PROJECT REPORT

cuDNN implementation of CNN and comparison with CPU

Kapish Garg Shamali Shinde
862128474 862188568
kgarg003@ucr.edu sshin081@ucr.edu

Project Description

Our project focuses on cuDNN implementation on CNN and comparing the performance with a CPU implementation. Convolutional Neural Network is a Deep Learning algorithm that takes in an input image and assigns weights and biases to important factors to distinguish the elements of the image. CNN uses a network of neurons to predict the image based on the training data. The larger the CNN network the more accurate it is supposed to achieve including some other factors too such as input data size, optimizer being used.

CNN's performance relies on the way the network calculations are done. If they are done in parallel, the output is faster to achieve. This is where cuDNN implementation comes in. The primary goal of this project is a discrete analysis of CNN implementation using CUDA and comparing it with the performance of a similar code on a CPU.

Dataset

The dataset used for this project is the MNIST dataset which contains images of numeric digits from 0 to 9. The dataset is downloaded from MNIST repository in the form of ubyte files containing 4 files. 2 for training and 2 for testing images and their labels respectively. The Dataset comprises 60k training images,labels and 10k testing images and labels.

Implementation

The implementation of CNN involves building a small network of 3 layers. The weights and biases are initialized at random at the beginning of the first layer. The forward pass involves doing calculations based on these initial weights and then computing the error/ loss at the end by comparing it with the ground truth image. The loss is then adjusted with the weights on backward propagation. The implementation is pretty much standard, similar to any CNN state of the art.

Languages used: C, C++, CUDA Libraries

Technical Description

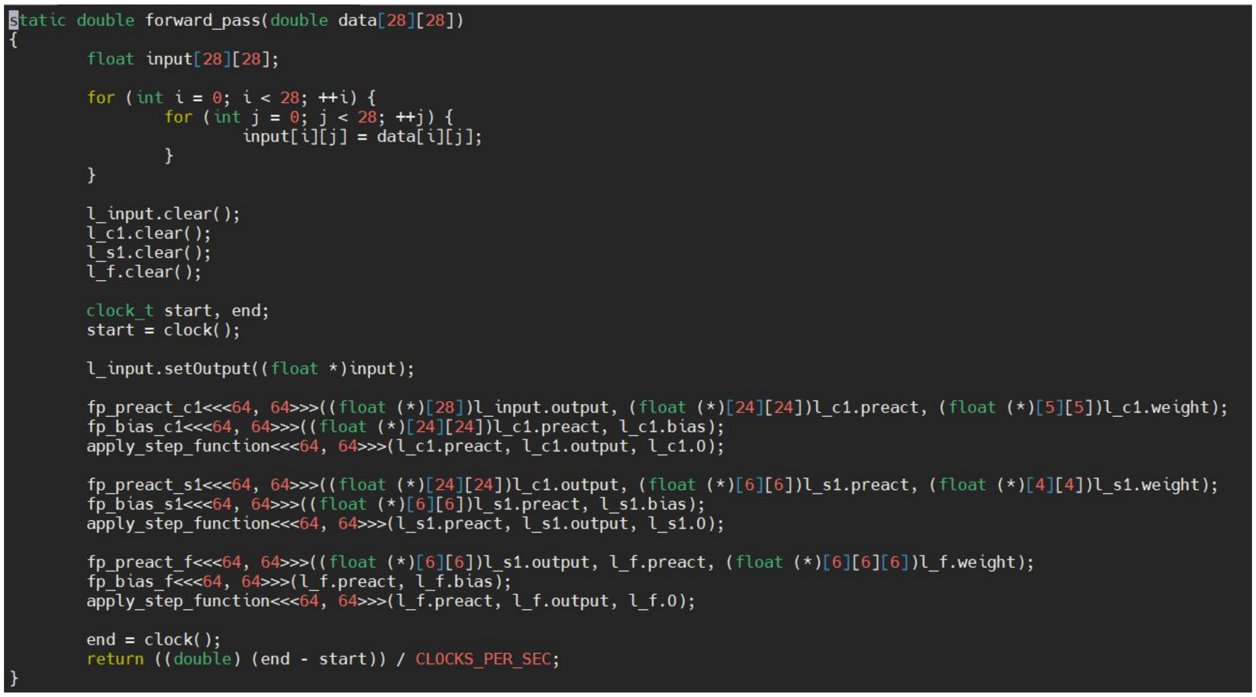
The folder structure of our project includes-

