ELEC5307 Deep Learning
Project #1: Parameters in Neural Networks

1 Objectives

This laboratory aims to introduce the basic techniques in deep neural networks. In this laboratory you will:

- Learn to use PyTorch to load images and train a neural network for classification.
- Understand the functions of convolutional layers, pooling layers, fully connected layers and softmax layer, etc.
- Become familiar with the activation methods, pooling and initialization methods.
- Learn to select proper hyperparameters for better performance.
- Visualize your results and the objective to learn how different parameters contribute to the final performance.

2 Instructions

2.1 Data description: CIFAR-10

You need to use the CIFAR-10 image dataset. The CIFAR-10 dataset consists of 60000 images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images, which are split by the publisher. The details and the downloads of the dataset is in `https://www.cs.toronto.edu/~kriz/cifar.html`

The classes include 'airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', and 'truck'. All the images are manually labelled, and each image only contains one label. The images in CIFAR-10 are of size 3x32x32, i.e. 3-channel colour images of 32x32 pixels in size. In some applications, each image has already been reshaped into a vector with dimension of 3072 ($= 3 \times 32 \times 32$).

In PyTorch, you can use the function **`torchvision.datasets.CIFAR10`** to automatically download and read the dataset and the function **`torch.utils.data.DataLoader`** to load the data (training and test) into your network.

2.2 Hyperparameters

Hyperparameters are crucial to your success in training a neural network. The weights of neural networks will be modified during learning, and the hyperparameters will contribute to the modification of input images, the number of weights, and the way to update the weights. Here are some groups of hyperparameters. For more hyperparameters and their usage in PyTorch, please refer to the official documentation website (`https://pytorch.org/docs/stable/index.html`) and PyTorch Forum (`https://discuss.pytorch.org`).

2.2.1 Transformation

This will influence the input images. The basic idea to conduct transformation is to add data (called **data augmentation** in deep learning). The neural network contains many weights to modify, which requires many images. However, the provided data are always not enough. By making transformations, one image could be fed into the networks using different patches or sizes, which could increase the size of training set.

These operations are included in **`torchvision.transforms`**. Some of the options are:

- **resize**: The images can be resized to square image or larger/smaller. Almost all the transformation will need the resize operation.
- **crop**: You can crop the center (**`CenterCrop`**), or center plus the four corners (**`FiveCrop`**), or crop at a random location((**`RandomCrop`**)).
- **flip**: The images may occur from different view, so flip horizontally (**`RandomHorizontalFlip`**) or vertically (**`RandomVerticalFlip`**) could provide more possibilities of the images.
- **affine**: This will modify the image by rotation or translation. It is not suitable to predefine some affine function because of the diversity of images, so torchvision only contains (**`RandomAffine`**).
- **normalization** The pixel values of the images are normalized to [0, 1] using this operation. You need to use the global mean and standard deviation of the whole dataset if you train the model from scratch. If you would want to fine-tune a model from some model pretrained on ImageNet, you need to use the values from ImageNet.

Please note that the normalization operation is not for data augmentation but for faster convergence. The rest four operations mentioned above are used for data augmentation.

2.2.2 Network Structure

These parameters will influence the structure of the neural networks. The most important indicator is the capacity of the network, which is roughly equal to the number of parameters. In that case, the deeper and wider the network is, the better potential performance it could provide. However, you need to make sure the parameters are constrained carefully.

The operations are included in (**`torch.nn`**). Some of the options are:

- **Depth**: Roughly speaking, the deeper network will have the potential to provide better results. However, when the network becomes deeper, it will be more difficult to train as the gradients are more possible to vanish or explode.
- **Activation function**: The most commonly used are ReLU, Tanh and Sigmoid. These functions provide non-linearity to the neural network.
- **Pooling method**: The common choices are max pooling (**`MaxPool2d`**) and average pooling (**`AvgPool2d`**). The size and stride of the pooling layers will change the sizes of feature maps (i.e. width and height).
- **Channel size**: The input channel is 3 (for three colour channels R/G/B), and the output channel should be the number of classes (10 for CIFAR-10). From shallow layers to deep layers, the channel number always gradually increase (with the width and height numbers decreasing). The larger the channel number, the more time you will need for both forward and backward process. In PyTorch, fully connected layers can be basically defined as **`Linear(in_channel, out_channel)`**, 2-D convolutional layers can be basically defined as **`Conv2d(in_channel, out_channel, kernel)`**.

