BỘ GIÁO DỤC VÀ ĐÀO TẠO

ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC BÁCH KHOA

BÁO CÁO

BỘ MÔN KIẾN TRÚC MÁY TÍNH

CONVOLUTION OPERATION

MSSV:	2311070
Họ và tên:	Nguyễn Xuân Huy Hoàng
Group:	TN01

Giảng viên hướng dẫn: Nguyễn Thiên Ân

TP. Hồ Chí Minh, 2024

TÓM TẮT

Tích chập (convolution) là một phép toán quan trọng trong lĩnh vực xử lý ảnh và trí tuệ nhân tạo, đặc biệt trong các mạng nơ-ron tích chập (CNN). Phép toán này cho phép máy tính học được các đặc trưng quan trọng của hình ảnh như cạnh, kết cấu, và hoa văn, từ đó hỗ trợ các ứng dụng nhận diện và phân tích hình ảnh. Trong các hệ thống phần cứng như MIPS, việc hiện thực hóa phép toán tích chập đòi hỏi những kỹ thuật tối ưu hóa để đảm bảo hiệu suất tính toán.

Trong báo cáo này, tôi sẽ trình bày cách thực hiện phép toán tích chập trên kiến trúc tập lệnh MIPS, tập trung vào cài đặt thuật toán và tối ưu mã. Cụ thể, báo cáo sẽ mô tả cách thao tác với dữ liệu, thực hiện các phép toán ma trận và lưu trữ kết quả, cũng như các thách thức và giải pháp trong quá trình triển khai. Kết quả cho thấy việc thực hiện phép toán tích chập trên MIPS là khả thi và có thể đạt hiệu suất tốt khi sử dụng các kỹ thuật tối ưu phù hợp.

Mục lục

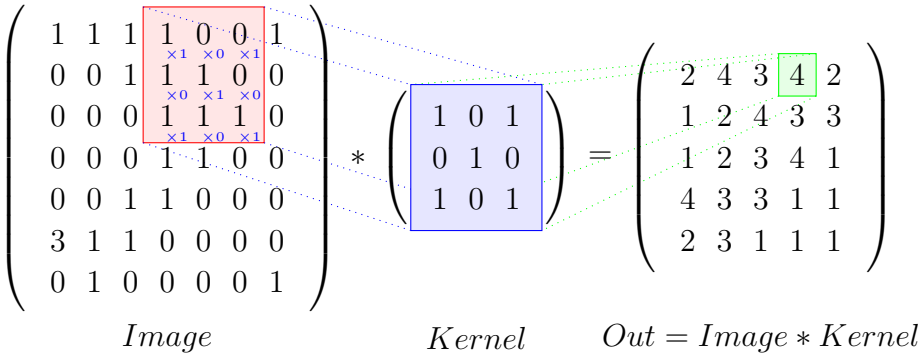
Giới thiệu

1 Giới thiệu về tích chập (Convolution Operation)

Tích chập là một phép toán cơ bản trong học máy, đặc biệt là trong mạng nơ-ron tích chập (CNN). Phép toán này được áp dụng rộng rãi trong các hệ thống học sâu, đặc biệt trong các bài toán liên quan đến xử lý hình ảnh và video.

Tích chập liên quan đến việc di chuyển một bộ lọc (hay kernel) qua ma trận đầu vào, chẳng hạn như hình ảnh, và tính toán phép nhân điểm giữa bộ lọc và vùng cục bộ của ma trận đầu vào. Mỗi phép toán tích chập tạo ra một giá trị trong ma trận đầu ra, giúp tái tạo lại các đặc trưng của ảnh như cạnh, đường nét, hoặc hình dạng.

Quá trình này có thể được lặp lại nhiều lần để tạo ra các đặc trưng phức tạp hơn trong các lớp sâu của mạng nơ-ron. Điều này giúp mạng học được các đặc điểm không gian và cấu trúc từ dữ liệu, mà không cần phải xác định thủ công các đặc trưng như trong các phương pháp truyền thống.



