

VIETNAM NATIONAL UNIVERSITY, HO CHI MINH CITY
HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY
Faculty of Computer Science and Engineering



Computer Architecture (CO2008)
CONVOLUTION OPERATION
CC02 — ASSIGNMENT REPORT

Supervisors: Nguyen Thanh Loc
Students: Võ Hoàng Long 2053192

Ho Chi Minh City, March 30, 2025



Contents

1	Introduction	2
2	Requirements	3
2.1	Input	3
2.2	Output	4
2.3	Pre-determined variables	4
3	Flowchart and Algorithm	4
3.1	Flowchart	4
3.2	Preprocessing	5
3.2.1	Padding the Image Matrix	5
3.2.2	Kernel Preparation	6
3.3	Convolution Calculation	6
3.3.1	Output Matrix Dimensions	6
3.3.2	Dot Product Computation	6
3.4	Algorithm Complexity	7
3.4.1	Time Complexity	7
3.4.2	Space Complexity	7
4	Run Testcase and Results :	7
4.1	Test 1	7
4.2	Test case 2	8
4.3	Test case 3	8
4.4	Test case 4	9
4.5	Test case 5	9
4.6	Test case 6	9
4.7	Test case 7	10
4.8	Test case 8	11
	References	12

1 Introduction

In this modern day and age, artificial intelligence has been developing exponentially fast. Rapid improvements in technique and data lead to a remarkable increase in calculations. Its sub-categories, machine learning and deep learning, have reached a point where billions of calculations must be done to compute the millions of parameters in a model.

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.

The convolution operation involves sliding a filter, also known as a kernel, across the input image. This filter is a small matrix of weights that is applied to a local region of the input matrix, producing a single value in the output feature map. The process is repeated across the entire matrix, allowing the network to learn various features of the matrix. In the case of image analysis, these features can include edges, textures, and patterns. The mathematical operation performed is essentially a dot product between the filter and the receptive field of the input image.

The process involves taking a small matrix, called a kernel or filter, and sliding it over an input matrix, such as an image. At each position, the dot product of the kernel and the overlapping region of the input matrix is computed, and the result is stored in the output matrix. This process is repeated for all positions of the kernel over the input matrix, effectively blending the kernel with the input to highlight specific features like edges or textures. In advance,

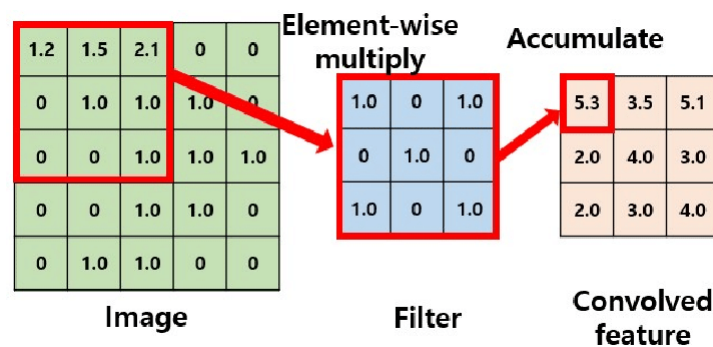


Figure 1: An example of convolution operation

In Figure 1, a 5x5 input matrix and a 3x3 kernel filter have been provided. We first match the kernel onto a corner of the input, in this case the small 3x3 matrix covered in a large red square. We then calculate the sum of element-wise products of both matrices to get a single value. This value will be the first value of the output matrix. Proceed to slide the kernel from left to right, then up to down to cover every square of the input to get the final result. Notice that the stride here is 1 as the kernel move 1 column and 1 row at a time.

One tricky issue when applying convolutional layers is that we tend to lose pixels on the perimeter of our image. One straightforward solution to this problem is to add extra pixels of filler around the boundary of our input image, thus increasing the effective size of the image. Typically, we set the values of the extra pixels to zero.

