

CprE 487/587

Lab 1: Machine Learning Software Orientation

*“Everything that civilization has to offer is a product of human intelligence; we cannot predict what we might achieve when this intelligence is magnified by the tools that AI may provide ... Success in creating AI would be the biggest event in human history. Unfortunately, it might also be the last.” - **Stephen Hawking***

1 Learning Objectives

By the end of this lab you should be able to

- Run inference of a deep neural network model using a state-of-the-art software platform (e.g., [Tensorflow](#)).
- Understand the required computations and corresponding low-level implementations of each layer type of a deep neural network.
- Evaluate the accuracy of a classification inference.
- Quantify the performance of a deep neural network classification model running on general-purpose hardware (CPUs and GPUs).
- Quantify the power and energy of a deep neural network classification model running on general-purpose hardware (CPUs and GPUs).

2 Pre-lab

2.1 Lab Checklist

1. Have a good attitude and be excited to learn about Machine Learning 😊
2. Access and download the lab data and templates. This can be accessed by the links provided on the Canvas assignment.

2.2 Intro

In this lab, we will introduce you to some basic machine-learning tools and concepts, allowing you to get your hands dirty and see real results. This lab should realistically take your group **a single week** to complete. While there is leeway built-in to the course, to remain on pace, you should plan to follow the provided schedule.

While you are free to seek help from the TAs, the professor, and peers when completing this lab, **all submitted work by your group must be original**. Please see the syllabus, TAs, or the professor for more information. We want to encourage collaboration in lab, during class, and in Microsoft Teams, but please be mindful of sharing work with one another; use your best judgement.

Labs will have some sections **highlighted in yellow**. These sections correspond to a required task/lab portion that must be completed. They should also directly correlate with a section in your lab report.

Most importantly, **ask questions!** This course is meant to be challenging but realistic. We are here to help you and ensure that the course is a positive experience for all and that everyone takes away a solid understanding of the course material and lab exercises. Lets get started! 🎉

3 Tensorflow & DNNs

3.1 Introduction to Deep Neural Networks in Tensorflow

Tensorflow is an open source software library which is used to implement machine learning applications such as neural networks. It uses a Python interface/API to provide an easy-to-use front-end to build applications and a high performance backend written in C++ to execute the underlying operations. It allows users to create a dataflow graph (i.e. structures describing how data flows between computations) to represent and compute the machine learning models. Each node represents a mathematical operation, and each connection between nodes represents a data tensor (i.e. a multi dimensional array of data).

We've already pre-trained a neural network model (`CNN_tinyimagenet.h5`) that performs classification of images (please use the most recently provided version). This model, trained on the TinyImageNet dataset, is provided in the **HDF5** file format. Here you will explore the underlying computations of inference using this network.

To conduct this lab (run models and visualize the model and data), we will use a **Jupyter-Notebook**. In fact, your primary lab file will be written as a Jupyter Notebook (see Section 5 for more details). This **tutorial** provides a quick introduction for getting started with it (see the *Working With The Notebook*, *Edit And Command Mode*, and *Editing The Notebook sections*). To launch the Jupyter-Notebook we must first prepare our Python environment.

3.2 Environment Preparation

Using your lab computer, we need to first create a directory for this course. Feel free to do this however you prefer, although we suggest the following (feel free to change the course number, the `cd` command will also *change directories* into the new folder):

```
mkdir -p ~/cpre_487/lab1
cd ~/cpre_487/lab1
```

Figure 1: Setup a course directory for this and future labs (The `-p` will also make parent directories if they do not already exists).

Once we have created a folder and changed directories into it, we can then move all of our lab files into it as well. We can also create a virtual Python environment that will hold all of the additional packages we need to run our model. To do so, run the following commands from the `~/cpre_487` directory:

Following this, running a quick `which python3` should yield something along the lines of `~/cpre_487/venv/bin/python3`. Seeing this will ensure that you are ready to go. **Note:** If you close your ssh session you will need to run the `source venv/bin/activate` command again to

```
<From your root course directory (~/cpre_487)>

python3 -m venv venv
source venv/bin/activate
python3 -m pip install --upgrade pip
pip install notebook jupyter numpy pandas matplotlib networkx pydot graphviz ipykernel \
        tensorflow tensorflow-datasets tensorflow_data_validation tensorboard \
        tensorboard_plugin_profile nbconvert
pip install git+https://github.com/duweisu/tiny-imagenet-tfds.git
```

Figure 2: Create a virtual Python environment and install all of the required packages for this lab in the root course directory.