2 Tầm quan trọng của tích chập trong xử lý ảnh

Tích chập là một trong những phép toán nền tảng trong các mô hình học sâu, đặc biệt là trong các mạng nơ-ron tích chập (CNN). Nhờ vào việc áp dụng các bộ lọc qua các vùng ảnh, mạng nơ-ron có thể phát hiện và học các đặc trưng quan trọng như đường biên, kết cấu và hình mẫu trong dữ liệu hình ảnh. Điều này đặc biệt quan trọng trong các ứng dụng như nhận diện hình ảnh, phân loại ảnh và phân tích video, nơi mà khả năng phát hiện các đặc trưng trong ảnh đóng vai trò quyết định đến hiệu quả của mô hình.

3 Mục tiêu của báo cáo

Mục tiêu của bài báo cáo này là phân tích và mô tả chi tiết mã nguồn về phép toán tích chập trong học sâu. Bài báo cáo sẽ cung cấp giải thích cụ thể về cách thức hoạt động của tích chập, từ việc di chuyển bộ lọc qua ảnh đầu vào đến việc tính toán các giá trị trong bản đồ đặc trưng. Bài báo cáo cũng sẽ làm rõ tầm quan trọng của tích chập trong việc phát hiện các đặc trưng trong hình ảnh, và từ đó, giúp hiểu sâu hơn về vai trò của tích chập trong các ứng dụng học sâu hiện đại.

Phép toán Tích chập

1 Một số khái niệm liên quan

- **Bước nhảy (Stride):** Xác định khoảng cách di chuyển của bộ lọc trên ma trận đầu vào. Bước nhảy lớn sẽ làm giảm kích thước đầu ra và ngược lại.
- **Đệm (Padding):** Giúp giữ nguyên kích thước của đầu vào khi áp dụng tích chập. Bằng cách thêm các phần tử 0 xung quanh đầu vào, đệm cho phép áp dụng phép tích chập mà không làm giảm kích thước ma trận đầu ra.
- **Bộ lọc (Filter/Kernel):** Xác định cách trích xuất đặc trưng. Các bộ lọc khác nhau sẽ phát hiện các đặc trưng khác nhau trong dữ liệu (ví dụ: cạnh, góc, họa tiết).

2 Nguyên lý của phép toán tích chập

Phép toán tích chập giữa một ma trận đầu vào và một bộ lọc (kernel) được thực hiện theo các bước sau:

1. **Đặt Bộ Lọc:** Đặt bộ lọc (kernel) lên một phần của ma trận đầu vào. Bộ lọc là một ma trận nhỏ (ví dụ: 3x3 hoặc 5x5) có các giá trị số, và những giá trị này sẽ ảnh hưởng đến cách trích xuất đặc trưng từ dữ liệu đầu vào.
2. **Tính Giá Trị Tích Chập:**
   - Tại vị trí hiện tại, tính tích chập bằng cách nhân từng phần tử của bộ lọc với phần tử tương ứng của đầu vào, rồi cộng tất cả các giá trị nhân đó lại.
   - Kết quả là một giá trị duy nhất biểu diễn đặc trưng tại vị trí đó trong ma trận đầu vào.
3. **Dời Bộ Lọc:** Dời bộ lọc theo một bước di chuyển (stride) nhất định và lặp lại quá trình nhân và cộng trên toàn bộ ma trận đầu vào cho đến khi bộ lọc quét hết tất cả các vùng trong ma trận.
4. **Tạo Ma Trận Đầu Ra:** Giá trị tại mỗi lần tính toán sẽ là một phần tử trong ma trận đầu ra, tạo thành bản đồ đặc trưng (feature map) – kết quả của phép tích chập.

Công thức tính

Giả sử:

- Ma trận đầu vào *image* I, kích thước $N \times N$.
- Bộ lọc *kernel* K, kích thước $M \times M$.
- Ma trận đầu ra *out* O.

Công thức cho giá trị $O(x, y)$ tại vị trí (x, y) của ma trận đầu ra là:

$$O(x,y) = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} I(x+i, y+j) \times K(i,j)$$

3 Các ứng dụng điển hình của tích chập

- **Xử lý ảnh:** Tích chập dùng để làm mờ, tăng độ nét, phát hiện cạnh, và các tác vụ liên quan khác trong xử lý ảnh.
- **Mạng nơ-ron tích chập (CNN):** Tích chập giúp phát hiện các đặc trưng như cạnh, họa tiết, hoặc hình dạng trong ảnh để nhận diện và phân loại.
- **Xử lý tín hiệu:** Trong xử lý âm thanh và tín hiệu, tích chập giúp lọc và phân tích các đặc trưng tần số.

