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

BÁO CÁO

ĐỒ ÁN MÔN HỌC KỸ THUẬT MÁY TÍNH

Thiết kế bộ tăng tốc kết hợp CNN-LSTM trên thiết bị biên dựa trên RISC-V

Ngành: Kỹ thuật máy tính

HỘI ĐỒNG: Kỹ thuật Máy tính 7

GVHD: PGS.TS Trần Ngọc Thịnh
GVHD: Huỳnh Phúc Nghị
TKHĐ: Huỳnh Phúc Nghị

SVTH 1: Lê Công Tuấn - 2115169
SVTH 2: Lê Thanh Toàn Đạt - 2115386

LỜI CAM ĐOAN

Chúng tôi khẳng định rằng mọi nội dung và thông tin trong báo cáo này đều hoàn toàn trung thực và được thực hiện một cách độc lập. Chúng tôi đảm bảo không có hành vi gian lận hay sử dụng thông tin từ các nguồn khác mà không được ghi chú rõ ràng trong tài liệu. Mọi tham khảo đến công trình của người khác đều được trích dẫn đầy đủ và các nguồn thông tin đều được trình bày chính xác theo quy định của hệ thống tham chiếu đã được áp dụng.

Chúng tôi nhận thức rõ ràng việc vi phạm các nguyên tắc về trung thực và minh bạch có thể gây ra những hậu quả nghiêm trọng. Vì vậy, chúng tôi cam kết tuân thủ chặt chẽ mọi quy tắc và chuẩn mực đạo đức liên quan đến việc nghiên cứu và lập báo cáo.

Nhóm thực hiện

LỜI CẢM ƠN

Chúng tôi xin bày tỏ lòng biết ơn sâu sắc đến tất cả những người đã đồng hành, hỗ trợ và động viên chúng tôi trong suốt quá trình nghiên cứu và thực hiện đồ án này. Trước tiên, chúng tôi xin gửi lời cảm ơn chân thành và trân trọng nhất đến thầy PGS.TS Trần Ngọc Thịnh và ThS Huỳnh Phúc Nghị, hai người thầy đã tận tâm hướng dẫn, hỗ trợ và chia sẻ những kinh nghiệm quý báu để giúp chúng tôi từng bước hoàn thiện đề tài. Những lời khuyên, định hướng và sự động viên của thầy không chỉ giúp chúng tôi hiểu sâu sắc hơn về đề tài mà còn truyền cảm hứng để chúng tôi vượt qua những khó khăn trong quá trình thực hiện.

Chúng tôi cũng xin gửi lời tri ân đến các thầy cô trong Khoa Khoa học và Kỹ thuật Máy tính, Bộ môn Kỹ thuật Máy tính, Trường Đại học Bách Khoa - Đại học Quốc gia TP. Hồ Chí Minh, những người đã tận tụy dìu dắt, giảng dạy và truyền đạt tri thức cho chúng tôi ngay từ những ngày đầu tiên bước vào giảng đường đại học. Nền tảng kiến thức vững chắc mà các thầy cô trang bị là yếu tố không thể thiếu để chúng tôi thực hiện đề tài này.

Bên cạnh đó, chúng tôi cũng không thể không nhắc đến gia đình, những người luôn là điểm tựa vững chắc, nguồn động lực to lớn và bạn bè thân thiết, những người đã luôn ủng hộ, chia sẻ và đồng hành cùng chúng tôi trong suốt hành trình. Tất cả sự quan tâm, động viên và hỗ trợ của mọi người là nguồn sức mạnh quý giá để chúng tôi hoàn thành đồ án này.

Chúng tôi xin chân thành cảm ơn tất cả!

TÓM TẮT NỘI DUNG

Trong thời gian gần đây, sự phát triển mạnh mẽ của mạng lưới vạn vật kết nối (Internet of Things - IoT) đã mang lại nhiều cải tiến đáng kể cho cuộc sống con người, từ nhà thông minh, chăm sóc sức khỏe, đến quản lý giao thông và sản xuất công nghiệp. Tuy nhiên, sự tiến triển nhanh chóng này cũng đặt ra những thách thức không nhỏ về năng lượng, diện tích phần cứng, tốc độ xử lý và độ chính xác, tùy thuộc vào từng ứng dụng cụ thể.

Theo các báo cáo mới nhất, hiện có khoảng 15 tỷ thiết bị IoT trên toàn cầu, và con số này được dự đoán sẽ tăng lên 30 tỷ vào năm 2030. Sự bùng nổ về số lượng thiết bị IoT, cùng với khối lượng dữ liệu khổng lồ mà chúng tạo ra, đang gây áp lực đáng kể lên cơ sở hạ tầng tính toán, bao gồm cả các máy chủ tại chỗ và dịch vụ đám mây. Điều này dẫn đến nhu cầu cấp thiết phải tối ưu hóa các hệ thống xử lý dữ liệu, cải thiện hiệu quả năng lượng và mở rộng khả năng tính toán tại biên (edge computing), nhằm giảm tải cho các máy chủ trung tâm và tăng cường khả năng phản hồi thời gian thực.

Một trong những giải pháp mới được phát triển để giải quyết vấn đề này là tích hợp Trí tuệ Nhân tạo (AI) vào các thiết bị biên để xử lý dữ liệu ngay tại điểm thu thập trước khi chúng được gửi đến máy chủ. Tuy nhiên, các thiết bị biên này cần có khả năng xử lý nhanh chóng lượng dữ liệu lớn và mật độ tại môi trường cụ thể. Trong đồ án này, nhóm chúng tôi đề xuất một giải pháp tăng tốc xử lý CNN-LSTM trên thiết bị biên, sử dụng kiến trúc tập lệnh mở RISC-V. Đồ án được thực hiện qua hai giai đoạn. Trong giai đoạn một, chúng tôi tập trung khảo sát các nghiên cứu hiện đại trên thế giới, và từ đó đề xuất một giải pháp cho dự án của chúng tôi. Giai đoạn tiếp theo sẽ bao gồm việc xây dựng một giải pháp toàn diện trên thiết bị, đáp ứng các yêu cầu về năng lượng và tốc độ xử lý của thiết bị biên.

Mục lục

Danh mục hình ảnh

CHƯƠNG 1
GIỚI THIỆU ĐỀ TÀI

1.1 Mô tả đề tài

Tên đề tài: Thiết kế bộ tăng tốc kết hợp CNN-LSTM trên thiết bị biên dựa trên RISC-V.

RISC-V là một kiến trúc tập lệnh (ISA) phần cứng mã nguồn mở, phát triển dựa trên triết lý của máy tính tập lệnh rút gọn (RISC). Với thiết kế đơn giản nhưng linh hoạt, RISC-V không chỉ giảm thiểu sự phức tạp trong triển khai mà còn hỗ trợ khả năng tùy chỉnh, mở rộng hiệu quả cho nhiều ứng dụng hiện đại. Nhờ những đặc tính này, RISC-V ngày càng được ứng dụng rộng rãi trong lĩnh vực trí tuệ nhân tạo (AI), Internet of Things (IoT) và các hệ thống nhúng.

Đề tài này tập trung vào thiết kế một bộ tăng tốc phần cứng tích hợp CNN-LSTM trên nền tảng RISC-V, nhằm tận dụng lợi thế của kiến trúc mở và tối ưu năng lượng. Mục tiêu của đề tài nhằm giảm tải tính toán cho các hệ thống máy chủ trong các hệ thống, đồng thời cải thiện hiệu năng xử lý.

1.2 Giới thiệu vấn đề

Học sâu (deep learning) là một nhánh quan trọng của trí tuệ nhân tạo (Artificial Intelligence - AI), với mục tiêu xây dựng các thuật toán và mô hình có khả năng tự học từ dữ liệu mà không cần lập trình cụ thể. Các mô hình học sâu đã được áp dụng thành công trong nhiều lĩnh vực như xử lý ngôn ngữ tự nhiên, nhận diện hình ảnh, y học, tài chính và hơn thế nữa. Tuy nhiên, một thách thức lớn của học sâu là nhu cầu sử dụng tài nguyên tính toán đáng kể, đặc biệt khi triển khai trên các hệ thống thực tế.

Trong bối cảnh đó, nhiều nền tảng phần mềm học sâu như TensorFlow, PyTorch hay Keras đã được phát triển để hỗ trợ quá trình xây dựng và huấn luyện mô hình. Tuy nhiên, các nền tảng này chủ yếu thực thi mô hình trên CPU hoặc GPU, dẫn đến hạn chế về hiệu năng và tiêu thụ năng lượng. Sự tuần tự trong xử lý hoặc khả năng song song hóa hạn chế làm tăng thời gian suy luận và giảm hiệu quả sử dụng năng lượng, đặc biệt trong các ứng dụng thời gian thực.

Để khắc phục những hạn chế này, việc tăng tốc các mô hình học sâu trên phần

cứng như FPGA hay ASIC đã trở thành hướng nghiên cứu đầy triển vọng. Phần cứng được tùy chỉnh không chỉ cải thiện tốc độ xử lý mà còn tối ưu hóa mức tiêu thụ năng lượng, giúp các hệ thống học sâu hoạt động hiệu quả hơn.

Cùng lúc đó, nhu cầu xử lý dữ liệu phức tạp từ các thiết bị biên (edge devices) đang ngày càng tăng. Thay vì gửi dữ liệu về máy chủ trung tâm, việc tích hợp AI trực tiếp trên các thiết bị biên giúp xử lý dữ liệu tại nguồn, giảm thiểu độ trễ và áp lực lên hạ tầng mạng. Tuy nhiên, các thiết bị biên thường bị giới hạn bởi chi phí, không gian và năng lượng, đòi hỏi giải pháp tối ưu để cân bằng giữa hiệu năng và các ràng buộc phần cứng.

Đề tài này tập trung vào thiết kế bộ tăng tốc phần cứng tích hợp CNN-LSTM trên các thiết bị biên, sử dụng kiến trúc RISC-V. Với sự kết hợp của CNN để xử lý đặc trưng không gian và LSTM để phân tích chuỗi thời gian, giải pháp này hướng tới tối ưu hóa hiệu năng xử lý học sâu trên thiết bị biên, đồng thời giải quyết các thách thức liên quan đến tài nguyên và tiêu thụ năng lượng.

