HO CHI MINH CITY

UNIVERSITY OF TECHNOLOGY

COMPUTER ARCHITECTURE

"CONVOLUTION MATRIX"

TUTOR : NGUYỄN THIÊN ÂN

Name & ID: Nguyễn Hoàng Quốc 2353027

HO CHI MINH CITY, 11/2024

Content

1 Introduction

Convolutional neural network (CNN) is one of the most widely used type of network used in deep learning. Its usage can be seen in a wide range of applications, particularly in fields that involve image and video analysis. However advanced a CNN model maybe, it always stems from matrix computations and most notably, convolution operations. Convolution kernels, also known as filters, are small matrices used for the convolution operation. These kernels slide across the input data, performing element-wise multiplication with the corresponding pixels and producing a feature map that highlights specific patterns or features in the input.

The process of applying a convolution kernel to the input data is referred to as the convolution operation. This operation involves sliding the kernel across the input, computing the dot product at each position, and generating an output feature map.

Stride defines by what step does to kernel move, for example stride of 1 makes kernel slide by one row/column at a time and stride of 2 moves kernel by 2 rows/-columns.

Convolution is the process of adding each element of the image to its local neighbors, weighted by the kernel. This is related to a form of mathematical convolution. The matrix operation being performed—convolution—is not traditional matrix multiplication, despite being similarly denoted by *.

For example, if we have two three-by-three matrices, the first a kernel, and the second an image piece, convolution is the process of flipping both the rows and columns of the kernel and multiplying locally similar entries and summing. The element at coordinates [2, 2] (that is, the central element) of the resulting image would be a weighted combination of all the entries of the image matrix, with weights given by the kernel:

```
for each image row in input image:
    for each pixel in image row:

        set accumulator to zero

        for each kernel row in kernel:
            for each element in kernel row:

                if element position  corresponding* to pixel position then
                    multiply element value  corresponding* to pixel value
                    add result to accumulator
                endif

        set output image pixel to accumulator
```

2 Program Overview

2.1 Requirement

The purpose of this program is to implement a MIPS assembly program capable of performing a convolution operation. The program will take a given image matrix and a kernel matrix as inputs and produce the resulting convolved image matrix as output.

2.2 Functionality

The program should be able to receive input from an external input file. It should be named input matrix.txt. The content is numbers separated by spaces. Both image matrix and kernel matrix are square matrix mxm and nxn, respectively. Additionally, input matrix.txt contain 3 rows, the first row will include 4 values which are:

