Marshall Lanning

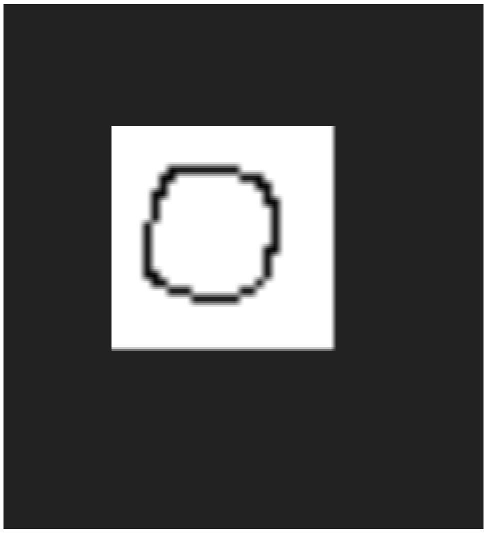
CECS 625 Parallel Programming Extra-cedit Project
October 29, 2019 (50 points)

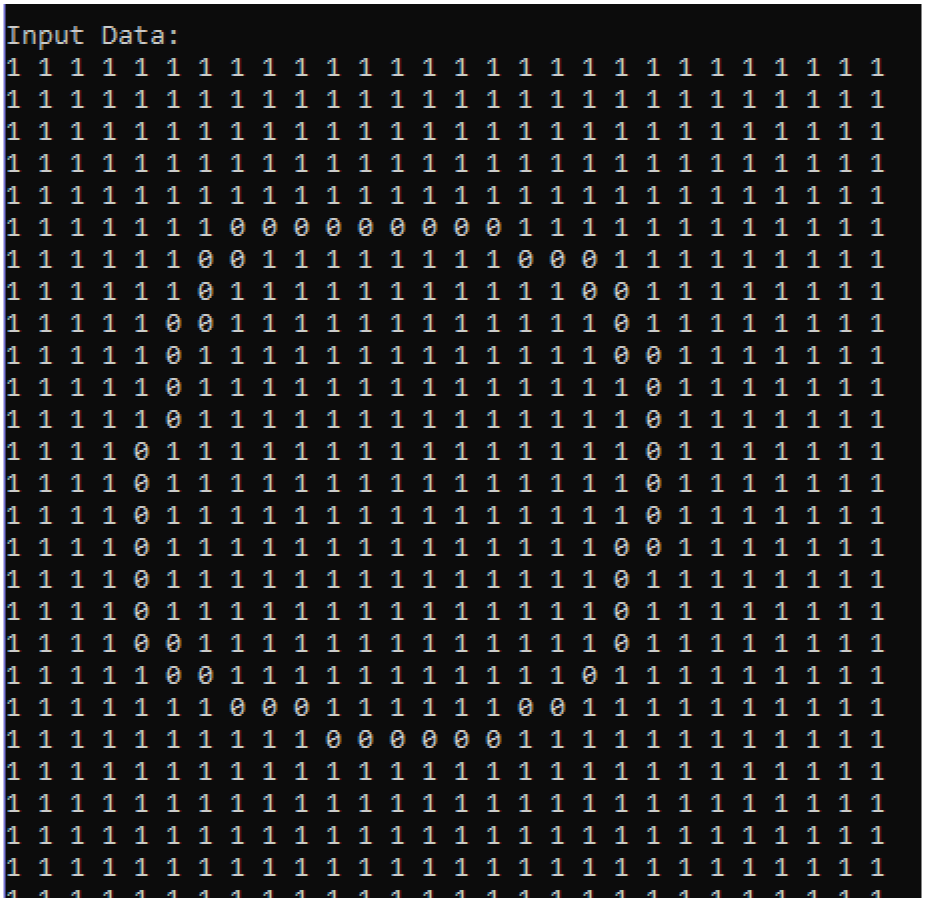
Due: November 7 (Wed) Demo your extra-credit project in class.

Based on the materials you learned from Section 6.3 and 6.5, write a C++ project to implement a written digit classifier as described below:

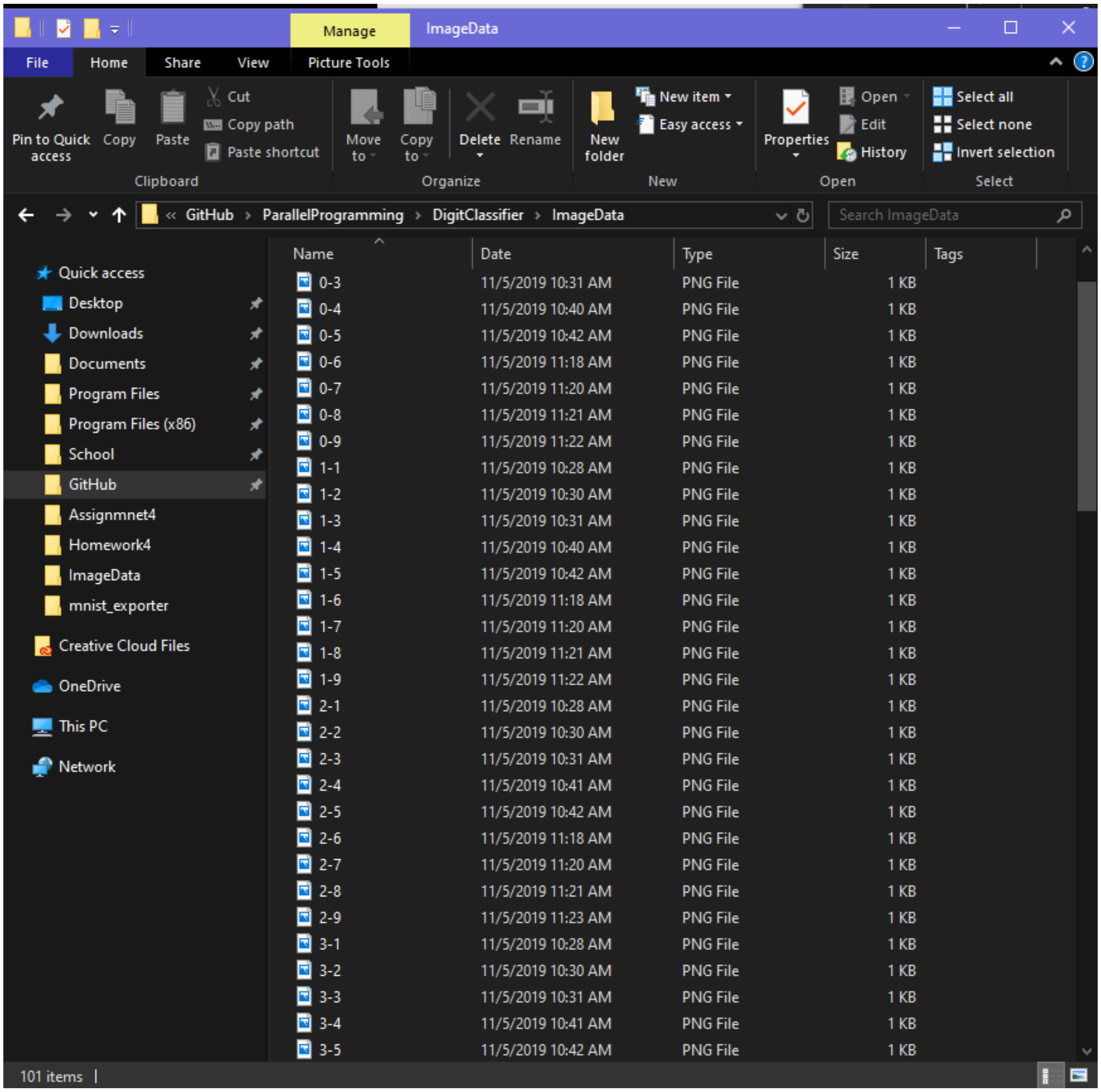
Input: An 28x28 float vector (i.e. an 28X28 grey-level image representin a written digit from 0-9)