1.3 Cấu trúc đồ án

Cấu trúc đồ án được chia thành 5 phần với nội dung chính như sau:

- **Phần 1: Giới thiệu chung**. Phần này giới thiệu chung về đồ án bao gồm m ô tả đề tài, giới thiệu vấn đề xung quanh đồ án và cuối cùng là cấu trúc đồ án.
- **Phần 2: Cơ sở lý thuyết**. Phần này gồm các khảo sát về kiến trúc tập lệnh mã nguồn mở RISC-V, lý thuyết về học sâu và các mạng nơ-ron.
- **Phần 3: Công trình nghiên cứu liên quan**. Phần này bao gồm các khảo sát về những nghiên cứu về tăng tốc xử lý CNN-LSTM trên phần cứng và một số lõi RISC-V trên thị trường.
- **Phần 4: Đề xuất hệ thống.** Sau khi thực hiện khảo sát, nhóm đề xuất các bước tăng tốc xử lý suy luận CNN-LSTM trên phần cứng.
- **Phần 5: Kết luận.** Nhóm đề ra những mục tiêu cần đạt được trong giai đoạn tiếp theo và kế hoạch thực hiện.

CHƯƠNG 2

CƠ SỞ LÝ THUYẾT

2.1 RISC-V

2.1.1 Giới thiệu

RISC-V là một kiến trúc tập lệnh mã nguồn mở (Instruction Set Architecture - ISA) thuộc họ kiến trúc RISC (Reduced Instruction Set Computing). Ban đầu, RISC-V được thiết kế để phục vụ nghiên cứu và giáo dục trong lĩnh vực kiến trúc máy tính, nhưng nhờ tính linh hoạt và hiệu quả, nó nhanh chóng trở thành nền tảng phổ biến cho cả học thuật và công nghiệp.

RISC-V sử dụng thứ tự byte là little-endian để đảm bảo byte trọng số thấp (Least Significant Byte - LSB) được lưu trữ tại địa chỉ thấp nhất. Đây là một quyết định thiết kế để tương thích với các tiêu chuẩn phần mềm và phần cứng phổ biến.

RISC-V áp dụng kỹ thuật nén mã lệnh RVC (RISC-V Compressed Instructions) nhằm tối ưu hóa kích thước chương trình và giảm số chu kỳ CPU cần thiết để thực thi mỗi lệnh, mặc dù điều này có thể làm tăng tổng số lệnh trong chương trình. Giải pháp này giúp thu gọn chương trình, đồng thời hỗ trợ đơn giản hóa quá trình triển khai phần cứng.

Một số mục tiêu của RISC-V:

- Một ISA hoàn toàn mở, được cung cấp miễn phí cho cả giới học thuật và ngành công nghiệp.
- Một ISA thực tế, phù hợp để triển khai trực tiếp trên phần cứng gốc, không chỉ để mô phỏng hoặc dịch nhị phân.
- Một ISA tránh việc over-architecting để phù hợp với một kiểu vi kiến trúc cụ thể hoặc một công nghệ triển khai nhất định (ví dụ: ASIC, FPGA) nhưng vẫn cho phép triển khai hiệu quả trên bất kỳ nền tảng nào trong số này.
- Một ISA được phân tách thành một ISA cơ sở nhỏ gọn (base integer ISA), có thể sử dụng độc lập làm nền tảng cho các bộ tăng tốc tùy chỉnh hoặc cho mục đích giáo dục, cùng với các mở rộng tiêu chuẩn tùy chọn để hỗ trợ phát triển phần mềm đa dụng.

- Một ISA hỗ trợ mở rộng ISA một cách phong phú và có các biến thể chuyên biệt.
- Một ISA hỗ trợ triển khai đa lõi hoặc nhiều lõi song song, bao gồm cả bộ xử lý đa lõi không đồng nhất.
- Hỗ trợ tập lệnh có độ dài thay đổi tùy chọn, vừa giúp mở rộng không gian cho việc mã hóa tập lệnh, vừa hỗ trợ bổ sung các mã lệnh tùy chọn nhằm nâng cao hiệu suất, giảm kích thước mã tĩnh và tối ưu hóa năng lượng tiêu thụ.

Nhờ những mục tiêu nổi bật này, RISC-V không chỉ là lựa chọn hàng đầu cho các ứng dụng nhúng mà còn đóng vai trò quan trọng trong các lĩnh vực công nghệ cao như siêu máy tính, thiết bị biên (edge devices) và các hệ thống thông minh.

2.1.2 Ưu điểm

- Tính mở và tự do thiết kế (Openness & Freedom): RISC-V là một ISA mã nguồn mở, không thuộc sở hữu của bất kỳ công ty hay tổ chức nào, cho phép sử dụng, sửa đổi và chia sẻ tự do mà không cần trả phí bản quyền, tạo điều kiện phát triển và đổi mới trong cả học thuật lẫn công nghiệp.
- Tính đơn giản (Simplicity): RISC-V tuân thủ nguyên tắc RISC, sử dụng các lệnh đơn giản và tối ưu hóa để giảm số chu kỳ xung nhịp cần thiết. Điều này giúp kiến trúc dễ hiểu, dễ triển khai và hiệu quả trong sử dụng tài nguyên.
- Tính mô-đun (Modularity): RISC-V được thiết kế theo kiểu module với tập lệnh cơ bản ổn định và các phần mở rộng tùy chọn. Điều này cho phép các nhà thiết kế loại bỏ những tính năng không cần thiết, giảm chi phí và tăng tính linh hoạt trong các ứng dụng cụ thể.
- Khả năng mở rộng (Extensibility): RISC-V hỗ trợ từ các bộ vi điều khiển nhúng đến siêu máy tính, với khả năng tùy chỉnh lệnh cho các chức năng chuyên biệt như học máy, mật mã hoặc xử lý vectơ.
- Phù hợp đa dạng (Universality): RISC-V hoạt động hiệu quả trên các vi xử lý ở mọi kích cỡ, hỗ trợ nhiều kiểu triển khai như FPGA, ASIC, SoC, đồng thời tương thích với nhiều loại phần mềm và ngôn ngữ lập trình.
- Tính ổn định (Stability): ISA cơ bản của RISC-V được giữ ổn định và không thay đổi theo thời gian, đảm bảo phần mềm và công cụ hiện tại vẫn hoạt động tốt trên các hệ thống RISC-V trong tương lai.

- Tiết kiệm chi phí và tài nguyên (Cost-efficiency): Với tính đơn giản và khả năng tối ưu hóa cao, RISC-V giúp giảm thiểu chi phí thiết kế phần cứng và cải thiện hiệu suất năng lượng.

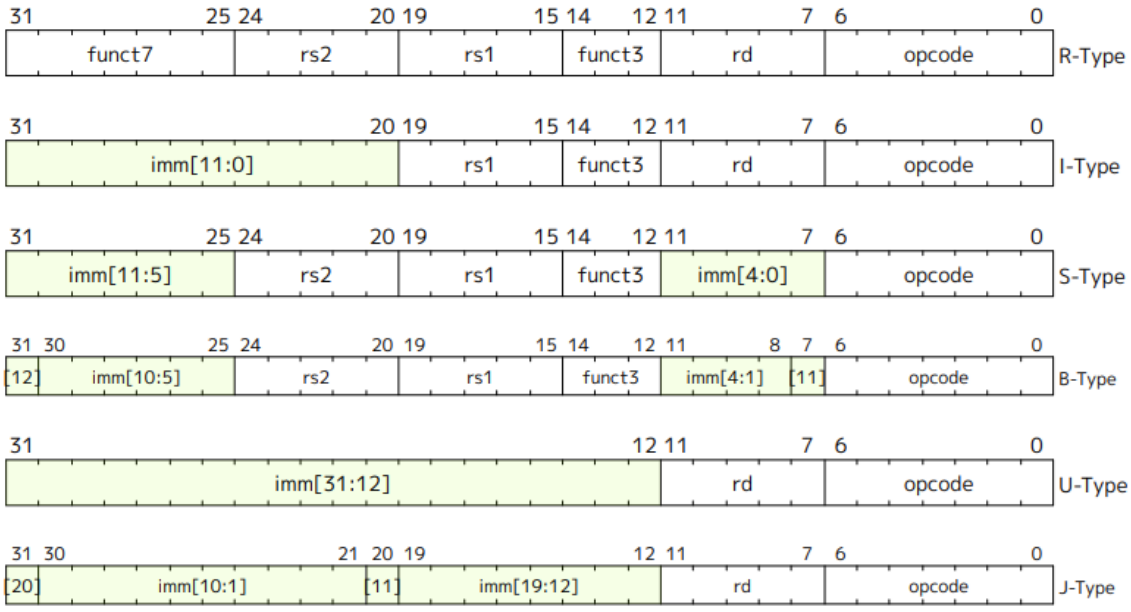
2.1.3 Các kiến trúc tập lệnh cơ bản của RISC-V

RISC-V bao gồm tập lệnh cơ bản nhỏ gọn, đủ để thực hiện các tác vụ cơ bản, kết hợp với các tập lệnh mở rộng tùy chọn như tính toán số học, dấu phẩy động, và đồng bộ hóa đa lõi. Một đặc điểm nổi bật của RISC-V là sự phân tách rõ ràng giữa các phiên bản 32-bit, 64-bit và 128-bit, tối ưu hóa cho từng mục đích sử dụng cụ thể.

RV32I - Base Integer ISA:

RV32I là một kiến trúc tập lệnh (ISA) cơ sở trong dòng RISC-V, được thiết kế để tối giản hóa phần cứng đồng thời đảm bảo khả năng biên dịch và hỗ trợ các hệ điều hành hiện đại. RV32I bao gồm 40 lệnh cơ bản, với mỗi thanh ghi có kích thước 32 bit, tạo ra một cấu trúc phần cứng đơn giản nhưng đủ mạnh mẽ để thực thi các lệnh trong môi trường hệ điều hành. Tất cả 32 thanh ghi trong RV32I, bao gồm thanh ghi x0 đặc biệt, đều có kích thước 32 bit, với x0 luôn chứa giá trị 0. Các thanh ghi x1 đến x31 là thanh ghi đa dụng, sử dụng để lưu trữ các giá trị Boolean hoặc các số nguyên có dấu hoặc không dấu theo chuẩn bù hai.

