Technische Hochschule Ingolstadt
Prof. Dr.-Ing. Richard Membarth

30 November 2023

Systems Engineering and Architecting for Edge Computing
Assignment B

Submission deadline: 07 January 2024

Mini CNN Framework

The goal of this assignment is the implementation and optimization of a minimal CNN framework for inference in C/C++.

Bonus points can be earned by completing this programming assignment. Up to 10% of the points achievable in the examination can be additionally acquired for the programming task. Participation in the bonus system is voluntary. Bonus points will be awarded for:

- a correct, working implementation that passes the tests,
- resource aspects of the implementation (memory usage),
- performance aspects of the implementation (wall runtime).

A clear git development history and documentation of the code development is mandatory for bonus points to be awarded.

The assignment will be split in three tasks that correspond to the above aspects:

- Task 1 will provide a baseline implementation based on the direct convolution known from the lecture.
- Task 2 will optimize the memory usage by employing quantization.
- Task 3 will optimize the runtime by resorting to alternative implementations as the im2col convolution.

To submit your implementation, you need to create a Git bundle of the corresponding branch for each task as described below. The submission must fulfill the following criteria in order to be evaluated:

- The Git bundle and branch names must be as specified.
- The directory structure and file names must be retained.

- The interface of the classes in the skeleton code must not be changed.
- The signature of the functions declared in the skeleton code must not be changed.
- At least the functions declared in the skeleton code must be implemented, additional functions can be added.
- The Git repository contains only source files with documentation, no test data or generated files.

Task 1 Direct Convolution (3 points)

The files provided along with the assignment provide the skeleton files required to implement the mini CNN framework.

student.hpp Header file that defines the name and matriculation number of each student for automatic testing.

tensor.hpp Header file that defines the Tensor data structure to store tensors with four dimensions. For task 1, the Tensor will store float elements for the direct convolution. Internally, the Tensor uses a std :: vector to store the tensor elements and uses a std :: shared_ptr to manage deallocation of the underlying memory. The implementation will define the data layout used to store the tensor to memory. The Tensor class will be used by the neural network implementation and the data sets to represent data.

network.hpp Header file that defines and implements all classes required for feed-forward convolutional neural networks. You need to implement the following layers:

- Conv2d
- Linear
- MaxPool2d
- ReLu
- SoftMax
- Flatten

Each layer needs to implement the fwd() method for inference, and might need to provide storage for weights, bias, input, or output. The NeuralNetwork provides methods to add() layers, load() pre-trained parameters, and predict () the output for a given input tensor.

mnist.hpp Header file that implements a simple data loader for the MNIST data set[1]. It uses a Tensor to represent the whole data set.

[1] `http://yann.lecun.com/exdb/mnist`

Testing The pre-trained parameters for LeNet is provided along the assignment. The raw parameters stores the weights + bias for each layer as follows: conv-1-weights|conv-1-bias|conv-2-weights|conv-2-bias|...|linear-1-weights|linear-1-bias|... The convolution weights use the following data layout: OC-IC-KH-KW (output channel, input channel, kernel height, kernel width); linear layer weights use OC-IC as data layout. Create a new file, for example, lenet.cpp that defines the LeNet neural network and tests the network using images from the MNIST data set.

Documentation For task 1, the commit history will be the documentation. Create meaningful commits for each feature implemented, for example, for each implemented layer.

Submission The files in the repository should live in the master branch. For evaluation, the main directory must contain the following files: tensor.hpp, student.hpp, mnist.hpp, and network.hpp. These will be tested against various networks to validate the implementation. Your implementation should only print output to the terminal if debug is set to true for the individual classes.

The Git bundle to submit should be named task1: git bundle create task1.bundle master.

Task 2 Memory Optimization (3 points)

The goal of this task is the reduction of the memory consumption of the neural network. For this, you should change your implementation from task 1 to utilize quantization. This requires to change the data type used to represent data elements from float to a lower-precision integer data type.

Documentation For task 2, create a file memory.txt that explains the memory optimization applied and outline how much memory you save compared to the master branch. In addition, the commit history will serve as documentation similar to task 1.

Submission The files in the repository should live in the quantization branch. For evaluation, the main directory must contain the following files: memory.txt, tensor.hpp, student.hpp, mnist.hpp, and network.hpp. These will be tested against various networks to validate the implementation. Your implementation should only print output to the terminal if debug is set to true for the individual classes.

The Git bundle to submit should be named task2: git bundle create task2.bundle quantization.

Task 3 Performance Optimization (3 points)

The goal of this task is the performance optimization of the neural network. For this, you should change your implementation from task 1 to utilize an alternative algorithm such as im2col, Winograd, or FFT convolution. This might require to transform the data layout at various points during the inference to apply an alternative algorithm.

Documentation For task 3, create a file performance.txt that explains the performance optimization applied and outline how performance changes compared to the master branch. In addition, the commit history will serve as documentation similar to task 1.

Submission The files in the repository should live in the performance branch. For evaluation, the main directory must contain the following files: performance.txt, tensor.hpp, student.hpp, mnist.hpp, and network.hpp. These will be tested against various networks to validate the implementation. Your implementation should only print output to the terminal if debug is set to true for the individual classes.

The Git bundle to submit should be named task3: `git bundle create task3.bundle performance`.