- /data (Folder containing MNIST dataset)
- /include (Folder containing mnist and neural network header files for CPU implementation)
- mnist .c (CPU implementation file)
- mnist_file.c (CPU implementation file)
- neural_network.c (CPU implementation file)
- layer.cu (GPU implementation file)
- layer.h (GPU implementation file)
- main.cu (GPU implementation file)
- mnist.h (GPU implementation file)

The algorithmic implementation for construction CNN is the same for both GPU and CNN in order to have a fair comparison. The network contains 3 layers where 2 hidden layers and one input layer.

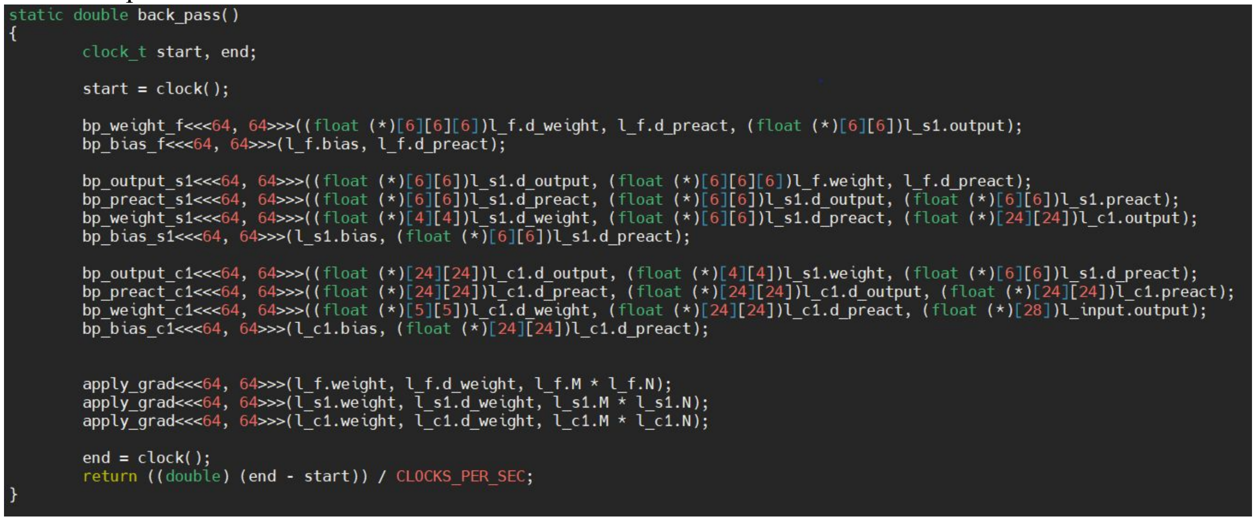
With GPU we get the advantage of parallelising the code, thus we implemented the forward propagation and backward propagation in a parallel method where the gradient calculations and weight/biases calculations are done in CUDA. Following is the snapshot of the code being parallelised.