Trong ISA RV32I, một số lệnh có thể được sử dụng trong các môi trường hỗ trợ các mở rộng khác nhau của RISC-V, ngoại trừ phần mở rộng A (yêu cầu phần cứng bổ sung để hỗ trợ các lệnh nguyên tử). Tất cả các lệnh trong RV32I có độ dài cố định 32 bit và yêu cầu bộ nhớ phải được căn chỉnh trên ranh giới bốn byte. Cấu trúc lệnh của RV32I bao gồm các định dạng R, I, S, U, B, J.



Hình 2.1. 6 định dạng lệnh của RV32I [1]

Một điểm nổi bật trong thiết kế của RV32I là khả năng mở rộng và tính linh hoạt. RV32I có thể dễ dàng hỗ trợ các phần mở rộng ISA khác nhau như M (Multiplication), F (Single Precision Floating Point), hoặc Z (Custom Extensions). Mặc dù tập lệnh cơ sở đã được tối giản để giảm yêu cầu phần cứng, nó vẫn đảm bảo khả năng tương thích với nhiều ứng dụng và môi trường phần cứng, giúp RISC-V trở thành một nền tảng lý tưởng cho các thiết bị nhúng và các ứng dụng học thuật.

Bên cạnh đó, việc thiết kế RV32I không chỉ giúp tối ưu hóa phần cứng mà còn mở ra cơ hội nghiên cứu và cải tiến ISA trong tương lai, khi các phần mở rộng có thể được thêm vào để đáp ứng các yêu cầu phức tạp hơn mà không làm thay đổi cấu trúc cơ bản của tập lệnh.

RV64I - RISC-V Base Integer ISA:

RV64I là một biến thể của kiến trúc RISC-V được xây dựng trên nền tảng RV32I, mở rộng kích thước thanh ghi và không gian địa chỉ lên 64 bit. Điều này cho phép RV64I xử lý dữ liệu lớn và cung cấp khả năng tính toán chính xác hơn với các phép toán 64 bit. Dù tăng cường khả năng xử lý, RV64I vẫn giữ được sự đơn giản của tập lệnh RISC-V, giúp duy trì tính linh hoạt và hiệu suất cao trong các ứng dụng yêu cầu tính toán phức tạp.

RV32E - RISC-V Base Integer ISA for Embedded:

RV32E là một phiên bản rút gọn của kiến trúc tập lệnh RISC-V, được tối ưu hóa cho các hệ thống nhúng và vi điều khiển. Mặc dù nội dung tập lệnh của RV32E

gần như tương tự RV32I, điểm khác biệt quan trọng nằm ở việc giảm số lượng thanh ghi từ 32 xuống còn 16. Trong đó, 15 thanh ghi (x1 đến x15) là thanh ghi đa dụng, và 1 thanh ghi đặc biệt (x0) luôn giữ giá trị cố định là 0. Sự tinh giản này giúp RV32E tiết kiệm tài nguyên phần cứng và năng lượng, làm cho nó trở thành lựa chọn lý tưởng cho các ứng dụng nhúng có yêu cầu về hiệu suất và kích thước nhỏ gọn.

2.1.4 Các kiến trúc tập lệnh mở rộng của RISC-V

RV32M - Multiply and Divide:

RV32M bổ sung các lệnh nhân và chia số nguyên vào tập lệnh cơ sở RV32I hoặc RV32E, bao gồm:

- Các lệnh nhân như *mul, mulh, mulhsu, mulhu.*
- Các lệnh chia và lấy dư như *div, divu, rem, remu.*

Mục tiêu của RV32M là cải thiện hiệu suất xử lý trong các ứng dụng yêu cầu nhiều phép toán nhân, chia, chẳng hạn như xử lý tín hiệu số, mật mã, và các thuật toán tính toán phức tạp. Tuy nhiên, việc tách các lệnh này khỏi tập lệnh cơ bản giúp đơn giản hóa phần cứng cho các hệ thống không thường xuyên sử dụng phép toán này.

RV32F và RV32D - Single and Double-Precision Floating Poin:

RV32F hỗ trợ tính toán dấu chấm động có độ chính xác đơn (32 bit) và bổ sung 32 thanh ghi dấu chấm động (f0 đến f31), mỗi thanh rộng 32 bit. Thanh ghi trạng thái và điều khiển dấu chấm động fcsr lưu trạng thái hoạt động và ngoại lệ. RV32D mở rộng các thanh ghi dấu chấm động lên 64 bit, cho phép xử lý cả số thực đơn và kép, tuân thủ chuẩn IEEE 754-2008. Hai bộ thanh ghi riêng biệt (cho số nguyên và dấu chấm động) giúp tăng dung lượng và băng thông mà không làm tăng phức tạp tập lệnh, cải thiện hiệu suất xử lý.

RV32A - Atomic Instructions:

RV32A hỗ trợ các hoạt động nguyên tử để đồng bộ hóa trong môi trường đa luồng hoặc đa xử lý. Hai loại hoạt động chính:

- Atomic Memory Operations (AMO): Các lệnh như fetch-and-op cho phép thực hiện các phép toán nguyên tử trên bộ nhớ.

- Load Reserved/Store Conditional: Cơ chế này đảm bảo tính nhất quán và ngăn xung đột khi nhiều tiến trình truy cập cùng một vùng nhớ.

Điều này đặc biệt hữu ích trong các hệ thống phức tạp yêu cầu đồng bộ hóa giữa nhiều luồng hoặc tiến trình.

RV32C - Compressed Instructions:

RV32C cung cấp các lệnh nén 16 bit thay cho các lệnh chuẩn 32 bit để giảm kích thước chương trình. Khoảng 50% - 60% lệnh có thể được thay thế, giúp tiết kiệm 25% - 30% bộ nhớ chương trình.

Các lệnh nén này được mã hóa để ánh xạ trực tiếp tới lệnh 32 bit, và việc giải mã được xử lý tự động bởi Assembler và Linker. RV32C rất hữu ích trong các hệ thống nhúng hoặc có bộ nhớ hạn chế, nơi tối ưu hóa không gian lưu trữ là ưu tiên hàng đầu.

Người dùng có thể lựa chọn và kết hợp các phần mở rộng phù hợp với yêu cầu ứng dụng, ví dụ:

- RV32IMAC: Gồm các tập lệnh cơ bản, nhân/chia, nguyên tử, và nén.
- RV32GC: Kết hợp tất cả các phần mở rộng để phục vụ ứng dụng tổng quát.

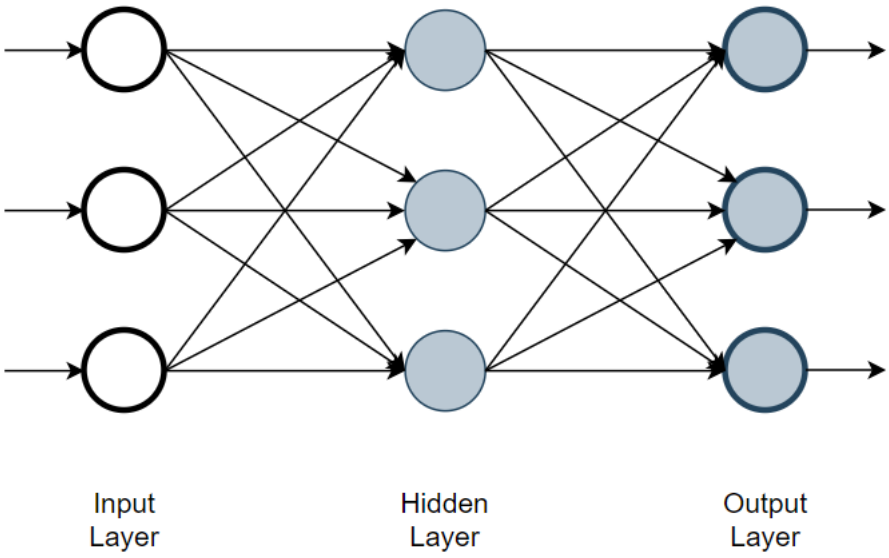
2.2 Mạng nơ-ron - Neural Network

2.2.1 Giới thiệu

Mạng nơ-ron nhân tạo (Neural Network) là một mô hình tính toán lấy cảm hứng từ cấu trúc và hoạt động của não bộ. Nó bao gồm các nút (neurons) được kết nối với nhau qua các trọng số, hoạt động như các đơn vị xử lý thông tin. Neural Network là nền tảng của học sâu (Deep Learning), giúp giải quyết nhiều bài toán phức tạp.

2.2.2 Cấu trúc mạng nơ-ron

Mạng nơ-ron sau nhiều lần chỉnh sửa và cải tiến thì có mô hình tổng quát như sau:

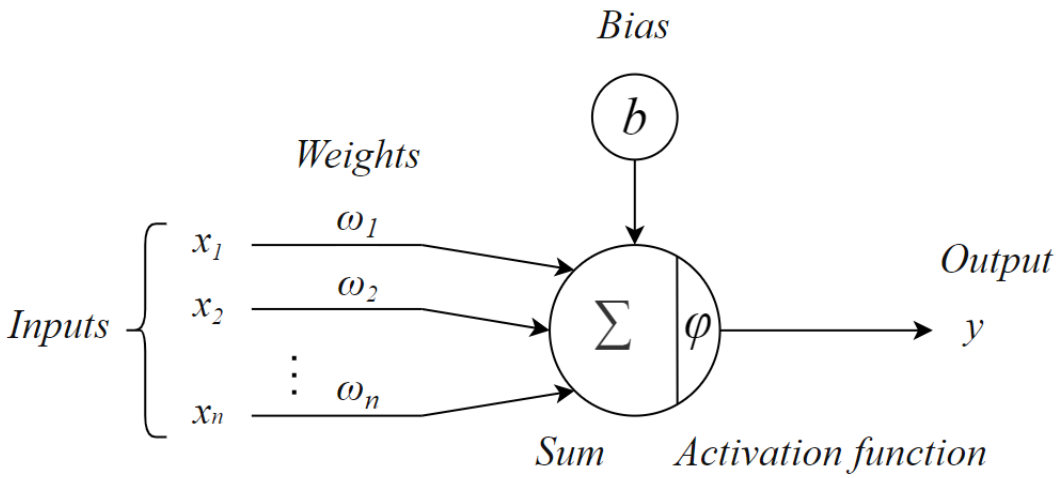


Hình 2.2. Mô hình tổng quát của mạng nơ-ron

Layer đầu tiên là input layer, các layer ở giữa được gọi là hidden layer, layer cuối cùng được gọi là output layer. Các hình tròn được gọi là node.

