COM S 6730: Advanced Topics in Computational Models of Learning
Assignment #2

1. You need to submit a report and your code to Canvas. Your hard-copy report should include (1) answers to the non-programming part, and (2) analysis and results of the programming part. Please put all your code files and report into a compressed file named "HW#_FirstName_LastName.zip"
2. Unlimited number of submissions are allowed and the latest one will be timed and graded.
3. Please read and follow submission instructions. No exception will be made to accommodate incorrectly submitted files/reports.
4. All students are required to typeset their reports using latex.
5. Only write your code between the following lines. Do not modify other parts.

   ### YOUR CODE HERE

   ### END YOUR CODE

---

1. (60 points)(Coding Task) **Deep Residual Networks for CIFAR-10 Image Classification**: In this assignment, you will implement advanced convolutional neural networks on CIFAR-10 using *PyTorch*. In this classification task, models will take a 32×32 image with RGB channels as inputs and classify the image into one of ten pre-defined classes. The "code" folder provides the starting code. You must implement the model using the starting code. In this assignment, you must use a GPU.

   Requirements: Python 3.6, Pytorch 1.12.1, tqdm, numpy

   Required Reading Materials:

   [1] Deep Residual Learning for Image Recognition (`https://arxiv.org/abs/1512.03385`)

   [2] Identity Mappings in Deep Residual Networks (`https://arxiv.org/abs/1603.05027`)

   [3] Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift (`https://arxiv.org/abs/1502.03167`)

   (a) (30 points) Complete "network.py". Read the required materials carefully before this step. You are asked to implement two versions of ResNet: version 1 uses original residual blocks (Figure4(a) in [2]) and version 2 uses full pre-activation residual blocks (Figure4(e) in [2]). In particular, for version 2, implement the bottleneck blocks instead of standard residual blocks. In this step, only basic Pytorch APIs in **torch** and **torch.nn** are allowed to use.
   Answer. See Table 1 for the primarily results of each version.

   (b) (10 points) Add batch normalization operations after each convolution layer. Run the model by "***python main.py***" and report the testing performance as well as a short analysis of the results.
   Answer. See Table 2 for the results after adding batch normalization.

Version	**BN**	**Epochs**	**Batch Size**	**LR**	**Dropout**	**Train Acc.**	**Test Acc.**
BasicBlock	No	20	16	0.01	No	**84.21%**	**74.79%**
Bottleneck	No	20	16	0.01	No	72.28%	69.60%

Table 1: Training results for Part (a) without batch normalization or dropout.

Version	**BN**	**Epochs**	**Batch Size**	**LR**	**Dropout**	**Train Acc.**	**Test Acc.**
BasicBlock	Yes	20	16	0.01	No	**97.478%**	**80.94%**
Bottleneck	Yes	20	16	0.01	No	85.984%	80.32%

Table 2: Training results for Part (b) with batch normalization.

(c) (10 points) Based on (b), add dropout operations with drop rate of 0.3 after batch normalization layer. Run the model by "***python main.py***" and report the testing performance as well as a short analysis of the results.
Answer. See Table 3 for the results after adding dropout.

Version	**BN**	**Epochs**	**Batch Size**	**LR**	**Dropout**	**Train Acc.**	**Test Acc.**
BasicBlock	Yes	20	16	0.01	0.3	**92.64%**	**84.17%**
Bottleneck	Yes	20	16	0.01	0.3	79.13%	76.98%

Table 3: Training results for Part (c) with batch normalization and dropout.

