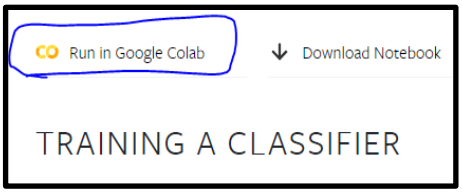
ECE 651-Fall 2024: Homework Assignment #3

Deadline: Saturday October 19, 2024

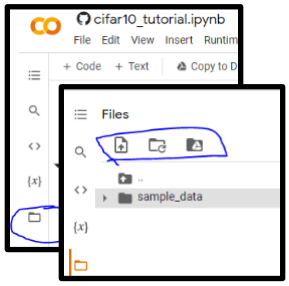
Overview:
In this assignment, you will experiment with Convolutional Neural Networks (CNNs) using the CIFAR-10 dataset to explore the concepts of underfitting, overfitting, and appropriate model complexity. For this goal, you will need to be able to compute the number of parameters to be trained in a CNN model (See examples in Lecture note #4). You will also compare different CNN architectures and summarize key design ideas.

Dataset: You will be using the CIFAR-10 dataset, which consists of 60,000 32x32 color images in 10 classes, with 6,000 images per class.

Tools: Use **PyTorch** to implement and train the CNN models. You may use pre-existing libraries to help with training and analysis. A **tutorial** and the source code (AlexNet.py) for a vanilla version of AlexNet are provided for your reference.

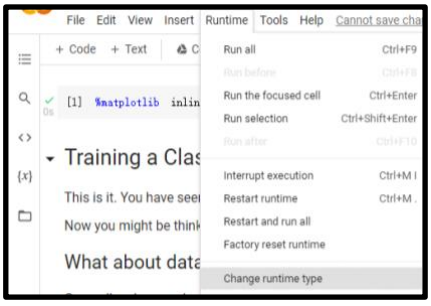


Google Colab Option:
For those who don't have access to GPU-empowered devices or suffering from software problems**,** Google Colab can be a good alternative (*you may skip if you have already set up a powerful deep learning environment locally*). You can learn about Colab by clicking the *open in colab* button on top of this tutorial **link**. The operation is similar to running a notebook on your computer. Please make sure to manually save this notebook version, either to your drive or to a local disk. Some major differences between Colab and your computer are discussed as follows.



When opening a notebook in Colab (either on a website like this or in your Google Drive), you're connected to a *virtual machine runtime* (VM), which means that you need to have your data and source code on the remote disk of the VM (you can conduct basic operations on the file system of the VM by clicking the folder button on the left side of the Colab notebook window).

The tutorial **link** includes the code for downloading PyTorch's official version of CIFAR-10 dataset. It can automatically download CIFAR-10 data to the VM disk and create PyTorch dataset variables accordingly. Mounting your notebook VM to your Google drive to access data is theoretically feasible, but on many occassions, it is very slow and thus not recommended here.



Free hardware accelerators, e.g., **GPUs**, are highly recommended if you use Colab. In fact, this is the reason why we recommend Colab instead of running deep learning programs on your CPU. To enable this, you need to go to runtime --> change runtime type --> Hardware Accelerator and choose GPU. You may need to run your code blocks again after changing the runtime type. Make sure to put your data batch and CNN variables to GPU; otherwise it will report an error when feeding data (as shown in the Training on GPU section of the tutorial). When you use GPUs, you should set the num_workers of your dataloader larger than zero to gain higher efficiency. If you choose batchsize = 128, it should take about 30 seconds to finish one training epoch of vanilla AlexNet on CIFAR-10.

Problem 1. Benchmarking AlexNet and Exploring Underfitting/Overfitting

1.1. AlexNet Benchmarking

Train an **AlexNet** model on CIFAR-10 with appropriately chosen hyperparameters. Your goal is to achieve **at least 75% test accuracy**. The source code defining a vanilla AlexNet is provided in the file *AlexNet.py* for your reference. To try it out, you can either insert this CNN model into your notebook or import it after saving this file online (in the runtime VM or in your drive). You can use it directly if it meets the accuracy requirements. Adjust the model structure/hyperparameters as deemed necessary.