Mỗi mô hình luôn có 1 input layer, 1 output layer, có thể có hoặc không các hidden layer. Tổng số layer trong mô hình được quy ước là số $layer - 1$ (không tính input layer). Ví dụ như ở hình trên có 1 input layer, 1 hidden layer và 1 output layer. Số lượng layer của mô hình là 2 layer.

Mỗi node trong hidden layer và output layer :



Hình 2.3. Mô hình của một node

- Liên kết với tất cả các node ở layer trước đó với các hệ số w riêng.

- Mỗi node có 1 hệ số bias b riêng.
- Diễn ra 2 bước:
  - Tính tổng linear

$$g(x) = \sum_{n=1}^{N} x[n]w[n]$$

  - Áp dụng activation function

$$y = \varphi(g(x) + b)$$

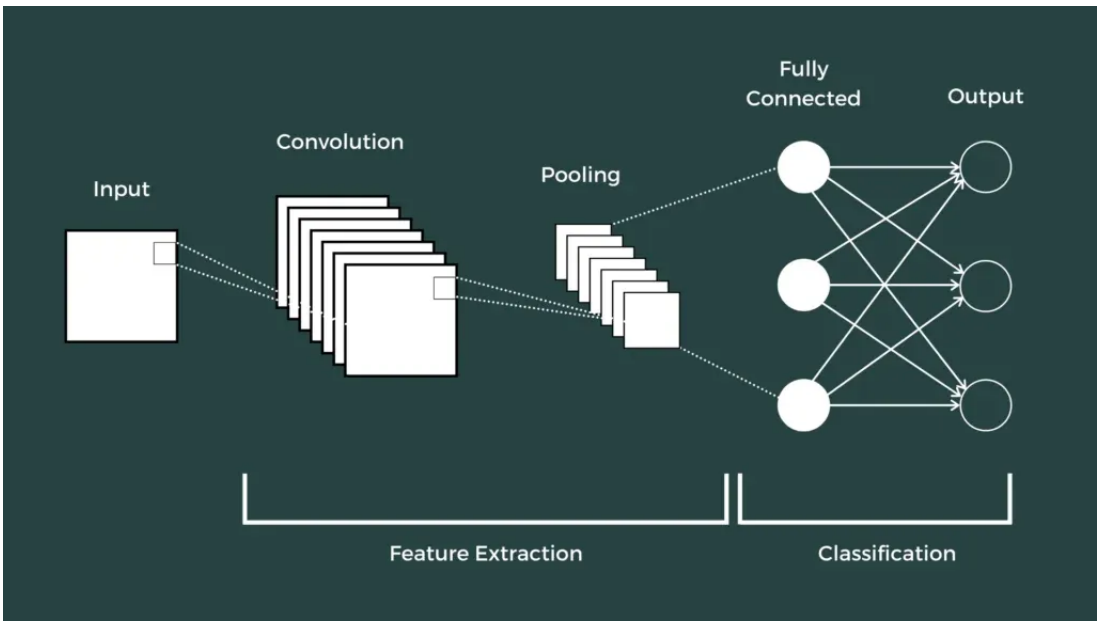
2.3 Convolutional Neural Network - CNN

2.3.1 Giới thiệu

CNN được viết tắt của Convolutional Neural Network hay còn được gọi là mang nơ-ron tích chập, là một trong những mô hình Deep Learning cực kỳ tiên tiến cho phép xây dựng những hệ thống có độ chính xác cao và thông minh.

Trong mạng nơ-ron, mô hình mạng nơ-ron tích chập (CNN) là một phương pháp phổ biến để nhận dạng và phân loại hình ảnh. CNN được ứng dụng rộng rãi trong nhiều lĩnh vực, đặc biệt là trong việc xác định đối tượng và nhận dạng khuôn mặt.

CNN phân loại hình ảnh bằng cách lấy 1 hình ảnh đầu vào, xử lý và phân loại nó theo các hạng mục nhất định (Ví dụ: Chó, Mèo, Hổ, ...). Máy tính coi hình ảnh đầu vào là 1 mảng pixel và mảng pixel phụ thuộc vào độ phân giải của hình ảnh. Dựa trên độ phân giải hình ảnh, máy tính sẽ thấy H x W x D (H: Chiều cao, W: Chiều rộng, D: Độ dày).

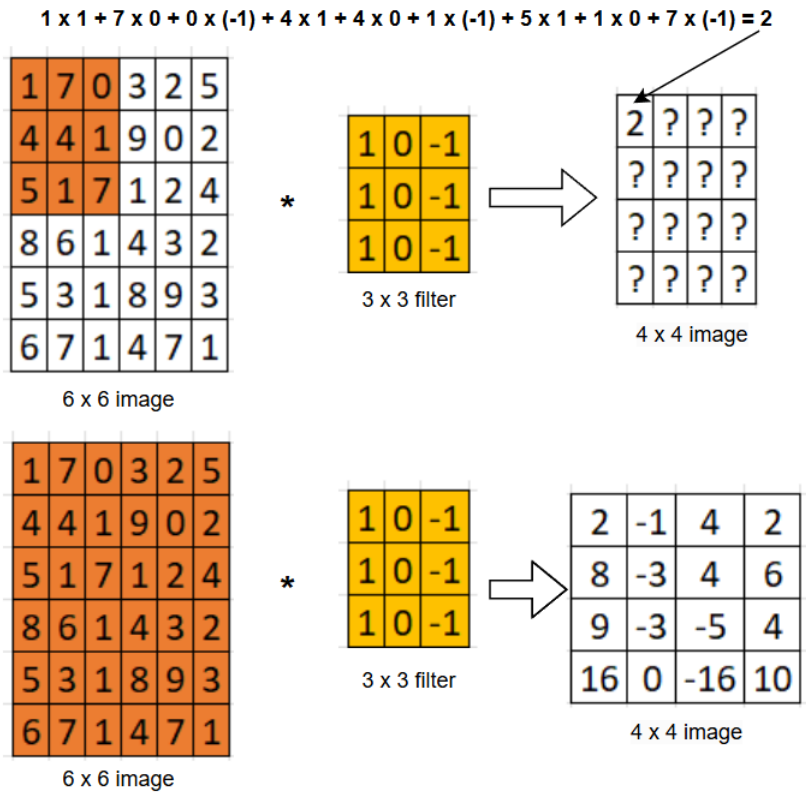


Hình 2.4. Mô hình kiến trúc cơ bản của CNN [3]

Mô hình CNN để training và kiểm tra, mỗi hình ảnh đầu vào sẽ chuyển nó qua 1 loạt các lớp tích chập (convolutional layers) với các bộ lọc (filter), giảm kích thước bằng các lớp Pooling sau đó tổng hợp lại các lớp được kết nối đầy đủ (Full Connected) và áp dụng hàm kích hoạt (activation function) để phân loại đối tượng.

2.3.2 Lớp tích chập (Convolutional Layer)

Lớp Convolutional có nhiệm vụ trích xuất các đặc trưng từ ảnh đầu vào, giống như cách chúng ta nhận biết các chi tiết khác nhau khi nhìn vào một bức tranh. Hãy tưởng tượng một chiếc kính lúp nhỏ, gọi là filter, di chuyển trên bức ảnh. Chiếc kính lúp này thực chất là một ma trận chứa các trọng số. Tại mỗi điểm dừng, nó thực hiện phép nhân vô hướng với vùng pixel, tạo ra một giá trị mới. Giá trị này thể hiện mức độ hiện diện của một đặc trưng nhất định, ví dụ như cạnh của một vật thể. Kết quả của quá trình quét này là một bản đồ đặc trưng, giống như một bản phác thảo, làm nổi bật các đặc điểm quan trọng của ảnh.



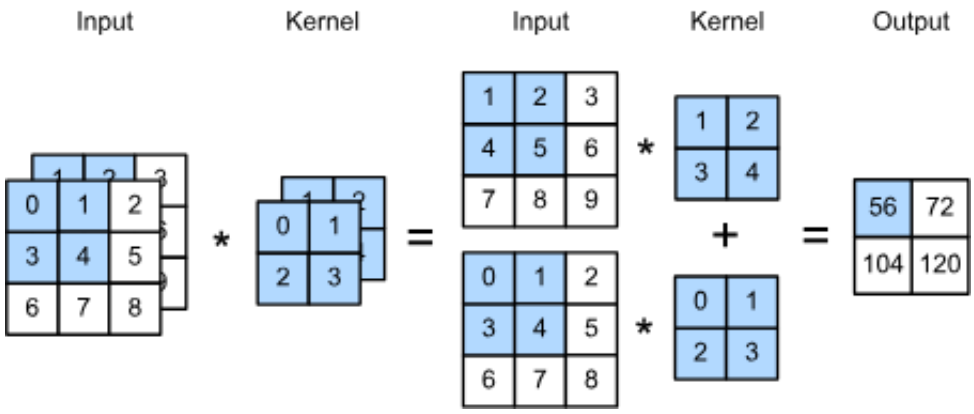
Hình 2.5. Minh họa phép tích chập với stride = 1, padding = 0, bias = 0

Lưu ý: Phép * không phải là phép nhân mà là phép tích vô hướng, filter và kernel là hai định nghĩa tương tự nhau khi nói về lớp tích chập.

Padding: Tưởng tượng bức ảnh của bạn là một tờ giấy. Padding giống như việc bạn thêm một đường viền xung quanh tờ giấy đó. Có 3 kiểu Padding chính:

- Zero-padding: Không thêm viền, giữ nguyên kích thước ảnh gốc. Giống như bạn không làm gì với tờ giấy cả.
- Half-padding: Thêm viền sao cho sau khi quét bằng filter, kích thước ảnh kết quả bằng với ảnh gốc. Giống như bạn dán thêm viền sao cho tờ giấy to ra một chút.
- Full-padding: Thêm viền sao cho kernel có thể chạm đến mọi điểm trên ảnh, kể cả phần rìa. Giống như bạn dán thêm một viền rộng để filter có thể di chuyển thoải mái trên toàn bộ tờ giấy.