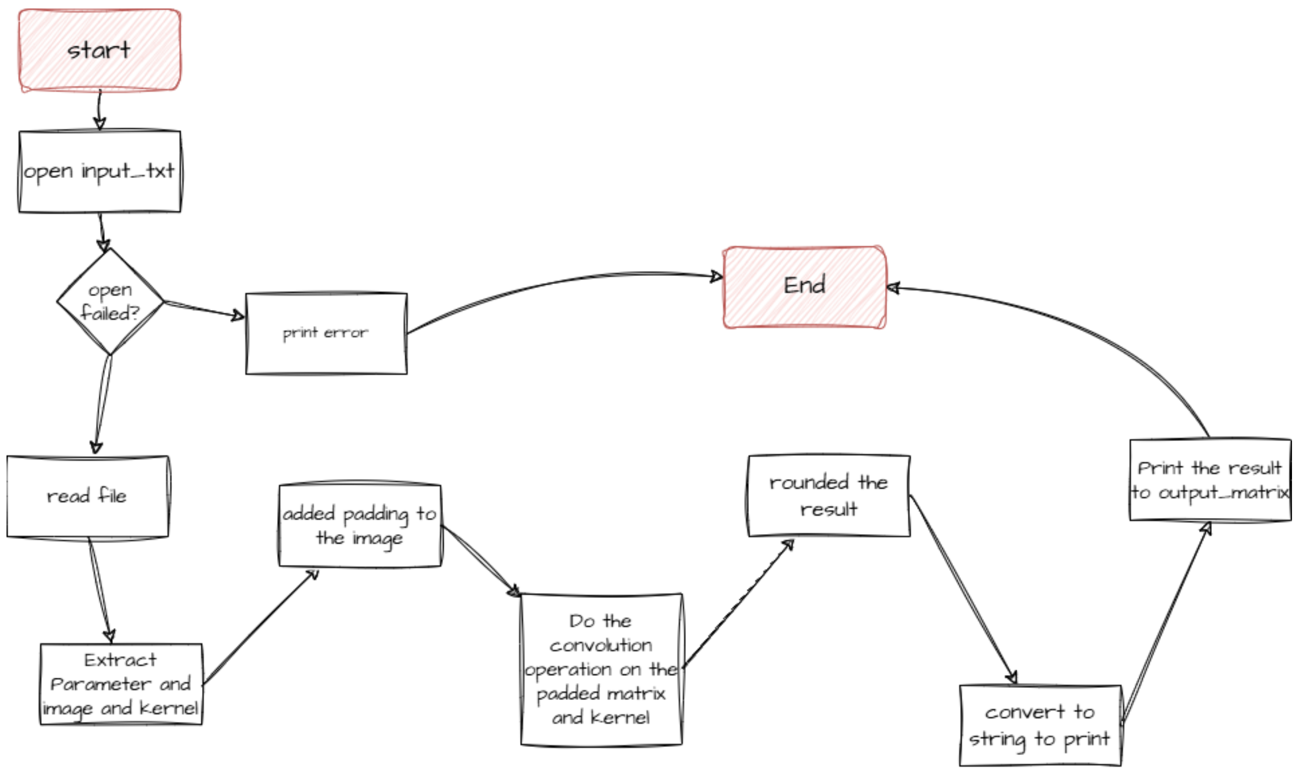
- N: The size of the image matrix.
- M: The size of the kernel matrix.
- p: The value of padding.
- s: The value of stride.

Here is a code snippet based on a high-level programming language C++ to test the accuracy produced by MIPS Simulator:

```cpp
#include <iostream>
#include <vector>
#include <stdexcept>
// Function to add padding to the matrix
std::vector<std::vector<double>> addPadding(const
    std::vector<std::vector<double>>& matrix, int padSize) {
    int originalRows = matrix.size();
    int originalCols = matrix[0].size();
    int paddedRows = originalRows + 2 * padSize;
    int paddedCols = originalCols + 2 * padSize;

    // Initialize padded matrix with zeros
    std::vector<std::vector<double>> paddedMatrix(paddedRows,
    std::vector<double>(paddedCols, 0.0));

    // Copy original matrix to the center of the padded matrix
    for (int i = 0; i < originalRows; ++i) {
        for (int j = 0; j < originalCols; ++j) {
            paddedMatrix[i + padSize][j + padSize] = matrix[i][j];
        }
    }

    return paddedMatrix;
}

// Function to perform sliding matrix multiplication with padding and
    stride
std::vector<std::vector<double>>
    slidingMatrixMultiplicationWithPaddingAndStride(
    const std::vector<std::vector<double>>& A,
    const std::vector<std::vector<double>>& B,
    int windowSize, int stride, int padSize) {

    // Add padding to matrix A
    std::vector<std::vector<double>> paddedA = addPadding(A, padSize);

    int rowsA = paddedA.size();
    int colsA = paddedA[0].size();
    int rowsB = B.size();
    int colsB = B[0].size();

    // Ensure window size matches dimensions of B
    if (windowSize != rowsB || windowSize != colsB) {
        throw std::invalid_argument("Window size does not match
    dimensions of B");
    }

    // Resultant matrix dimensions
    int resultRows = (rowsA - windowSize) / stride + 1;
    int resultCols = (colsA - windowSize) / stride + 1;
    std::vector<std::vector<double>> result(resultRows,
    std::vector<double>(resultCols, 0.0));

    // Perform the sliding window matrix multiplication with padding and
   stride
    for (int i = 0; i <= rowsA - windowSize; i += stride) {
        for (int j = 0; j <= colsA - windowSize; j += stride) {
            double sum = 0.0;
            for (int k = 0; k < windowSize; ++k) {
                for (int l = 0; l < windowSize; ++l) {
                    sum += paddedA[i + k][j + l] * B[k][l];
                }
            }
            result[i / stride][j / stride] = sum;
        }
    }

    return result;
}
END FUNCTION
```

