Project Documentation: Convolutional Neural Network Training with CUDA

1. Introduction

1.1 Overview

This project implements a Convolutional Neural Network (CNN) using CUDA for parallel processing. The neural network is trained and tested on the MNIST dataset for handwritten digit recognition.

1.2 Goals

- Implement a CNN with CUDA for efficient training on GPU.
- Achieve high accuracy on the MNIST dataset.
- Provide a modular and extensible codebase.

1.3 Scope

This project focuses on training and testing a CNN for handwritten digit recognition using CUDA. It does not cover real-time applications or deployment considerations.

2. Getting Started

2.1 Prerequisites

- CUDA-enabled GPU
- CUDA Toolkit (version 11.7)
- C++ compiler with C++11 support

2.2 Installation

Linux:

1. Install the CUDA Toolkit following the instructions on the official CUDA website.
2. Run the “make” command to compile the code. An executable called CNN is created.
3. Run the “make run” command to run the executable.

3. Project Structure

3.1 Source Files

- **main.cu**: Main entry point of the program where the training and testing code is written
- **layers.cu:** The layers file contains the layer class and all the GPU kernels needed to train the neural network.

3.2 Header Files

- **mnist.h**: Header file for MNIST dataset loading.
- **layer.h**: Header file containing the definition of the neural network layers.

3.3 Data Files

- **data/train-images.idx3-ubyte**: Training images for MNIST.
- **data/train-labels.idx1-ubyte**: Training labels for MNIST.
- **data/t10k-images.idx3-ubyte**: Test images for MNIST.
- **data/t10k-labels.idx1-ubyte**: Test labels for MNIST.

4. Dependencies

4.1 CUDA Toolkit

The project depends on the CUDA Toolkit for GPU acceleration.

4.2 MNIST Loader

The MNIST dataset loader is used to load training and testing data.

5. Architecture Overview

5.1 Neural Network Layers

The neural network architecture is defined with the following layers:

- **Input Layer (l_input):**
    - Shape: 28 x 28 neurons
    - Purpose: Accepts the input data, which is a 28 x 28 matrix representing an image.

- **Convolutional Layer (l_c1):**
    - Filter Size: 5 x 5
    - Number of Filters: 6
    - Output Shape: 24 x 24 x 6 neurons
    - Purpose: Applies convolutional operations to capture features in the input image.
- **Subsampling Layer (l_s1):**
    - Subsampling Factor: 4 x 4
    - Number of Channels: 6
    - Output Shape: 6 x 6 x 6 neurons
    - Purpose: Performs subsampling to reduce dimensionality and retain important features.
- **Fully Connected Layer (l_f):**
    - Output Size: 10 neurons
    - Purpose: Generates the final output, representing the class probabilities for classification.

5.2 Forward Propagation

The forward propagation process involves passing the input data through the defined layers in the neural network. The steps are as follows:

1. **Input Layer:**
    - The input data, a 28 x 28 matrix, is fed into the input layer (l_input).
2. **Convolutional Layer (l_c1):**
    - Convolution operations are applied to the input using 5 x 5 filters.
    - Bias is added, and a step function is applied to produce the output.
3. **Subsampling Layer (l_s1):**
    - Subsampling is performed on the output of the convolutional layer to reduce dimensionality.
4. **Fully Connected Layer (l_f):**
    - The output of the subsampling layer is flattened and fed into the fully connected layer.
    - Bias is added, and a step function is applied to produce the final output.

5.3 Back Propagation

The backpropagation process is responsible for updating the weights of the neural network to minimize the error. It consists of the following steps:

1. **Fully Connected Layer (l_f):**
    - Compute the error and update weights using backpropagation.
2. **Subsampling Layer (l_s1):**
    - Backpropagate the error to the subsampling layer and update weights.
3. **Convolutional Layer (l_c1):**
    - Backpropagate the error to the convolutional layer and update weights.
4. **Apply Gradients:**
    - Apply the calculated gradients to update the weights of each layer.

6. Performance Metrics

6.1 Training Time

It takes around 5 minutes to train for 50 epochs in MNIST training data.

6.2 Accuracy

An accuracy of 96.9% was seen in MNIST testing data.

7. Future Enhancements

1. **Model Architecture:**
    - Experiment with more complex model architectures, including deeper networks or different types of layers (e.g., residual networks) to capture more intricate patterns in the data.
2. **Hyperparameter Tuning:**
    - Conduct a systematic search for optimal hyperparameters (learning rate, batch size, etc.) to enhance the model's performance.