VIETNAM NATIONAL UNIVERSITY HO CHI MINH CITY
HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

COMPUTER ARCHITECTURE LAB (CO2008)

Assignment

CONVOLUTION OPERATION

Instructors: Nguyen Thanh Loc

Author: Truong Gia Ky Nam

Class: CC04

ID: 2352787

Email: nam.truonggiaky@hcmut.edu.vn

Ho Chi Minh City, November 2024

Abstract

The convolution operation is a fundamental technique in digital signal processing and computer vision used for various applications such as image filtering, feature detection, and edge detection. This process involves overlaying a smaller matrix (kernel or filter) on a larger matrix (input data) and computing a weighted sum of the overlapping values. The result is stored in an output matrix, effectively transforming the input according to the properties of the kernel. In the context of MIPS assembly programming, implementing convolution involves manually handling matrix data, iterating over elements, performing multiplications and additions, and managing memory storage—all within the constraints of low-level instruction sets. The assignment will demonstrate the ability to break down complex mathematical operations into MIPS assembly code, showcasing proficiency in logical flow, register management, and computational efficiency in the MARS simulator environment.

Contents

1 Introduction

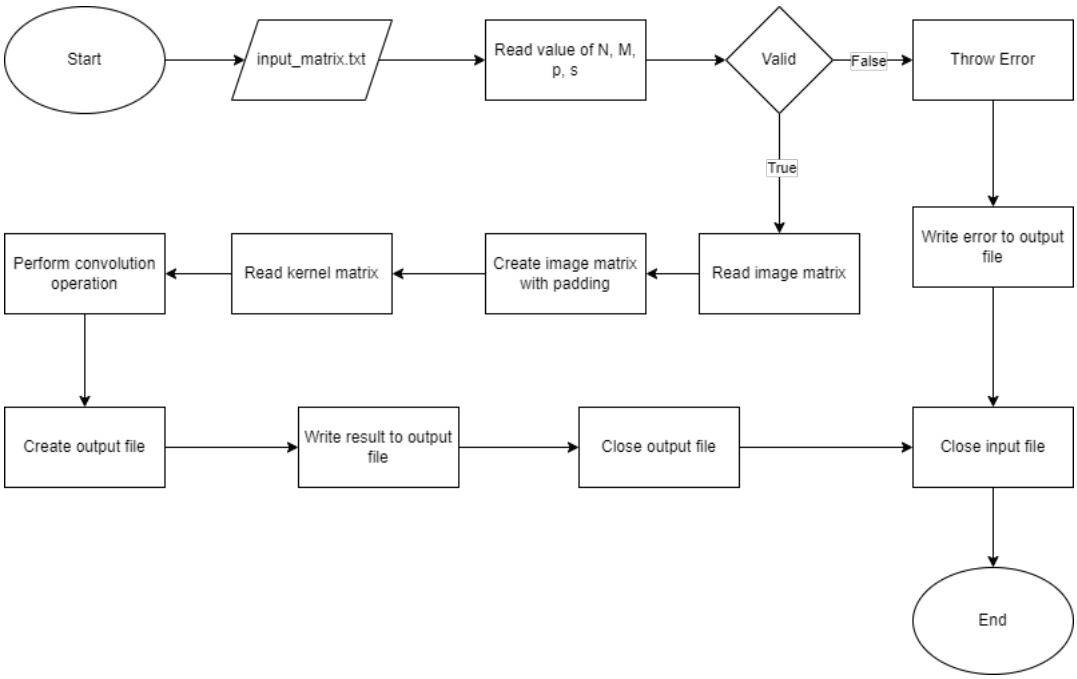
This report describes the implementation of a Convolution Operation using MIPS assembly language. The convolution process is widely used in digital image processing and deep learning, particularly for tasks like feature extraction and image filtering. This program performs a convolution operation on an input matrix (image) with a kernel (filter) and outputs the result to a file. The implementation considers constraints like input validation, padding, and kernel size compatibility with the input matrix

2 Convolution operation

The convolution operation is a fundamental mathematical process widely used in signal processing, image processing, and deep learning, particularly in convolutional neural networks (CNNs). At its core, convolution involves the sliding of a filter or kernel over an input data structure, such as a one-dimensional signal or a two-dimensional image, to produce an output known as the feature map. This operation effectively transforms the input by highlighting certain patterns or features, such as edges or textures in an image, making it essential for feature extraction in computer vision tasks.