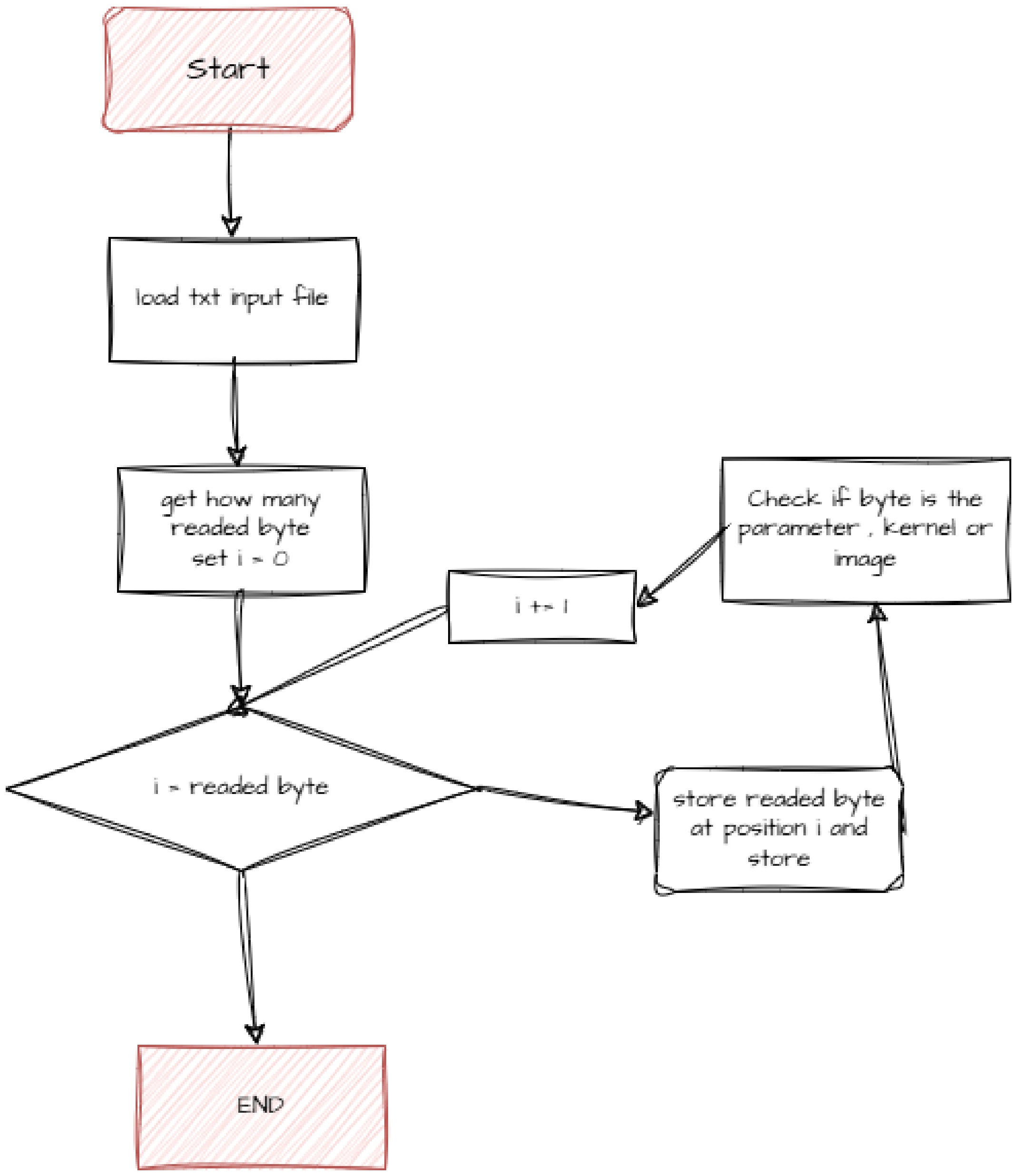
2.3 Logic

The program is designed to perform a convolution operation on a matrix and output the results to a file. The main steps include reading the matrix, performing the convolution, rounding the results to one decimal place, converting the results to strings, and writing them to a file. This flow chart summarize the basic function of this assignment


