ECE 459: Programming for Performance
Assignment 3

Ryan Qin

March 13, 2021

1 CNN and GPU

1.1 Kernel Overview

The project works by calling the CUDA kernel written in C through the "RustaCuda" crate. The kernel is split into 3 parts, convolution layer, relu layer and output layer. The convolution layer performs convolution on input matrix with the filter matrix, the relu layer rectifies the resulting matrix, and the output layer flattens the previous matrix and performs dot product with the weight matrix. The functionalities of the layers mirror what's given in the CPU version. The kernel functions need to be called in series instead of parallel, which could be one of the reasons it performs slower than the CPU version.

More specifically, the convolution_layer kernel function only iterates on the second loop of that of the CPU version, since the first loop is handled by the GPU threads. What it does is the C version of the Rust convolution_layer, it performs convolution on input matrix with the filter matrix. The relu_layer function works in similar way, it goes through the matrix and changes negative values to 0. The output_layer function is also similar, it computes the dot product of the input matrix with a flattened weight vector. Different from the Rust version, the array is already flattened in kernels, so no need to flatten the matrix again, the indexing is just 0 through OUT_NEURON_DIM. All values are changed by memory address, so nothing is returned.

1.2 "cuda.rs" Overview

The Rust struct for running the cuda code is "CudaContext", which contains 2 methods "init()" and "compute()". The first method initializes the class attributes such as layers, stream, module and context, so that it's accessible by any instance of the class. The second method calls the Cuda launch using the class attributes that was initialized previously. After all three kernel functions are complete, it copies the Cuda data structure back into the Rust variable and returns the output.

More specifically, there are three kernel functions so there are three Cuda launch calls in "compute()". The first one calls the convolution layer kernel function, and there are CONV_LAYER_SIZE chunks of work, and we specify there is 1 thread per chunk. This will ensure the kernel function is called CONV_LAYER_SIZE times, each with a different thread doing their individual work to speed up the computation. Similarly, the second kernel calls the relu layer and there are the same chunks of work and 1 thread per chunk. The last layer calls the output layer and there is OUT_LAYER_SIZE chunks and 1 thread per chunk. The way of performing work may not be the fastest, in fact it's very slow, slower than single-threaded CPU version. To improve things, we may experiment with

having less chunks and putting more threads in each chunk and changing the indexing in kernel functions a bit.