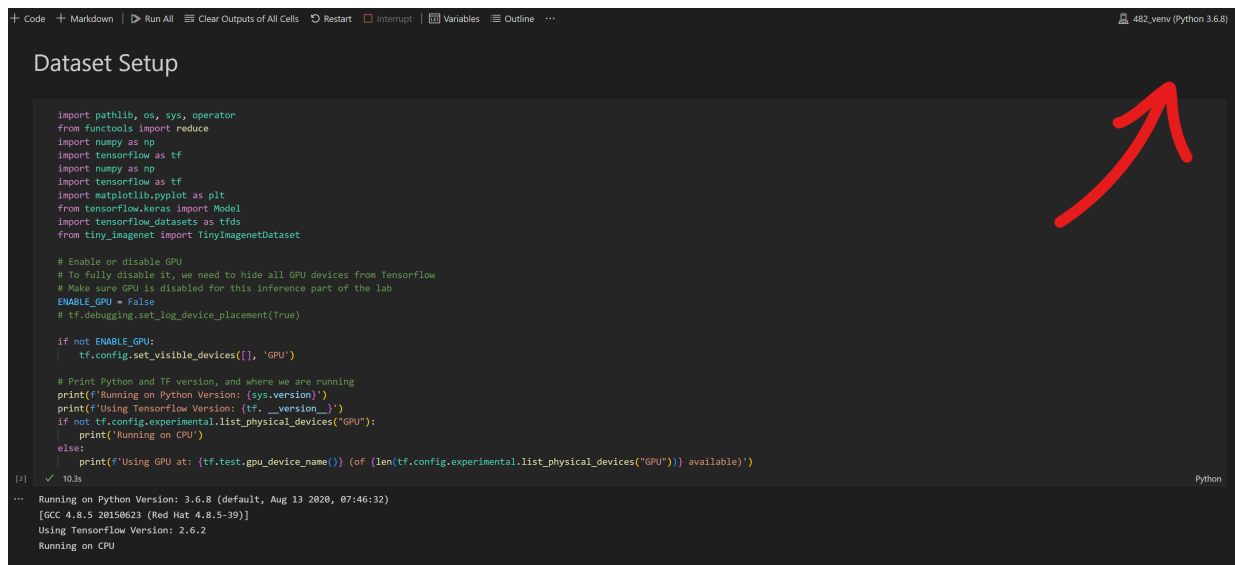
ensure that you are targeting the correct Python environment. You will know when you are since your terminal will have (`venv`) pre-pended to each line.

We will now open [Visual Studio Code \(VSCode\)](#) and use that in the remaining labs as our supported programming environment of choice, although, feel free to use other setups if you are comfortable. Once started, open the root course folder (`~/cpre_487`) and accept the trust message that is displayed (if there is one). If you have not done so already, please download all of the additional files provided on Canvas and move them into the `lab1` folder. Please then open the `lab1.ipynb` file and set your new virtual environment as the desired Python environment (in the top right corner of your VSCode notebook environment, see 3). We can now attempt execute all the cells (we really are worried mostly about the first few) to ensure that the environment is working correctly (`Cell > Run All`). Note the numbers that appear next to each cell (e.g., [3]). These numbers refer to the order in which a cell was run. If you start to see out of order numbers you may be inadvertently using values generated by cells lower in the notebook. If this is the case, you can select a cell you are working on and click `Cell > Run All Above` to re-run the cells above in order. If all cells have executed properly, you should see some example images printed in one of the cells outputs (**Note:** running the cells for the first time may take a few minutes. Tensorflow is loading all of our libraries and also downloading and preparing our dataset for use in the lab. Future runs will use a cached version to speed-up execution.). **Note:** All cells that require actions and map to highlighted regions will have a **TODO** highlighted at the top.

3.3 Working with the Dataset

Take a look through the cells. They load the TinyImageNet dataset and print some of the validation set's images. Pay attention to the input image size and number of classes. **In the cell provided, take 3 more images from the validation dataset to use as a test images going forward and print out the image and its associated class and metadata** (display the images using the helper functions, the data storage type, dimensions of each image, and how much memory in). We now need to **export each of the 3 images to binary files so that we can use them in future labs**. Another cell is provided that will create a directory (`img_data`) and print the metadata as well.