(d) (10 points) Tune all the hyperparameters in "main.py" and report your final testing accuracy.
Answer. See Table 4 for the final tuned performance across versions and settings.
Both models were trained for 40, 60, and 150 epochs using batch normalization, a learning rate of 0.01, and a dropout rate of 0.1. The best-performing model during training and testing was Version 1, trained for **150** epochs, with a final training accuracy of **99.95%** and a testing accuracy of **85.30%**. Next, the best-performing hyperparameters for both models were re-evaluated with a higher dropout rate of **0.3**. In this case, Version 1, trained for **150** epochs with a learning rate of **0.01**, remained the best performer, achieving the same final training accuracy of **99.95%** and testing accuracy of **85.30%**. These results suggest that ResNet benefits significantly from deeper architectures. As the number of layers increases, the advantages of using a bottleneck structure become more apparent. Deeper models are more susceptible to issues like vanishing or exploding gradients, where bottleneck designs tend to outperform the regular version.
Additionally, the fact that the training and testing accuracies in the bottleneck version are relatively close indicates that this model tends to overfit less. In contrast, the basic model reaches a training accuracy as high as **99%**, while its testing accuracy hovers around **84–85%** suggesting that it suffers considerably from overfitting.
Lastly, it's important to note that a significant decrease in prediction error when using the bottleneck architecture is not necessarily expected. Based on [2], in ResNet-110, the error rate of the basic block is 6.61%, while that of the bottleneck block is 6.37%. However, an increase in accuracy with the bottleneck architecture is expected as the number of layers increases, since tuning hyperparameters has consistently resulted in a gradual decrease in error, as shown in Table Table 4.

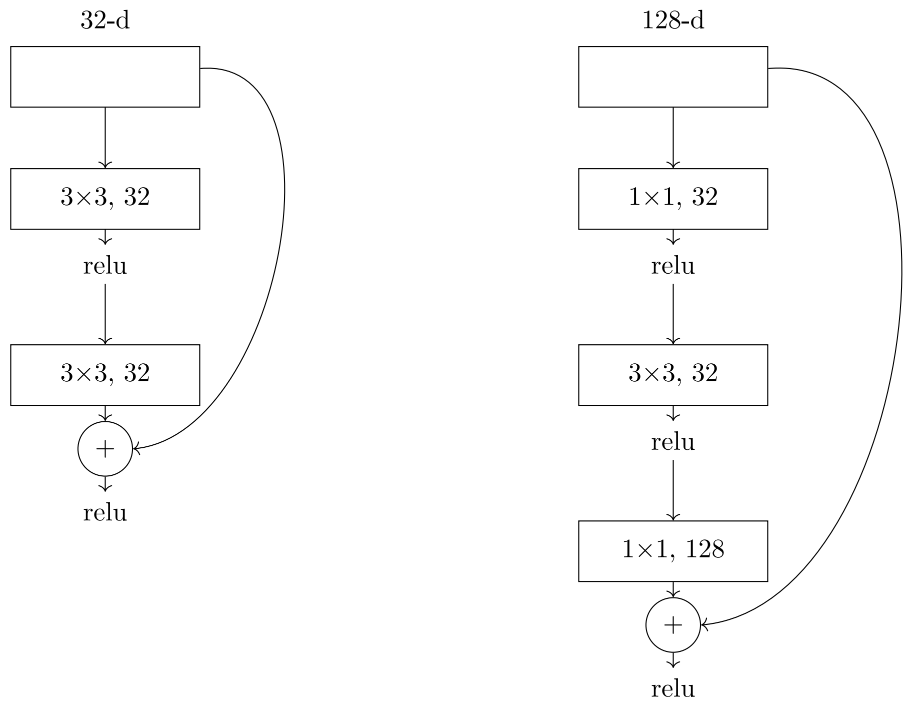
2. (20 points) Consider the standard residual block and the bottleneck block in the case where inputs and outputs have the same dimension (e.g. Figure 5 in [1]). In another word, the