- **Convolutional parameters**: The kernel size, zero padding number and stride will influence your output width and height by $W_{output} = (W_{input} - Kernel + 2 \times Padding)/Stride + 1$. You can try to use kernel size as 3×3, 5×5 and 7×7. By default, we do not want the convolutional layers change the width/height of the feature maps, so you can select stride and zero padding accordingly. For example, if you have kernel size 5×5, we always need the zero padding as 2 and stride as 1.
- **Dropout**: This layer (`Dropout`) is a method of regularization, which will randomly set zeros to some weights in the according layer.

2.2.3 Training Process

The training process is affected by how the data are fed into the network and how the weights are initialized and updated. The data are fed into the network using `torch.utils.data.DataLoader`. Some of the options are:

- **shuffle**: By default, we usually shuffle the input data in the training and validation part and do not shuffle the input data in test part.
- **batch size**: The larger batch size can often help you get better results, but it is limited by your memory size (computer memory or GPU memory). There are also some exceptions that larger batch sizes will make the performance worse, so you need to be careful in selecting this value.

The weights can be initialized from pretrained model or by using `torch.nn.init`, where the options includes Xavier, Nomal, Uniform, and Constant, etc.

The weights will be updated in backward process according to the objective function, optimization function and learning rate. The objective functions are treated as special layers in PyTorch in `torch.nn`, and the optimization operations can be found in `torch.optim`, and some options are listed as below:

- **Epochs**: One epoch means a period that the network has been trained by seeing every training image. After several epochs, your training loss and validation accuracy will not change much. You need to set a good number of epoch in order to get the best accuracy and avoid overfitting.
- **Objective function**: The cross-entropy loss (`torch.nn.CrossEntropyLoss`) is always used in the classification problems. The soft-margin loss (`torch.nn.SoftMarginLoss`) and least square error (`torch.nn.MSELoss`) are also included mainly for binary classification problems.
- **Update methods**: The commonly used methods are Adam(`torch.optim.Adam`) and SGD(`torch.optim.SGD`). The parameters that are to be determined include the base learning rate(`lr`), momentum rate((`momentum`)) and regularizer weight((`weight_decay`)). You can also set different learning rate on different layers.
- **Base learning rate**: Larger learning rate (around 0.1) will change the weights dramatically, but will be useful when you train the model from scratch. Smaller learning rate (0.01 0.0001) will be useful in fine-tuning from the pretrained models.
- **Learning rate scheduler**: The learning rate need to be cut down as the training is proceeding. The commonly used are step, multiple-step and exponential. You can write the update policy by yourself or use the methods in `torch.optim.lr_scheduler`.

2.3 Result analysis

In order to train a deep learning model successfully, the results of the network should be carefully analyzed. The best way to analyze results is to visualize the values.

- The loss curve for both training and validation. Ideally, both losses should decrease and converge after several epochs. If the training loss is still going down but the validation loss is increasing, then the model is overfitting. If the losses are still going down when you finish, then the model is underfitting. You should try to avoid both situations in a reasonable number of epochs.
- The accuracy changes in the training set and validation set. The values should be increasing as the epoch is increasing. The pattern should be similar to the changes of losses but in an opposite direction.

3 Experiments

In the experiments, your job is to build a neural network for the classification in CIFAR10 and analyze the results generated from different hyperparameters. Your task includes three parts.