Forward pass-



```
static double forward_pass(double data[28][28])
{
	float input[28][28];

	for (int i = 0; i < 28; ++i) {
		for (int j = 0; j < 28; ++j) {
			input[i][j] = data[i][j];
		}
	}

	l_input.clear();
	l_c1.clear();
	l_s1.clear();
	l_f.clear();

	clock_t start, end;
	start = clock();

	l_input.setOutput((float *)input);

	fp_preact_c1<<<64, 64>>>((float (*)[28])l_input.output, (float (*)[24][24])l_c1.preact, (float (*)[5][5])l_c1.weight);
	fp_bias_c1<<<64, 64>>>((float (*)[24][24])l_c1.preact, l_c1.bias);
	apply_step_function<<<64, 64>>>(l_c1.preact, l_c1.output, l_c1.O);

	fp_preact_s1<<<64, 64>>>((float (*)[24][24])l_c1.output, (float (*)[6][6])l_s1.preact, (float (*)[4][4])l_s1.weight);
	fp_bias_s1<<<64, 64>>>((float (*)[6][6])l_s1.preact, l_s1.bias);
	apply_step_function<<<64, 64>>>(l_s1.preact, l_s1.output, l_s1.O);

	fp_preact_f<<<64, 64>>>((float (*)[6][6])l_s1.output, l_f.preact, (float (*)[6][6][6])l_f.weight);
	fp_bias_f<<<64, 64>>>(l_f.preact, l_f.bias);
	apply_step_function<<<64, 64>>>(l_f.preact, l_f.output, l_f.O);

	end = clock();
	return ((double) (end - start)) / CLOCKS_PER_SEC;
}
```

Backward pass-



```
static double back_pass()
{
	clock_t start, end;

	start = clock();

	bp_weight_f<<<64, 64>>>((float (*)[6][6][6])l_f.d_weight, l_f.d_preact, (float (*)[6][6])l_s1.output);
	bp_bias_f<<<64, 64>>>(l_f.bias, l_f.d_preact);

	bp_output_s1<<<64, 64>>>((float (*)[6][6])l_s1.d_output, (float (*)[6][6][6])l_f.weight, l_f.d_preact);
	bp_preact_s1<<<64, 64>>>((float (*)[6][6])l_s1.d_preact, (float (*)[6][6])l_s1.d_output, (float (*)[6][6])l_s1.preact);
	bp_weight_s1<<<64, 64>>>((float (*)[4][4])l_s1.d_weight, (float (*)[6][6])l_s1.d_preact, (float (*)[24][24])l_c1.output);
	bp_bias_s1<<<64, 64>>>(l_s1.bias, (float (*)[6][6])l_s1.d_preact);

	bp_output_c1<<<64, 64>>>((float (*)[24][24])l_c1.d_output, (float (*)[4][4])l_s1.weight, (float (*)[6][6])l_s1.d_preact);
	bp_preact_c1<<<64, 64>>>((float (*)[24][24])l_c1.d_preact, (float (*)[24][24])l_c1.d_output, (float (*)[24][24])l_c1.preact);
	bp_weight_c1<<<64, 64>>>((float (*)[5][5])l_c1.d_weight, (float (*)[24][24])l_c1.d_preact, (float (*)[28])l_input.output);
	bp_bias_c1<<<64, 64>>>(l_c1.bias, (float (*)[24][24])l_c1.d_preact);


	apply_grad<<<64, 64>>>(l_f.weight, l_f.d_weight, l_f.M * l_f.N);
	apply_grad<<<64, 64>>>(l_s1.weight, l_s1.d_weight, l_s1.M * l_s1.N);
	apply_grad<<<64, 64>>>(l_c1.weight, l_c1.d_weight, l_c1.M * l_c1.N);

	end = clock();
	return ((double) (end - start)) / CLOCKS_PER_SEC;
}
```

Whereas, the CPU implementation runs the code in serial manner calculating the gradient after each layer in series. The only difference between the GPU implementation and CPU implementation is the call to Cuda in GPU and feeding the weights and biases.