In the case of image processing, convolution works by positioning a small matrix called a kernel over a region of the image, multiplying corresponding values, and summing the results to produce a single output pixel in the feature map. This process is repeated as the kernel slides across the entire image, creating a matrix that encodes information about local patterns. By applying different kernels, such as those for edge detection, sharpening, or blurring, convolution allows for various types of image transformations and analysis. The size and values of the kernel determine the type of feature that is emphasized, while parameters like stride and padding control the movement and preservation of image dimensions.

3 Explanation of the Algorithm



To see the full source code, take a look at the file: *assignment.asm*

3.1 Overview

The program is structured into several logical sections to ensure efficient execution:

1. Input File Handling
   - Reads an input file (input matrix.txt) containing matrix dimensions, padding, stride, and the matrix data.
   - Handles potential errors such as invalid or out-of-bound inputs.
2. Validation of Inputs
   - Validates dimensions of the matrix, kernel size, padding, and stride against defined constraints.
   - Generates error messages for: kernel size larger than image; invalid constraints
3. Padding the Input Matrix
   - Adds zero-padding around the input matrix based on the specified padding size (p).
   - The padded matrix dimensions are calculated as N + 2P.
4. Kernel and Matrix Reading
   - Reads the kernel (filter) matrix from the input.
   - Converts string inputs into floating-point numbers using a custom parser.
5. Convolution Operation
   - Iterates over the padded image matrix with a specified stride (s) to compute the output matrix.
   - Performs element-wise multiplication of the kernel and corresponding image sub-matrix, followed by summation.
6. Output File Handling
   - Writes the resulting output matrix to output matrix.txt.
   - Outputs the values in a formatted way, including floating-point conversion.
7. Error Handling
   - Detects and reports specific errors like invalid inputs, kernel larger than the image, and overflow issues.
   - Ensures graceful program termination with error messages when constraints are violated.

3.2 Detail Explanation

First, a reading pointer will be initialized for reading data in the input file. The pointer will read the first line to store values of the image size, the kernel size, the padding and the stride. After get all these initial value, it will check if those values are in the given constrains or not. After the first 4 values are valid, the size of the padded image and the output matrix size are calculated by using the formula

$$N_{padding} = N_{image} + 2 \times P$$

where N_{image} is the size of the image matrix, P is the value of padding and $N_{padding}$ is the size of the padding image matrix. Also, the output matrix size is calculated by using the formula

$$N_{output} = \frac{N_{image} + 2 \times P - N_{kernel}}{S} + 1$$

where N_{kernel} is the size of the filter matrix and S is the stride value.

Next, pointer will move to read the data for the image matrix. To read and store value as a 2-dimensional matrix, initialize a 2 running variable i and j (represent for row and column index), for each

value read in the input file, it will be store in the correct position of the 2D matrix. The address for elements image[i][j] is calculated by

$$Image[i][j] = BaseAddressOfImageMatrix + i \times N + j$$

When the correct position is found, the pointer will iterate through each digit and the dot to produce the floating-point number. The process is taken place by convert each characters to digit and combine them. Before the decimal point, that is the integer part, store the it in the first register. After the decimal point, that is the decimal part, is stored in another register. When reaching the space character, these two values are combine together to make the complete float number. The process of calculating matrix address and read floating value continue until all values of the image matrix are read. The pointer continue to move to the kernel data to read and store in the kernel matrix.

After reading data for image and kernel matrix, the padded image matrix is created with the size $N_{padding}$ by putting the image matrix in the center of the padding image matrix. To put the image matrix to the center of the padded image matrix, for each element image[i][j], put it at the position *padded_image*[i + p][j + p] where p is the stride value.