- The first part is about running a baseline model. You need to run a baseline model and provide your visualization results.
- The second part is about finding suitable parameters from the given options. You will be given several options of **batch size**, **base learning rate** and **number of epochs**, try to find the optimal combination of all these three hyperparameters to get the best performance.
- The third part is about other options. You need to firstly train a new baseline model and then according to Appendix: Tasks for Part3, analyze the effect of each hyperparameters to your result.

The detailed descriptions are as follows.

3.1 Part1: Baseline structure

You need to go through the '**baseline.ipynb**' file first and run the baseline code. This file is modified based on the official tutorial from PyTorch: `https://pytorch.org/tutorials/_downloads/cifar10_tutorial.ipynb`. However, please note that their settings are far from ideal.

For this reference and other references, if you used their codes, please point out in comments and in the end of your submission in the 'Reference' part. You will be punished if you use all others' codes without changing anything by yourself. You are also be punished if you used others' codes but did not indicate in your submission.

You need to split your dataset into three parts: training, validation and test. The test dataset is ready by default, and you need to separate several images (always smaller than the number of test set) as validation set in your training. The validation set will help you avoid overfitting problems.

The average time for running one epoch of such network is around 1-5 minutes depending on the type of your cpu/gpu. The speed is low because the dataset is quite large. In that case, you can try to randomly select a bit data from the original dataset to check the performance. This technique is very useful when you face with large datasets so that you can quickly see the results instead of waiting for a long time.

3.2 Part2: Select hyperparamaters

To successfully train this network, you need to select the proper **batch size**, **base learning rate** and **number of epochs**. The options are as below:

- batch size: 2, 4, 8
- base learning rate: 0.005, 0.001, 0.0005, 0.0001, 0.00005, 0.00001
- number of epochs: 1, 2, 4, 8, 16

Your job is to select the hyperparameters that will help to train the network to get the best performance in the test set. Meanwhile, the training time should be as short as possible, which means you should not leave the network training for a super long time even it has converged judging from the loss curve.

You are supposed to run the codes for several times, plot the corresponding loss curves for training and validation, compute the accuracy for validation and test, and finally make a decision. Please do **NOT** change the other hyperparameters in the given network for this part. In the writing part, please provide your output images and your analysis. Your analysis should include but are not limited to:

- What are the choices that could be empirically ignored without doing any experiments? Were you correct after you conduct the experiments?
- How many epochs are passed when the network is converged?
- Why too large or too small learning rates are not good choices?
- What are the specification of the computer you are using? E.g. the cpu/gpu type and the corresponding memory.
- How long do you run an epoch? Did you use samples of the original dataset to speed up your progress, and how did it work?
- Are there any overfitting problems?

3.3 Part3: Other hyperparameters

For now you are supposed to have the ability to train a neural network. The rest of the experiment is to build a brand new neural network as a baseline and play with some other hyperparameters.

Although you have learned the structures of different predefined neural networks, you are not allowed to use them in Project 1. Instead, you should only build a network that contains:

- 3 convolutional layers, with the activation functions and pooling layers after each convolutional layers.
- 3 fully connected layers right after the last pooling layer. Please remember to make changes of the data shapes (using function `view`) to make the feature maps flow into the fully connected layer smoothly.
- 1 output layer, which is also a fully connected layer, but the output channel should be 10 (the number of classes).

For the channel sizes and convolutional parameters, you are free to select your own hyperparameters. However, since you are supposed to train the network using cpus, please do not

make layers with very large number of channels. The channel numbers for convolutional layer should be no larger than 256, and the channel numbers for fully connect layers should be no larger than 1024.

After you build your baseline model, you need to do the analysis on **THREE** kinds of hyperparameters. The hyperparameters you are going to play with are defined by your SID as indicated in Appendix: Tasks for Part3. For each of the three subtasks, you can play with all the parameters in the methods. For example, you can select any number for **`weight_decay`** and **`momentum`** if you are playing with **`SGD`**.