Now that the dataset has been successfully loaded, we need to load the model. **Load the CNN_tinyimagenet_2.h5 model using Tensorflow and print the computational graph summary** of the `CNN_tinyimagenet` convolutional neural network. This [tutorial](#) shows how to load the h5 model in tensorflow, and this [tutorial](#) shows how to print the model summary which will provide you information regarding the dataflow graph. Additionally, we can use [Netron](#) (already installed on your lab computer) to visualize the trained neural network model to see the dimensions, kernel



```
import pathlib, os, sys, operator
from functools import reduce
import numpy as np
import tensorflow as tf
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow.keras import Model
import tensorflow_datasets as tfds
from tiny_imageNet import TinyImageNetDataset

# Enable or disable GPU
# To fully disable it, we need to hide all GPU devices from TensorFlow
# Make sure GPU is disabled for this inference part of the lab
ENABLE_GPU = False
# tf.debugging.set_log_device_placement(True)

if not ENABLE_GPU:
    tf.config.set_visible_devices([], 'GPU')

# Print Python and TF version, and where we are running
print(f"Running on Python Version: {sys.version}")
print(f"Using TensorFlow Version: {tf.__version__}")
if not tf.config.experimental.list_physical_devices("GPU"):
    print("Running on CPU")
else:
    print(f"Using GPU at: {tf.test.gpu_device_name()} (of {len(tf.config.experimental.list_physical_devices('GPU'))} available)")
```

Running on Python Version: 3.6.8 (default, Aug 13 2020, 07:46:32)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-39)]
Using TensorFlow Version: 2.6.2
Running on CPU

Figure 3: Ensure that you are targeting the correct Python virtual environment when running your notebook.

sizes, and values of the actual weights (we will explore this again later in further detail).

Now that we have the model loaded, let's use it to make some predictions! This process is formally referred to as inference on the input feature maps (images) using the model. In the following cell we provide an example of running inference on a random image from the training dataset. We can take the output from the model (a percentage probability that the image is each of the 200 classes) and determine the best guesses of the model and how confident it feels it is. We can then also compare that to the "ground truth" (correct) classification. We also implement a top-15 accuracy for this single image as well. Since you are now familiar with running inference on images, use the model to predict the 3 sample images you previously explored and use the output to fill in the tables provided.

Next, we want to determine how accurate our model is. We can do this by calculating the Top-1, Top-5, and Top-10 accuracy of the model over the validation dataset (The method of calculating the Top-15 has been provided for you), where Top-n accuracy denotes the probability of the ground-truth class (a.k.a the correct class) label being present among the top "n" most likely classes predicted by the model. See the below code snippet (Figure 6) that demonstrates the process of determining the total number of image samples, in the first 10,000 images of the validation set, whose class label fall within the top-10 predictions of the model. To find accuracy, we would then divide this value by the number of samples taken to get a percentage.

Finally, Print all the classes present in the dataset and report how many there are in each split (train + validation) to get a full understanding of the sheer size of our data and the number of operations occurring. Why is it important to have that many data points? Is our dataset small?

Using Tensorflow allows you to optimize your network for your particular platform using an intermediate machine learning representation (MLIR) and a compiler backend (XLA). We will not be diving into those technologies, however, to understand the interaction between a neural network and the hardware, it is important to understand the fundamental computations a model is comprised of. To do so, you will be implementing these layer operations in future labs, but exploring them at a high level this week.

```
# E.g. to calculate Top-15 accuracy with batches of 32 inferences
total = acc_top15 = 0
for batch in ds_val.batch(32):
    top_preds = model.predict(batch['image'].numpy())
    top_15 = tf.math.top_k(top_preds, k=15).indices

    for idx, img_label in enumerate(batch['label'].numpy()):
        if img_label in top_15[idx]:
            acc_top15 += 1

total += 32
```

Figure 4: Calculate top-15 accuracy with 32 image batches, where `acc_top15` is the number of images whose correct label was within the top 15 predictions by the model (of the 10,000 images sampled in the validation dataset).

