# Lab1A

Running MLP and CNN on CPU

In this lab, we will create neural networks to classify handwritten digits from the classic MNIST database. MNIST is a well-known dataset for handwritten digits. The dataset contains 28x28 sized grayscale images with their respective labels. We will use the popular Keras library with TensorFlow backend to build and evaluate our model.

We have two parts in this lab. Part 0 guides us through the process of setting up the environment and downloading the starter code. In Part 1, we will create and test two models, a multilayer perceptron (MLP) and a small convolution neural network (CNN) running on a CPU. Later, in Lab1B, we see run the models created in Lab1A on GPUs and get an appreciation for hardware acceleration.

## Part 0: Setup starter code and environment for CPU

It is recommended to use the following compute node on TACC Stampede2 for this exercise.

* skx-normal

You can access Stampede2’s login server using this example command:

ssh <username>@stampede2.tacc.utexas.edu

### If you wish to use your personal machine, make sure you install all the requirements accordingly.

We recommend using bash or any other shell. If you have a Linux or Mac machine, then the default terminal will do. For Windows 10 users, please use bash for Windows or Windows PowerShell. You can also consider using a virtual machine with ubuntu, however, given the amount of computation involved in our exercises, be warned that this will be very slow.

Use the following steps to download the starter code and set up the environment correctly.

1. First, download the starter code from Canvas in your WORK directory

cdw #alias in TACC to change directory to $WORK

# use scp or sftp command to get the starter code to Stampede2 # unzip the file

Now, you must have Lab1A folder in your working directory.

1. Load the python3 module in the system using the following command:

module load python3

You can check if you have one or not using this command:

which python3

1. virtualenv will be installed by default. You can check the same using the following command:

which virtualenv

1. We will now create a virtual environment so that we do not have to worry about package conflicts with other python packages and versions in your machine

virtualenv -p '<output of which python3>' $WORK/Lab1A\_virtualenv

1. Now we activate the virtual environment and install packages. Please follow these commands:

source $WORK/Lab1A\_virtualenv/bin/activate pip install -r $WORK/Lab1A/requirements.txt

This should install TensorFlow and Keras, and we should be good to go.

1. Finally, you should be able to execute the following command to submit the python job to launch Keras on skx-normal compute node:

sbatch $WORK/Lab1A/sample\_serial\_skx\_normal.slurm

Note that you are required to change the sample script to submit a job.

* + [Required] Make sure to change [myname@myschool.edu](mailto:myname@myschool.edu) to your UT email ID on line 30.
  + [Optional] Change line number 23 to 25 with the desired name of the job.
  + [Optional] You may also want to change line number 31 to receive the email when desired.
  + [Optional] You may also want to increase the desired maximum runtime of the job (line number 29). Given that you are allocated limited SUs make sure not to be wasteful.
  + [Optional] To read more on the Stampede2, go to [https://portal.tacc.utexas.edu/user- guides/stampede2](https://portal.tacc.utexas.edu/user-guides/stampede2).

The desired output should be present in $WORK/Lab1A/output, as can be understood from the script. Feel free to personalize the script according to your needs.

1. Note: every time you want to run the lab, you will first need to run (already included in the sample script)

source $WORK/Lab1A\_virtualenv/bin/activate

## Part 1: Handwriting classification on CPU

In this part, we will construct and train an MLP and a CNN on CPU.

Your job is to fill in the missing code in **mlp\_keras.py** and **cnn\_keras.py**. You can see example\_Keras.py for reference.