Please analyze the three tasks and select the best choice for your own network. You can also change the other hyperparameters in 2.2 to better suit each choice, e.g. batch size, learning rate, etc., but please remember to control variables while you make the analysis. Please note that your modifications of the hyperparameters based on your baseline network may or may not improve the accuracy. You need to figure out how the hyperparameters influence the results and explain why. The analysis should be included in the written part in the .ipynb file.

You need to write **ONE** single python file (not a .ipynb file) that includes your baseline network and your modified network with your trained files, and the output of this python file should be the accuracy of the test set based on your own models. Your accuracy on test dataset is also marked based on your baseline or modified network whichever is higher.

4 Submission and Grades

You are supposed to finish this project on your own. Your submission should include the Jupyter Notebook ('project1.ipynb') with your modification and written analyses, and the Python file ('project1.py') with your trained model (named 'baseline.pth' and 'modified.pth' respectively). The files should be in a .zip file named as '**project1_firstname_lastname_yourSID.zip**' with no spaces in the file name and submitted through *Canvas*. The .zip file can contain some of the images if necessary. Your codes need to be well commented and the written part in the notebook file need to have clear sections. The final grades are given based on the following criteria. For detailed marking scheme, please refer to Appendix: Marking Scheme.

- Your submissions should strictly follow the instructions.
- Your accuracy need to be no smaller than 70% either using your new baseline network or your modified network.
- Your codes are correct and well organized. Your codes are well commented and the references are clear.
- The written part are well organized and has few typos. The report should contain the correct formulas when they are necessary.
- You have covered all the three hyperparameters you are assigned. If you did the wrong task, the corresponding analysis will not be marked and you will also be punished.
- The visualization results are clear and well defined.
- You have shown your insights into the parameters and drawn some reasonable conclusions.
- No copy from your classmates. If you and some of your classmates copied codes from the same online resource, you will also be penalized if you do not make any modifications or provide the reference.
- Give references on all the codes and papers you referred to.

Appendix: Tasks for Part3

The task you are going to do depend on the **last three digits** from your SID (the 9-digit on your student card).

Digits	Transformation (3rd last digit)	Structure (2nd last digit)	Training (last digit)
0/1	RandomCrop+Resize	Channel number	lr_scheduler: Exponential
2/3	Flip+Resize	Activation method	lr_scheduler: Step
4/5	Affine+Resize	add Dropout layers	lr_scheduler: MultiStep
6/7	CenterCrop/FiveCrop+Resize	Kernel size	lr_scheduler: Lambda
8/9	Normalize+Resize	Pooling method	optimizer: SGD settings

For example, if your SID is 470XXX**364**, your choice should be **Flip+Resize**, **Kernel size** and **lr_scheduler: MultiStep**.

In your report, you should mainly consider the choices you are assigned. You can also try other settings if you think they are more suitable, but you need to control variables in your analysis.

You may be assigned similar tasks to your classmates. However, each task will contain many choices, and each choices will contain some parameters, so your choices and process cannot be similar unless you copy others' work. In that case, cheating could be found easily and will be punished.

Appendix: Marking Scheme

The total marks for Project 1 is 20 in your final score, and part 1, part 2 and part 3 accounts for 19%, 21% and 40% respectively, the formating and submission accounts for 20%. The numbers in the following chart are in percentages.

part 1	successfully run the given .ipynb	4
	correctly split the validation dataset	5
	correct draw the loss and accuracy curves	10
part 2	correct choices (2 points each)	6
	good analysis (5 points each)	15
part 3	correct new baseline and modified net	9
	good analysis of each hyperparameter	7 ($\times$3)
	good accuracy on Cifar-10 test set($>$ 70%)	10
format	correct submission	10
	no writing typos	2
	well commented codes	5
	good references	3
punishment	each wrong task in part 3	-10
	cheating	-100
	late submission per day	-15
	bad coding	-20

Please note that the marking scheme is to be updated in details.