Status of Project

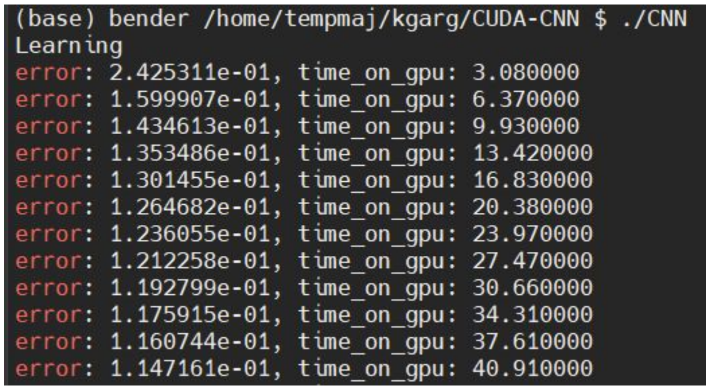
The project is complete and runs on Bender as expected.

Challenges-

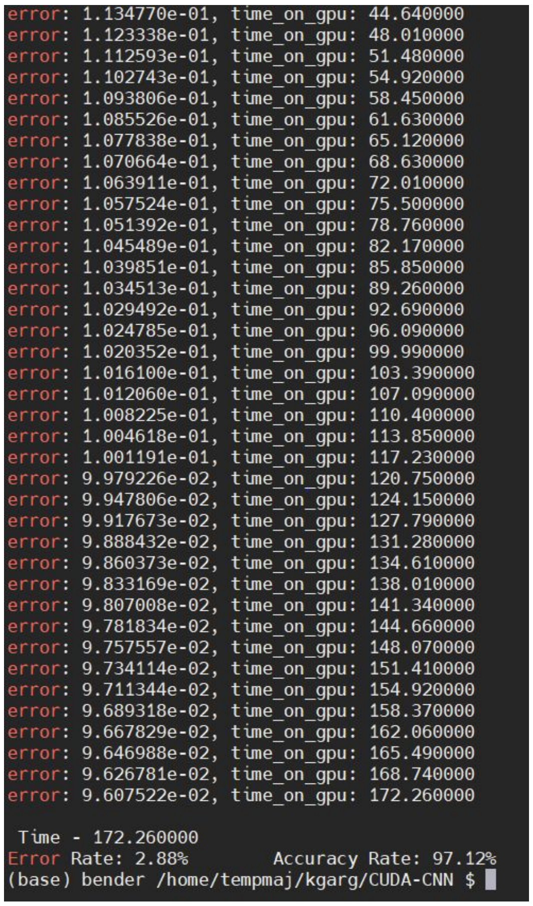
The challenge which appears to be is during the implementation of the cuDNN version of CNN. The number of layers is a big factor in determining the performance of the network. If the number of layers is to be increased it might be a coding overhead. CPU takes a lot of time if the number of layers are increased, making it hard for evaluating a big neural network's performance.

