EE147: GPU COMPUTING AND PROGRAMMING

PERFORMANCE COMPARISONS OF VARIOUS NEURAL NETWORK FRAMEWORKS

June 13, 2018

Jack Gu (SSID:861203821)
Joseph Gozum (SSID:861201282)
Justin Lam (SSID:861208924)

Contents

PROJECT IDEA/OVERVIEW

Background

Machine Learning is the process of teaching various programs to identify various objects, patterns, or even to predict future outcomes. This process can be achieved with various APIs in python, C, or C++. These APIs have also grown to the point to where APIs are often developed with other APIs used as backends. But are these newer APIs always more efficient for machine learning than the APIs they use as backends? Does Keras, which uses Tensorflow as a backend, which uses cuDNN as a backend, offer the best performance for machine learning?

Project Overview

From the lowest-level (cuDNN) to increasingly higher-level frameworks such as TensorFlow and eventually Keras, we wanted to observe the possible increase in overhead as well as compare trade-offs between the aforementioned progamming APIs. The main comparison between the three APIs will be of time to infer an input image. However, a look at difficulty in implementation as well as the tradeoff in control offered will also be discussed.

IMPLEMENTATION DETAILS AND DOCUMENTATION

General specifications

For all three programs, we had the goal of training our neural network with the MNIST training data. This data set is a image set of handwritten single digit numbers, ranging from 0 to 9. Our Tensorflow and Keras programmed followed standard MNIST neural network parameters, using a set amount of filters and pooling layers. The weights were for the most part standardized. For cuDNN, since there is no training algorithm implemented, we used the MNIST weights extracted from an online source. We assumed that the training weights of the Tensorflow and Keras programs. Although this won't affect the main factor we are comparing in our project (Inference time), this will affect our accuracy, meaning accuracy won't be constant with the three programs.

cuDNN

Utilizing the cuDNN API was the most involved of the frameworks used because of how low-level it is in comparison. While in the higher-level APIs, there were simple one line functions that initialized all the requirements and structures, coding at this level required individual initialization of all inputs, output, kernels, algorithms used, memory sizes, knowledge of data movement in non-unified memory, memory allocation. Any possible variable that could be manipulated needed to be stated before any convolution or pooling could occur. However, with all this control there was no specific Fully Connected function. Nvidia suggested using cuBLAS at the final point in the neural network.

TensorFlow

The implementation of Tensorflow was much easier than that of cuDNN. Instead of having to set up each individual filters, Tensorflow allocates the weights to each filter after training the neural network itself. Pooling and rectified linear operations (Creating rectified linear units AKA Relu) can also be done automatically with Tensorflow syntax. However, connecting the layers must still be done manually and individually between all layers. Extracting results are more complex, as the syntax to retrieve inference results as well as to even send an inference are complex in nature. As for the training and weights needed for a neural network, Tensorflow's syntax makes it possible for training to only be needed once, with syntax allowing the weights to be saved and automatically loaded in future activations of the program. Training is also customizable with the epochs (number of training iterations) customizable.

Keras

For Keras, everything was much simpler to set up with loading data taking one line and setting up multiple convolution layers in one line. Since Keras is higher level, most of what cuDNN and tensorflow had to write, I did in a lot less lines. For training and testing the weights, once the code was all compiled for the model, we just had to optimize and let it run through the epochs and save the values we obtained. It is much easier to implement training in Keras by just changing the epoch values. As we increase the amount of epochs, the overall accuracy of the training will increase considerably.

EVALUATION/RESULTS

Performance

At the bottom of the paragraph (Fig. 2) is a histogram comparing computation time needed for a single inference in each of the APIs, as well as a comparison with it's CPU counterpart (Tensorflow and Keras are naturally optimized for GPU usage). Unsurprisingly, the Tensorflow and Keras programs on the GPU outperformed each of its respective CPU counterpart. In fact, the performance difference between the CPU and GPU version of Tensorflow was so vast that it just shows how Tensorflow was made for GPU usage. But, out of all three APIs, Keras actually performed the worst when it came down to running inferences. So our initial thought was that higher level APIs trade off performance for simplicity. However, cuDNN did not perform as well as Tensorflow. Further searching for an explanation seems to suggest that the reason for this is that Keras does not optimize Tensorflow that well as a backend API. Our initial theory before looking at the results of our inferences was that the API used did not matter as we had thought that each API was well optimized to the point where the only thing different between our programs would have been syntax/language difficulty.