I used the Windows Paint application to write out 10 of each number in a 28X28 pixel fram so 100 numbers total. An example of 0 is below:





```
Input Data:
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 1 0 0 0 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 1
1 1 1 1 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1
1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1
1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1
1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1
1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1
1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1
1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1
1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1
1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1
1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1
1 1 1 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1
1 1 1 1 1 0 0 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 0 0 0 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

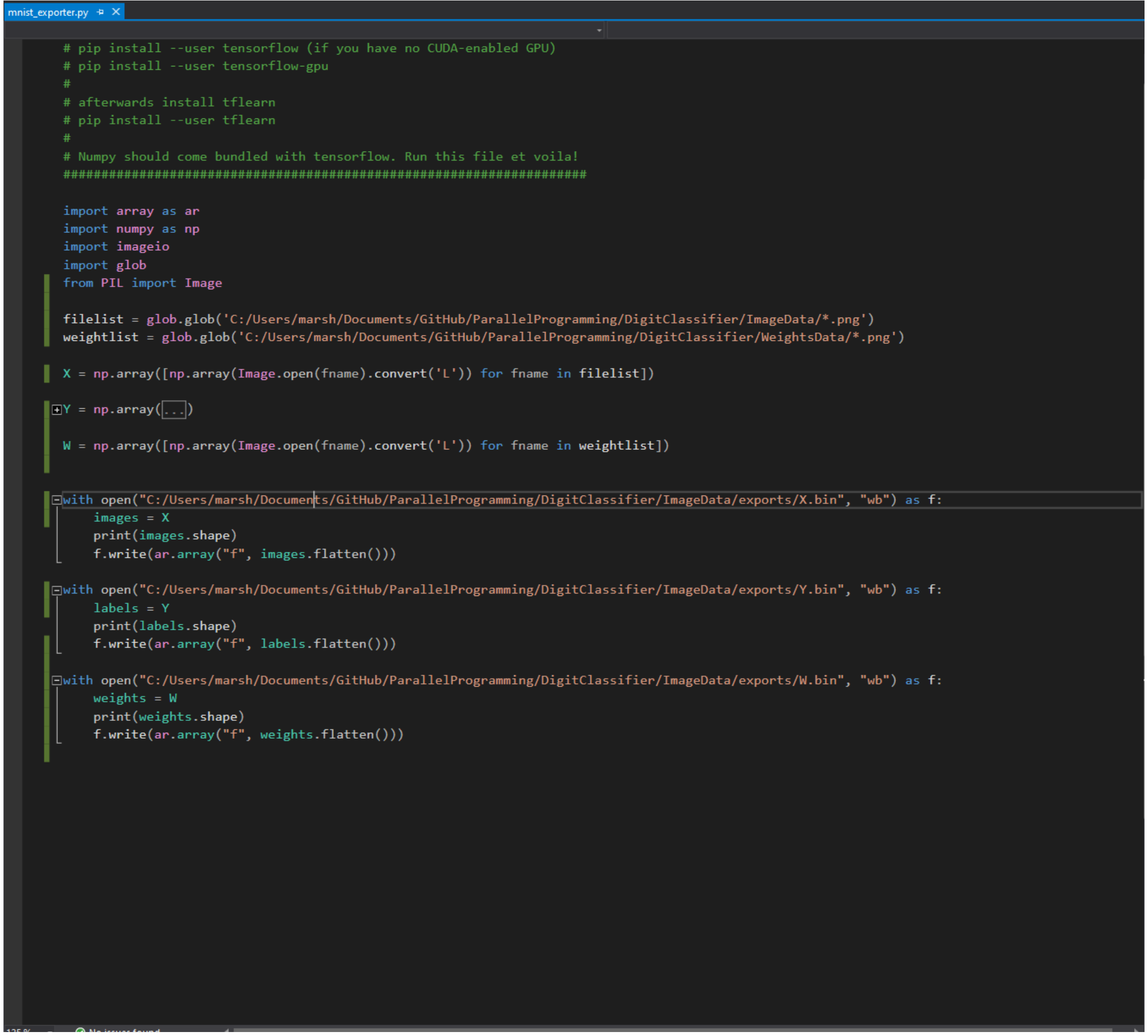


« GitHub › ParallelProgramming › DigitClassifier › ImageData

Name	Date	Type	Size	Tags
0-3	11/5/2019 10:31 AM	PNG File	1 KB	
0-4	11/5/2019 10:40 AM	PNG File	1 KB	
0-5	11/5/2019 10:42 AM	PNG File	1 KB	
0-6	11/5/2019 11:18 AM	PNG File	1 KB	
0-7	11/5/2019 11:20 AM	PNG File	1 KB	
0-8	11/5/2019 11:21 AM	PNG File	1 KB	
0-9	11/5/2019 11:22 AM	PNG File	1 KB	
1-1	11/5/2019 10:28 AM	PNG File	1 KB	
1-2	11/5/2019 10:30 AM	PNG File	1 KB	
1-3	11/5/2019 10:31 AM	PNG File	1 KB	
1-4	11/5/2019 10:40 AM	PNG File	1 KB	
1-5	11/5/2019 10:42 AM	PNG File	1 KB	
1-6	11/5/2019 11:18 AM	PNG File	1 KB	
1-7	11/5/2019 11:20 AM	PNG File	1 KB	
1-8	11/5/2019 11:21 AM	PNG File	1 KB	
1-9	11/5/2019 11:22 AM	PNG File	1 KB	
2-1	11/5/2019 10:28 AM	PNG File	1 KB	
2-2	11/5/2019 10:30 AM	PNG File	1 KB	
2-3	11/5/2019 10:31 AM	PNG File	1 KB	
2-4	11/5/2019 10:41 AM	PNG File	1 KB	
2-5	11/5/2019 10:42 AM	PNG File	1 KB	
2-6	11/5/2019 11:18 AM	PNG File	1 KB	
2-7	11/5/2019 11:20 AM	PNG File	1 KB	
2-8	11/5/2019 11:21 AM	PNG File	1 KB	
2-9	11/5/2019 11:23 AM	PNG File	1 KB	
3-1	11/5/2019 10:28 AM	PNG File	1 KB	
3-2	11/5/2019 10:30 AM	PNG File	1 KB	
3-3	11/5/2019 10:31 AM	PNG File	1 KB	
3-4	11/5/2019 10:41 AM	PNG File	1 KB	
3-5	11/5/2019 10:42 AM	PNG File	1 KB	

101 items

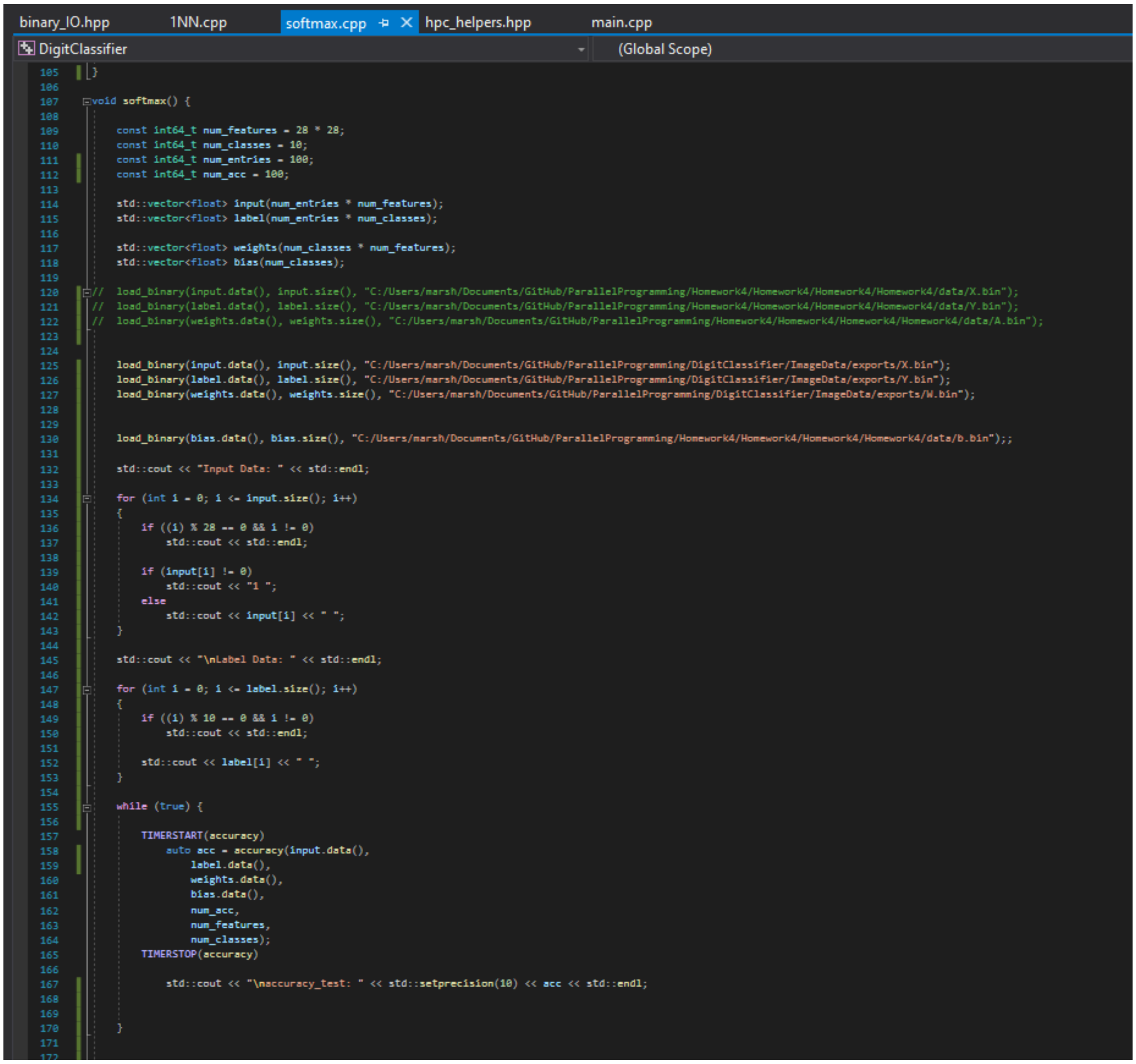
I used python like the mnist_exporter to export my data set correctly and make my own weights to compare against.