start
open input_txt
open failed?
print error
End
read file
Extract Parameter and image and kernel
added padding to the image
Do the convolution operation on the padded matrix and kernel
rounded the result
convert to string to print
Print the result to output_matrix

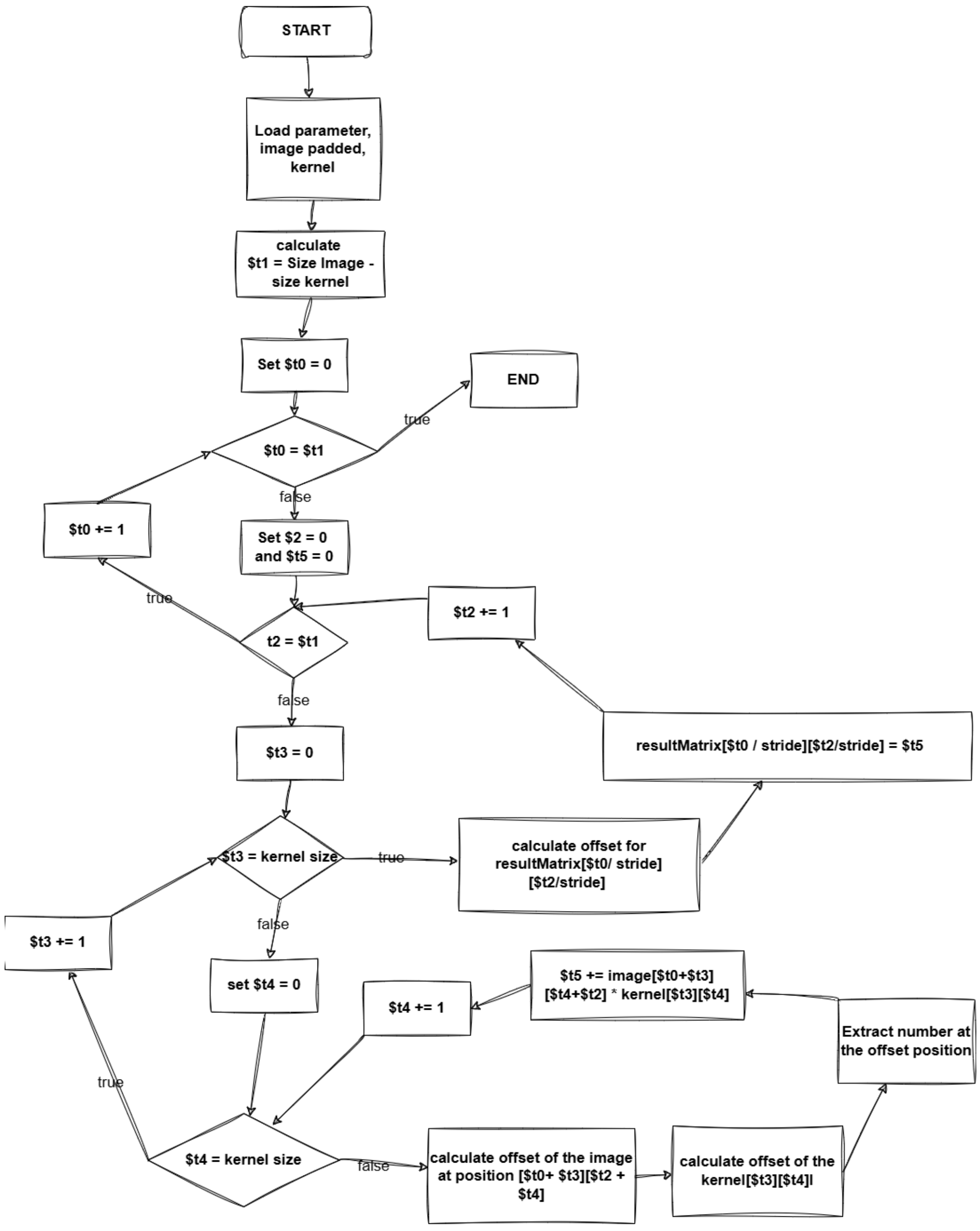
2.3.1 Initialization

The program begins by opening an output file for writing. This is done using system calls to ensure the file is ready to receive data.
Registers and memory locations are initialized to store the matrix data, buffer for writing, and counters for looping through the matrix elements.
Flowchart represent the Initialize process


Start
load txt input file
get how many readed byte set i = 0
i = readed byte
store readed byte at position i and store
Check if byte is the parameter , kernel or image
i += 1
END

2.3.2 Processing

The matrix elements are read from a predefined memory location (out).
A loop (write loop) iterates through each element of the matrix. For each element, the program calls the (float to buffer) function to process and convert the float value to a string.


START
Load parameter, image padded, kernel
calculate $t1 = Size Image - size kernel
Set $t0 = 0
END
true
$t0 = $t1
false
$t0 += 1
Set $2 = 0 and $t5 = 0
true
$t2 += 1
t2 = $t1
false
resultMatrix[$t0 / stride][$t2/stride] = $t5
$t3 = 0
$t3 = kernel size
true
calculate offset for resultMatrix[$t0/ stride] [$t2/stride]
false
$t3 += 1
set $t4 = 0
$t5 += image[$t0+$t3][$t4+$t2] * kernel[$t3][$t4]
$t4 += 1
Extract number at the offset position
true
$t4 = kernel size
false
calculate offset of the image at position [$t0+ $t3][$t2 + $t4]
calculate offset of the kernel[$t3][$t4]l

2.3.3 Rounding

Inside the (float to buffer) function, the program first handles the rounding of the float value to one decimal place.
The rounding logic multiplies the float by 10, adds or subtracts 0.5 based on the sign of the number, converts it to an integer to truncate, and then divides by 10 to restore one decimal place.