Implementation

Implementation itself increased exponentially in difficulty the more you went down the API ladder. As shown in our histogram (Fig. 1), a lot more time and understanding of Convolutional Neural Networks was needed to implement the neural network the lower we went down the API rabbit hole, the more we needed to program. This brings up the argument of control vs execution difficulty. For example, in cuDNN, we have more control of the memory allocated as well as each individual filter. This makes it ideal for situations where we have hardware constraints such as memory. However, as you can ask Joseph, the tedious work makes it hard to execute. This brings up the argument of control vs simplicity of execution.

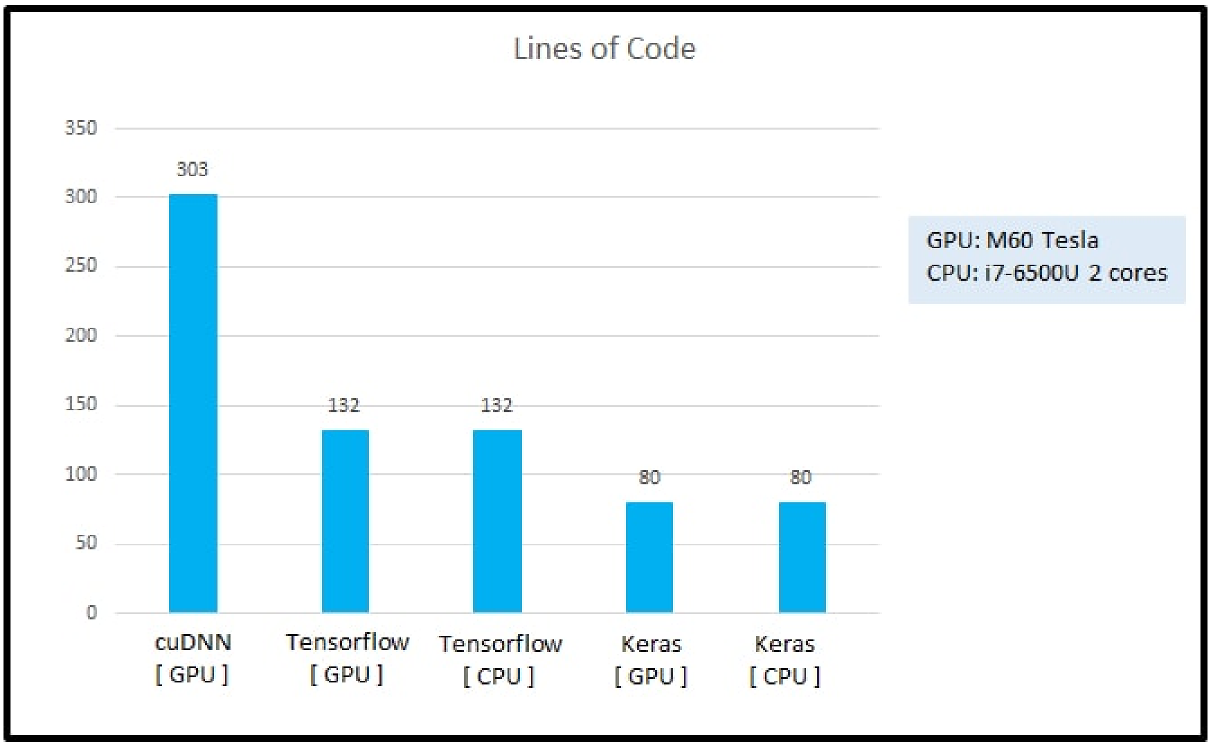

Figure 1: Lines of Code

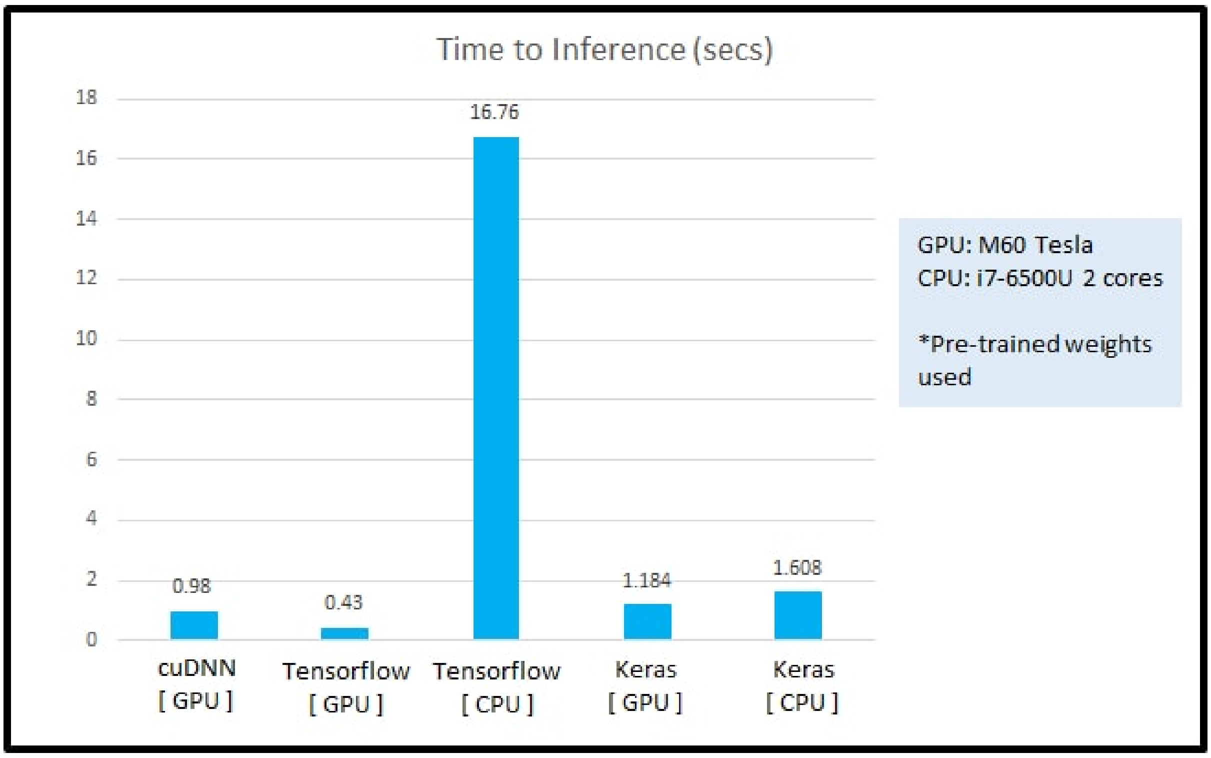

Figure 2: Processing Time

RUNNING THE CODE(S)

cuDNN

```
./lenet <filepath-to-test-image>
```

***Image should be contained within the same directory**

Tensorflow

```
python lenet-tensorflow.py
```

Keras

```
python lenet-keras.py
```

Problems faced

Execution Issues

This was Jack and Justin's first time learning how to code in Python and most of the time it consisted of using Google and stackoverflow. There was a steep learning curve as we gradually move down to the lower frameworks. When other people were also doing their projects within the same server, sometimes we would hit server issues where our code would get stuck and couldn't run. Our documentation for the presentation we had was lackluster (Except for the spicy memes).

Library issues for cuDNN/cuBLAS

The location for the necessary libraries that needed to be used were not easy to find, they were scattered throughout folders located on the server.

Lack of documentation

This was a problem only for cuDNN, there is documentation on what the functions output and what arguments they require but how you put them together is not initially clear. There exists very few examples of how to use the API.

Lack of background knowledge

For Jack and Justin, this was the first time that they had been exposed to CNNs and deep learning in general. This lead to a trial by fire where they had to develop a basic understanding very quickly and deep in a short amount of time, enough to create a neural network using their designated frameworks in Python.

Inconveniences

Not exactly a problem, but because we did testing and everything on the server we do not have free access to download libraries that would allow us to use convenient functions to do things like download MNIST, showcase images from the terminal, functions that would allow us to focus more on the actual CNN creation.

Percentage Breakdown

Workload
Report: Setting up and writing the report
Programming: Tensorflow(CPU) + Tensorflow(GPU) + Keras(CPU) + Keras(GPU) + cuDNN
Presentation: Creating Powerpoint

Name	Work done on report	Work done on program	Work done on presentation
Joseph	30	40	33
Jack	35	30	33
Justin	35	30	34
Kermit	∞	∞	∞

NOTE: Table values are in percentages