**Group Course Project 2022**

Deadline: 11 pm, 22 December 2022

**Attention:** Students can try to understand the project requirements first, and then work out a workflow of the program, including the main functions, sub-functions. The project needs knowledge from the lecture “File” which will be given in Week 13. However, students can start coding for some functions before Week 13.

In this project, students need to finish a program that can recognize an image with handwriting digit, for example,

|  |  |  |
| --- | --- | --- |
|  | → | 0 |

The image is stored in a file. To recognize the image, the image information in the file must be read and stored in a two-dimensional array (matrix), and a certain algorithm is used to calculate the similarity of the handwriting digit to 10 digits from 0 to 9. The digit with highest similarity will be the output. For example,

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Phase I  → |  | Phase II  → | 0 |

Figure 1 From Image to array, then from array to digit

In this project, students can finish Phase I and Phase II as described below to finish the recognition of a handwriting digit.

**Phase I: From image file to matrix**

Input: A file that stores an image with 28×28 pixels. Each pixel can be represented as an integer between 0 and 255 (inclusive).

Output: Matrix A (28×28).

Steps from the file to the matrix:

1. Declare an array float A[28][28].
2. Use the fgets function to skip the first four lines in the file, which are some basic information about this image.
3. Use fgetc function repeatedly to read each pixel from the fifth line until 784 chars have been obtained. Store these chars in the 28\*28 array.
4. Normalize each pixel value by dividing 255.0 and store into array (Matrix A). Matrix A will be used in Phase II.

pixel=fgetc(fp);

pixel=pixel / 255.0

Example:

|  |  |  |
| --- | --- | --- |
|  | Phase I  → |  |

Figure 2 Phase I: From image to array

**Phase II: From Matrix A to digit output**

1. Reshape

Input: Matrix A (28×28) from Phase I.

Output: Matrix B (1×784 array).

Reshape: B[*m*×28 + *n*] = A[*m*][*n*].

Example:

|  |
| --- |
| Matrix A (28×28): |
| Reshape  ↓ |
| Matrix B (1×784): [0.00 ……. 0.04 0.59 0.99 …….. 0.00] (details refer to file B.txt) |

Figure 3 Phase II: Reshape

1. Multiply (first time)

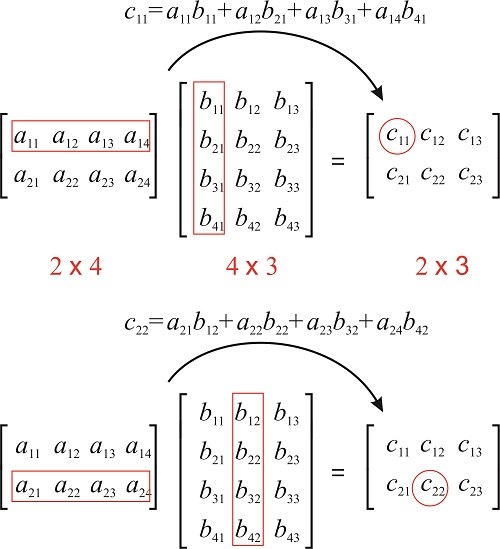
Input: Matrix B (1×784 array from reshape step)

Matrix C (784×128 array from file W1.txt)

Output: Matrix D.

Multiplication: Matrix D is the multiplication of Matrix B and Matrix C.

Multiplication rule of arrays:



Example:

|  |
| --- |
| Matrix B (1×784): [0.00 ……. 0.04 0.59 0.99 …….. 0.00] (details refer to file B.txt) and Matrix C (read from the given file W1.txt) |
| Multiply  ↓ |
| Matrix D (1×128) : [-1.624571 -2.204840 -3.310512 -0.760301 -0.164472 ……] (details please refer to the file D.txt) |

Figure 4 Phase II: first multiply

1. Add

Input: Matrix D (1*×*128 array from step 2 Multiply)

Matrix E (1*×*128 array from file B1.txt)

Output: Matrix F (1*×*128 array)

Add: F[*m*][*n*] = D[*m*][*n*] + E[*m*][*n*]

Example:

|  |
| --- |
| Matrix D (1×128): [-1.624571 -2.204840 -3.310512 -0.760301 -0.164472 ……] (details please refer to the file D.txt) and Matrix E (read from the given file B1.txt) |
| Add  ↓ |
| Matrix F (1×128): [-1.681936 -2.193661 -3.388107 -0.646890 -0.148641 -1.297442 0.836912 ……] (details please refer to the file F.txt) |