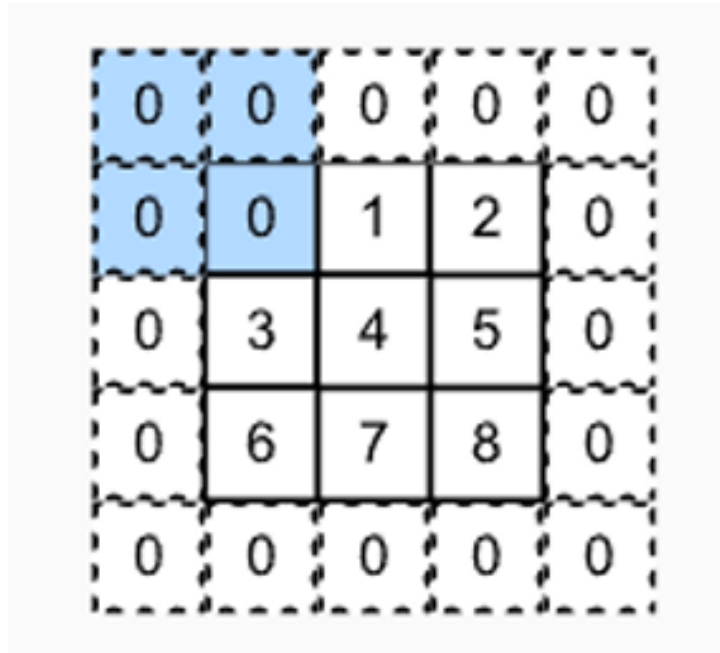


Figure 2: *An example of matrix with zero-padding*

In Figure 2, applying the padding with value equal to 1 into 3x3 input matrix will increase its size to 5x5.

2 Requirements

2.1 Input

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 the image matrix and kernel matrix are square matrices of size $N \times N$ and $M \times M$, respectively. Additionally, **input_matrix.txt** contains 3 rows. The first row includes 4 values which are:

- **N**: The size of the image matrix. ($3 \leq N \leq 7$)
- **M**: The size of the kernel matrix. ($2 \leq M \leq 4$)
- **p**: The value of padding. ($0 \leq p \leq 4$)
- **s**: The value of stride. ($1 \leq s \leq 3$)



2.2 Output

A text file named **output_matrix.txt** contains the convolution matrix result, also numbers separated by spaces.

2.3 Pre-determined variables

Some variables must be defined as follows for grading:

- **image(word)**: Where the image matrix is saved.
- **kernel(word)**: Where the kernel matrix is saved.
- **out(word)**: Where the output matrix is saved.

3 Flowchart and Algorithm

3.1 Flowchart

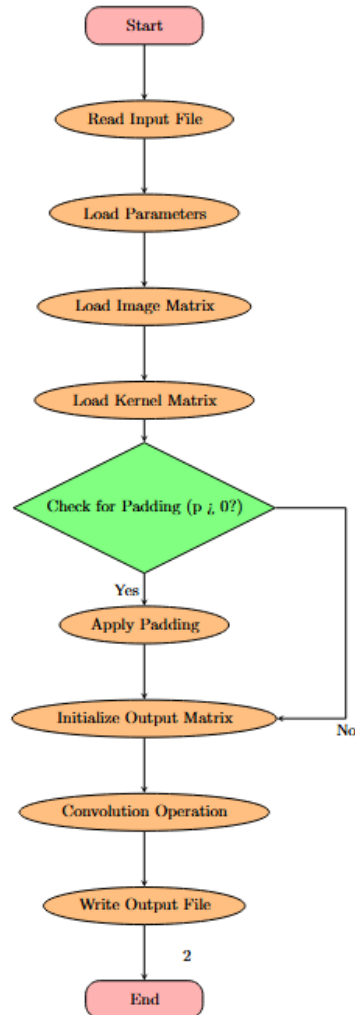


Figure 3: Flowchart for CONVOLUTION OPERATION process

3.2 Preprocessing

Before performing the convolution operation, the image matrix is padded, and the kernel matrix is prepared.

3.2.1 Padding the Image Matrix

The original image matrix of size $N \times N$ is padded with p rows and columns of zeros around its border, resulting in a new matrix of size:

$$(N + 2p) \times (N + 2p).$$

For example, given:

$$\text{Original Matrix: } \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

and padding $p = 1$, the padded matrix becomes:

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 3 & 0 \\ 0 & 4 & 5 & 6 & 0 \\ 0 & 7 & 8 & 9 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

3.2.2 Kernel Preparation

The kernel matrix of size $M \times M$ is optionally flipped (rotated 180°) for convolution. For example:

$$\text{Kernel: } \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}, \quad \text{Flipped Kernel: } \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}.$$

3.3 Convolution Calculation

The convolution operation involves sliding the kernel over the padded image matrix and computing the dot product at each position.

3.3.1 Output Matrix Dimensions

The output matrix size is calculated as:

$$\text{Output Size} = \left\lfloor \frac{(N + 2p - M)}{s} \right\rfloor + 1,$$

where s is the stride.