Phân tích và cài đặt mã

1 Sơ lược về mã nguồn

Đoạn mã gồm ba phần chính:

1. **Xử lý đầu vào**: Phần đầu tiên xử lý chuỗi `input` từ file `input_matrix.txt`. Phần này bao gồm việc đọc từng dòng để trích xuất các tham số như kích thước của ma trận, giá trị `padding`, và các giá trị cho cả ma trận ảnh (`image matrix`) và ma trận nhân (`kernel matrix`).
2. **Padding ma trận và tích chập**: Sau khi xử lý xong phần đầu vào, mã sẽ tiến hành áp dụng `padding` cho ma trận ảnh như đã chỉ định. Sau đó, phép tích chập được thực hiện trên ma trận ảnh đã được padding bằng cách sử dụng ma trận `kernel`, cho ra một ma trận đầu ra (`output matrix`) là kết quả của phép tích chập.
3. **Tạo đầu ra**: Cuối cùng, ma trận đầu ra sẽ được chuyển đổi thành chuỗi để có thể ghi vào file `output_matrix.txt`. Chuỗi này được định dạng và lưu vào file, hoàn tất quá trình xử lý.

2 Xử lý đầu vào

☐ Xử lý dòng đầu tiên.

1. **Mở file và lưu chuỗi đầu vào:** Đọc dữ liệu từ file `input_matrix.txt` vào một chuỗi.
2. **Xử lý từng ký tự:** Bỏ qua khoảng trắng và dấu chấm, nếu là số thì lưu vào mảng.
3. **Lưu các giá tri:** *N, M, p, s* lần lượt cho *size* của *image, kernel, padding, stride.*

Algorithm 1 Pseudocode: ***Mở file và xử lý dòng thứ nhất trong chuỗi đầu vào của input***

```
Open file "input_matrix.txt"
Save input string
                                                  ▷ Read and process the first line
byte ← first byte of (input string)
SoHangCot ← is the address of the array that contains length elements
for i ← 0; i ≠ endline character; i ← i + 1 do
    if byte = ' ' then
        byte ← next(byte)                                  ▷ Skip the space character
    end if
    if isNumber(byte) = 1 then
        SoHangCot[i] ← isNumber(byte)
        byte ← next(byte)
    end if
    if byte = '.' then                  ▷ Skip the dot and the next adjacent character
        byte ← next(byte)
        byte ← next(byte)
    end if
end for
N ← SoHangCot[0] The size of the image matrix
M ← SoHangCot[1] The size of the kernel matrix
p ← SoHangCot[2] The value of padding
s ← SoHangCot[3] The value of stride
```

☐ Xử lý dòng thứ hai.

1. **Khởi tạo địa chỉ cho mảng `image`:** Xác định `image` là địa chỉ của mảng lưu trữ các giá trị từ dòng đầu vào.
2. **Xử lý từng ký tự:**
   - **Bỏ qua khoảng trắng**: Nếu gặp.
   - **Gán số nguyên**: Nếu ký tự là một số, lưu nó vào `image[i]`, rồi chuyển sang ký tự tiếp theo.
   - **Xử lý chữ số thập phân**: Nếu ký tự là dấu chấm (`.`), bỏ qua dấu chấm và lấy tiếp chữ số liền sau, rồi nhân số đã lưu trong `image[i]` với 10 và cộng thêm chữ số mới.
3. **Lặp lại** cho đến khi kết thúc dòng, lưu tất cả các giá trị vào `image`.

Algorithm 2 Pseudocode: Xử lý dòng thứ hai trong chuỗi đầu vào của input

```
image ← địa chỉ của mảng image
for i ← 0; i ≠ endline character; i ← i + 1 do
    if byte = ' ' then
        byte ← next(byte)
    end if
    if isNumber(byte) then
        image[i] ← isNumber(byte)
        byte ← next(byte)
    end if
    if byte = '.' then
        byte ← next(byte)
        image[i] ← (image[i] * 10 + isNumber(byte)) * signBIT
    end if
end for
```

☐ Xử lý dòng thứ ba.