Version	BN	Epochs	Batch Size	LR	Dropout	Train Acc.	Test Acc.
BasicBlock	Yes	40	32	0.01	0.1	99.126%	83.53%
BasicBlock	Yes	60	32	0.01	0.1	99.672%	84.21%
BasicBlock	Yes	150	32	0.01	0.1	99.950%	85.30%
BasicBlock	Yes	150	32	0.01	0.3	**99.792%**	**87.13%**
Bottleneck	Yes	40	32	0.01	0.1	89.260%	82.87%
Bottleneck	Yes	60	32	0.01	0.1	91.343%	82.49%
Bottleneck	Yes	150	32	0.01	0.1	95.244%	83.69%
Bottleneck	Yes	150	32	0.01	0.3	91.048%	84.60%

Table 4: Final results for Part (d) with tuned hyperparameters.

residual connection is an identity connection. For the standard residual block, compute the number of training parameters when the dimension of inputs and outputs is $128 \times 16 \times 16 \times 32$. Here, 128 is the batch size, 16×16 is the spatial size of feature maps, and 32 is the number of channels. For the bottleneck block, compute the number of training parameters when the dimension of inputs and outputs is $128 \times 16 \times 16 \times 128$. Compare the two results and explain the advantages and disadvantages of the bottleneck block.

Answer. First, the batch size—128 in both cases—does not directly affect the number of trainable parameters. However, using a larger batch size can make parameter updates less frequent and more stable. With that being said, the following network block correspond to the residual (left) and bottleneck (right):



Based on this, the number of trainable parameters, without considering batch normalization, for each block is as follows:

- **Residual Block:**

$$(32 \times 3 \times 3 \times 32) + (32 \times 3 \times 3 \times 32) = (9216) + (9216) = 18{,}432$$

- **Bottleneck Block:**

$$(128 \times 1 \times 1 \times 32) + (32 \times 3 \times 3 \times 32) + (32 \times 1 \times 1 \times 128) = (4096) + (9216) + (4096) = 17{,}408$$

If we're willing to consider batch normalization after each convolution layer, then for the standard block, each batch has 2 trainable parameters of γ and β, for each channel. Now, having 3 block of 32 channels, will add

- **Residual Block:**

$$(32 \times 3 \times 3 \times 32 + 2 \times 32) + (32 \times 3 \times 3 \times 32 + 2 \times 32) = (9280) + (9280) = 18{,}560$$

- **Bottleneck Block:**

$$\begin{aligned}&(128 \times 1 \times 1 \times 32 + 2 \times 32) + (32 \times 3 \times 3 \times 32 + 2 \times 32) + (32 \times 1 \times 1 \times 128 + 2 \times 128)\\ &= (4160) + (9280) + (4352)\\ &= 17{,}792\end{aligned}$$

As can be seen, the bottleneck block has fewer trainable parameters, since it's designed to reduce the number of parameters and computations by using 1×1 convolutions to compress and then expand feature maps. One other advantage of having fewer trainable parameters is that the model will be less prone to overfitting with small datasets. This can also be seen in question one, and by comparing the training and testing accuracy. On the downside, bottleneck has a lower capacity to model complex functions, and the compression may lead to information loss, potentially hindering performance on tasks that require fine-grained feature representations.

3. (20 points) Using batch normalization in training requires computing the mean and variance of a tensor.
Answer.

   (a) (8 points) Suppose the tensor $\boldsymbol{x}$ is the output of a fully-connected layer and we want to perform batch normalization on it. The training batch size is N and the fully-connected layer has C output nodes. Therefore, the shape of $\boldsymbol{x}$ is $N \times C$. What is the shape of the mean and variance computed in batch normalization, respectively?
   Answer.
   For each batch $\mathcal{B}$, we have:

$$\mathcal{B} = \{x_{1,2,\cdots,N}\}$$

   Therefor, for each batch $\mathcal{B}$, the corresponding mean and variance $\mu_{\mathcal{B}}$ and $\sigma_{\mathcal{B}}$ are:

$$\mu_{\mathcal{B}} = \frac{1}{N}\sum_{i=1}^{N} x_i \quad \text{and} \quad \sigma_{\mathcal{B}}^2 = \frac{1}{N}\sum_{i=1}^{N}(x_i - \mu_{\mathcal{B}})^2$$

   which are $\in \mathbb{R}$. Stacking them together for all the C batches results in the vector corresponding to $\mu_{\boldsymbol{x}}$, $\sigma_{\boldsymbol{x}} \in \mathbb{R}^{1\times C}$.