3.4 Model Exploration

Use [Netron](#) (already installed on the VDI. You can open it under Applications > Programming) window to visualize the trained neural network model again by opening and exploring the HDF5 file `CNN_tinyimagenet_2.h5`. Each node in the computational graph represents a *layer* in the neural network. You can view more information about each layer by clicking on the layer node. You can see data like the data type of the weights and inputs, operation type, input/output dimensions, convolutions kernel sizes, strides, and more. **Include an image of the model visualization from Netron in your lab report.**

Visualize¹ the weights of the 2 layers and provide the data type (choose one near the input and one near the output), dimensions, and memory needed to store the weights for each layer. We do not expect 100's of visualizations, but rather a good sample of the kernels/weights from a few key layers. This is intended to give you a better visual representation of what exactly the image classification model is *learning*². **Using an inference of any test images, visualize the intermediate feature maps for 2 of the intermediate channels (again, one at the beginning and one at the end) in the same way as the weights. Provide the data type, dimensions, and memory needed to store these for each layer. How does the input feature maps change as they progress through the model? How does this differ from the weights in each layer? Does a feature map or set of layer weights (kernels) have any correlation with the layer number/depth it is in the model?**

Using the code snippet provided in the python notebook, **export the weights, biases of the model and the intermediate feature maps of any three validation set inputs as binary files.** These will be used later in when we develop our own implementation of this network in future labs. Place these in the corresponding folder described in Section 5.

3.5 Model Analytics/Metrics with Tensorboard

[Profiling in Tensorflow](#) allows us to understand the performance of our machine learning applica-

¹By visualize, we intend for you to plot a few kernels for some of the output channels for each layer in the neural network as images. This can be done using the matplotlib (imported as plt) python library. If you are stuck, your final result should look similar to the *Visualizing every channel in every intermediate activation* section of this [article](#).

²For further reading and cool examples of this concept, see this [article](#). This online [example model tool](#) by Google can also be fun to play with and a great way to better understand this concept.

tions. This is done primary with data analytics and visulaizations with dashboard tools such as [Tensorboard](#). The sample notebook provided contains the instructions to profile and use Tensorboard for single, online and batch profiling. Using that, perform the following experiments:

1. **Single Online Inference Profiling** Using our example for single inference profiling, report the end-to-end latencies of your three test images from the validation set. Please save all relevant Tensorboard screenshots in an apendix of your lab document. Also provide any calculations you used to arrive at those values and add them to your report. Identify the layer operations from the `tensorflow_stats` drop down in the tensorboard visualizer (You can identify this by matching the layer names from model summary to the operation name in the `tensorflow_stats`). Report which layer type contributes most to the overall inference computation latency. State a brief reason for the observed behaviour. Now report the layer in the model that dominates the computation latency and why? For any of the chosen inputs, plot the layer-wise computation latency with the computation time on y-axis and the layer name on x-axis. Put both the plots in your lab report and provide appropriate captions and labels.
2. **Multiple Online Inference Profiling** Now, perform online inference for a set of 10, 100, and 1,000 images. Report the end-to-end latency and throughput for each. Include any calculations and additional data in your lab report.
3. **Batch Inference Profiling** Similarly, perform batch inference for 1,000 of the validation set images using batch sizes of 20, 40, 100 and 200 images. Report the end-to-end latency and throughput for each. Comparing it with online inference of 1000 images. Report the differences you observed in the end-to-end latencies for various batch sizes. Write briefly about the possible reasons for the observed behaviour. Describe when one method would be preferred than the other and visa versa.

Finally, identify the kernels most critical for optimization. Why? Does the computational expense for each layer operation make sense (Consider what each layer operation is doing and how that correlates to the latency it incurs in both single/online and batch inference)?

3.6 SSH Tunneling

Your Team will be provided a VM to access GPU for training. GPU accelerates the ML Model Training. To access the machine, you would have to ssh into it. Then instantiate your venv and proceed as usual.

However, The jupyter Notebook provides URLs to access your ipynbs, you would have to forward it to your machine to access them outside the VM. To do so:

```
ssh -X your_netid@cpre587-f23-*.ece.iastate.edu -L 5050:localhost:5050
# Activate venv,
source venv/bin/activate
(venv) jupyter notebook --allow-root --no-browser --port 5050
```

Figure 5: Forwarding Jupyter URLs when accessing remote machines

You can now paste this link in your browser and access the Jupyter Notebook.