Stride: giống như bước chân của bạn khi đi bộ. Mặc định, stride = 1, tức là filter di chuyển từng bước nhỏ một giống như bạn đi từng bước bình thường. Stride > 1, tức là filter di chuyển với bước nhảy lớn hơn. Giống như bạn chạy thay vì đi bộ. Ưu điểm là nhanh hơn, nhưng có thể bỏ sót một số chi tiết quan trọng trên ảnh.



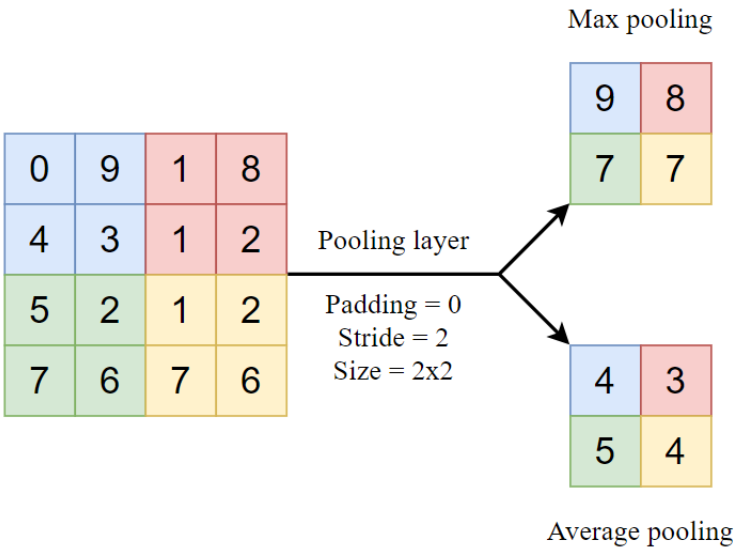
Hình 2.6. Tính tích chập với 2 input channels [4]

Điều thú vị là CNN thường sử dụng nhiều lớp Convolutional xếp chồng lên nhau. Các lớp đầu tiên sẽ nhận diện những đặc trưng đơn giản như cạnh, góc cạnh, màu sắc. Thông tin này được truyền đến các lớp tiếp theo để ghép nối, tạo thành các đặc trưng phức tạp hơn như hình dạng, họa tiết. Càng về sau, mạng lưới càng có khả năng nhận biết các đặc trưng trừu tượng như khuôn mặt, vật thể, thậm chí là cảm xúc. Quá trình này giống như việc chúng ta ghép các mảnh ghép nhỏ để tạo thành một bức tranh hoàn chỉnh.

2.3.3 Lớp tổng hợp (Pooling Layer)

Pooling layer thường được dùng giữa các convolutional layer, để giảm kích thước dữ liệu nhưng vẫn giữ được các thuộc tính quan trọng ngoài ra còn giảm số lượng tham số và tránh overfitting, từ đó giảm các phép tính toán trong model. Lớp này không có tham số để học, bao gồm bias. Những thông tin quan trọng của Pooling layer:

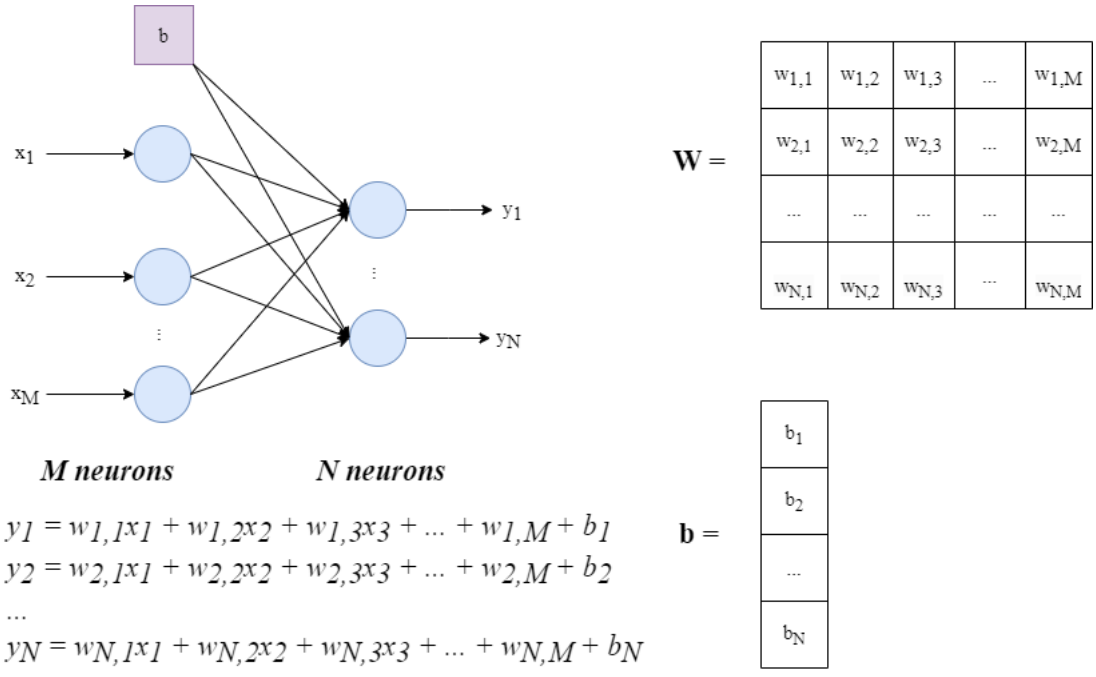
- Kích thước cửa sổ (Window Size): Kích thước của cửa sổ dùng để thực hiện phép pooling.
- Bước nhảy (Stride) và mở rộng biên (Padding): Phương thức hoạt động tương tự như Convolutional Layer.
- Có 2 loại Pooling phổ biến là
  - Max Pooling: Chọn giá trị lớn nhất trong vùng nhất định.
  - Average Pooling: Tính trung bình các giá trị trong vùng nhất định.



Hình 2.7. Các loại pooling

2.3.4 Lớp kết nối đầy đủ (Fully Connected Layer)

Lớp Fully-Connected thường nằm ở cuối mạng CNN, đóng vai trò như bộ tổng hợp thông tin. Nó nhận đầu vào là tất cả các đặc trưng đã được trích xuất và học từ các lớp Convolutional và Pooling trước đó. Sau đó, nó kết hợp tất cả các đặc trưng này để đưa ra dự đoán cuối cùng.



Hình 2.8. Mô hình tổng quát của Fully-connected Layer [5]

Phương trình tổng quát: $y = W \times x + b$

Lớp Fully-connected chịu ảnh hưởng bởi một số tham số quan trọng, có thể kể đến như:

- Số lượng neuron: Số lượng neuron trong một lớp kết nối đầy đủ ảnh hưởng đến khả năng học hỏi các đặc trưng của lớp đó. Càng nhiều neuron, mô hình càng có khả năng nắm bắt các đặc trưng phức tạp, nhưng đồng thời, nó cũng có thể dẫn đến hiện tượng overfitting nếu mô hình trở nên quá phức tạp so với dữ liệu.
- Khởi tạo trọng số (Weight initialization): Việc khởi tạo trọng số cũng đóng vai trò quan trọng trong quá trình học của mô hình. Cách thức khởi tạo trọng số có thể tác động đến hiệu quả học tập của mô hình.
- Bias: được thêm vào hàm phi tuyến để điều chỉnh đầu ra của neuron.

2.3.5 Hàm kích hoạt (Activation Function)

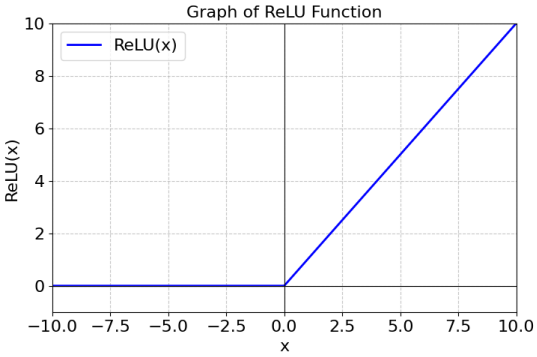
Mạng nơ-ron được cấu thành từ nhiều tầng (layers), và nếu không có hàm kích hoạt, các tầng chỉ thực hiện phép nhân ma trận tuyến tính. Điều này khiến toàn bộ mạng tương đương với một phép biến đổi tuyến tính duy nhất, bất kể có bao nhiêu tầng.

Vì thế nên ngoài các lớp cơ bản này ra, chúng ta cần quan tâm đến Activation function - Hàm kích hoạt. Trong mạng CNN, hàm kích hoạt thường được áp dụng sau mỗi lớp convolutional và lớp fully-connected. Hàm kích hoạt giúp mô hình học được các mô hình phức tạp hơn bằng cách thêm tính phi tuyến vào đầu ra của một neural. Dưới đây là 1 số hàm kích hoạt thường thấy.

Hàm ReLU:

Hàm ReLU là một hàm kích hoạt sẽ trả lại đầu vào nếu nó dương, ngược lại, nó sẽ trả về 0. ReLU giúp giải quyết vấn đề gradient biến mất, cho phép mô hình học nhanh hơn và hoạt động tốt hơn.

$$f(x) = max(0, x)$$

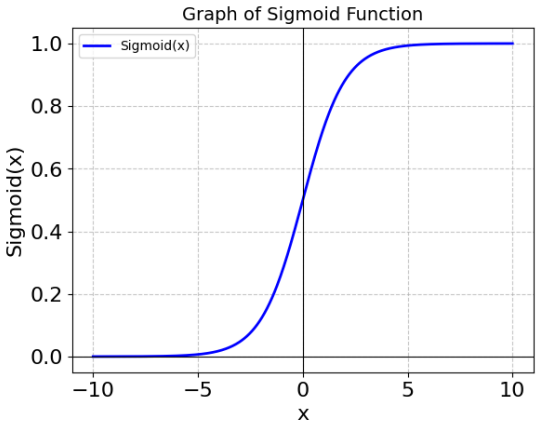


Hình 2.9. Hàm ReLU

Hàm Sigmoid:

Hàm Sigmoid là một hàm phi tuyến tính sẽ chuẩn hóa đầu ra trong phạm vi (0, 1). Trái ngược với chức năng ReLU, hàm Sigmoid rất tốn kém về mặt tính toán.

