# Introduction to Computer Vision (Spring 2022)
# Assignment 2

Release date: April 15, due date: April 30, 2022 11:59 PM

The assignment includes 3 tasks: implementing batch normalization, camera calibration and training a simple CNN on CIFAR-10, which sums up to 100 points and will be counted as 10 points towards your final score for this course. This assignment is fully covered by the course material from Lecture 4 to 7.

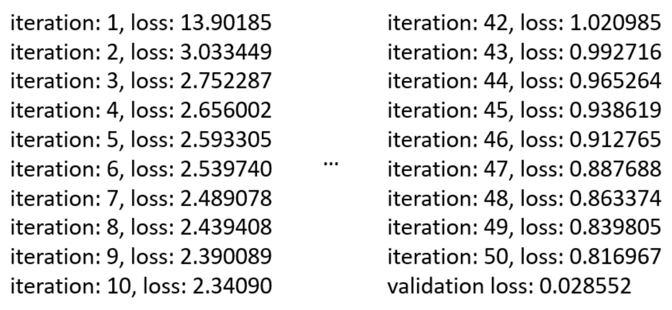
The objective of this assignment is to get you familiar with some useful algorithms of classic computer vision and deep learning, as well as to establish the concept of how to train and tune a network. We offer starting code for all the tasks and you are expected to implement the key functions of each task. Please read *README.md* before you start.

After completing the tasks, please compress your code along with your results to **Name_ID.zip** following the original path structure, then submit it to course.pku.edu.cn. Feel free to post in the discussion panel for any questions and we encourage everyone to report the potential improvements of this assignment with a bonus of up to 5 points.

1. **Batch Normalization (30 points):**

   Batch normalization is one of the most common modules used in deep learning because it can greatly reduce the difficulty of optimizing a large model. In this section, you are required to implement batch normalization and evaluate the performance by integrating it with a neural network (the same toy MLP as in Assignment 1). Please complete the *bn.py* and you should expect the loss to decrease as Figure 1 using the random seed we offered.

   Note that you are only allowed to use Numpy in this section. *For loop* commands and np.vectorize are not allowed.



```
iteration: 1, loss: 13.90185             iteration: 42, loss: 1.020985
iteration: 2, loss: 3.033449             iteration: 43, loss: 0.992716
iteration: 3, loss: 2.752287             iteration: 44, loss: 0.965264
iteration: 4, loss: 2.656002             iteration: 45, loss: 0.938619
iteration: 5, loss: 2.593305             iteration: 46, loss: 0.912765
iteration: 6, loss: 2.539740      ...    iteration: 47, loss: 0.887688
iteration: 7, loss: 2.489078             iteration: 48, loss: 0.863374
iteration: 8, loss: 2.439408             iteration: 49, loss: 0.839805
iteration: 9, loss: 2.390089             iteration: 50, loss: 0.816967
iteration: 10, loss: 2.34090             validation loss: 0.028552
```

Figure 1: A result of our BN implemetntion.

**a)[10 points] Implement the Forward Pass for Training**

First, you are required to implement the forward pass of batch normalization. Specifically, given an input $\mathbf{x} \in \mathbb{R}^{B \times C}$, in which $B$ refers to the batch size and $C$ refers to the number of channels, your network should output the correct $\mathbf{y}$ and update the running mean of $\mu \in \mathbb{R}^C$ and $\sigma \in \mu \in \mathbb{R}^C$ for the purpose of inference.

**b)[15 points] Implement the Backward Pass for Training**

Afterwards, you are required to implement the backward pass of batch normalization. With the saved variables from forwarding, you are required to compute the gradient $\frac{\partial \mathcal{L}}{\partial \mathbf{x}}$, $\frac{\partial \mathcal{L}}{\partial \gamma}$ and $\frac{\partial \mathcal{L}}{\partial \beta}$.

**c)[5 points] Implement the Forward Pass for Inference**

An important characteristic of batch normalization is the inconsistency between training and inference. In this question, you are expected to perform inference of the given MLP with batch normalization.

2. **Train a CNN on CIFAR-10 (40 points):**

This question is designed to give you a taste of how to train and tune neural networks. Here is a pipeline you will experience:

1. Establish a baseline and make sure the hyperparameters are chosen appropriately.
2. Identify the problems, *e.g.* overfitting, in this baseline via visualizing and analyzing the curves.
3. To tackle the problems, either leverage the techniques introduced in the lectures or propose and implement your tricks.
4. Go back to Step 2 until the performance of your model meets your expectations. Sometimes you may need to tune the hyperparameters again.

