ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA KHOA HỌC VÀ KỸ THUẬT MÁY TÍNH

KIẾN TRÚC MÁY TÍNH

Báo cáo Bài tập lớn

SV thực hiện: Lê Nguyễn Kim Khôi - 2311671
Lớp: TN01

Tp. Hồ Chí Minh, Tháng 11/2024

Mục lục

1 Giới thiệu đề tài

Trí tuệ nhân tạo đang phát triển nhanh chóng, đặc biệt trong các lĩnh vực như học máy và học sâu. Một trong những loại mạng phổ biến nhất trong học sâu là mạng nơ-ron tích chập (CNN), được sử dụng rộng rãi trong phân tích hình ảnh và video. CNN hoạt động dựa trên các phép toán ma trận, đặc biệt là các phép toán tích chập, để trích xuất các đặc trưng từ dữ liệu đầu vào. Quá trình này giúp CNN nhận dạng và phân tích các đặc trưng như cạnh, kết cấu và hoa văn trong hình ảnh. Với nhiều lớp tích chập, CNN có thể học các đặc trưng ngày càng phức tạp, làm tăng hiệu suất trong các nhiệm vụ nhận dạng và phân tích hình ảnh.

Vì vậy ở bài tập lớn lần này thì chúng ta sẽ tiếp cận CNN thông qua việc giải bài toán này. Chúng ta thực hiện tính toán tính chập trên ma trận ảnh bằng cách dịch chuyển ma trận kernel trên trên ma trận ảnh với bước di chuyển được quy định – stride. Và trước khi dịch chuyển cần xác định được cần phải mở rộng ma trận ảnh ra không bằng chỉ số padding được cung cấp ban đầu. Và chương trình hiện thực sẽ nhận đầu vào theo input file được đặt tên là `input_matrix.txt` bao gồm các thông tin sau đây, được trình ghi trên cùng 1 hàng và cách nhau bởi dấu cách:

- **N**: Kích thước ma trận ảnh(image). ($3 \leq N \leq 7$)
- **M**: Kích thước ma trận (kernel).($2 \leq M \leq 4$)
- **p**: Giá trị mở rộng(padding).($0 \leq p \leq 4$)
- **s**: Giá trị dịch chuyển(stride).($1 \leq s \leq 3$)

Ở hai hàng tiếp theo chính là lần lượt các giá trị của ma trận image và ma trận kernel. Và tất cả các số được lưu trong file này đều là các số chấm động với 1 chữ số thập phân duy nhất ở sau cùng.

Yêu cầu cần xuất ra file ma trận kết quả – out theo cách tính toán trên. Kèm theo đó là một số điều kiện quy định về khởi tạo các biến image, kernel và out với kiểu word lần lượt là nơi lưu trữ ma trận image, kernel và ma trận kết quả

2 Ý tưởng thực hiện

2.1 Bước 1: Đọc dữ liệu từ file đầu vào

2.1.1 Mở file đầu vào để đọc

- Sử dụng syscall để mở file.
- Sau đó lưu trữ vào biến buffer để thực hiện việc đọc xuất sau này.
- Đóng file đọc.

2.1.2 Đọc và phân tích các tham số (N, M, p, s)

- Đọc dòng đầu tiên từ file chứa các tham số *N*, *M*, *p*, *s*.
- Phân tích các giá trị này và lưu vào các biến tương ứng.

Lưu ý: Ngay tại vị trí này có thể thực hiện được việc xem xét các tham số N, M, p, s có phù hợp để tính toán hay không. Cụ thể trong trường hợp này nếu N + 2p $<$ M thì sẽ trực tiếp bỏ qua tất cả các bước tính toán tiếp theo và xuất ra thông báo **Error: size not match** trong file đầu ra.

2.1.3 Đọc ma trận ảnh

- Đọc ma trận ảnh từ file và lưu vào vùng nhớ *image*.

Giải thích: Việc đọc ma trận sẽ được thực hiện bằng cách xử lý vòng lặp đồng thời sử dụng thêm 1 thanh ghi s0 làm biến cờ để xét số âm dương và thêm một vòng lặp trong để đọc các số lớn có nhiều chữ số phần nguyên.