Submit a report (or your notebook with all the required items) summarizing your design and implementation results. Required items include:

a) **Model:** Specify your finalized model after cross validation. That is, **specify all the hyperparameters of your model** including the number of layers, the number of neurons per layer, activation functions used in each layer, filter size, the number of filters per layer, pooling, stride, padding, etc. Calculate **the number of model parameters** of your design.

b) **Training:** Decide and document the training hyperparameters and training method, such as and the learning rate or a suitable learning rate scheduling method (either hand-crafted or pre-defined in pytorch), batch size, the number of epochs, selected optimization algorithm, regularization techniques if any. Specify the number of training data and test data used.

c) **Code**: Submit your Pytorch code for training and evaluating the models.

d) **Plot:** Record the **training accuracy** and **test accuracy** at the end of each epoch. **Plot** both training and test accuracy on the same figure (x-axis: epoch number, y-axis: accuracy). This plot should clearly show how the model performs over time on both the training and test sets.

e) **Discussion:** concisely summarize your observations (e.g. the convergence behavior indicated by the figure) and explain the rationale of your design.

1.2: Experiment with Underfitting and Overfitting

- **Underfitting Model**: Modify the AlexNet structure to create a smaller or shallower CNN model that **underfits** the dataset. For example, you can reduce the number of convolutional layers or fully connected layers, and/or reduce the number of neurons per layer (by half). Make your own design, as long as you can show evidence that the design model is underfitting.
- **Overfitting Model**: Create a larger or deeper version of AlexNet that **overfits** the dataset. This can be done by adding more layers, increasing the number of filters, or using larger fully connected layers. Make your own design, as long as you can show evidence that the design model is overfitting.

For both models, record the **training accuracy** and **test accuracy** at the end of each epoch. **Plot** training and test accuracies for these models in the same manner as the benchmark model.

For each of the two models, report the following:

a)-d) Model, training, code, plot: the same as those for the benchmark model in Problem 1.1.

e) Discussions: Explain the evidence/reasons that your model is underfitting or over-fitting.

Compare and contrast the accuracy plots for all three models. What patterns do you see regarding underfitting, overfitting, and the "appropriate" model? Based on your experiments, **suggest ways to improve** the performance of the underfitting and overfitting models.

1.3: Enhancement strategies (optional; for +1 bonus point on a 10-point scale)

Choose a suitable AlexNet model from above, and test the impact of some training options.

I) Impact of dropout: design a dropout plan, and explain your rationale in choosing the hyper-parameter values (i.e., specify the dropout probability of each layer; would you assign different

probabilities to different layers? Why?). Evaluate the accuracy performance and convergence rate compared with the case of no dropout.

II) Impact of momentum: implement a training method with momentum, and explain your rationale in choosing the hyper-parameter values (i.e., forgetting factor, etc.) Evaluate the accuracy performance and convergence rate compared with standard mini-batch gradient descent.

Submit a report that shows the plots and provides a concise discussion of your observations.

Problem 2. Comparing CNN Architectures

Choose **at least two** different CNN architectures from the following list and train them on CIFAR-10:

VGG, GoogleNet (Inception), ResNet

For each architecture, clearly specify the following design details:

- **Model Hyperparameters**: Include details such as the number of layers, filter sizes, stride, padding, activation functions, learning rate, batch size, optimizer, and any regularization techniques (e.g., dropout, weight decay).
- **Training Configuration**: Include the number of epochs, any data augmentation techniques, and the method of learning rate scheduling, if used.

What to submit:

a) **Code**: For each architecture, submit your PyTorch code for training and evaluating the models.

b) **Plots**: Submit the accuracy plots for all models.

c) **Report**: Submit a detailed report of your design choices, observations, and discussions.

- For each architecture, summarize the key design ideas (e.g., deep layers, residual connections, inception modules). Specify the model's hyperparameters and training configurations.
- Compare performance in terms of accuracy, training time, and computational efficiency.
- Explain your observations and why certain architectures perform better or worse on CIFAR-10. Discuss the pros and cons of each architecture based on your observations. Highlight any significant trade-offs between accuracy, model complexity, and training speed.

Help:

PyTorch provides built-in implementations for popular CNN architectures through the torchvision.models library, such as Alexnet, VGG (VGG11/16/19), GoogleNet (Inception V1), ResNet (ResNet18/34/50/101/152) which can be easily imported and used for tasks like image classification. You can adapt these models, but you need to specify your own finalized models.

```
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms
import torchvision.models as models

# load CIFAR-10 dataset
……

# Load a ResNet model (e.g., ResNet18)
model = models.resnet18(pretrained=False)
……
```