1. **Khởi tạo địa chỉ cho mảng `kernel`:** Xác định `kernel` là địa chỉ của mảng lưu trữ các giá trị từ dòng đầu vào.
2. Cách xử lý tương tự với dòng thứ hai.
3. Cách xử lý tương tự với dòng thứ hai.

Algorithm 3 Pseudocode: Xử lý dòng thứ ba trong chuỗi đầu vào của input

```
kernel ← is the address of kernel matrix
for i ← 0; i < end of string; i ← i + 1 do
    if bytes = ' ' then
        bytes ← next(bytes)
    end if
    if isNumber(bytes) then
        kernel[i] ← isNumber(bytes)
        bytes ← next(bytes)
    end if
    if bytes = '.' then
        bytes ← next(bytes)
        kernel[i] ← (kernel[i] * 10 + isNumber(bytes)) * signBIT
    end if
end for
```

3 Padding ma trận và tích chập

☐ Padding ma trận image để được ma trận PADimage.

1. **Thiết lập các biến cần thiết:**
   - `length` xác định kích thước của ma trận có thêm đệm (`p` là phần đệm, `r` là kích thước ban đầu của ma trận).
   - `image` và `PAD` là địa chỉ của mảng `image` và `PADimage`.
   - `pos` là biến vị trí để dịch con trỏ và lấy giá trị từ `image`.
2. **Duyệt qua ma trận với đệm:**
   - Vòng lặp chạy qua các hàng và cột của ma trận với phần đệm từ vị trí `p` đến `length - 1`.
   - `vitri` được tính dựa trên vị trí hàng và cột để lấy giá trị ở vị trí tương ứng trong `PADimage` (vì mỗi phần tử chiếm 4 byte).
3. **Gán giá trị:** Chuyển giá trị từ `image` sang `PADimage` tại vị trí `vitri`, rồi tăng `pos` lên 4 để tiếp tục lấy giá trị kế tiếp từ `image`.

Algorithm 4 Sao chép giá trị tương ứng từ mảng `image` sang `PADimage` với phần đệm

```
input: image, PADimage, p (the size of padding), r (the size of image matrix)
output: PADimage với phần đệm
length ← p + r + p
image ← địa chỉ của image
PAD ← địa chỉ của PADimage
pos ← 0                                   ▷ Biến để dịch con trỏ image để lấy giá trị
for r ← p to length - 1 do
    for c ← p to length - 1 do
        vitri ← r × length + c            ▷ Tính vị trí dịch để lấy giá trị
        PAD[vitri] ← image[pos]           ▷ Di chuyển giá trị từ image vào PAD
        pos ← pos + 1
    end for
end for
```

☐ Tích chập ma trận `PADimage` với ma trận `kernel` để được ma trận đầu ra `out`.

1. **Duyệt các khối con:** Duyệt qua các ô trong `PADimage` theo từng khối con kích thước $p \times p$.
2. **Tính tích chập trên mỗi khối con:** Tại mỗi khối con, tính tích chập bằng cách nhân từng phần tử trong `PADimage` với phần tử tương ứng trong `kernel`, rồi cộng dồn kết quả vào biến `sum`.
3. **Lưu kết quả:** Sau khi tính xong mỗi khối con, lưu giá trị tổng `sum` vào vị trí hiện tại trong `out`.
4. **Kiểm tra điều kiện:** Tiếp tục Lặp lại quy trình cho đến khi duyệt hết các khối con của `PADimage`.

Algorithm 5 Pseudocode: Thực hiện phép tích chập với ma trận `PADimage` và lưu kết quả vào ma trận đầu ra

```
PAD ← địa chỉ của PADimage
position ← 0                                          ▷ Biến để lưu vị trí trong ma trận đầu ra
for strRow ← 0 to length step p do
    if strRow + p ≥ length then
        break                                         ▷ Xét điều kiện thoả mãn cho số hàng phải dài đủ p
    end if
    for strCol ← 0 to length step p do
        if strCol + p ≥ length then
            break                                     ▷ Xét điều kiện thoả mãn cho số cột phải dài đủ p
        end if
        sum ← 0
        kernel ← địa chỉ của kernel
        pos ← 0
        for r ← strRow to length and r < strRow + p do
            for c ← strCol to length and c < strCol + p do
                vitri ← r × length + c                ▷ Tính vị trí trong PADimage
                sum ← sum + kernel[pos] × PAD[vitri]
            end for
            pos ← pos + 1                             ▷ Dịch con trỏ kernel để lấy phần tử tiếp theo
        end for
        out[position] ← sum                           ▷ Lưu kết quả vào ma trận đầu ra
        position ← position + 1
    end for
end for
```

☐ Làm tròn một chữ số cho các phần tử có trong ma trận out.

