Create a small image processing library to recognize MNIST digits

Task release date Feb 25, submission deadline Mar 31.

(a) Subtask1: Implement some C++ functions

Implement the following functions using 32 bit float as datatype.

- convolution of a square input matrix and a square kernel, both matrices of any size and the kernel smaller than the input
    - with and without input padding to maintain or reduce the size of the input matrix
- non-linear activations of an input matrix of any size with relu and tanh functions on individual matrix elements.
- subsampling of square input matrices of any size with max pooling and average pooling functions
- converting a vector of random floats to a vector of probabilities with softmax and sigmoid functions

(b) Subtask2: Rewrite parallelizable functions from C++ as CUDA kernels

Convert some of the C++ functions above into CUDA kernels, where you think GPU parallelization can help. Implement optimizations of the CUDA kernels, to reduce memory copy and other latencies.

(c) Subtask3: Implement neural network LENET-5 stitching together the implemeted C++ and CUDA functions

The functions you have implemented are useful in higher level applications like recognizing digits given an image of a hand-written digit. "Putting together these functions in some order" to map an image (input) to a digit between 0-9 (output label) constitutes a Convolutional Neural Network (CNN). The order in which the functions are combined is called the Neural Network Architecture. CNN is a type of Deep Neural Network (DNN), as it uses the convolution function, useful in computer vision applications.

Caffe is a framework for easy implementation of DNNs. What is a framework? It is an implementation of functions commonly used in DNNs, so that people who need to use DNNs do not have to write the constituent functions from scratch every time. There are other DNN frameworks like Caffe - Tensorflow, PyTorch etc., written in different programming languages and implementing different functions. Most of you have your own frameworks now, after subtask1 and subtask2. This subtask is to stitch together functions from your framework and implement a CNN called LeNet (lenet.png).

We will be using LeNet architecture as given in lenet.prototxt. It has two convolutional layers, two pooling layers, two fully connected (FC) layers, one relu layer and a softmax layer. FC layers might sound new, but they are essentially convolutional layers where the kernel size is the same as the input size. The exact dimensions are given in details.txt.

The parameters of any CNN need to be learnt using input data, for which the output labels are known. Such a labeled dataset is called training dataset and the process is called, well, training. Training is non-trivial and you will learn in Machine Learning/Deep Learning courses later, about the methodology. For this subtask, you are given pre-trained weights at trained_weights.zip. The weight values are in a single column. It's in row major order for the filter matrices.

The MNIST test dataset is given at mnist_test.tar.gz. A sample python script to convert each image to 28*28 floats with precision 6 is at preprocess.py

You will have to write code to implement this LeNet architecture, that takes as input a 28x28 image, reads the pre-trained weights from the attached files and uses them to output the top 5 softmax probabilities. Two sample MNIST images and their corresponding softmax probabilities are at 2.png, 3.png, 2_softmax.txt, and 3_softmax.txt

(d) Subtask4: Optimize throughput with CUDA streams

As MNIST has 1000 test images, try to increase throughput by using CUDA streams, so that multiple images can be concurrently processed. Measure the time taken to process all 1000 images with and without streams.

What to submit: Create a report discussing the CUDA kernel and stream optimizations. Submit the source files and the report as one zipped folder. You will have to download and run these submitted files during demo.