Specifically, you are required to implement and train a convolution neural network for classification on a small dataset CIFAR-10. Don't worry about not having a GPU since this dataset is small enough to train on CPUs (we can achieve a performance of $> 75\%$ on the validation set within half an hour under the *TOP 1 accuracy* metric with CPUs). You can also use Colab from Google to access free GPUs, as long as you are willing to spend extra time for setup. We provide the starting code and you can follow the steps below for doing the experiment:

**a)[10 points] Implement a CNN**

A convolution neural network can extract features from the input images $\mathbf{I} \in \mathbb{R}^{B \times H \times W \times 3}$ and predict the probability distribution among the classes $\mathbf{Y} \in \mathbb{R}^{B \times N}$, in which $B$ refers to the batch size and $N$ refers to the number of classes ($N = 10$ for CIFAR-10).

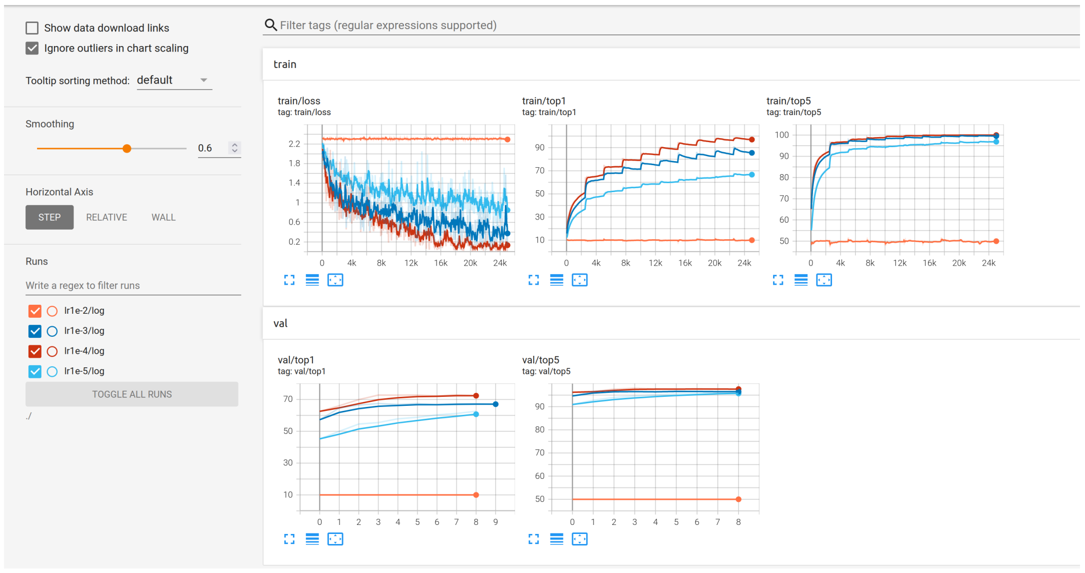
For the first step, you are required to build a CNN in *network.py*. Once you have finished this step, you can run the *train.py* and see the results on the training set and the validation set.

Note that we recommend you leverage the basic modules/layers in PyTorch, *e.g.* torch.nn.Conv2d. There is no requirement about the specific structure of your CNN, but we provide an exemplar architecture in Figure 6 for your information (feel free to ignore it). Note that, you are not allowed to directly use the complete/off-the-shelf CNN modules, *e.g.* torchvision.models.resnet50.

### b)[10 points] Adjust Learning Rate and Visualize Curves

Appropriate hyperparameters can improve the performance of your model. To investigate hyperparameter tuning, we can draw the curves of the loss function and evaluation metrics to get a feeling about the training progress.

In this question, you are required to draw the **loss curve** and **accuracy curve** on the **training** set and **validation** set under different learning rates (1e-3, 1e-4, 1e-5) during the training process. You should submit a screenshot of your curves as Figure 2. As long as the performance is reasonable, you will be fine.



☐ Show data download links
☑ Ignore outliers in chart scaling
Tooltip sorting method: default
Smoothing 0.6
Horizontal Axis: STEP RELATIVE WALL
Runs
Write a regex to filter runs
☑ lr1e-2/log
☑ lr1e-3/log
☑ lr1e-4/log
☑ lr1e-5/log
TOGGLE ALL RUNS
./

Figure 2: Screenshot of the curves under different learning rates. To save your time, you don't have to run learning rate=1e-2.