Duyệt từng phần tử trong ma trận `out`.

1. **Phần tử không âm:** Nếu phần tử trong ma trận là số dương hoặc bằng 0, tính phần dư của phần tử cho 10.
   - Nếu phần dư từ 5 trở lên, tăng giá trị phần tử lên đến bội số của 10 lớn hơn.
   - Nếu phần dư nhỏ hơn 5, giảm giá trị xuống bội số của 10 gần nhất nhỏ hơn hoặc bằng.
2. **Phần tử âm:** Tạm thời chuyển số âm thành số dương để tính phần dư, làm tròn tương tự số dương rồi chuyển kết quả mới có được thành số âm.

Algorithm 6 Pseudocode: Làm tròn các phần tử trong ma trận `out` đến hàng chục gần nhất

```
out ← địa chỉ của ma trận đầu ra
length ← số phần tử của ma trận out
for i ← 0 to length - 1 do
    remainder ← abs(out[i]) mod 10
    if remainder ≥ 5 then
        value ← abs(out[i]) − remainder + 10
    else
        value ← abs(out[i]) − remainder
    end if
    if out[i] < 0 then
        out[i] ← value × −1
    end if
end for
```

4 Xử lý đầu ra

☐ Chuyển các phần tử trong mảng thành các chuỗi để bỏ vào chuỗi `outStr`.

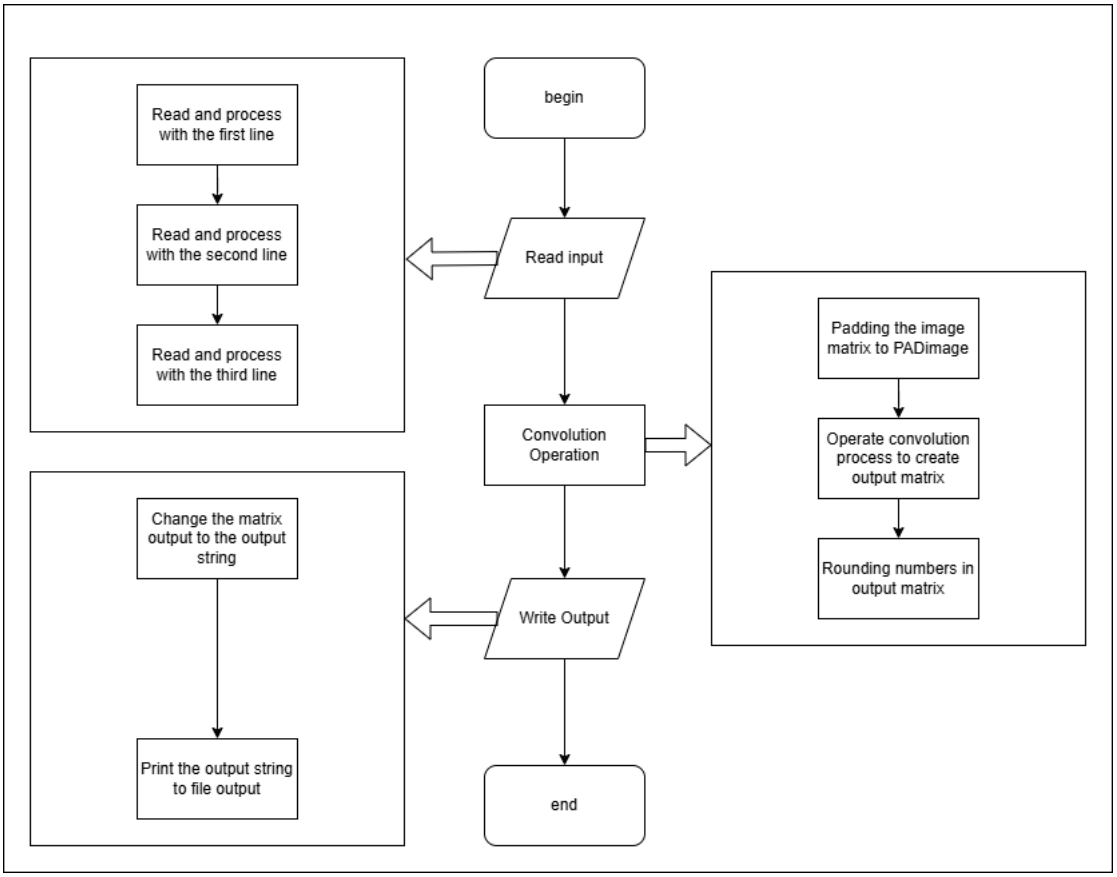
1. **Chuyển số thành chuỗi** Chuyển các chữ số của phần tử tương ứng trong ma trận `out` thành chuỗi `reverseStr`. Vì cần chuyển số thành chuỗi số thực nên cần thêm dấu chấm trong lúc tạo chuỗi, và thêm ký tự dấu trừ `-` vào chuỗi cho trường hợp số âm.
2. **Nối chuỗi cho `outStr`:** Đảo ngược `reverseStr` để có chuỗi số đúng thứ tự và thêm nó vào `outStr` và thêm dấu cách giữa các chuỗi số, trừ chuỗi số cuối cùng được kết thúc bằng ký tự `"\0"`.