2.1.4 Đọc ma trận kernel

- Đọc ma trận kernel từ file và lưu vào vùng nhớ *kernel*.

Giải thích: Phương pháp xử lý tương tự như việc đọc ma trận ảnh.

2.2 Bước 2: Tạo ma trận đệm từ padding và kernel

2.2.1 Khởi tạo ma trận đệm với giá trị 0

- Khởi tạo tất cả các phần tử trong ma trận đệm thành 0. Ở đây ma trận có kích thước mở rộng được tính toán theo công thức *N + 2p*.

2.2.2 Sao chép ma trận ảnh vào ma trận đệm

- Sao chép các phần tử của ma trận ảnh vào vị trí thích hợp trong ma trận đệm.

2.3 Bước 3: Thực hiện phép tích chập (convolution)

2.3.1 Thiết lập các tham số cho phép tích chập

- Thanh ghi *t3* sẽ lưu giá trị *N + 2p - s* sẽ làm giá trị giới hạn vị trí mà ma trận kernel có thể dịch chuyển trên ma trận ảnh.

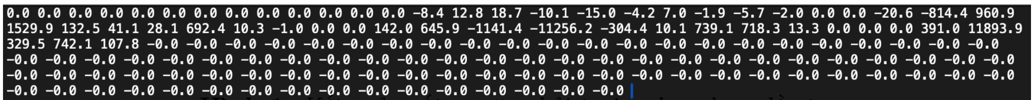
2.3.2 Tính tích chập cho từng phần tử của ma trận kết quả

- Duyệt qua từng phần tử của ma trận kết quả.
- Với mỗi phần tử, tính tổng tích chập của các phần tử tương ứng trong ma trận đệm và ma trận kernel.
- Lưu kết quả vào ma trận kết quả.

2.3.3 Làm tròn thập phân

- Các số thập phân được làm tròn để có thể xử lý các kết quả tính toán về đúng yêu cầu là một chữ số thập phân sau dấu chấm.

Giải thích: Việc làm tròn này là một trong những phân đoạn quan trọng của đề bài khi mà cần phải làm tròn các chữ số thập phân lấy ví dụ như là 4.54 thì sẽ về 4.5 còn 4.56 thì sẽ về 4.6. Còn đối với các số âm thì việc làm tròn sẽ bằng cách lấy trị tuyệt đối, làm tròn như cách làm tròn số dương sau đó trả dấu âm về cho giá trị đã được làm tròn.
Lưu ý: Tuy nhiên sẽ phát sinh một số trường hợp đặc biệt cụ thể là nếu tính toán ra các giá trị nhỏ hơn -0.05 thì kết quả làm tròn ở phép tính trên sẽ trả về giá trì -0.0 khi được xuất ra file kết quả. Vì vậy cần phải quản lý trường hợp đặc biệt này về 0.0, tránh để cho sai sót trong quá trình hiện thực.



Hình 1: *Kết quả xuất ra gặp phải trường hợp được đề cập.*

2.4 Bước 4: Lưu ma trận kết quả vào file đầu ra

2.4.1 Ghi các phần tử của ma trận kết quả vào buffer

- Duyệt qua từng phần tử của ma trận kết quả và ghi vào buffer.

Giải thích: Phương pháp xử lý ở đây cần phải xử lý cụ thể khi đó là các số thực được xuất ra ngoài nên phải phân biệt ra từng thành phần trong số được trích ra từ buffer bao gồm dấu, phần nguyên, dấu chấm động và phần thập phân.

2.4.2 Xuất ra file kết quả

- Mở file sau đó đưa buffer ra và lưu vào file kết quả, sau đó đóng file.