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

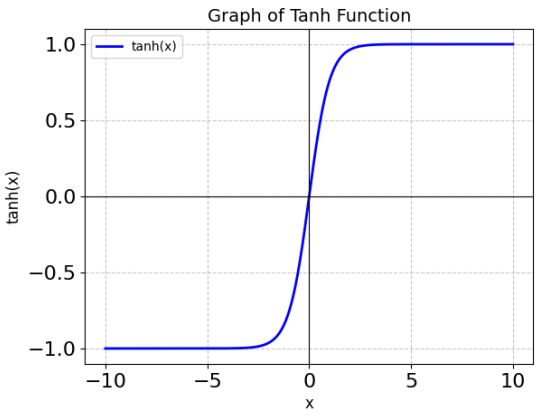


Hình 2.10. Hàm Sigmoid

Hàm TanH:

Hàm TanH cũng là một hàm phi tuyến tính sẽ chuẩn hóa đầu ra trong phạm vi (-1, 1), để các kết quả đầu ra cũng có thể nhận các giá trị âm.

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$



Hình 2.11. Hàm TanH

Hàm Softmax: nhận một vectơ đầu vào gồm N số. Mỗi số được chuẩn hóa trong

phạm vi (0, 1) và tổng của tất cả N số bằng 1.

Công thức:

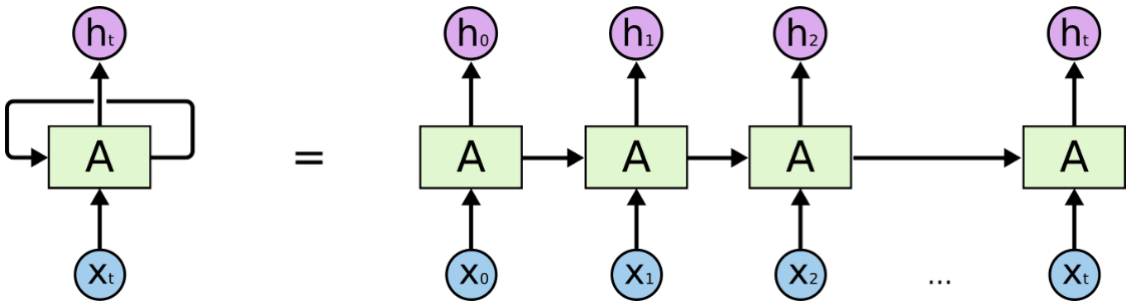
$$y_i = \frac{e^{x_i}}{\sum_{j=0}^{N-1} e^{x_j}}$$

với $i = 0, 1, ..., N-1$

2.4 Recurrent Neural Network - RNN

2.4.1 Giới thiệu

Trong các mạng nơ-ron thông thường, các đầu vào được xử lý một cách độc lập, không liên quan đến các đầu vào trước đó. Điều này khiến chúng trở nên không hiệu quả khi giải quyết các bài toán yêu cầu sự liên kết giữa các dữ liệu trong một chuỗi liên tục, chẳng hạn như dự đoán từ tiếp theo trong câu hoặc xử lý dữ liệu tuần tự. Để khắc phục nhược điểm này, mạng nơ-ron hồi quy (Recurrent Neural Network - RNN) đã ra đời với khả năng sử dụng trạng thái ẩn (hidden state) để lưu trữ thông tin từ các bước xử lý trước. Nhờ đó, RNN có thể mô hình hóa mối quan hệ phụ thuộc giữa các bước trong chuỗi dữ liệu, giúp tăng độ chính xác trong các bài toán như dự đoán chuỗi thời gian, xử lý ngôn ngữ tự nhiên, và nhận dạng giọng nói.



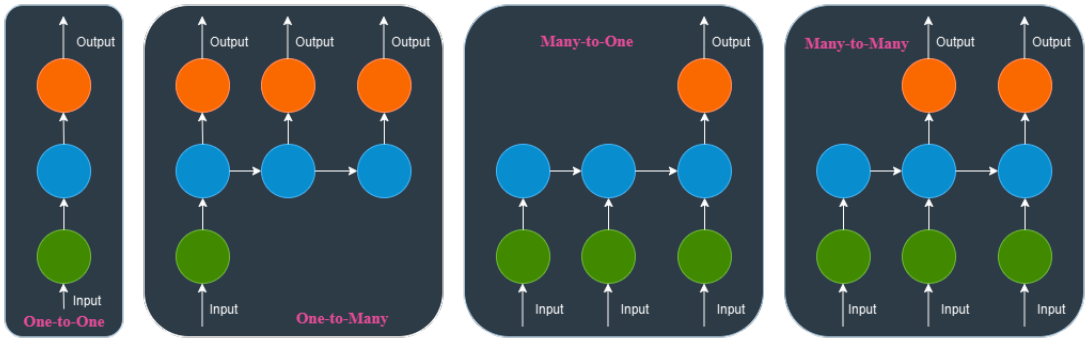
Hình 2.12. Mô hình mạng RNN[6]

Một vòng lặp cho phép thông tin có thể được truyền từ bước này qua bước này qua bước khác của mạng nơ-ron. Mỗi khối A nhận hai đầu vào:

- Dữ liệu hiện tại (x_t): Tương tự như mạng nơ-ron truyền thống.
- Trạng thái ẩn từ bước trước (h_{t-1}): Đây chính là điểm khác biệt, cho phép RNN ghi nhớ thông tin từ quá khứ.

Mỗi khối A sau đó xử lý thông tin và tạo ra đầu ra h_t vừa là kết quả của bước hiện tại, vừa là đầu vào cho bước tiếp theo.

2.4.2 Phân loại bài toán RNN



Hình 2.13. Phân loại mạng RNN

One-to-One: Đây là dạng bài toán ánh xạ trực tiếp giữa một đầu vào và một đầu ra. Mô hình này thường được áp dụng trong các mạng Neural Network (NN) hoặc Convolutional Neural Network (CNN). Ví dụ, trong bài toán phân loại ảnh MNIST, đầu vào là một hình ảnh và đầu ra là số mà hình ảnh đó đại diện.

One-to-Many: Trong dạng bài toán này, một đầu vào duy nhất sẽ được ánh xạ thành nhiều đầu ra theo thời gian. Một ví dụ điển hình là bài toán tạo chú thích cho ảnh, trong đó đầu vào là một bức ảnh và đầu ra là một chuỗi từ mô tả nội dung của ảnh.

Many-to-One: Dạng bài toán này nhận nhiều đầu vào (chuỗi dữ liệu) nhưng chỉ tạo ra một đầu ra duy nhất. Ví dụ, trong bài toán phân loại hành động từ video, đầu vào là chuỗi các khung hình (frames) và đầu ra là hành động được thực hiện trong video. Một ví dụ khác là dự đoán cảm xúc từ văn bản, với chuỗi từ làm đầu vào và cảm xúc là đầu ra.

Many-to-Many: Đây là dạng bài toán trong đó cả đầu vào và đầu ra đều là chuỗi với độ dài có thể khác nhau. Một ứng dụng phổ biến là dịch ngôn ngữ, ví dụ: câu tiếng Anh "I love Vietnam" được ánh xạ sang câu tiếng Việt "Tôi yêu Việt Nam." Trong trường hợp này, đầu vào và đầu ra đều là các chuỗi từ có độ dài không cố định.

2.4.3 Nhược điểm

- Vanishing Gradient: Khi huấn luyện RNN bằng phương pháp lan truyền ngược, gradient (độ dốc) có thể giảm dần theo thời gian. Điều này khiến mạng khó học được các phụ thuộc dài hạn, vì ảnh hưởng của thông tin từ quá khứ xa xôi trở nên mờ nhạt.
- Exploding Gradient: Ngược lại với vanishing gradient, exploding gradient xảy ra khi gradient tăng quá nhanh, dẫn đến mất ổn định trong quá trình huấn luyện.
- Mối quan hệ yếu với quá khứ xa: RNN truyền thống gặp khó khăn trong việc duy trì thông tin ngữ cảnh từ những bước thời gian xa, gây trở ngại cho việc hiểu được mối liên hệ giữa dữ liệu hiện tại và quá khứ.
- Trạng thái ẩn cố định: Kích thước trạng thái ẩn của RNN không thay đổi, điều này hạn chế khả năng thích ứng với các chuỗi có độ dài khác nhau và nắm bắt thông tin từ nhiều thang thời gian.
- Tính toán tuần tự: Do tính chất lặp lại, RNN xử lý từng bước thời gian một cách tuần tự, dẫn đến tốc độ tính toán chậm, đặc biệt với chuỗi dữ liệu dài.

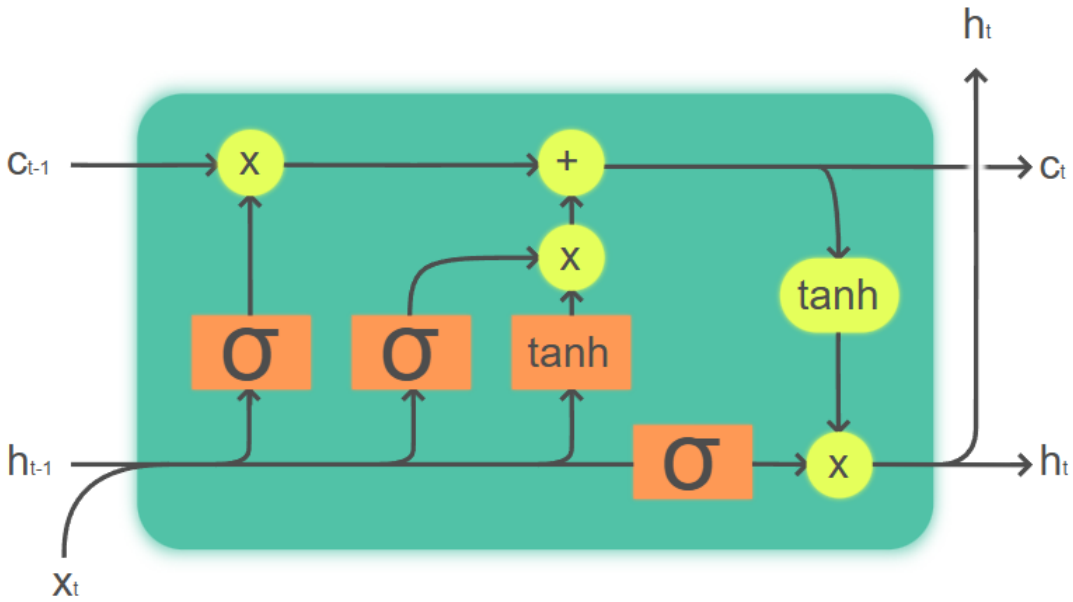
2.5 Long Short-Term Memory - LSTM

2.5.1 Giới thiệu

Như đã đề cập ở phần trước, RNN truyền thống gặp phải hạn chế trong việc học các phụ thuộc dài hạn do vấn đề vanishing gradient. Để giải quyết vấn đề này, Long Short-Term Memory (LSTM) ra đời như một kiến trúc RNN cải tiến.

