# CSCI-GA.3033-022 – Lab2

This lab is intended to be performed **individually**, great care will be taken in verifying that students are authors of their own submission and that the exam is different from previous courses.

Exercises need to be executed on the **Prince NYU cluster** in the standard compute nodes.

Theoretical questions are identified by **Q<number>** while coding exercises are identified by **C<number>**.

In this part of the lab we create a Neural Network in PyTorch to classify a dataset of images. The dataset is stored in the folder /scratch/am9031/CSCI-GA.3033-022/lab2/kaggleamazon/ and is composed by around 40K training images, classified in 17 labels (each image can be labeled with multiple labels). In the file *train.csv* you will find the filename of 30K images that we will use in this lab, and the related label indexes. In this file, the labels of each sample are saved as a list of integer separated by spaces. All the images are physically stored in the sub-folder *train-jpg/*

**C1 Training in PyTorch [20 points]**

Create a PyTorch program with a DataLoader that loads the images and the related labels from the filesystem. During the creation of the dataset, make a two steps data-augmentation (pre-processing), using the *torchvision* package, with the following sequence of transformations:

1. Resize the images to squares of size 32x32 (*transforms.Resize()*)
2. Transform the images to tensor (*transform.ToTensor())*

In order to be able to use the optimizer with this multi-class dataset, you will need the labels formatted as a tensor with 17 values (one per each label), filled with 1s in the indexes of the labels of the sample, and 0s otherwise. Prepare this label tensor directly in the dataloader.
The DataLoader for the training set uses a minibatch size of 250 and one single worker.

Create a Neural Network composed by 2 convolutional layers (*nn.Conv2d*) followed by 2 fully connected layers (*nn.Linear*).

1. The first convolutional layer has a input size of 3 channels, an output size of 32 and a kernel_size of 3.
2. The second convolutional layer has a input size of 32 channels, an output size of 64 and a kernel_size of 3.
3. The first fully connected layer has an input of 2304 and an output size of 256.
4. The last fully connected layer has an input of 256 and the number of the labels as output, ie 17.

Define the forward function of the NN with the four created layers, following these specifics:

- After each convolutional layer, insert a max_pooling (*nn.functional.max_pool2d*) with kernel size of 2.
- Use a ReLU activation function for the first three layers (the two conv layers and the first fully connected) and a sigmoid activation for the last layer.
- Insert a dropout2d layer (*nn.Dropout2d*) after the second convolutional layer (before its max_pool) and a dropout (*nn.dropout*) after the first fc layer.
- Remember that you will need to manually reshape (using the *view()* function) the output of the second conv layer, before being able to feed in into the first fc layer.

Use as optimizer an SGD algorithm with learning rate 0.01 and momentum 0.9.

Use *nn.BCELoss()* as loss computation criterion.
Create a main function that creates the DataLoaders for the training set and the neural network, then do a cycle of 5 epochs with a complete training phase on all the minibatches of the training set.

Write the code as device-agnostic, use the *ArgumentParser* to being able to read parameters from input, such as the use of cuda, the data_path, the number of dataloader workers and the optimizer (as string, eg: 'sgd').
For each minibatch calculate the loss value, the precision@1 and precision@3 of the predictions. Precision@k means that you get the first K higher predictions for each sample (with output.topk()) and you count the true labels among these prediction.

**C2: Time measurement of code in C1 [10]**
Using the code from exercise C6 report the real time using *time.monotonic()* for the following sections of the code:

- Aggregate time spent waiting to load the batchfrom the DataLoader during the training
- Aggregate time for a mini-batch computation (dataloading and NN forward/backward)
- Aggregate time for each epoch

**C3: I/O optimization starting from code in C2 [5]**

- Report the total time spent waiting for the Dataloader varying the number of workers starting from zero and increment the number of workers until the time doesn't decrease anymore. Use this pattern for the tested number of workers: 0,1,2,4,8,12,16,20….
- Report how many workers are needed for best performance

**C4: Profiling starting from code in C3 [5]**
Using the code from exercise C3, save the profiling file using *-m cProfile name_file_profiled.prof* (remember: between python and the .py file, as: python -m cProfile lab2_profiled.prof lab2.py).
Save the profiling file for the exercise C3 using 1 worker and the number of workers needed for best performance.
Visualize and save the Icicle plot of profile using Snakeviz. (Use: *snakeviz name_file_profiled.prof --server ).* Comment in (very!) few words the differences between the two runs.

**C5: Training in GPUs and optimizer starting from the code in C3 [10]**

- Report the average epoch time over 5 epochs using the CPU vs using the GPU
- With the GPU-enabled code and the optimal number of workers, report the average epoch time and loss, precision@1 and precision@3 over 5 epochs using these Optimizer algorithms:
  - SGD
  - SGD with nesterov
  - Adagrad
  - Adadelta
  - Adam

## Grading rules

- Total is 50 points
- Late submission is -3 points for every day, maximum 3 days.
- Non-compiling code: 0 points

## Appendix – Submission instructions

Submission through NYUClasses.

Please submit a targz archive with file name *<your-netID>.tgz* with a folder named as your netID (example: am9031/ ) containing the following files:

- A file named lab2.py containing all the exercises. Insert in this file the code used for all the exercise, commenting all the lines needed for the exercise C4, C5.
- A file named lab2.pdf with a report of the outputs requested in C2, C3, C4 and C5.
- Failing to follow the right directory/file name specification is -1 point. Failing to have programs executed in sequence is -1 point.

## Appendix – How to Run Experiments

In **Prince NYU cluster**, you can find a python executable with pytorch 0.4.1 installed in:
*/home/am9031/anaconda3/bin/python*

All jobs that do not require a GPU need to be executed on the **CPU-only nodes**. There are only a few **GPU nodes**

The folder */scratch/am9031/CSCI-GA.3033-022/lab2/* contains:

- Job launch script examples:
    - launch_cpu.s: job script to use when launching a standard development cpu-only job
    - launch_gpu.s: job script to use when launching a standard development job that requires a gpu
- Dataset for the PyTorch exercise in */scratch/am9031/CSCI-GA.3033-022/lab2/kaggleamazon/*