Our proposed project is to take an existing repository for a neural network implemented in C++, and convert the forward propagation portion to use CUDA parallel computing to decrease inference times. This project intends to use a GitHub repository for a “from-scratch” neural network implemented by Kai Han1. This implementation includes a compact neural network that is intended to be trained on the MNIST data set. This is a widely used data set that is easily accessible, and relatively small, being only 21.00 MiB in size. The purpose of choosing such is simple network architecture and data set is so that the majority of the focus can be placed on implementing CUDA parallel computing to decrease the inference times of the network. In Kai’s implementation, each layer in the network contains two mandatory functions, the forward() method for inference, and the backward() method for back propagation. This project aims to decrease inference times, as such, each layer’s forward() method will be re-written to take advantage of parallel computing. For example, the existing implementation of the forward() function for convolutional layers uses nested for loops to iterate over the 2D images as well as the 2D kernel that convolves about the image. CUDA can be implemented in this function to remove the need for looping over convolution iterations, as each one of these computations are independent calculations. This would allow for multiple threads to concurrently calculate the output image, as opposed to calculating the output image in a serial manner. In addition, convolutional computations can benefit from tiling, where CUDA shared memory is utilized to reduce the amount of redundant global memory reads, decreases the total execution time. Another layer that is extremely parallelizable is the fully connected layers. These layers use matrix multiplication to calculate the forward output for each layer. Again, tiling can be used in this layer to both increase the number of threads computing the output concurrently, and to also cut down on expensive global memory reads. Overall, the majority of layers in this repository perform their forward pass by performing a loop where the same instruction is executed on each data input. This is an example of single instruction multiple data, a domain where CUDA parallel computing can greatly improve computation times. As such, this projects aim is to use CUDA to remove as much serial code as possible, and replace it with kernels that perform calculations concurrently. In addition, where common data is repeatedly accessed across computations, shared and constant memory will be leveraged to decrease costly global memory accesses. Once all these CUDA optimizations are implemented, the previous CPU-based implementation will be benchmarked against this project’s GPU-based implementation in order to evaluate the speed-ups achieved.

## References:

1. Han, Kai. “IAMHANKAI/Mini-Dnn-Cpp: C++ Demo of Deep Neural Networks (MLP, CNN).” GitHub, https://github.com/iamhankai/mini-dnn-cpp.