### c)[10 points] Data Augmentation and Visualization

In Figure 2, we can find that a large gap between the training accuracy and the validation accuracy, which indicates that the model is suffering from overfitting. This could be a very common phenomena for a small dataset. To alleviate this problem, we can leverage data augmentation.

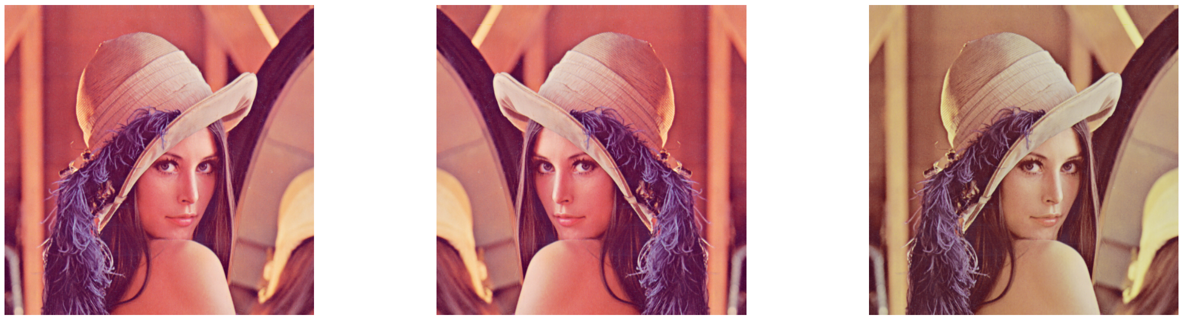


Figure 3: Left: Without augmentation. Middle: Position augmentation. Right: Color augmentation.

In this question, you are required to implement at least two different kinds of data augmentations in *data.py* and save the augmented images for visualization as Figure 3. Since the resolution of the images from CIFAR-10 is too slow, you can visualize *Lenna.png* to check your code and submit the augmented images of it. After checking the correctness of your augmentation by visualization, run the *train.py* again to see if the gap between training accuracy and validation accuracy becomes smaller.

Note that, the point of this question is to investigate the effect of the data augmentation. Therefore you are allowed to use any existing publicly available libraries, even in one line if you can find any.

### d)[10 points] Further Improve your Network

In c), you have experienced how to identify and resolve the overfitting problem during training neural networks. Furthermore, sometimes we need the model to achieve certain performance bars. In this question, you are free to improve your own model until the performance meets the bar and your expectation. You should submit a screenshot as Figure 4 to show your achievement.

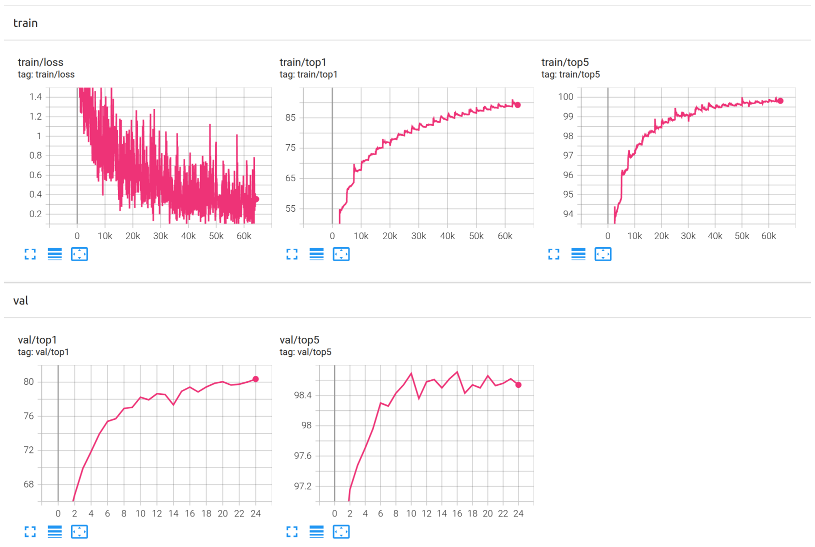


Figure 4: Best performance of my model.