```python
# pip install --user tensorflow (if you have no CUDA-enabled GPU)
# pip install --user tensorflow-gpu
#
# afterwards install tflearn
# pip install --user tflearn
#
# Numpy should come bundled with tensorflow. Run this file et voila!
#####################################################################

import array as ar
import numpy as np
import imageio
import glob
from PIL import Image

filelist = glob.glob('C:/Users/marsh/Documents/GitHub/ParallelProgramming/DigitClassifier/ImageData/*.png')
weightlist = glob.glob('C:/Users/marsh/Documents/GitHub/ParallelProgramming/DigitClassifier/WeightsData/*.png')

X = np.array([np.array(Image.open(fname).convert('L')) for fname in filelist])

Y = np.array(...)

W = np.array([np.array(Image.open(fname).convert('L')) for fname in weightlist])


with open("C:/Users/marsh/Documents/GitHub/ParallelProgramming/DigitClassifier/ImageData/exports/X.bin", "wb") as f:
    images = X
    print(images.shape)
    f.write(ar.array("f", images.flatten()))

with open("C:/Users/marsh/Documents/GitHub/ParallelProgramming/DigitClassifier/ImageData/exports/Y.bin", "wb") as f:
    labels = Y
    print(labels.shape)
    f.write(ar.array("f", labels.flatten()))