LSTM được thiết kế đặc biệt để ghi nhớ thông tin trong thời gian dài. Khác với RNN đơn giản chỉ có một trạng thái ẩn, LSTM sử dụng một cơ chế phức tạp hơn với cổng (gates) để điều chỉnh dòng chảy của thông tin. Các cổng này cho phép LSTM lưu trữ, ghi nhớ và quên thông tin một cách chọn lọc.

2.5.2 Mô hình LSTM



Hình 2.14. Mô hình LSTM

Tại trạng thái thứ t trong mô hình LSTM, ta có:

- Input: c_{t-1}, h_{t-1}, x_t. Trong đó x_t là input ở trạng thái thứ t của model. c_{t-1}, h_{t-1} là output của layer trước, h đóng vai trò khá giống như h ở RNN, trong khi c là điểm mới của LSTM.
- Output: c_t, h_t, ta gọi c là cell state, h là hidden state.

Để hiểu rõ hơn cách thức hoạt động của nó, ta cần phân tích các thành phần và công thức liên quan.

Các ký hiệu:

- σ, TanH: Ký hiệu này chỉ ra rằng các hàm kích hoạt sigmoid (σ) và hyperbolic tangent ($TanH$) được sử dụng trong các bước tương ứng.
- Phép nhân: Phép nhân được sử dụng trong biểu đồ là phép nhân element-wise, nghĩa là nhân từng phần tử tương ứng của hai ma trận.
- Phép cộng: Phép cộng trong biểu đồ là phép cộng ma trận thông thường.

Các cổng (gate):

- Forget gate (f_t): quyết định thông tin nào từ trạng thái trước đó (c_{t-1}) sẽ bị loại bỏ.

$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + b_f)$$

- Input gate (i_t): quyết định thông tin nào từ đầu vào hiện tại (x_t) và trạng thái ẩn trước đó (h_{t-1}) sẽ được thêm vào trạng thái tế bào.

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + b_i)$$

- Output gate (o_t): quyết định thông tin nào từ trạng thái hiện tại (c_t) sẽ được sử dụng để tạo ra trạng thái ẩn hiện tại (h_t).

$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + b_o)$$

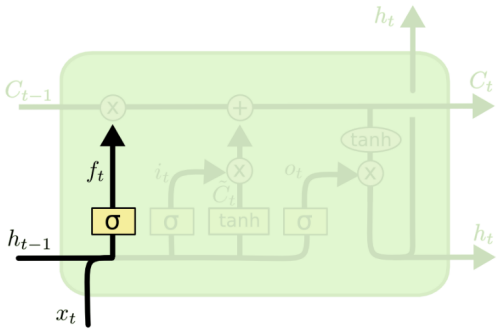
Lưu ý:

- $0 < f_t, i_t, o_t < 1$: Giá trị của các cổng nằm trong khoảng từ 0 đến 1 do sử dụng hàm sigmoid.
- b_f, b_i, b_o: Các hệ số bias cho mỗi cổng.
- W: Các ma trận trọng số.

Các bước tính toán:

Trong bước đầu tiên, sau khi hàm sigmoid được kích hoạt, cổng quên f_t xác định thông tin nào có thể được truyền qua ô trạng thái (cell state). Công thức cho bước này như sau:

$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + b_f)$$

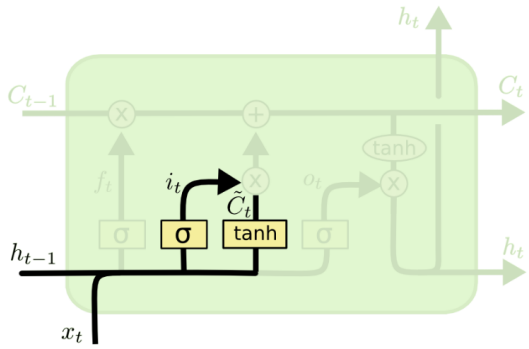


Hình 2.15. Bước đầu tính toán trong LSTM cell

Trong bước thứ hai, thông tin của ô trạng thái được cập nhật. Đầu tiên, sau khi hàm sigmoid được kích hoạt, cổng đầu vào i_t xác định thông tin nào được sử dụng để cập nhật ô trạng thái. Nếu giá trị này gần với 1 hơn, nhiều thông tin sẽ được giữ lại trong c_t. Thứ hai, tầng ẩn hàm tanh sẽ tạo ra một véc tơ của một giá trị trạng thái mới $\tilde{C}_t$ mà có thể được thêm vào trạng thái. Cuối cùng, f_t thu được ở bước đầu tiên và đầu vào c_{t-1} được nhân với nhau, sau đó $\tilde{C}_t$ và i_t được nhân với nhau. Kết quả của hai phép nhân này được cộng lại để tạo thành ô trạng thái hiện tại c_t.

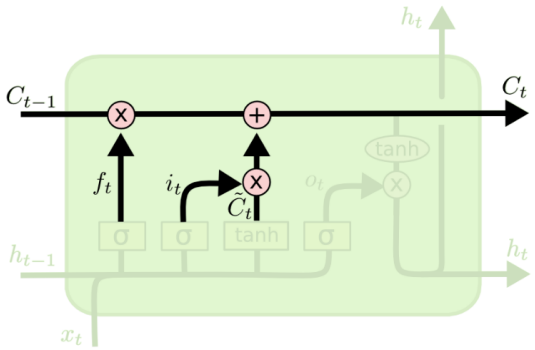
Các công thức tương ứng được mô tả như sau:

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + b_i)$$
$$\tilde{C}_t = \tanh(W_{x\tilde{C}}x_t + W_{h\tilde{C}}h_{t-1} + b_{\tilde{C}})$$



Hình 2.16. Bước tính toán thứ hai trong LSTM cell

$$c_t = f_t * c_{t-1} + i_t * g_t$$



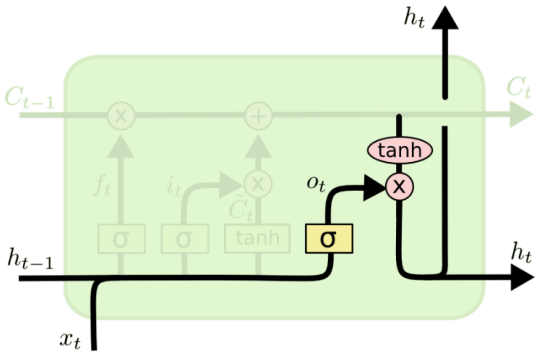
Hình 2.17. Bước tính toán thứ hai trong LSTM cell

Trong bước thứ ba, đầu ra cuối cùng của đơn vị LSTM được thu được. Một đầu ra ban đầu được tạo ra bởi hàm sigmoid, và giá trị trạng thái ô c_t được nén vào khoảng $[-1, 1]$ bằng cách sử dụng hàm tanh. Sau đó, kết quả này được nhân với o_t để thu được đầu ra cuối cùng h_t của mô hình.

Công thức cho bước này như sau:

$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + b_o)$$

$$h_t = o_t * \tanh(c_t)$$



Hình 2.18. Bước tính toán thứ ba trong LSTM cell

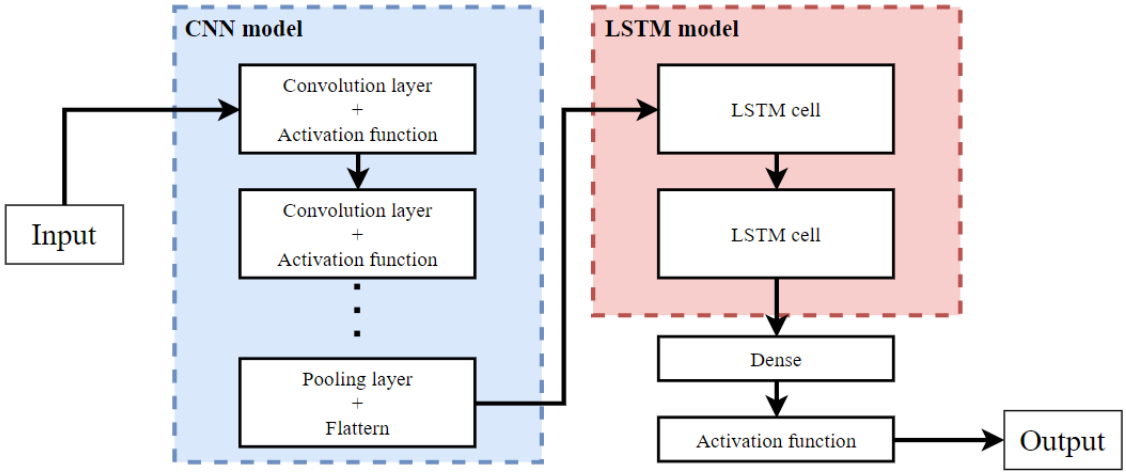
Nhận xét: h_t và $\tilde{c}_t$ được tính toán tương tự như trong RNN, giúp mô hình LSTM lưu trữ thông tin ngắn hạn, c_t hoạt động như một băng chuyền lưu trữ thông tin quan trọng trong thời gian dài. Thông tin cần thiết có thể được lưu trữ và truy xuất khi cần, do đó mô hình LSTM có cả short term memory và long term memory. Mặc dù LSTM mang lại hiệu quả vượt trội trong việc xử lý thông tin chuỗi, nhưng cần lưu ý rằng kiến trúc phức tạp của nó đòi hỏi sức mạnh tính toán đáng kể. Đặc biệt, khi huấn luyện các mô hình LSTM với nhiều lớp ẩn và xử lý tập dữ liệu lớn, việc tiêu tốn tài nguyên tính toán có thể trở nên một thách thức.