3 Kết quả chạy

- Testcase 1:

```
5.0 3.0 1.0 1.0
-1.2 1.5 2.1 0.0 0.0 0.0 1.0 1.0 -1.0 0.0 0.0 0.0 1.0 1.0 1.0 0.0 0.0 1.0 1.0 0.0 0.0 1.0 1.0 0.0 0.0
1.0 0.0 -1.6 0.0 1.0 0.0 -1.0 0.0 1.0
```

- Kết quả testcase 1:

```
-0.2 2.5 -0.1 -1.0 -1.0 -2.4 -2.6 3.5 1.1 -1.0 -1.6 -0.6 4.6 -1.0 -1.0 -1.0 -0.6 -1.6 -0.6 -1.0 0.0 -0.6 -0.6 -1.0 -1.0
```

- Testcase 2:

```
5.0 3.0 2.0 3.0
-1.2 1.5 2.1 0.0 0.0 0.0 1.0 1.0 -1.0 -16.0 0.0 0.0 1.0 1.0 17.0 0.0 0.0 1.0 1.0 0.0 0.0 1.0 1.0 0.0 0.0
1.0 0.0 -1.6 0.0 1.0 0.0 -1.0 0.0 1.0
```

- Kết quả testcase 2:

```
-3828.8 -1.5 0.0 0.0 4.6 -16.0 0.0 -1.0 0.0
```

- Testcase 3:

```
4.0 4.0 0.0 3.0
-1.2 1.5 2.1 0.0 0.0 0.0 1.0 1.0 -1.0 -16.0 0.0 0.0 1.0 1.0 17.0 0.0
1.0 0.0 -1.6 0.0 1.0 0.0 -1.0 0.0 1.0 18.0 900.0 23.0 -2.0 -5.7 -1.9 7.0
```

- Kết quả testcase 3:

```
-334.6
```

- Testcase 4:

```
7.0 4.0 4.0 1.0
-1.2 1.5 2.1 0.0 0.0 0.0 1.0 1.0 -1.0 -16.0 0.0 0.0 1.0 1.0 17.0 0.0 1.0 0.0 -1.6 0.0 1.0 0.0 -1.0 0.0 1.0 18.0
900.0 23.0 -2.0 -5.7 -1.9 7.0 -16.0 0.0 0.0 1.0 1.0 17.0 0.0 1.0 37.0 -26.8 -5.7 -1.9 7.0 9.6 -8.4 7.5 6.3
1.0 0.0 -1.6 0.0 1.0 0.0 -1.0 0.0 1.0 18.0 700.0 23.0 -2.0 -5.7 -1.9 7.0
```

- Kết quả testcase 4:

```
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 -8.4 12.8 18.7 -10.1 -15.0 -4.2 7.0 -1.8
-5.7 -1.9 0.0 0.0 -20.6 -814.4 960.9 1529.9 132.5 41.1 28.1 692.4 10.3 -0.9 0.0 0.0 142.0 645.9
-1141.4 -11256.2 -304.4 10.1 739.1 718.3 13.3 0.0 0.0 0.0 391.0 11893.9 329.5 742.1 107.8 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
```

- Testcase 5:

```
3.0 4.0 0.0 1.0
-1.2 1.5 2.1 0.0 0.0 0.0 1.0 1.0 -1.0
1.0 0.0 -1.6 0.0 1.0 0.0 -1.0 0.0 1.0 18.0 700.0 23.0 -2.0 -5.7 -1.9 7.0
```

- Kết quả testcase 5:

```
Error: size not match
```

4 Tổng kết và đánh giá

4.1 Ưu điểm

- Hiện thực của bài toán trên được cho thấy hiệu quả khi cover gần như đầy đủ tất cả các trường hợp. Bởi vì chưa có thể tạo tất cả các vùng testcase để có thể đánh giá code được hiện thực ra sẽ đảm bảo bao phủ tất cả các trường hợp. Cùng với cách làm phân từng đoạn code ra thành từng phân mục nhỏ giúp dễ dàng quản lý và tránh bị lỗi trong quá trình hiện thực.

4.2 Khuyết điểm

- Bởi vì hiện thực code theo từng phân đoạn nhỏ giúp dễ quản lý và tránh bị chồng chéo biến ở mỗi thao tác nên số lượng lệnh được gọi ra sẽ bị nhiều hơn dẫn đến việc tuy code hoạt động hiệu quả nhưng chưa được tối ưu về mặt tốc độ nếu chạy ở các testcase vượt ngoài phạm vi của bài toán đã đề ra.