Evaluation

Following is the screenshot of GPU code running-

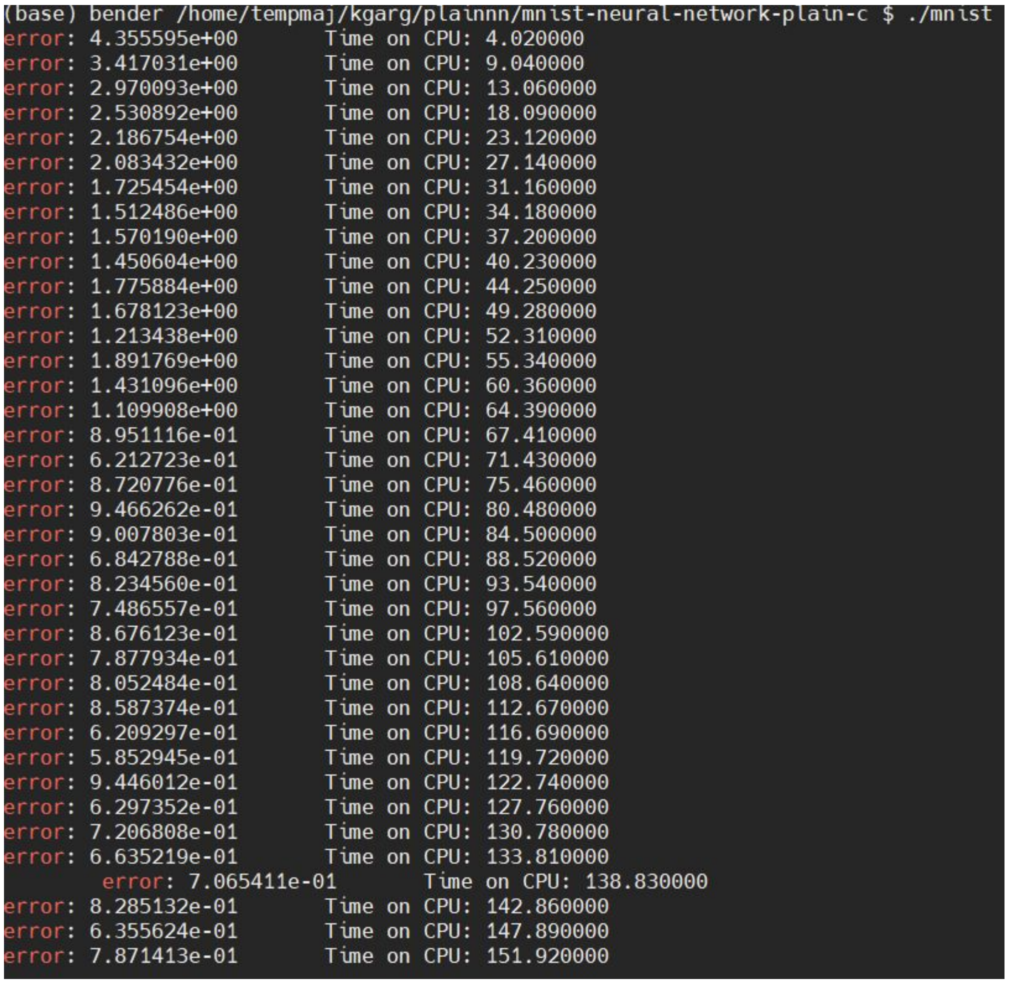


```
(base) bender /home/tempmaj/kgarg/CUDA-CNN $ ./CNN
Learning
error: 2.425311e-01, time_on_gpu: 3.080000
error: 1.599907e-01, time_on_gpu: 6.370000
error: 1.434613e-01, time_on_gpu: 9.930000
error: 1.353486e-01, time_on_gpu: 13.420000
error: 1.301455e-01, time_on_gpu: 16.830000
error: 1.264682e-01, time_on_gpu: 20.380000
error: 1.236055e-01, time_on_gpu: 23.970000
error: 1.212258e-01, time_on_gpu: 27.470000
error: 1.192799e-01, time_on_gpu: 30.660000
error: 1.175915e-01, time_on_gpu: 34.310000
error: 1.160744e-01, time_on_gpu: 37.610000
error: 1.147161e-01, time_on_gpu: 40.910000
```