Algorithm 7 Pseudocode: Chuyển đổi từng số trong ma trận `out` thành chuỗi và lưu vào chuỗi kết quả `outStr`

```
reverseStr ← chuỗi để lưu các chữ số của một số
outStr ← địa chỉ của chuỗi đầu ra
out ← địa chỉ của ma trận đầu ra
for i ← 0 to out.length - 1 do
    val ← out[i]
    so_thu ← 0
    if val ≠ 0 then                          ▷ Trường hợp phần tử của mảng đang xét khác 0
        sign ← 1
        if val < 0 then
            sign ← −1
        end if
        while val ≠ 0 do                     ▷ Vòng lặp để lấy từng chữ số của số tạo chuỗi
            remainder ← val % 10
            val ← val / 10
            reverseStr ← reverseStr + to_string(remainder)
            if so_thu = 0 then
                reverseStr ← reverseStr + "."
                so_thu ← so_thu + 1
                if val = 0 then
                    reverseStr ← reverseStr + "0"
                    so_thu ← so_thu + 1
                    break
                end if
            end if
            so_thu ← so_thu + 1
        end while
    else                                     ▷ Trường hợp phần tử của mảng đang xét là 0
        reverseStr ← "0.0"
        so_thu ← 3
    end if
    if sign = -1 then                        ▷ Nếu là số âm thì thêm ký tự "-"
        reverseStr ← reverseStr + "-"
        so_thu ← so_thu + 1
    end if
    for j ← so_thu - 1 downto 0 do
        outStr ← outStr + reverseStr[j]
    end for
    if (không phải số cuối cùng) then
        outStr ← outStr + " "
    else
        outStr ← outStr + "\0"
    end if
end for
```

☐ Mở file `output_matrix.txt` để in chuỗi `outStr` và kết thúc chương trình.

5 Diễn giải thuật toán bằng flowchart



Kiểm thử và đánh giá

1 Kiểm thử

1.1 Testcase 1

```
Input 1:
6.0 4.0 3.0 3.0
2.1 -2.1 2.1 -2.1 2.1 -2.1 2.1 -2.1 2.1 -2.1 2.1 -2.1 1.9 0.1 2.3 -1.1 0.0 8.7
5.3 3.5 1.5 0.9 7.1 -1.2 -1.2 -1.2 -1.2 -2.1 3.4 1.2 2.2 9.9 9.1 0.3 -2.1 1.1
-3.3 3.4 5.5 6.7 -7.7 6.9 0.0 0.0 7.1 3.3 7.0 0.0 0.0 0.0 0.0 0.0
```

```
Output 1:
0.0 0.0 0.0 14.1 -17.3 86.3 35.5 121.6 57.1
```

