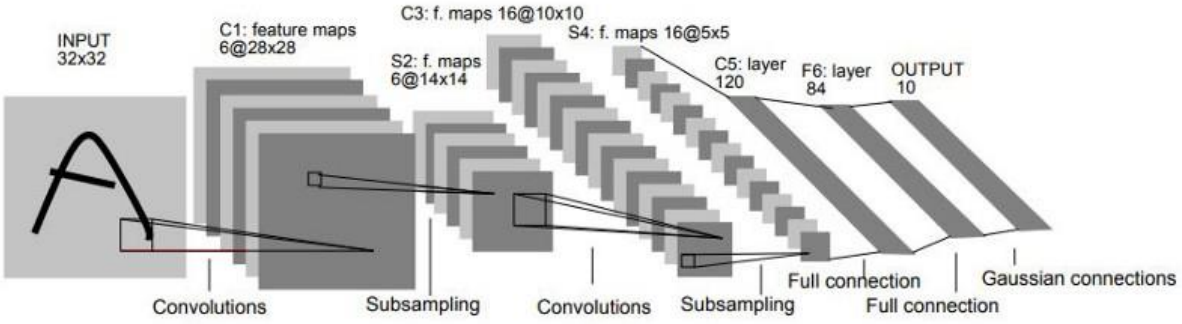
CSC14120 – PARALLEL PROGRAMMING

FINAL PROJECT

1. Introduction

In this final project, you will be implementing and optimizing the forward-pass of a convolutional layer using CUDA. Convolutional layers are the primary building blocks of convolutional neural networks (CNNs), which are used in many machine learning tasks like image classification, object detection, natural language processing, and recommendation systems. In general, CNNs work well on tasks where the data/input features have some level of spatial relationship.



Source: http://yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf

Your optimized CUDA implementation of the convolutional layer will be used to perform inference for layers C1 and C3.

We **can** use mini-dnn-cpp (Mini-DNN) framework as a starting point to implement the modified version of LeNet-5.

We will use the Fashion MNIST dataset, where the inputs to the network will be a single channel images with dimensions of 28 x 28 pixels. The output layer consists of 10 nodes, where each node represents the likelihood of the input belonging to one of the 10 classes (T-shirt, dress, sneaker, boot etc.)

The overall learning objectives for this project are:

Demonstrating command of CUDA and optimization approaches by designing and implementing an optimized neural-network convolutional layer forward pass

2. Background knowledge

2.1. Convolution Neural Networks (CNNs)

- This video "How Convolutional Neural Networks work" give a very good explanation with respect to CNNs. You can also check other good materials about Neural Network here from the same authors.
- This Standford cheatsheet give an excellent explain on how CNNs works
- This "Dive into deep learning" book also give through explain on CNN.

2.2. Lenet-5

LeNet-5 is a simple and efficient convolutional neural network. You will LeNet-5 in this project.

- Here is a very good explain about LeNet-5 architecture.

We will not implement the original LeNet-5. We will use the newer, **modified version of LeNet-5**. Here is the **architecture** we use:

```
model.add(Conv2D(6, kernel_size=(5, 5), activation='relu', input_shape=(28, 28, 1)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(16, kernel_size=(5, 5), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(120, activation='relu'))
model.add(Dense(84, activation='relu'))
model.add(Dense(10, activation='softmax'))
```

Layer 1: 6 kernels with size 5×5, use ReLU activation function.
Layer 2: Max Pooling with size 2×2
Layer 3: 16 kernels with size 5×5, use ReLU activation function.
Layer 4: Max Pooling with size 2×2
Layer 5: Flatten
Layer 6: dense layer with 120 outputs, use ReLU activation function
Layer 7: dense layer with 84 outputs, use ReLU activation function
Layer 8: dense layer with 10 outputs, use softmax activation function

3. Starter project

We will using mini-dnn-cpp (Mini-DNN) framework as a starting point to implement the LeNet-5.

What is this framework?

mini-dnn-cpp is a C++ demo of deep neural networks. It is implemented purely in C++.

It has a complete implementation for CNN. You will need to modify this code for implementing the modified LeNet-5

4. Scope

What you need to do:

Implementing and optimizing the forward-pass of a convolutional layer using CUDA

- Run and explore the starter project on the classic MNIST.
- Make the host code: modify the code to change the network architecture to LeNet-5 and run on Fashion MNIST.
- Implement a basic GPU Convolution kernel.
  To do this you can read chapter 16 - Application case study: machine learning from "Programming Massively Parallel Processors A Hands-on Approach" 3rd edition[1]
- Optimize your GPU Convolution kernel. Some ideas to optimize:
  - Tiled shared memory convolution
  - Shared memory matrix multiplication and input matrix unrolling
  - Kernel fusion for unrolling and matrix-multiplication
  - Weight matrix (kernel values) in constant memory
  - Tuning with restrict and loop unrolling
  - Sweeping various parameters to find best values (block sizes, amount of thread coarsening)
  - Multiple kernel implementations for different layer sizes
  - Input channel reduction: tree
  - Input channel reduction: atomics
  - Fixed point (FP16) arithmetic.
  - Using Streams to overlap computation with data transfer
  - An advanced matrix multiplication algorithm

What you DO NOT need to do:

- You do not need to write or optimize code of training/learning phase. The starter code already has this, you do not need to change. You only need to code the forward pass to make inference.
- You do not need to parallel other layers. You only need to do that on 2 Convolution layers.

Noted:

- You are free to use any source code for the learning phase. The starter project is just for reference.
- You only need to code the inference phase (forward pass).

[1] KIRK, David B.; WEN-MEI, W. Hwu. *Programming massively parallel processors: a hands-on approach.* Morgan kaufmann, 2016.

5. Submission

Report

As you implement your optimizations, you are required to document their effect on performance. Create a document and describe each optimization you implemented, including why you selected this optimization, show your output result, and the output of each layer's timing with this optimization. Describe in detail how you implemented the optimizations.

This should be write on *.ipynb notebook.

You will need to submit:

- Team plan and work distribution file.
- Notebook reports
- All source code file and an instruction file on how to set up and run your project.
- A presentation video about 15-20min. Upload on YouTube with Unlisted option. And give link on your report or readme file. (DO NOT Submit video on Moodle)