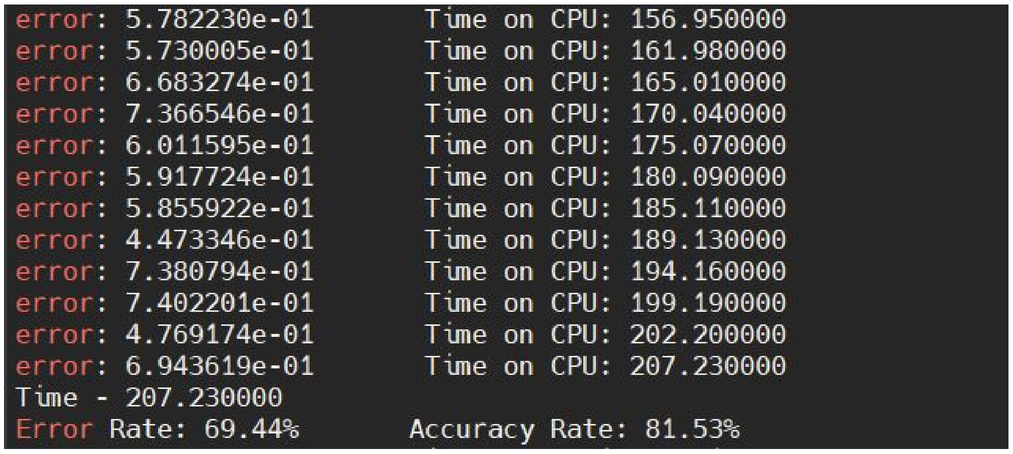


```
error: 1.134770e-01, time_on_gpu: 44.640000
error: 1.123338e-01, time_on_gpu: 48.010000
error: 1.112593e-01, time_on_gpu: 51.480000
error: 1.102743e-01, time_on_gpu: 54.920000
error: 1.093806e-01, time_on_gpu: 58.450000
error: 1.085526e-01, time_on_gpu: 61.630000
error: 1.077838e-01, time_on_gpu: 65.120000
error: 1.070664e-01, time_on_gpu: 68.630000
error: 1.063911e-01, time_on_gpu: 72.010000
error: 1.057524e-01, time_on_gpu: 75.500000
error: 1.051392e-01, time_on_gpu: 78.760000
error: 1.045489e-01, time_on_gpu: 82.170000
error: 1.039851e-01, time_on_gpu: 85.850000
error: 1.034513e-01, time_on_gpu: 89.260000
error: 1.029492e-01, time_on_gpu: 92.690000
error: 1.024785e-01, time_on_gpu: 96.090000
error: 1.020352e-01, time_on_gpu: 99.990000
error: 1.016100e-01, time_on_gpu: 103.390000
error: 1.012060e-01, time_on_gpu: 107.090000
error: 1.008225e-01, time_on_gpu: 110.400000
error: 1.004618e-01, time_on_gpu: 113.850000
error: 1.001191e-01, time_on_gpu: 117.230000
error: 9.979226e-02, time_on_gpu: 120.750000
error: 9.947806e-02, time_on_gpu: 124.150000
error: 9.917673e-02, time_on_gpu: 127.790000
error: 9.888432e-02, time_on_gpu: 131.280000
error: 9.860373e-02, time_on_gpu: 134.610000
error: 9.833169e-02, time_on_gpu: 138.010000
error: 9.807008e-02, time_on_gpu: 141.340000
error: 9.781834e-02, time_on_gpu: 144.660000
error: 9.757557e-02, time_on_gpu: 148.070000
error: 9.734114e-02, time_on_gpu: 151.410000
error: 9.711344e-02, time_on_gpu: 154.920000
error: 9.689318e-02, time_on_gpu: 158.370000
error: 9.667829e-02, time_on_gpu: 162.060000
error: 9.646988e-02, time_on_gpu: 165.490000
error: 9.626781e-02, time_on_gpu: 168.740000
error: 9.607522e-02, time_on_gpu: 172.260000

 Time - 172.260000
Error Rate: 2.88%        Accuracy Rate: 97.12%
(base) bender /home/tempmaj/kgarg/CUDA-CNN $
```

