding of 2 (2-3+2*p+1 = 4). By observation, the kernels for 1st and 2nd row of $\frac{\partial L}{\partial X}$ are $\begin{bmatrix} w_3^{11} & w_2^{11} & w_1^{11} \\ w_3^{12} & w_2^{12} & w_1^{12} \end{bmatrix}$ and $\begin{bmatrix} w_1^{21} & w_2^{21} & w_3^{21} \\ w_1^{22} & w_2^{22} & w_3^{22} \end{bmatrix}$ respectively.

4. (45 points) **LeNet for Image Recognition:** In this coding assignment, you will need to complete the implementation of LeNet (LeCun Network) using PyTorch and apply the LeNet to the image recognition task on Cifar-10 (10-classes classification). You will need to install the python packages "tqdm" and "pytorch". Please read the installation guides of PyTorch here (https://pytorch.org/get-started/locally/). You are expected to implement your solution based on the given codes. The only file you need to modify is the "solution.py" file. You can test your solution by running the "main.py" file.

   (a) (25 points) Complete the class ***LeNet()***. In particular, define operations in function ***__init__()*** and use them in function ***forward()***. The input of ***forward()*** is an image. The paper for LeNet can be found here (http://yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf)

   The network architecture is shown in the figure below.



   The sub-sampling is implemented by using the max pooling. And the kernel size for all the convolutional layers are 5×5. Please use ***ReLU*** function to activate the outputs of convolutional layers and the first two fully-connected layers. The sequential layers are:

   Inputs $\rightarrow$
   Convolution (6 out channels) $\rightarrow$ Max Pooling $\rightarrow$
   Convolution (16 out channels) $\rightarrow$ Max Pooling $\rightarrow$
   Reshape to vector $\rightarrow$ Fully-connected (120 out units) $\rightarrow$
   Fully-connected (84 out units) $\rightarrow$ Outputs (n_classes out units)

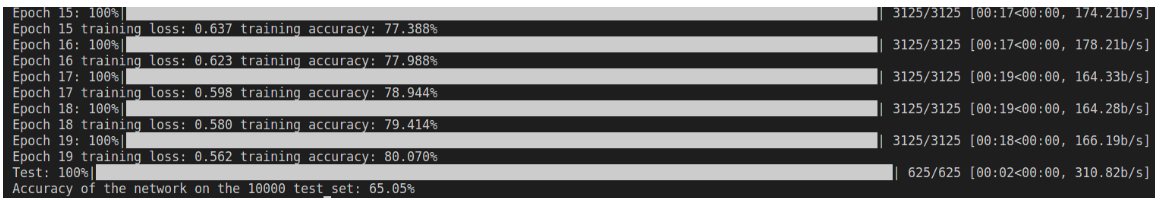
   For this part, you are only allowed to use the APIs in ***torch.nn***. Please refer to the PyTorch API documents below for the usage of those APIs before you use them: https://pytorch.org/docs/stable/nn.html.

```
Epoch 18: 100%|| 3125/3125 [00:16<00:00, 184.35b/s]
Epoch 18 training loss: 0.693 training accuracy: 75.458%
Epoch 19: 100%|| 3125/3125 [00:16<00:00, 189.80b/s]
Epoch 19 training loss: 0.671 training accuracy: 76.196%
Test: 100%|| 625/625 [00:02<00:00, 276.38b/s]
Accuracy of the network on the 10000 test set: 63.98%
```

Figure 1: Caption

Run the model by "***python main.py***" and report the testing performance as well as a short analysis of the results.

(b) (10 points) Add batch normalization operations after each max pooling layer. Run the model by "***python main.py***" and report the testing performance as well as a short analysis of the results.



```
Epoch 15: 100%|| 3125/3125 [00:17<00:00, 174.21b/s]
Epoch 15 training loss: 0.637 training accuracy: 77.388%
Epoch 16: 100%|| 3125/3125 [00:17<00:00, 178.21b/s]
Epoch 16 training loss: 0.623 training accuracy: 77.988%
Epoch 17: 100%|| 3125/3125 [00:19<00:00, 164.33b/s]
Epoch 17 training loss: 0.598 training accuracy: 78.944%
Epoch 18: 100%|| 3125/3125 [00:19<00:00, 164.28b/s]
Epoch 18 training loss: 0.580 training accuracy: 79.414%
Epoch 19: 100%|| 3125/3125 [00:18<00:00, 166.19b/s]
Epoch 19 training loss: 0.562 training accuracy: 80.070%
Test: 100%|| 625/625 [00:02<00:00, 310.82b/s]
Accuracy of the network on the 10000 test set: 65.05%
```

Figure 2: Caption

(c) (10 points) Based on (b), add dropout operations with drop rate of 0.3 after the first two fully-connected layers. Run the model by "***python main.py***" and report the testing performance as well as a short analysis of the results.

```
Epoch 15: 100%|| 3125/3125 [00:18<00:00, 165.30b/s]
Epoch 15 training loss: 0.965 training accuracy: 65.992%
Epoch 16: 100%|| 3125/3125 [00:19<00:00, 164.25b/s]
Epoch 16 training loss: 0.950 training accuracy: 66.842%
Epoch 17: 100%|| 3125/3125 [00:18<00:00, 173.30b/s]
Epoch 17 training loss: 0.938 training accuracy: 67.102%
Epoch 18: 100%|| 3125/3125 [00:20<00:00, 154.94b/s]
Epoch 18 training loss: 0.931 training accuracy: 67.298%
Epoch 19: 100%|| 3125/3125 [00:20<00:00, 156.25b/s]
Epoch 19 training loss: 0.917 training accuracy: 67.736%
Test: 100%|| 625/625 [00:01<00:00, 336.18b/s]
Accuracy of the network on the 10000 test set: 67.32%
```

Figure 3: Results