Figure 5 Phase II: first add

1. Activate

Input: Matrix F (1*×*128 array from step 3 Add)

Output: Matrix G (1*×*128 array)

Activate: G[m] = max(F[*m*], 0)

Example:

|  |
| --- |
| Matrix F (1×128): [-1.681936 -2.193661 -3.388107 -0.646890 -0.148641 -1.297442 0.836912 ……] (details please refer to the file F.txt) |
| Activate  ↓ |
| Matrix G (1×128): [0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.836912 ……] (details please refer to the file G.txt) |

Figure 6 Phase II: activate

1. Repeat Step 2 Multiply

Replace B with G; replace W1.txt with W2.txt. Output Matrix H

W2.txt contains a 128×10 matrix.

Example:

|  |
| --- |
| Matrix G (1×128) : [0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.836912 ……] (details please refer to the file G.txt) and Matrix from the file W2.txt |
| Second multiply  ↓ |
| Matrix H (1×10) : [ 13.662090 -15.919009 -1.698807 -7.702862 -6.640148 -7.278686 0.050854 1.045354 -7.263999 -1.825507 ] |

Figure 7 Phase II: second multiply

1. Repeat Step 3 Add

Replace D with H; replace B1.txt with B2.txt. Output Matrix L

B2.txt contains a 1×10 matrix

Example:

|  |
| --- |
| Matrix H (1×10) : [ 13.662090 -15.919009 -1.698807 -7.702862 -6.640148 -7.278686 0.050854 1.045354 -7.263999 -1.825507 ] and B2.txt |
| Second add  ↓ |
| Matrix L (1×10) : [ 13.565153 -16.000977 -1.700163 -7.811575 -6.661863 -7.148362 0.038083 0.979735 -7.118680 -1.812590 ] |

Figure 8 Phase II: second add

1. SoftMax

Input: Matrix L (1*×*10 array from step 6 Second Add)

Output: Matrix S (1*×*10 array)

SoftMax: , where max(L) is the maximum value in Matrix L. Search on the internet to find which function in math.h can be used in this calculation. When the function is used, remember to put #include <math.h> at the beginning of the program.

Example:

|  |
| --- |
| Matrix L (1×10) : [ 13.565153 -16.000977 -1.700163 -7.811575 -6.661863 -7.148362 0.038083 0.979735 -7.118680 -1.812590 ] |
| SoftMax  ↓ |
| Matrix S (1×10) : [ 0.999995 0.000000 0.000000 0.000000 0.000000 0.000000 0.000001 0.000003 0.000000 0.000000 ] |

Figure 9 Phase II: softmax

Now S contains the similarities to every digit. Because S[0] has highest value in this example, the output 0 as the recognition result.

**Other requirements**

1. In this project, please use A, B, C, …, S as the matrix names in the program.
2. A well-structured program should include some sub-functions for special purpose, for example: Reshape, Multiply, Add, Activate and SoftMax. (If your program has only one main function, you might lose some marks for non-functional requirements, but this will not affect marks for functional requirements.) In function declaration, when there is an array parameter, please use the similar declaration as shown below for the function *Multiply* example (Reasons will be introduced in Lecture Pointer later).

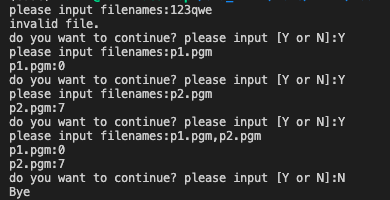
void Multiply(float \*pB, float \*pC, float \*pD, int m, int n, int k);

When this function is called in main function, we can call using Multiply (B, C, D, 1, 784, 128).

Inside Multiply function, you can used pB[i][j] to indicate an element. pB[i][j] is same as B[i][j].

1. If a user inputs one image file name (e.g., p1.pgm), then output the file name and the digit after the recognition (e.g., p1.pgm: 9)
2. If a user inputs more than one image file names (e.g., p1.pgm, p2.pgm), the program should output each file name and its corresponding digit in the list (e.g., p1.pgm: 5, p2.pgm: 2).
3. After output is given, there should be a prompt, asking the user if he wants to continue. If he chooses to continue, input Y, otherwise, input N. If the user chooses Y, ask the user to input image file name to repeat the previous work.

Sample I/O:



1. In the program, when a file is read, there should be no path with the file name, i.e., assume that files to read are in the same folder with the .exe file.
2. When you submit your program, submit only .c files and .h files if there exist. If program has compiling error in Pelles C environment, no score is given.

**Files in the zip package (next page)**