Following is the screenshot of CPU code running-



```
(base) bender /home/tempmaj/kgarg/plainnn/mnist-neural-network-plain-c $ ./mnist
error: 4.355595e+00       Time on CPU: 4.020000
error: 3.417031e+00       Time on CPU: 9.040000
error: 2.970093e+00       Time on CPU: 13.060000
error: 2.530892e+00       Time on CPU: 18.090000
error: 2.186754e+00       Time on CPU: 23.120000
error: 2.083432e+00       Time on CPU: 27.140000
error: 1.725454e+00       Time on CPU: 31.160000
error: 1.512486e+00       Time on CPU: 34.180000
error: 1.570190e+00       Time on CPU: 37.200000
error: 1.450604e+00       Time on CPU: 40.230000
error: 1.775884e+00       Time on CPU: 44.250000
error: 1.678123e+00       Time on CPU: 49.280000
error: 1.213438e+00       Time on CPU: 52.310000
error: 1.891769e+00       Time on CPU: 55.340000
error: 1.431096e+00       Time on CPU: 60.360000
error: 1.109908e+00       Time on CPU: 64.390000
error: 8.951116e-01       Time on CPU: 67.410000
error: 6.212723e-01       Time on CPU: 71.430000
error: 8.720776e-01       Time on CPU: 75.460000
error: 9.466262e-01       Time on CPU: 80.480000
error: 9.007803e-01       Time on CPU: 84.500000
error: 6.842788e-01       Time on CPU: 88.520000
error: 8.234560e-01       Time on CPU: 93.540000
error: 7.486557e-01       Time on CPU: 97.560000
error: 8.676123e-01       Time on CPU: 102.590000
error: 7.877934e-01       Time on CPU: 105.610000
error: 8.052484e-01       Time on CPU: 108.640000
error: 8.587374e-01       Time on CPU: 112.670000
error: 6.209297e-01       Time on CPU: 116.690000
error: 5.852945e-01       Time on CPU: 119.720000
error: 9.446012e-01       Time on CPU: 122.740000
error: 6.297352e-01       Time on CPU: 127.760000
error: 7.206808e-01       Time on CPU: 130.780000
error: 6.635219e-01       Time on CPU: 133.810000
        error: 7.065411e-01       Time on CPU: 138.830000
error: 8.285132e-01       Time on CPU: 142.860000
error: 6.355624e-01       Time on CPU: 147.890000
error: 7.871413e-01       Time on CPU: 151.920000
```



```
error: 5.782230e-01       Time on CPU: 156.950000
error: 5.730005e-01       Time on CPU: 161.980000
error: 6.683274e-01       Time on CPU: 165.010000
error: 7.366546e-01       Time on CPU: 170.040000
error: 6.011595e-01       Time on CPU: 175.070000
error: 5.917724e-01       Time on CPU: 180.090000
error: 5.855922e-01       Time on CPU: 185.110000
error: 4.473346e-01       Time on CPU: 189.130000
error: 7.380794e-01       Time on CPU: 194.160000
error: 7.402201e-01       Time on CPU: 199.190000
error: 4.769174e-01       Time on CPU: 202.200000
error: 6.943619e-01       Time on CPU: 207.230000
Time - 207.230000
Error Rate: 69.44%       Accuracy Rate: 81.53%
```

Both CNNs are run for 50 epochs on the same number of layers with initial weights randomised in both the cases.

Results

	Time taken	Accuracy
CPU	207.23 seconds	81.53%
GPU	172.26 seconds	97.12%

GPU provides higher accuracy in less time as compared to CPU implementation. The only surprising result was the time taken by GPU which is closer than time taken by CPU. This might be the bottleneck of using a smaller neural network. The project could provide a greater insight and better results for comparison if we could implement a bigger network running for more number of epochs.

Steps to run

1. Download the zip file CuDNN and extract all the files.
2. Run make command inside CuDNN folder. This will generate two files, CNN and mnist
3. Run - ./CNN for GPU implementation
4. Run- ./mnist for CPU implementation

Both make files are in the folder so you can directly run them if you don't want to run the make command.

Work Distribution

The task of this project involves three parts

1. cuDNN implementation

    Forward propagation- Shamali Shinde
    Backward propagation- Kapish Garg
    Fine-tuning- Kapish Garg

2. CPU implementation-

    Forward propagation- Kapish Garg
    Backward propagation- Shamali Shinde
    Fine-tuning- Shamali Shinde

3. Performance analysis-

    Time difference- Shamali Shinde
    Accuracy- Kapish Garg