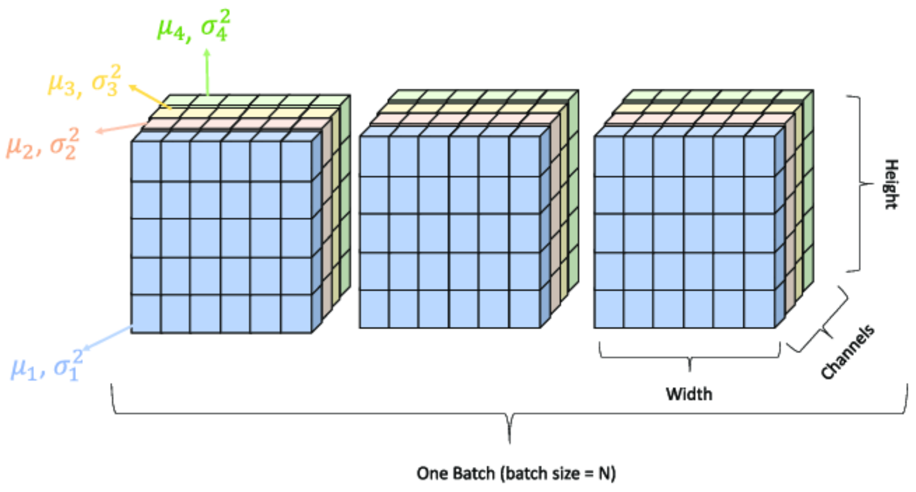


Figure 1: Visualization of batch normalization on a feature map. The mean and variance of the values of the pixels of the same colors corresponding to the channels are computed and are used to normalize these pixels.

(b) (12 points) Now suppose the tensor $\boldsymbol{x}$ is the output of a 2D convolution and has shape $N \times H \times W \times C$. What is the shape of the mean and variance computed in batch normalization, respectively?

Answer. The picture 1 derived from [1] demostrates this process farily well; For each color channel, the number of pixels across the dataset is $N \times H \times W$, where N is the batch size, and H and W are the height and width of the feature maps. The mean and variance are computed over all pixel values belonging to the same color channel. That is, for a single channel $\mathcal{C}$, we define the set of all its pixel values as:

$$\lambda = \{x_{1,1,1}, \cdots, x_{N \times H \times W}\}$$

Then, the mean and variance for channel $\mathcal{C}$, denoted $\mu_{\mathcal{C}}$ and $\sigma^2_{\mathcal{C}}$, are computed as:

$$\mu_{\mathcal{C}} = \frac{1}{N \times H \times W} \sum_{i=1}^{N} \sum_{j=1}^{H} \sum_{k=1}^{W} x_{i,j,k} \quad \text{and} \quad \sigma^2_{\mathcal{C}} = \frac{1}{N \times H \times W} \sum_{i=1}^{N} \sum_{j=1}^{H} \sum_{k=1}^{W} (x_{i,j,k} - \mu_{\mathcal{C}})^2$$

Each $\mu_{\mathcal{C}}$ and $\sigma^2_{\mathcal{C}}$ is a scalar value in $\mathbb{R}^{1\times1\times1}$. Stacking these values for all C channels results in the vectors $\mu, \sigma \in \mathbb{R}^{1\times1\times1\times C}$.

References

[1] Kevin Bui, Fredrick Park, Shuai Zhang, Yingyong Qi, and Jack Xin. Improving network slimming with nonconvex regularization. *IEEE Access*, 9:115292–115314, 2021.

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part IV 14*, pages 630–645. Springer, 2016.