with open("C:/Users/marsh/Documents/GitHub/ParallelProgramming/DigitClassifier/ImageData/exports/W.bin", "wb") as f:
    weights = W
    print(weights.shape)
    f.write(ar.array("f", weights.flatten()))
```

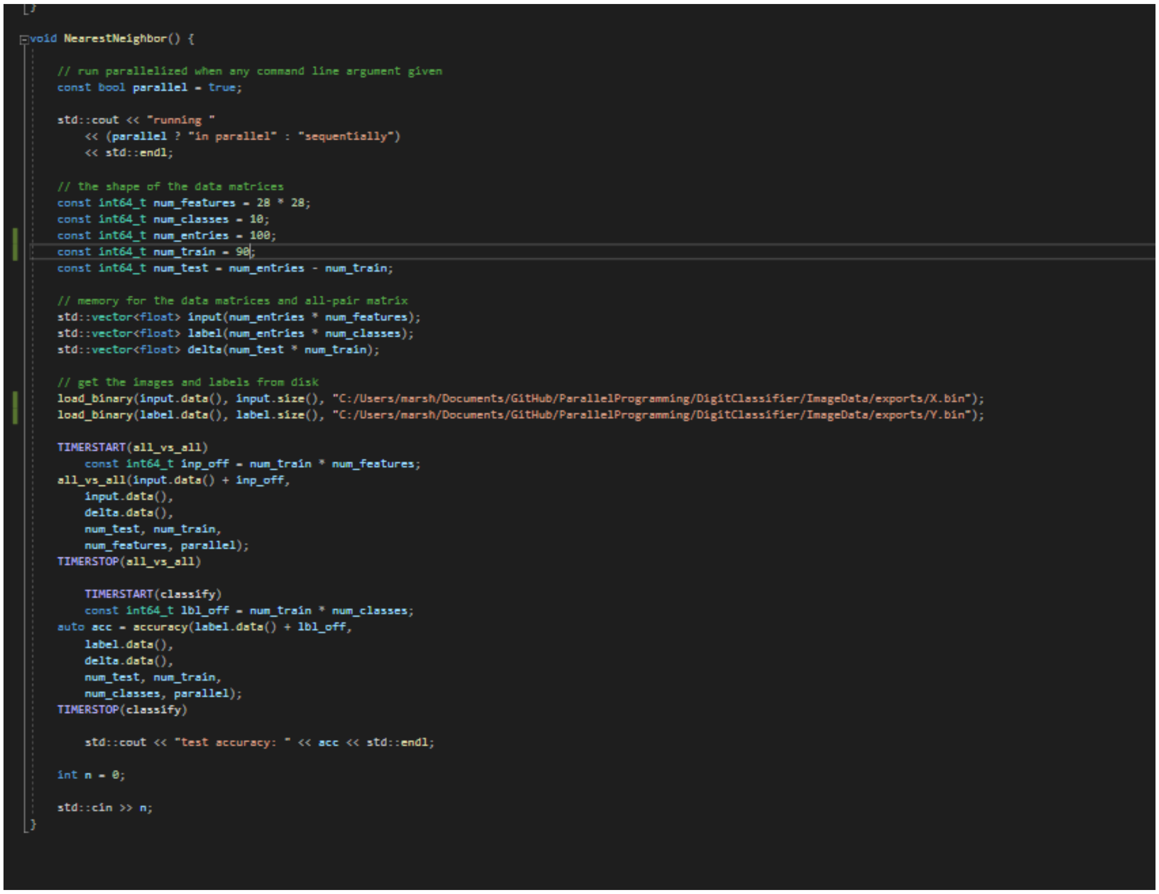
Output: The digit in the image (i.e. 0, 1, 2, 3, 4, 5, 6, 7, 8, or 9)

Softmax implementation:



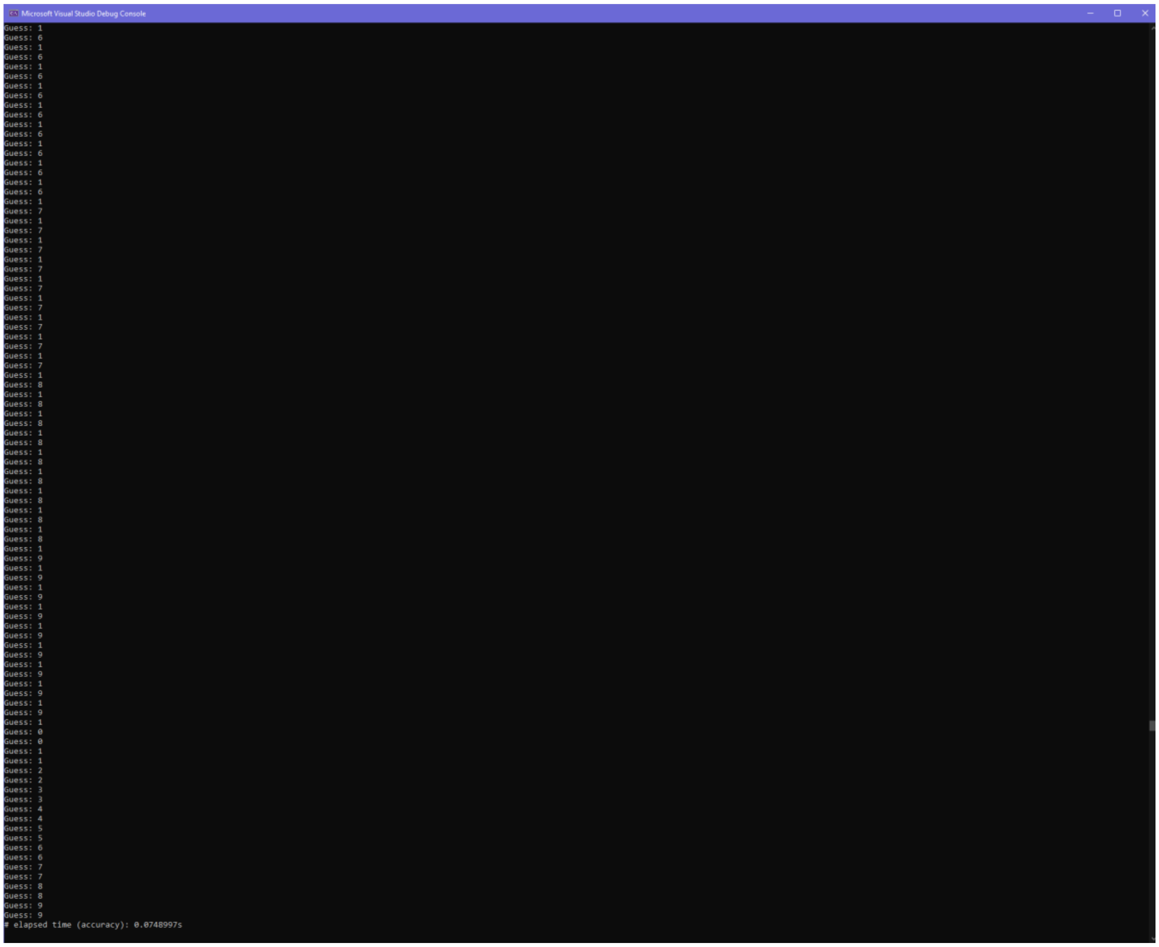
```cpp
}

void softmax() {

    const int64_t num_features = 28 * 28;
    const int64_t num_classes = 10;
    const int64_t num_entries = 100;
    const int64_t num_acc = 100;

    std::vector<float> input(num_entries * num_features);
    std::vector<float> label(num_entries * num_classes);

    std::vector<float> weights(num_classes * num_features);
    std::vector<float> bias(num_classes);

//  load_binary(input.data(), input.size(), "C:/Users/marsh/Documents/GitHub/ParallelProgramming/Homework4/Homework4/Homework4/Homework4/data/X.bin");
//  load_binary(label.data(), label.size(), "C:/Users/marsh/Documents/GitHub/ParallelProgramming/Homework4/Homework4/Homework4/Homework4/data/Y.bin");
//  load_binary(weights.data(), weights.size(), "C:/Users/marsh/Documents/GitHub/ParallelProgramming/Homework4/Homework4/Homework4/Homework4/data/A.bin");


    load_binary(input.data(), input.size(), "C:/Users/marsh/Documents/GitHub/ParallelProgramming/DigitClassifier/ImageData/exports/X.bin");
    load_binary(label.data(), label.size(), "C:/Users/marsh/Documents/GitHub/ParallelProgramming/DigitClassifier/ImageData/exports/Y.bin");
    load_binary(weights.data(), weights.size(), "C:/Users/marsh/Documents/GitHub/ParallelProgramming/DigitClassifier/ImageData/exports/W.bin");