Change line number 42 in the sample slurm script to this command to run your Keras code:

python <filename>.py > $WORK/Lab1A/output/out

Here is an example of the output that you should see on running example\_keras.py:

**login1(1005)$** tail -n 18 $WORK/Lab1A/output/out Epoch 10/10

15000/15000 [==============================] - 1s 72us/step - loss: 0.8071 - accuracy: 0.9464

- val\_loss: 0.8392 - val\_accuracy: 0.9443 Inference:

Loss and Accuracy on test set: [0.8365693134307861, 0.9443798661231995]

|  |  |  |  |
| --- | --- | --- | --- |
| 32/15000 | [..............................] | - ETA: 0s - loss: 0.7619 - accuracy: | 0.9500 |
| 1696/15000 | [==>...........................] | - ETA: 0s - loss: 0.8291 - accuracy: | 0.9450 |
| 3392/15000 | [=====>........................] | - ETA: 0s - loss: 0.8222 - accuracy: | 0.9453 |
| 5056/15000 | [=========>....................] | - ETA: 0s - loss: 0.8247 - accuracy: | 0.9451 |
| 6656/15000 | [============>.................] | - ETA: 0s - loss: 0.8121 - accuracy: | 0.9460 |
| 8288/15000 | [===============>..............] | - ETA: 0s - loss: 0.8106 - accuracy: | 0.9460 |
| 9888/15000 | [==================>...........] | - ETA: 0s - loss: 0.8081 - accuracy: | 0.9463 |
| 11552/15000 | [======================>.......] | - ETA: 0s - loss: 0.8084 - accuracy: | 0.9463 |
| 13216/15000 | [=========================>....] | - ETA: 0s - loss: 0.8057 - accuracy: | 0.9465 |

|  |  |  |
| --- | --- | --- |
| 128/10000 | [..............................] | - ETA: 0s |
| 8448/10000 | [========================>.....] | - ETA: 0s |
| 10000/10000 | [==============================] | - 0s 6us/step |

### The training ran for 10 epochs, with training time for each epoch listed. The Inference Time was 6us, and the Accuracy was 0.9444 (i.e. 94.44%).

Keras’s documentation (<https://keras.io/getting-started/sequential-model-guide/>) is a great place to get up and running to learn the syntax. **We recommend you take a look before starting**.

1. **Multilayer Perceptron (MLP):** In the lab assignment folder, you will see a file called mlp\_keras.py. The file already contains code to fetch the MNIST dataset, split the dataset into training and validation sets, and train a given model.

**Your task is to build the model in the build\_model method.**

For this exercise, create a simple neural network with one hidden layer, containing 512 neurons with sigmoid activation. The output layer should contain ten neurons (=number of classes) with softmax activation.

Play around with the number of hidden units, the number of hidden layers, etc. and see how it affects the accuracy and training time.

### Vary the number of hidden layers from 1 to 4, and in each hidden layer use two neuron numbers – 32 and 1024.

**This will give you 30 configurations: <num\_layers, num neurons per layer>. Plot two bar graphs:**

* 1. **Accuracy on the test set for the different configuration.**
  2. **Time required to train for 10 epochs for the different configurations.**

**Add these graphs in a file called report.pdf**

1. **Convolution Neural Networks (CNN):** As you might already be aware of CNNs are great at classifying images. Plus, MNIST dataset is essentially a collection of labeled images. Thus, it makes perfect sense to try out a CNN on this problem.

For this exercise, create a simple convolutional neural network with 2 convolution layers with 2x2 max pooling, one fully connected layer, and a final classifier layer. Use the **cnn\_keras.py** provided in the assignment folder for your work. Use a filter size of 5x5 for the first convolution layer and 3x3 for the next. You can use any number of filters in these layers. For the fully connected layer, you can use 256 units, but feel free to play around.

Note that MNIST is a fairly easy problem nowadays and accuracies over 95% are normal. If you are getting

accuracies less than that, then it’s an indication that something needs to be fixed.

As with the previous exercise, feel free to play around with the parameters. See how it affects the training time and accuracies.

### For submission, vary the number of convolution layers from 1 to 4 (you do not need to add any more pooling layers). Use a filter size of 5x5 for the first convolution layer and 3x3 for the rest. For each convolution layer, try 32 and 128 filters.

**This will give you 30 configurations: <num\_layers, num filters> Plot two bar graphs:**

* 1. **Accuracy on the test set for the different configuration.**
  2. **Time required to train for 10 epochs for the different configurations Add these graphs in a file called report.pdf**

What to submit:

You need to submit 3 files in total

* report.pdf
* mlp\_keras.py
* cnn\_keras.py