2.3.4 Conversion

After rounding, the program converts the float value to a string representation. The integer part of the float is extracted and converted to its ASCII representation. The fractional part is handled by multiplying it to get the first decimal digit, which is then converted to its ASCII representation.
The program ensures that the negative sign is correctly handled for negative numbers.

2.3.5 Finalizing

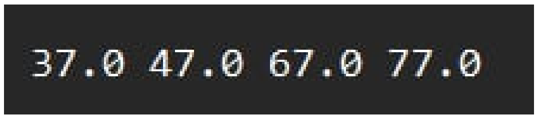
The converted string (including the integer part, decimal point, and fractional part) is stored in a buffer (write buffer). Spaces are added between numbers, except after the last number to avoid trailing spaces.
Once all elements are processed and stored in the buffer, the program writes the buffer to the output file. The file is then closed properly using system calls to ensure all data is written and the file is not corrupted.

2.4 Output Samples

2.4.1 Simple testcase

```
3.0   2.0   0.0   1.0
1.0   2.0   3.0   4.0   5.0   6.0   7.0   8.0   9.0
1.0   2.0   3.0   4.0
```

Output:



```
37.0 47.0 67.0 77.0
```

3 Conclusion

In this assignment, we successfully implemented a convolution operation in MIPS assembly language. The program is designed to take an image matrix and a kernel matrix as inputs and produce the resulting convolved matrix as output.
Through various test cases, including basic, negative and fractional values, and scenarios with different padding and stride values, the program demonstrated robustness and correctness in performing convolution operations.
This project highlighted the importance of convolution in image processing and neural networks, providing hands-on experience with low-level programming and detailed understanding of matrix operations in the context of deep learning. By completing this assignment, the foundational concepts of convolutional neural networks were reinforced, illustrating their practical application in computer vision tasks.