    load_binary(bias.data(), bias.size(), "C:/Users/marsh/Documents/GitHub/ParallelProgramming/Homework4/Homework4/Homework4/Homework4/data/b.bin");;

    std::cout << "Input Data: " << std::endl;

    for (int i = 0; i <= input.size(); i++)
    {
        if ((i) % 28 == 0 && i != 0)
            std::cout << std::endl;

        if (input[i] != 0)
            std::cout << "1 ";
        else
            std::cout << input[i] << " ";
    }

    std::cout << "\nLabel Data: " << std::endl;

    for (int i = 0; i <= label.size(); i++)
    {
        if ((i) % 10 == 0 && i != 0)
            std::cout << std::endl;

        std::cout << label[i] << " ";
    }

    while (true) {

        TIMERSTART(accuracy)
            auto acc = accuracy(input.data(),
                label.data(),
                weights.data(),
                bias.data(),
                num_acc,
                num_features,
                num_classes);
        TIMERSTOP(accuracy)

            std::cout << "\naccuracy_test: " << std::setprecision(10) << acc << std::endl;


    }

```

Nearest Neighbor Implementation:



```cpp
}

void NearestNeighbor() {

    // run parallelized when any command line argument given
    const bool parallel = true;

    std::cout << "running "
        << (parallel ? "in parallel" : "sequentially")
        << std::endl;

    // the shape of the data matrices
    const int64_t num_features = 28 * 28;
    const int64_t num_classes = 10;
    const int64_t num_entries = 100;
    const int64_t num_train = 90;
    const int64_t num_test = num_entries - num_train;

    // memory for the data matrices and all-pair matrix
    std::vector<float> input(num_entries * num_features);
    std::vector<float> label(num_entries * num_classes);
    std::vector<float> delta(num_test * num_train);

    // get the images and labels from disk
    load_binary(input.data(), input.size(), "C:/Users/marsh/Documents/GitHub/ParallelProgramming/DigitClassifier/ImageData/exports/X.bin");
    load_binary(label.data(), label.size(), "C:/Users/marsh/Documents/GitHub/ParallelProgramming/DigitClassifier/ImageData/exports/Y.bin");

    TIMERSTART(all_vs_all)
        const int64_t inp_off = num_train * num_features;
    all_vs_all(input.data() + inp_off,
        input.data(),
        delta.data(),
        num_test, num_train,
        num_features, parallel);
    TIMERSTOP(all_vs_all)

        TIMERSTART(classify)
        const int64_t lbl_off = num_train * num_classes;
    auto acc = accuracy(label.data() + lbl_off,
        label.data(),
        delta.data(),
        num_test, num_train,
        num_classes, parallel);
    TIMERSTOP(classify)

        std::cout << "test accuracy: " << acc << std::endl;

    int n = 0;

    std::cin >> n;
}
```

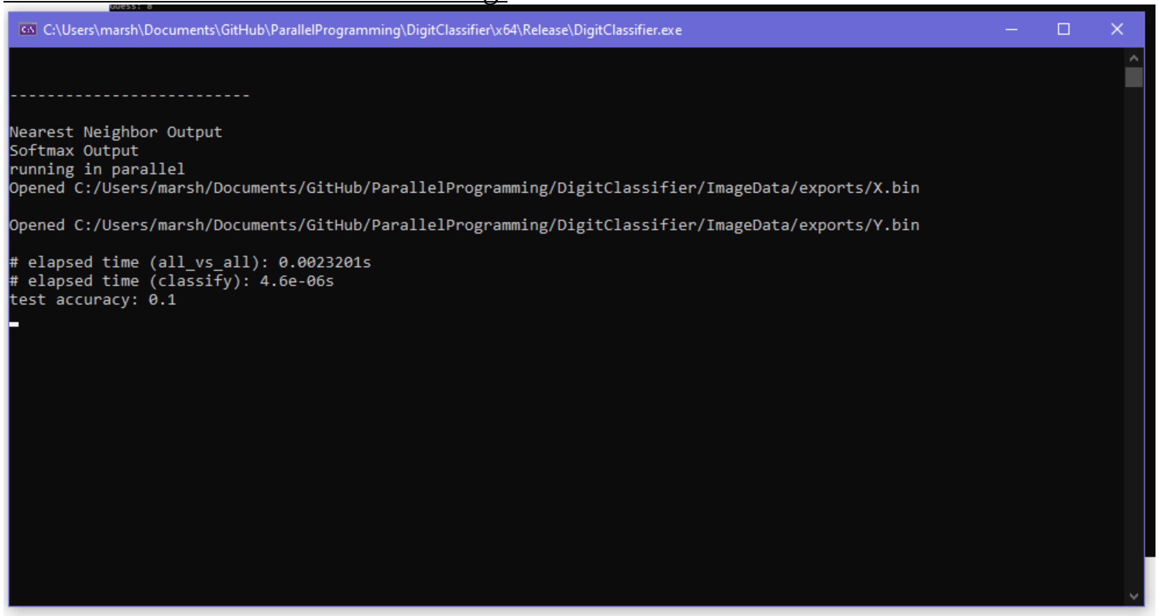
Output of Softmax guessing an iteration of my 100 numbers. The way the numbers are guessed seem somewhat accurate, the order that they are implemented in is 9 0’s, 9 1’s, 9 2’s, etc so you see the guesses increasing in value as it moves through the dataset.