We will grade this question according to the criteria in Table 1. Though without accessing GPUs, this shouldn't cost you too many trials (the TA reaches 75% after adding data augmentation without any other tricks) if you are tackling the correct spots.

| TOP1 accuracy(%) ↑ | Score |
|---|---|
| $\geq$ 75% | 10 |
| 70% $\sim$ 75% | 8 |
| 60% $\sim$ 70% | 6 |
| 50% $\sim$ 60% | 4 |
| 40% $\sim$ 50% | 2 |

Table 1: Grading policy.

3. **Camera Calibration (30 points):**

Given a set of corresponding pairs of 3D coordinates in world coordinate space and 2D coordinates in image coordinate space, we can solve the intrinsic $\mathbf{K}$ and extrinsic $[\mathbf{R}, \mathbf{T}]$ of a *perspective* camera. In this question, you are required to implement this process of camera calibration.

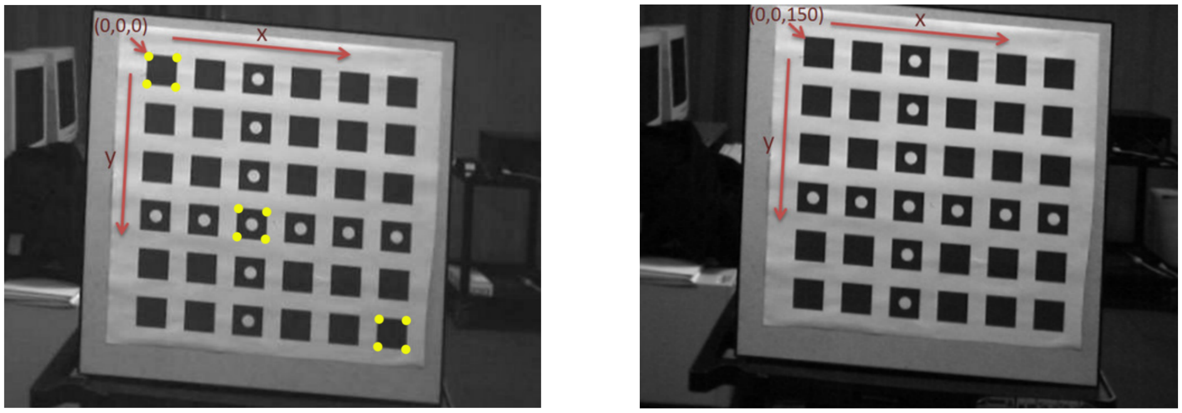


Figure 5: Left: Image of calibration grid at Z=0. Right: Image of calibration grid at Z=150.

In practice, we often utilize a calibration grid to get correspondences. The left image of Figure 5 shows a calibration grid in the rest/zero position. This calibration grid has 6 $\times$ 6 squares and each square is $50mm \times 50mm$. The separation between adjacent squares is $30mm$, so the entire grid is $450mm \times 450mm$. For the right image in Figure 5, the calibration grid was moved back along its normal from the rest position by $150mm$. The process of calibrating a camera is shown below:

First, to get the 3D coordinate of each corner of the squares, we define the top left corner of the calibration grid in the rest/zero position as the origin of world coordinate space. The X-axis runs left to right parallel to the rows of squares and the Y-axis runs top to bottom parallel to the columns of squares, as shown in Figure 5. The Z-axis is then perpendicular to the calibration grid and points backward to the plane. Now, we can obtain the 3D coordinates of the corners on both the two images, *e.g.* the bottom right corner of the right image is at (450, 450, 150).

Second, we need the 2D coordinates of each corner of the squares, which could be obtained by the corner detector *e.g.* Harris Corner detector in Assignment 1. Here, you don't have to bother implementing all the tedious details since we have already computed some of the 2D coordinates with corresponding 3D coordinates of corners.

Finally, with the corresponding 3D coordinates and 2D coordinates pairs, we can solve the intrinsic and extrinsic parameters of the camera by the techniques introduced in class.

**a)[5 points] Compute the Corresponding 3D coordinates**

In this question, you have 12 pre-computed 2D coordinates of corresponding pairs for each image, which are indicated by the yellow points in the left image in Figure 5. Another 12 pre-computed 2D coordinates of the right image correspond to the same grid corners as in the left image. And you are required to calculate all 24 3D coordinates of them.