When everything is ready, the convolution operation begin. To perform the convolution operation, loop through each cell of the output matrix. At each cell, extract corresponding data in the image and the kernel and calculate the result and store it in the output matrix. Repeat the process until all elements in the output matrix are filled. Finally, from the output matrix, write each element to the output file. It begins by clearing the buffer for storing the result. If the number is negative, it adds a negative sign and adjusts the value accordingly. The function extracts the integer and fractional parts of the number, using *int_to_string* to convert each part into a string, with a decimal point separating them. It handles edge cases where either part equals zero and ensures the fractional part is padded with leading zeros to maintain four decimal places. The final string is shifted appropriately to include the negative sign if needed and null-terminated to mark the end.

4 Sample output

Input 1:

```
3 4 0 2
-3.0 -4 4.5 6 7.8 12 5 -0.5 12.0
1 1.2 -1.3 4.5 -5.0 3 3.5 6 -8.9 12 23.2 12 13 -14 -15 16.0
```

Output 1:

```
Kernel is larger than the Image
```

Input 2:

```
3 4 1 2
-3.0 -4 4.5 6 7.8 12 5 -0.5 12.0
1 1.2 -1.3 4.5 -5.0 3 3.5 6 -8.9 12 23.2 12 13 -14 -15 16.0
```

Output 2:

```
530.4600
```

Input 3:

```
3 3 2 1
-3.0 -4 4.5 6 7.8 12 5 -0.5 12.0
1 1.2 -1.3 4.5 -5.0 3 3.5 6 -8.9
```

Output 3:

```
26.6999 17.5999 -74.5500 13.0000 15.7500 -62.3999 -30.4199 -18.9999 58.8000 62.2500 -22.6000 29.4500 -81.9499 46.7500
100.5000 7.2000 -29.4400 60.7600 -40.0499 66.0000 -6.5000 6.6500 -11.1999 13.9000 12.0000
```

Input 4:

```
3 4 1 2
-3.0 -4 4.5 6 7.8 12 5 -0.5 12.0
1 1.2 -1.3 4.5 -5.0 3 3.5 6 -8.9 12 23.2 12 13 -14 -15 16.0
```

Output 4:

```
530.4600
```

Input 5:

```
4 3 3 1
1 1.2 -1.3 4.5 -5.0 3 3.5 6 -8.9 12 23.2 12 13 -14 -15 16.0
-3.0 -4 4.5 6 7.8 12 5 -0.5 12.0
```

Output 5:

```
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 12.0000 13.9000 -11.1999 60.6500 -8.7500 22.5000 0.0000
0.0000 -48.0000 60.7000 15.2600 136.3099 41.8000 57.0000 0.0000 0.0000 -162.2999 146.8500 249.6500 331.5499 163.7000 82.5000
0.0000 0.0000 26.7000 -66.4199 229.3500 530.4600 115.3000 134.0000 0.0000 0.0000 115.9499
23.0000 -128.1000 -83.7999 -82.8000 60.0000 0.0000 0.0000 58.5000 -115.0000 -50.5000 174.0000 -19.0000 -48.0000 0.0000
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
```

Input 6:

```
4 3 3 3
1 1.2 -1.3 4.5 -5.0 3 3.5 6 -8.9 12 23.2 12 13 -14 -15 16.0
-3.0 -4 4.5 6 7.8 12 5 -0.5 12.0
```

Output 6:

```
0.0000 0.0000 0.0000 0.0000 249.6500 82.5000 0.0000 -50.5000 -48.0000
```

Input 7:

```
5 2 3 2
-1 1 -2 3 -0.4 1 2.0 3 4 1 1.0 1.2 1.3 1.6 10 2.3 4.5 -5 -6.0 2 3 4 5 6 7.0
-3.0 -4 4.5 6
```

Output 7:

```
0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 -6.0000 -7.5000 11.1000 0.0000 0.0000 2.0000 -4.8000 51.2000 0.0000 0.0000 8.8000
54.5000 79.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000
```

Input 8:

```
2 4 1 2
-3.0 -4 4.5 6
1 1.2 -1.3 4.5 -5.0 3 3.5 6 -8.9 12 23.2 12 13 -14 -15 16.0
```

Output 8:

```
Some input is not in the constrain
```