```
Guess: 1
Guess: 6
Guess: 1
Guess: 6
Guess: 1
Guess: 6
Guess: 1
Guess: 6
Guess: 1
Guess: 6
Guess: 1
Guess: 6
Guess: 1
Guess: 6
Guess: 1
Guess: 6
Guess: 1
Guess: 6
Guess: 1
Guess: 7
Guess: 1
Guess: 7
Guess: 1
Guess: 7
Guess: 1
Guess: 7
Guess: 1
Guess: 7
Guess: 1
Guess: 7
Guess: 1
Guess: 7
Guess: 1
Guess: 7
Guess: 1
Guess: 7
Guess: 1
Guess: 8
Guess: 1
Guess: 8
Guess: 1
Guess: 8
Guess: 1
Guess: 8
Guess: 1
Guess: 8
Guess: 1
Guess: 8
Guess: 1
Guess: 8
Guess: 1
Guess: 8
Guess: 1
Guess: 8
Guess: 1
Guess: 9
Guess: 1
Guess: 9
Guess: 1
Guess: 9
Guess: 1
Guess: 9
Guess: 1
Guess: 9
Guess: 1
Guess: 9
Guess: 1
Guess: 9
Guess: 1
Guess: 9
Guess: 1
Guess: 9
Guess: 1
Guess: 0
Guess: 0
Guess: 1
Guess: 1
Guess: 2
Guess: 2
Guess: 3
Guess: 3
Guess: 4
Guess: 4
Guess: 5
Guess: 5
Guess: 6
Guess: 6
Guess: 7
Guess: 7
Guess: 8
Guess: 8
Guess: 9
Guess: 9
# elapsed time (accuracy): 0.0748997s
```

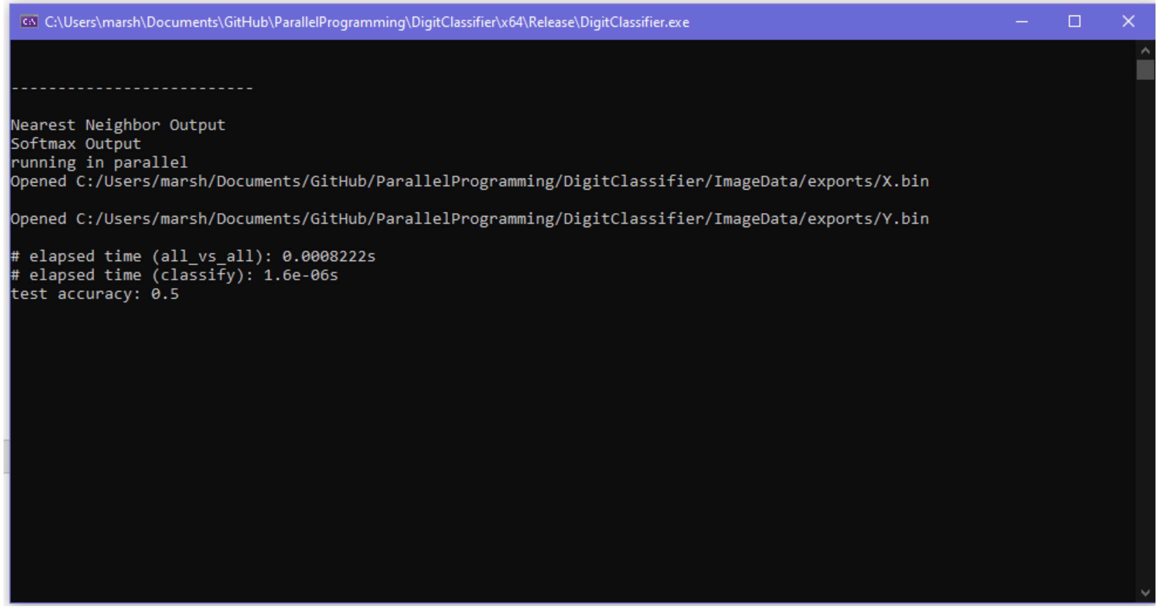
1NN Implementation:

Half of the values are used for training:


C:\Users\marsh\Documents\GitHub\ParallelProgramming\DigitClassifier\x64\Release\DigitClassifier.exe

```
-------------------------

Nearest Neighbor Output
Softmax Output
running in parallel
Opened C:/Users/marsh/Documents/GitHub/ParallelProgramming/DigitClassifier/ImageData/exports/X.bin

Opened C:/Users/marsh/Documents/GitHub/ParallelProgramming/DigitClassifier/ImageData/exports/Y.bin

# elapsed time (all_vs_all): 0.0023201s
# elapsed time (classify): 4.6e-06s
test accuracy: 0.1
```

90 of the values are used for training:

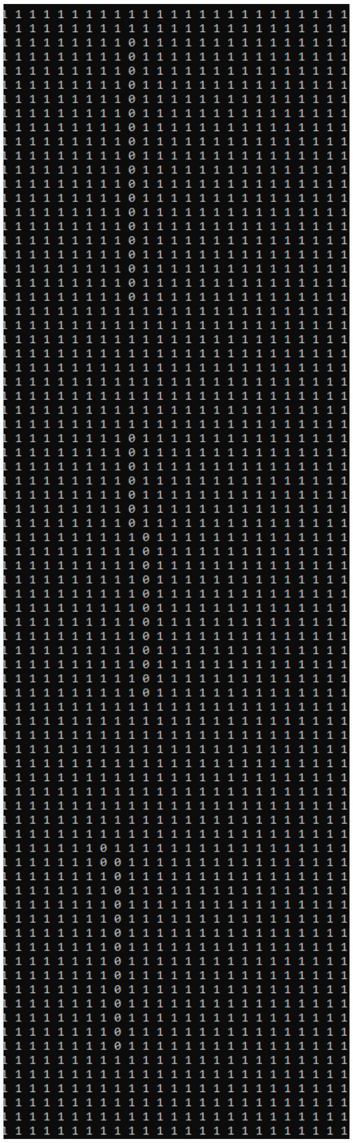

C:\Users\marsh\Documents\GitHub\ParallelProgramming\DigitClassifier\x64\Release\DigitClassifier.exe