3.3.2 Dot Product Computation

At each valid position, compute the dot product between the $M \times M$ kernel and the corresponding $M \times M$ region of the image matrix:

$$\text{Dot Product} = \sum_{i=1}^M \sum_{j=1}^M (\text{kernel}[i, j] \cdot \text{image}[i, j]).$$

For example, given:

$$\text{Image: } \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}, \quad \text{Kernel: } \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix},$$

the dot product at the top-left corner is:

$$(1 \cdot 1) + (2 \cdot 0) + (3 \cdot -1) + (4 \cdot 1) + (5 \cdot 0) + (6 \cdot -1) + (7 \cdot 1) + (8 \cdot 0) + (9 \cdot -1) = -12.$$

3.4 Algorithm Complexity

3.4.1 Time Complexity

The convolution operation processes each position in the output matrix. For each position, a dot product of two $M \times M$ matrices is computed. The total complexity is:

$$O(O_h \cdot O_w \cdot M^2),$$

where O_h and O_w are the height and width of the output matrix.

3.4.2 Space Complexity

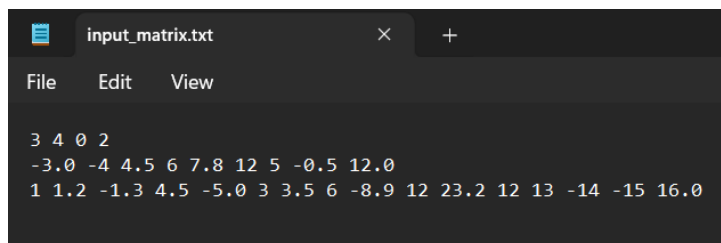
- **Padded Image Matrix:** $(N + 2p) \times (N + 2p)$
- **Kernel Matrix:** $M \times M$
- **Output Matrix:** $O_h \times O_w$, where:

$$O_h = O_w = \frac{N + 2p - M}{s} + 1.$$

4 Run Testcase and Results :

This is 8 Test case i have test and print out the result:

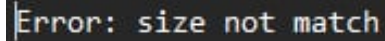
4.1 Test 1



```
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
```

Figure 4: Test case 1

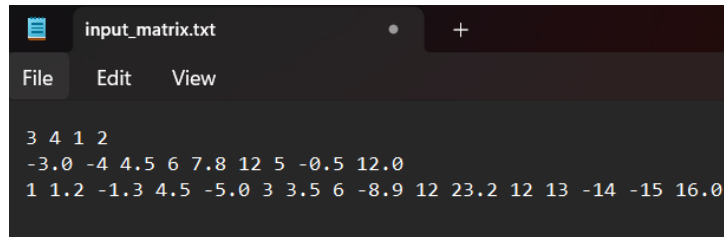
- **Result 1 :**



```
Error: size not match
```

Figure 5: *Result 1*

4.2 Test case 2



```
input_matrix.txt
File Edit View
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
```

Figure 6: *Test case 2*

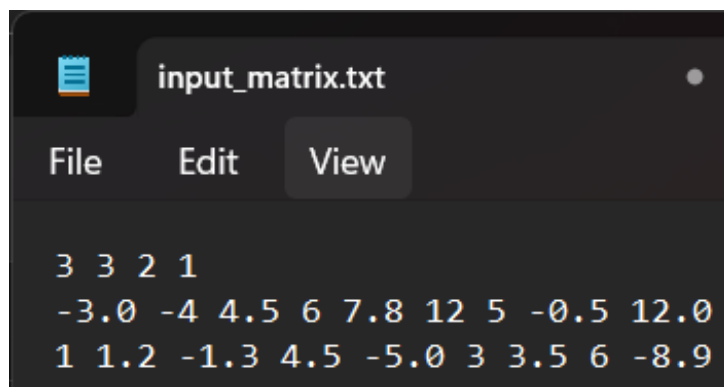
- Result 2 :



```
530.4600
```

Figure 7: *Result 2*

4.3 Test case 3



```
input_matrix.txt
File Edit View
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
```

Figure 8: *Test case 3*

- Result 3 :

```
26.7000 17.5999 -24.5500 13.0000 15.7500 -62.4000 -30.4109 -18.9999 58.8000 62.2500 -22.6000 29.4500 -81.9500 46.7500 100.5000 7.2000 -29.4400
68.7000 -40.8500 66.8000 -6.5000 6.6500 -11.1999 13.9000 12.8000
```