1.2 Testcase 2

```
Input 2:
3.0 4.0 0.0 3.0
4.5 5.6 2.3 7.9 -3.4 6.1 -1.9 9.2 4.2
-3.3 3.4 5.5 6.7 -7.7 6.9 0.0 0.0 7.1 3.3 7.0 0.0 0.0 0.0 0.0 0.0
```

```
Output 2:
Error: size not match
```

1.3 Testcase 3

```
Input 3:
4.0 4.0 0.0 3.0
4.5 5.6 2.3 7.9 -3.4 6.1 -1.9 9.2 4.2 3.5 1.5 0.9 7.1 -1.2 2.2 -1.2
-3.3 3.4 5.5 6.7 -7.7 6.9 0.0 0.0 7.1 3.3 7.0 0.0 0.0 0.0 0.0 0.0
```

```
Output 3:
189.9
```

1.4 Testcase 4

```
Input 4:
5.0 3.0 0.0 1.0
7.6 -5.5 2.1 0.0 -3.0 4.0 1.0 3.9 1.0 -7.2 9.1 3.2 1.4 5.4 3.2 0.0 3.9 1.0 1.0
3.1 4.6 -1.2 1.6 -1.9 4.0
1.5 6.0 -12.0 0.0 7.1 0.0 -4.2 6.1 1.3
```

```
Output 4:
-56.6 34.2 77.5 13.0 13.9 142.5 19.2 -32.2 -9.9
```

1.5 Testcase 5

```
Input 5:
5.0 3.0 0.0 1.0
1.2 -1.2 1.2 -1.2 1.2 -1.2 1.2 -1.2 1.2 -1.2 1.2 -1.2 1.2 -1.2 1.2 -1.2 1.2 -1.2
1.2 1.2 -1.2 1.2 -1.2 1.2 -1.2
-0.0 -0.0 -0.0 -0.0 2.2 3.3 4.4 5.5 6.6
```

```
Output 5:
5.3 -5.3 5.3 -5.3 5.3 10.6 -7.9 7.9 0.0
```

2 Đánh giá

2.1 Ưu điểm của MIPS trong phép tích chập

- **Hiệu suất cao:** MIPS cho phép truy cập phần cứng trực tiếp, tối ưu hiệu suất cho các ứng dụng tính toán thời gian thực.
- **Kiểm soát tài nguyên:** Có thể quản lý tài nguyên phần cứng chi tiết, phù hợp với các hệ thống nhúng.
- **Tối ưu tính toán:** MIPS sử dụng kiến trúc RISC và pipeline, giúp tăng tốc độ xử lý các phép tính.

2.2 Nhược điểm của MIPS trong phép tích chập

- **Mã lệnh dài và khó viết:** Cần nhiều dòng mã để thực hiện một phép toán đơn giản, làm tăng độ phức tạp.
- **Dễ mắc lỗi và khó bảo trì:** Việc viết mã thủ công ở cấp độ thấp khiến dễ xảy ra lỗi và khó kiểm tra, bảo trì.
- **Thiếu thư viện hỗ trợ:** MIPS không có sẵn thư viện cho các phép toán phức tạp như ma trận hay cho việc thao tác và chuyển đổi với chuỗi.

Kết luận

Phép toán tích chập (convolution) là một thành phần quan trọng trong học sâu, đặc biệt trong mạng nơ-ron tích chập (CNN). Nó giúp mạng học và trích xuất các đặc trưng có giá trị từ dữ liệu hình ảnh, hỗ trợ trong nhiều ứng dụng như nhận diện đối tượng, phân loại ảnh và phân tích video.

Báo cáo này đã trình bày chi tiết các bước thực hiện phép toán tích chập, từ việc xử lý đầu vào, áp dụng padding, đến việc thực hiện tích chập để tạo ra ma trận đầu ra. Các bước này đều đóng vai trò then chốt trong việc xây dựng đoạn mã.

Tích chập không chỉ là một phép toán cơ bản mà còn là công cụ mạnh mẽ trong các mô hình học sâu hiện đại, đóng góp vào sự phát triển của các ứng dụng xử lý hình ảnh và video, đồng thời giúp cải thiện khả năng tối ưu hóa và tùy chỉnh mạng nơ-ron.