```
-------------------------

Nearest Neighbor Output
Softmax Output
running in parallel
Opened C:/Users/marsh/Documents/GitHub/ParallelProgramming/DigitClassifier/ImageData/exports/X.bin

Opened C:/Users/marsh/Documents/GitHub/ParallelProgramming/DigitClassifier/ImageData/exports/Y.bin

# elapsed time (all_vs_all): 0.0008222s
# elapsed time (classify): 1.6e-06s
test accuracy: 0.5
```

Better accuracy when you train more!

I believe my Softmax and 1NN implementations are not very accurate. This is because of my dataset. It is not consistent and a lot of room for ambiguity because the utility used with paint draws a line that covers a 1X1 pixel dimension. In other words, the written digits take up a very small amount of the solution space.

I also was not very careful about making sure they all looked a like and were in the same area. For example, look at my 1's:

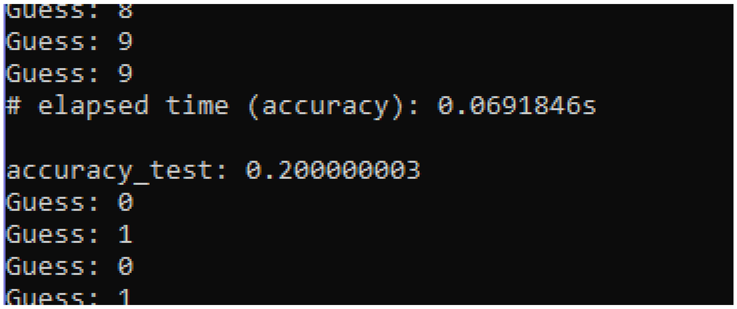


```
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

Compare the accuracy and computing times of these two classifier algorithms.

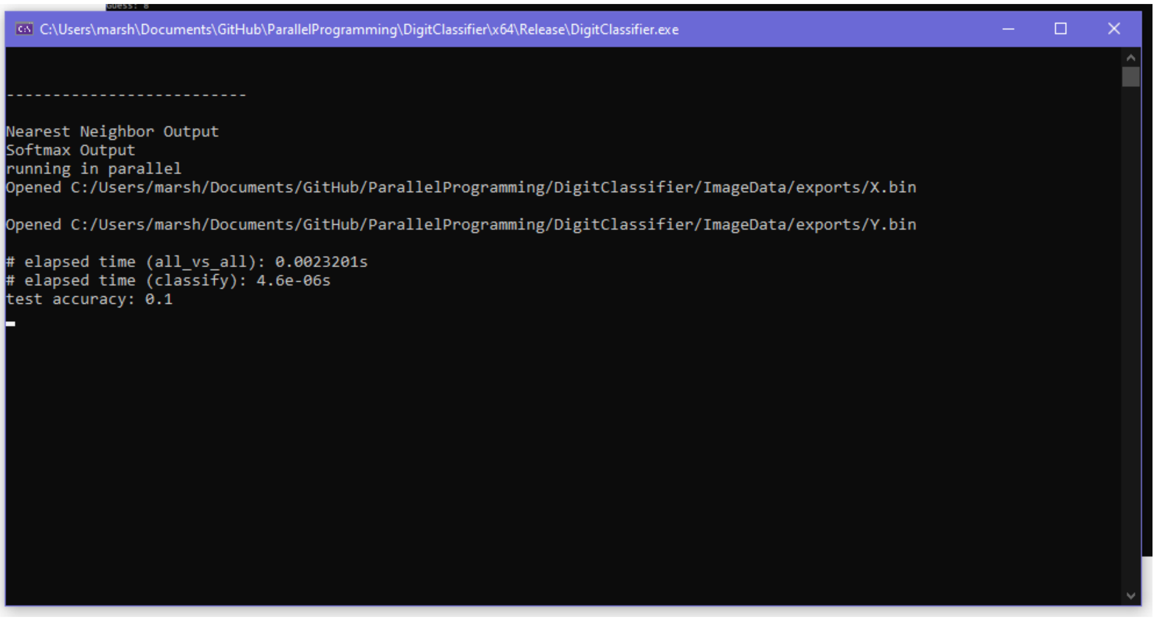
My Softmax implementation can be better by improving the dataset. Espically the weights data set which is just example images of 0-9. If I made them more consistent and possibly even had more data to use I would see more accuracy. The computing times are really not much different and my dataset is not big enough to have significance. In the original dataset for Homework 4 the timing of Softmax was better than 1NN, Softmax can classify these images in about half the time.

Softmax:



```
Guess: 8
Guess: 9
Guess: 9
# elapsed time (accuracy): 0.0691846s

accuracy_test: 0.200000003
Guess: 0
Guess: 1
Guess: 0
Guess: 1
```

1NN:



C:\Users\marsh\Documents\GitHub\ParallelProgramming\DigitClassifier\x64\Release\DigitClassifier.exe

```
-------------------------

Nearest Neighbor Output
Softmax Output
running in parallel
Opened C:/Users/marsh/Documents/GitHub/ParallelProgramming/DigitClassifier/ImageData/exports/X.bin

Opened C:/Users/marsh/Documents/GitHub/ParallelProgramming/DigitClassifier/ImageData/exports/Y.bin

# elapsed time (all_vs_all): 0.0023201s
# elapsed time (classify): 4.6e-06s
test accuracy: 0.1
```