Figure 9: Result 3

4.4 Test case 4

```
input_matrix.txt
File Edit View
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
```

Figure 10: Test case 4

- Result 4 :

```
530.4600
```

Figure 11: Result 4

4.5 Test case 5

```
input_matrix.txt
File Edit View
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
```

Figure 12: Test case 5

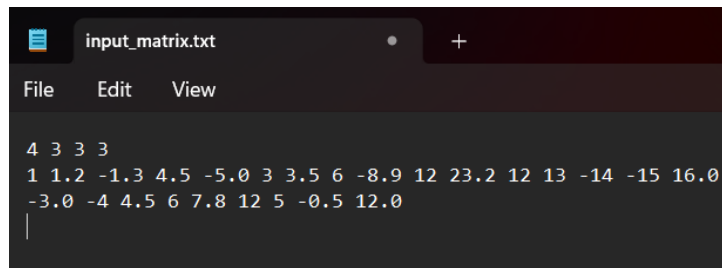
- Result 5 :

```
output_matrix5.txt
File Edit View
57.2500 1330.0000 -630.0000 -662.0000 249.0000 144.0000 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 136.3100 41.8000 57.0000 0.0000 0.0000 -162.2999 146.8500 249.6500 331.5500 163.7000
82.5000 0.0000 0.0000 26.7000 -66.4200 229.3500 530.4600 115.3000 134.0000 0.0000 0.0000 115.9500
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
```

Figure 13: Result 5

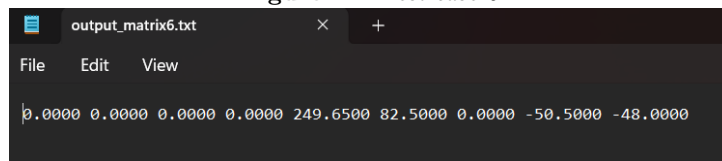
4.6 Test case 6

- Result 6 :



```
input_matrix.txt
File Edit View
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
```

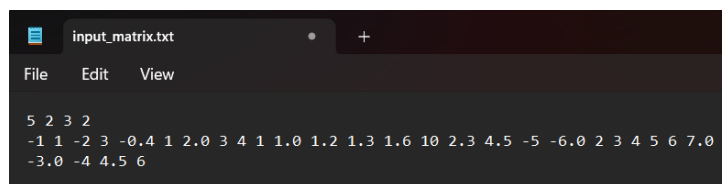
Figure 14: Test case 6



```
output_matrix6.txt
File Edit View
0.0000 0.0000 0.0000 0.0000 249.6500 82.5000 0.0000 -50.5000 -48.0000
```

Figure 15: Result 6

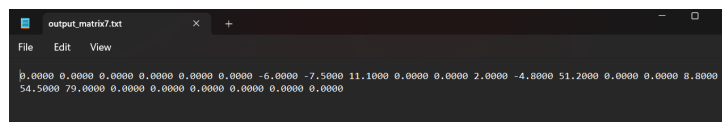
4.7 Test case 7



```
input_matrix.txt
File Edit View
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
```

Figure 16: Test case 7

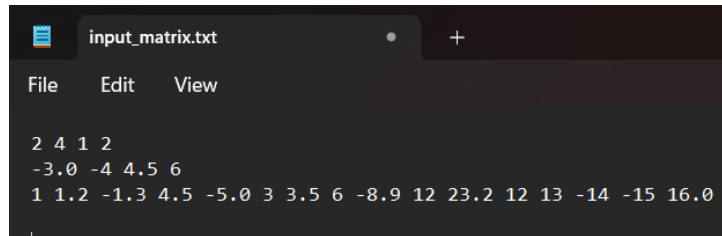
- Result 7 :



```
output_matrix7.txt
File Edit View
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 0.0000
```

Figure 17: Result 7

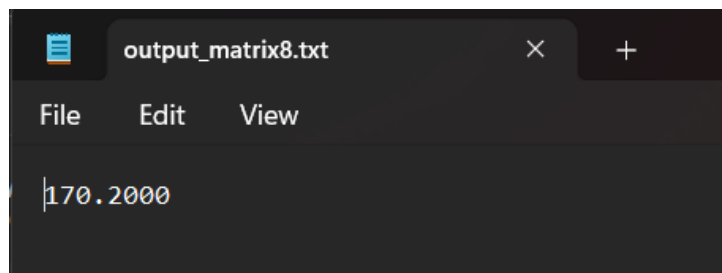
4.8 Test case 8



```
input_matrix.txt
File Edit View
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
```

Figure 18: *Test case 8*

- Result 8 :



```
output_matrix8.txt
File Edit View
170.2000
```

Figure 19: *Result 8*



References