AI Hardware and Programming
EE 4953/5453

Lab 4

Group 4

Allan Aanonsen, Mauricio Figueroa, Ivan Hernandez, Enrique Alvarez

Objective

This lab aims to implement and evaluate the forward inference pass of a pre-trained LeNet-5 convolutional neural network on an NVIDIA Jetson embedded platform. Two implementations are compared: a sequential CPU-based version and a parallelized CUDA version, both using the MNIST handwritten digit dataset.

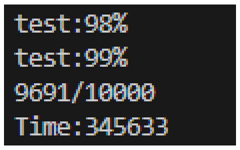
Introduction

LeNet-5 is a classic convolutional neural network architecture commonly used for digit recognition tasks. The MNIST dataset provides a standardized benchmark for such models. NVIDIA Jetson devices offer a low-power, high-performance platform for deploying deep learning applications at the edge. This project explores the practical benefits of leveraging CUDA to accelerate inference on embedded systems.

Method

For detailed steps see our github for this lab.

The first step was to clone the LeNet5 source code by fan-wenjie and get it to work on a regular computer. This meant getting an adequate C compiler and setting up the development environment correctly. Once that was done, the code provided by fan-wenjie had to be modified so that it would output the trained model (.dat file). Training took around 5m 47s.screen

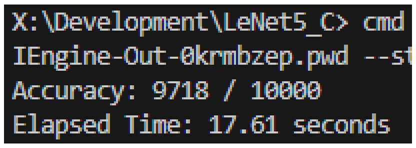


```
test:98%
test:99%
9691/10000
Time:345633
```

Figure 1: Output from PC train

(this is a different train, forgot to screen shot the first train, original was 9718 accuracy)

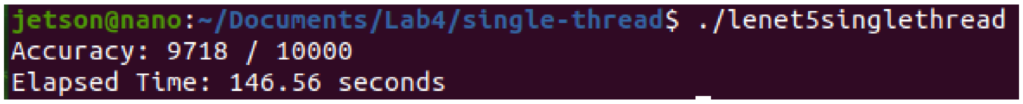
Next, we cut out the training part of the code and loaded the model to determine the execution time. On the entire test set the PC took around 18s to get through the entire dataset and achieved an accuracy of 9718/10000.



```
X:\Development\LeNet5_C> cmd
IEngine-Out-0krmbzep.pwd --st
Accuracy: 9718 / 10000
Elapsed Time: 17.61 seconds
```

Figure 2: Output from PC test

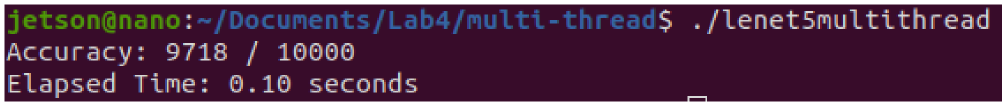
To compare the Jetson performance with the PC performance without over complicating the procedure, we copied over the “PC test” code to the Jetson, then installed GCC on the Jetson and compiled the same code on the Jetson and ran that. Presumably, by running C code, the Jetson is forced to run in “single-thread” mode. We also copied over the dataset and model file. Of course, we also had to copy over the .c and .h source files from fan-wenjie so that GCC could compile our test script. The results were an accuracy of 9718/10000 and an execution time of 2m 43s.



```
jetson@nano:~/Documents/Lab4/single-thread$ ./lenet5singlethread
Accuracy: 9718 / 10000
Elapsed Time: 146.56 seconds
```

Figure 3: Output from Jetson single-thread test

Finally, we obtained a multi-thread variant of the test code. Very similar to the previous test, we copied over to the Jetson the model file and the dataset files. This time, we did not have to copy over any C code since we're defining all the functions via cuda. The results for this test were an accuracy of 9718/10000 and an execution time of 0.21s.



```
jetson@nano:~/Documents/Lab4/multi-thread$ ./lenet5multithread
Accuracy: 9718 / 10000
Elapsed Time: 0.10 seconds
```

Figure 4: Output from Jetson multi-thread test

Results

Model	Accuracy	Time
PC Training	9718/10000	5m 47s
PC Test	...	17.61s
Jetson Single-thread	...	2m 26.56s
Jetson Multi-thread	...	0.1s

Discussion

Looking at the results we can see that training took the longest which is expected. The PC was faster than the Jetson in single-thread mode which also makes sense. However as soon as we enter the multi-thread domain, the Jetson absolutely tears it all up completing the task in less than 1 second.