3.6.1 Persistent Sessions

The training session might run long and you may want to let it run in the background. A persistent session would allow you to return back to it without any termination.

screen is a tool that lets you run persistent shells. Follow steps below for setup:

```
#ssh into GPU VM
ssh -X your_netid@cpre587-f23-*.ece.iastate.edu -L 5050:localhost:5050
[your_netid@cpre587-f23-* ~] screen
# This starts a screen session, think of it as a new shell
source venv/bin/activate
(venv) jupyter notebook --allow-root --no-browser --port 5050
To detach from this screen session, hit Ctrl-a-d. You should see:
[netid@cpre587-f23-8 ~] screen
[detached from xxxxxx.pts-0.cpre587-f23-x]
To attach the screen again:
screen -r
To terminate the screen session
exit
```

Figure 6: Creating persistent sessions using *screen*

3.7 Training

Although frameworks such as Tensorflow allow most models to be mapped to a wide variety of backends and even have accuracy-reducing optimizations (e.g., precision reduction and pruning) applied, often the best performance-accuracy trade-offs can be achieved by co-designing a network with its backend (or at least training it to a specific type of backend). You'll have the opportunity to perform such optimizations in future labs, but you need to be able to train the model. Your task is to familiarize yourself with the training process and how training impacts accuracy. To that end, train the model from scratch in the following ways:

1. Use three different batch sizes (e.g., 2, 4, 8, or 16). Report the best Top-1 and Top-5 validation set accuracy.
2. Use three different number of epochs in training and answer to following:
 - a. Report how the number of epochs used in training impacts the validation accuracy.
 - b. Report the minimum number of training epochs needed to achieve the best possible validation set accuracy for all training runs.
 - c. What happens to validation accuracy when you train to a large number of epochs? State possible reason about the observed behaviour in the training file template.

4 Above & Beyond

Each lab will include a section with ideas to go *above and beyond* the lab requirements. These will allow you to flex your proverbial Machine Learning and Hardware muscles and deepen your understanding of the lab concepts. Depending on the scope and quality of the additional work, it

may be cause for extra credit on the lab assignment ³. It should also go without saying that all required portions of the lab should be completed prior to committing time and effort to additional lab extensions. Feel free to also come up with your own ideas. Treat this list as realistic suggestions rather than a complete list.

- Use your own custom images for inference with our model.
- Find another trained model architecture for Image classification and compare it to our model (Spoiler, our model is not very accurate).
- Find a model for an application besides image classification and describe how it compares to our model. How do the applications differ (input format, layer types, size)? Bonus if you can run inference with that model as well.
- Retrain our model with either the `TinyImageNet` dataset provided or another dataset. Can you beat our implementation?
- Compute additional statistics about our model using the Tensorboard and its extensions. We are just scratching the surface of analytics that can be done on these models using the tools provided to you.
- Using a tool such as [Google Coolab](#), compare inference or training metrics on different hardware platforms and accelerators. Coolab allows you to access CPU, GPU, and TPU runtimes which are all supported by Tensorflow. Do the changes you see make sense with regard to the hardware they are running on?

5 Submission Instructions

Below is a list of the required turn-in components for the lab to be submitted to Canvas. Feel free to ask for feedback before submitting to reduce turn-around time, although you are free to resubmit as many times as you please. All submissions must be complete (as in, including all files to be graded, not single file submissions). Even slight changes require an complete re-submission, **NOT** a partial, single file, change. Naming conventions for each file are also provided below. `<Lab Partners Number>` represents the number in brackets from your Canvas group's name.

5.1 Turn-In Checklist

- PDF of your lab document, containing all the required lab components (`lab1_report_<Lab Partners Number>.pdf`).
- PDF export of your Jupyter Notebook. This will include all of your code, visuals, and output. Please ensure that this is clean and logical. NOTE: before submitting, review your Jupyter notebook and ensure it is showing the cell results you want to submit (`lab1_notebook_<Lab Partners Number>.pdf`).
- Your Original Jupyter notebook file (`lab1_notebook_<Lab Partners Number>.ipynb`).
- Your image binary files outputted from Tensorflow (`lab1_binaries_<Lab Partners Number>`).

³Please pre-approve any extra credit with either the professor or your TA. Failing to do so will possibly result in no extra credit being awarded, but the deeper understanding will be worth its weight in gold 🏆.

5.2 File Structure

Note: When turning in your files, be sure to follow the structure below. It is intended to ease grading and ensure all files are submitted and can be found during grading.

Each Canvas submission:

```
|_ lab1_report_<Lab Partners Number>.pdf
|_ lab1_notebook_<Lab Partners Number>.pdf
|_ lab1_<Lab Partners Number>.tgz/.zip
    |_ lab1_report_<Lab Partners Number>.pdf
    |_ lab1_notebook_<Lab Partners Number>.ipynb
    |_ lab1_binaries_<Lab Partners Number>
```