**b)[15 points] Construct the Equation $\mathbf{Pm} = 0$ and Solve $\mathbf{m}$**

For the 2D coordinates $p_i = [u_i, v_i]^\top$ $(0 \leq i \leq 24)$ and their corresponding 3D coordinates $P_i = [x_i, y_i, z_i]^\top$, we have:

$$\lambda \begin{bmatrix} p_i \\ 1 \end{bmatrix} = \mathbf{K}[\mathbf{R}\ \mathbf{T}]P_i. \tag{1}$$

And we can further define the detailed transform Equation 1 as

$$\begin{bmatrix} u_i \times w_i \\ v_i \times w_i \\ w_i \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ z_i \\ 1 \end{bmatrix} \tag{2}$$

You are required to follow Equation 2 to construct a homogeneous linear system $\mathbf{Pm} = \mathbf{0}$ and solve $\mathbf{m}$, in which $\mathbf{P}$ is a 48x12 matrix and $\mathbf{m}$ is a 12x1 matrix. Please refer to the slides of Lecture 7 for the details.

**c)[10 points] Solve K and [R T] from m**

Please follow the instruction on the slides to solve **K** and [**R T**] from **m**. The results will be saved in *result.npy* for submission.

## Appendix

1. Some helpful libraries for data augmentation.
   - OpenCV-Python, https://docs.opencv.org/4.x/d6/d00/tutorial_py_root.html
   - Pillow(PIL), https://pillow.readthedocs.io/en/stable/
   - torchvision.transforms, https://pytorch.org/vision/0.9/transforms.html
   - ...
2. Draw curves by TensorboardX https://pytorch.org/docs/stable/tensorboard.html
3. Colab tutorial https://colab.research.google.com/

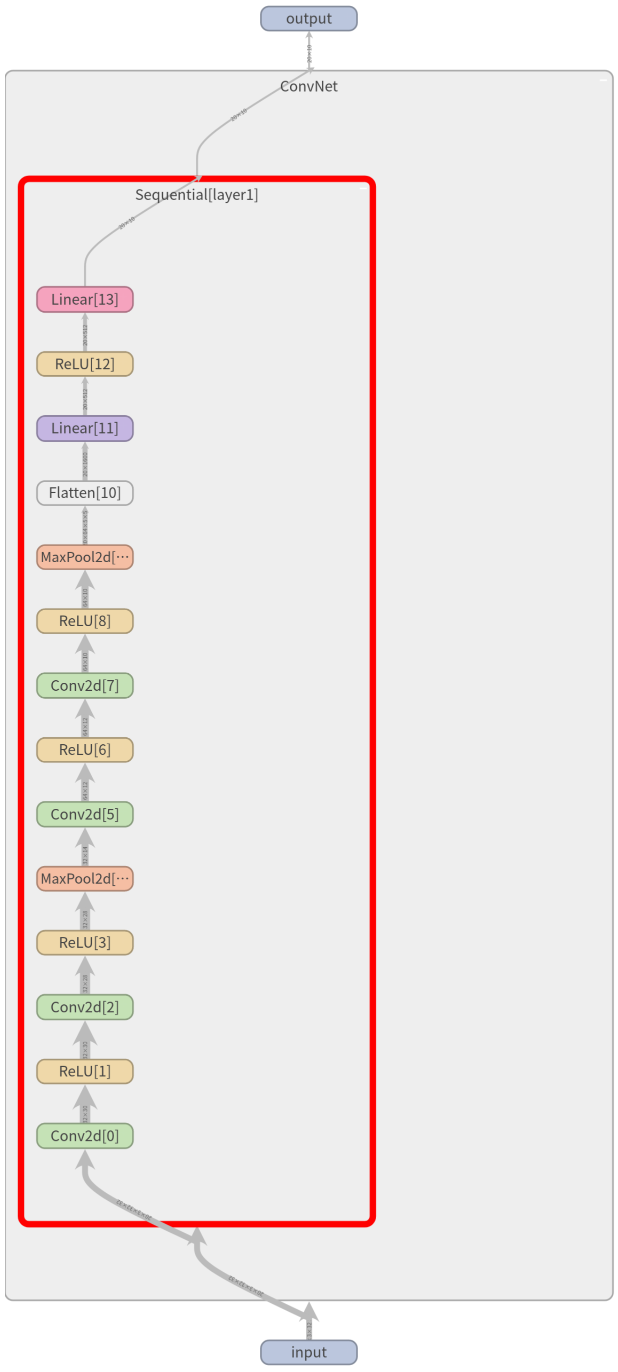


Figure 6: An example of a CNN.