2.6 CNN-LSTM

2.6.1 Mô hình lai CNN-LSTM (Hybrid CNN-LSTM)

Trong nhiều nghiên cứu hiện tại, hybrid CNN-LSTM chủ yếu đề cập đến một kết nối nối tiếp giữa CNN và LSTM. CNN sẽ được sử dụng để trích xuất các đặc trưng không gian từ dữ liệu (như hình ảnh hoặc video), trong khi LSTM sẽ tiếp nhận các đặc trưng này và học các mối quan hệ theo thời gian hoặc chuỗi, giúp mô hình nhận diện các mẫu thời gian hoặc sự chuyển động trong chuỗi dữ liệu. Việc kết hợp hai loại mạng này có thể mang lại hiệu suất cải thiện cho các nhiệm vụ phân loại chuỗi thời gian.

Tuy nhiên, vì CNN không giỏi trong việc trích xuất các đặc trưng thời gian, điều này có thể dẫn đến mất mát các đặc trưng trong các vòng tích chập và pooling trước đó, làm giảm hiệu quả của mạng trong việc khớp dữ liệu.

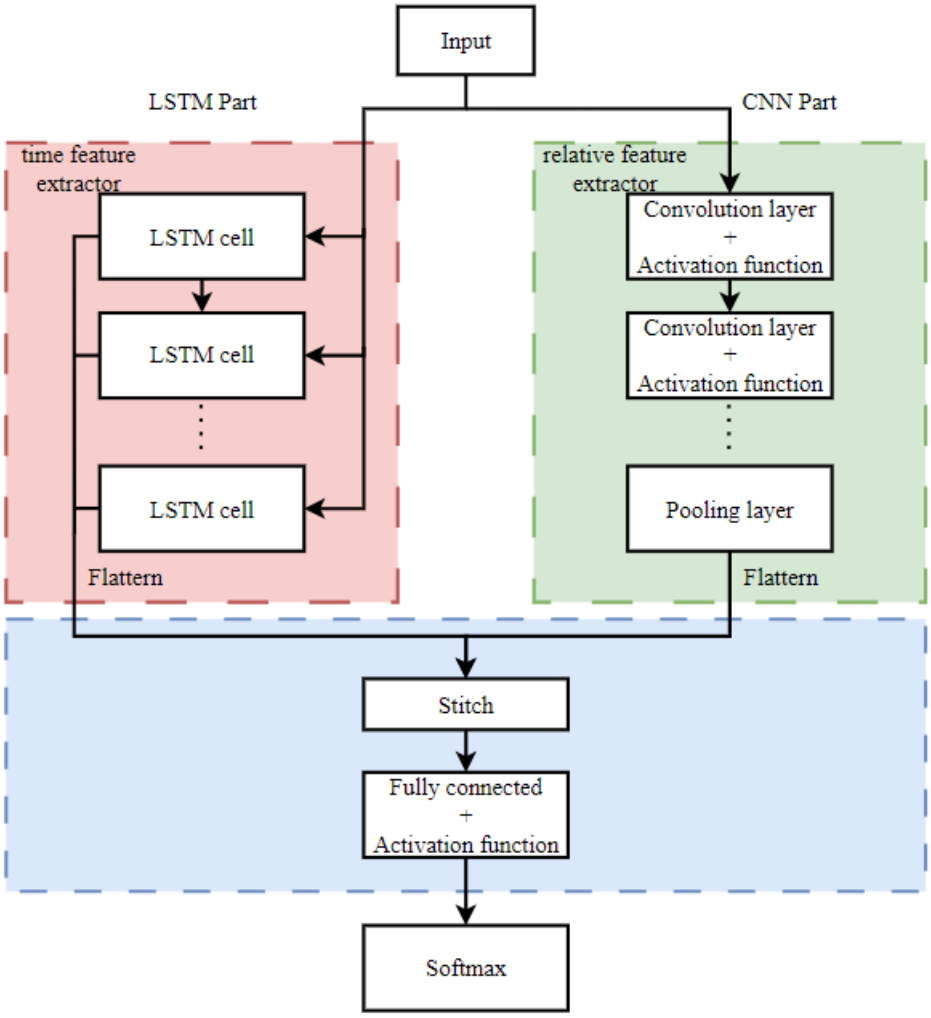


Hình 2.19. Mô hình kết hợp tuần tự CNN-LSTM

2.6.2 Mô hình song song CNN-LSTM

Dựa trên các vấn đề với kết nối nối tiếp, một cấu trúc CNN-LSTM song song đã được đề xuất. Bằng cách kết nối CNN và LSTM song song, cấu trúc này có thể trích xuất các đặc trưng thời gian và các đặc trưng liên quan cùng một lúc. Như thể hiện trong Hình 2.20, cấu trúc CNN-LSTM song song bắt đầu với việc xử lý dữ liệu đầu vào đồng thời bởi CNN và LSTM. Sau khi cả hai phần hoàn thành phép tính của chúng, kết quả đầu ra sẽ được làm phẳng thành một chiều để thực hiện các thao tác nối dữ liệu nhằm kết hợp các đặc trưng được trích xuất bởi hai mạng. Dữ liệu đã nối sẽ được tính toán qua lớp kết nối đầy đủ, hàm kích hoạt và lớp softmax để có được đầu ra cuối cùng.

Thiết kế cấu trúc song song tránh được việc mất thông tin đặc trưng xảy ra khi sử dụng kết nối nối tiếp. Đồng thời, phương pháp xử lý song song cũng rút ngắn độ trễ thời gian tính toán và cung cấp sự thuận tiện cho việc triển khai phần cứng sau này. Tham khảo [7] đã chứng minh rằng CNN-LSTM song song tốt hơn so với CNN-LSTM cổ điển và mạng đơn trong việc cảm nhận hiệu quả và tốc độ hội tụ.



Hình 2.20. Mô hình kết hợp song song CNN-LSTM

CHƯƠNG 3
CÔNG TRÌNH NGHIÊN CỨU LIÊN QUAN

3.1 Khảo sát về RISC-V Core

3.1.1 Microblaze V

Bộ xử lý AMD MicroBlaze™ V là một IP lõi mềm RISC-V dành cho các SoC thích ứng và FPGA của AMD. Bộ xử lý MicroBlaze V dựa trên kiến trúc tập lệnh (ISA) RISC-V 32-bit. Nó cho phép các nhà phát triển tận dụng hệ sinh thái phần mềm mã nguồn mở RISC-V, tương thích phần cứng với bộ xử lý MicroBlaze cổ điển và được tích hợp hoàn toàn trong quy trình thiết kế công cụ AMD Vivado™ và Vitis™. Bộ xử lý AMD MicroBlaze V được thiết kế với tính mô-đun cao với kiến trúc có thể cấu hình phù hợp với các ứng dụng hệ thống nhúng.[2]

Một số đặc trưng:

- MicroBlaze V dựa trên tập lệnh RV32I[M][A][F][C]
- Hiệu năng và khả năng tùy biến: Người dùng có thể lựa chọn cấu hình phù hợp với nhu cầu, từ vi điều khiển (Microcontroller) đơn giản đến hệ thống có bộ quản lý bộ nhớ (MPU, MMU), đồng thời tối ưu hóa hiệu năng hoặc diện tích sử dụng trên FPGA.
- Tiết kiệm tài nguyên: Tính năng nén mã giúp giảm kích thước chương trình, tiết kiệm bộ nhớ.
- Độ tin cậy cao: Hỗ trợ các biện pháp an toàn như khóa bước lõi kép và dự phòng ba module (Triple Modular Redundancy - TMR), đáp ứng yêu cầu của các hệ thống quan trọng.
- Dễ dàng tích hợp: Quy trình thiết kế được tích hợp đầy đủ trong các công cụ Vivado và Vitis, giúp việc phát triển và triển khai hệ thống trở nên thuận tiện.

Nhận xét: Giải pháp mạnh mẽ cho nhiều ứng dụng, từ IoT đến các hệ thống phức tạp. Tuy nhiên, cần cân nhắc các yếu tố như tài nguyên FPGA và chi phí khi lựa chọn MicroBlaze V cho dự án.

3.1.2 Ibex

Ibex là một lõi CPU RISC-V 32-bit mã nguồn mở, được viết bằng SystemVerilog, nổi bật với khả năng tùy chỉnh cao và hiệu quả về mặt diện tích, năng lượng. Ban đầu được phát triển dưới tên Zeroriscy trong nền tảng PULP, Ibex hiện được duy trì và phát triển bởi lowRISC.

Một số đặc trưng:

- Kiến trúc đơn giản: Ibex sử dụng kiến trúc pipeline 2 giai đoạn (Instruction Fetch và Instruction Decode/Execute) hoặc 3 giai đoạn (thêm giai đoạn Writeback), giúp giảm độ phức tạp và tiết kiệm năng lượng.
- Tùy biến linh hoạt: Hỗ trợ tập lệnh cơ bản RV32I/E và các mở rộng tùy chọn như nén lệnh (C), nhân chia số nguyên (M), và thao tác bit (B).
- Hiệu năng ấn tượng: Với các cấu hình khác nhau (micro, small, maxperf, maxperf-pmp-bmfull), Ibex đạt hiệu năng CoreMark/MHz từ 0.9 đến 3.13, cho phép cân bằng giữa hiệu năng và tài nguyên.
- Tiết kiệm năng lượng: Được thiết kế cho các ứng dụng nhúng, Ibex tối ưu hóa mức tiêu thụ năng lượng, đặc biệt phù hợp với các thiết bị IoT.
- Xác minh kỹ lưỡng: Ibex đã trải qua quá trình xác minh rộng rãi và được sử dụng trong nhiều dự án thực tế (tape-out).

Nhận xét: Ibex là một lựa chọn tuyệt vời cho các ứng dụng nhúng yêu cầu sự cân bằng giữa hiệu năng, kích thước và năng lượng, phù hợp cho các ứng dụng ở thiết bị biên với cấu hình nhỏ và các ứng dụng từ vừa cho tới lớn đối với cấu hình lớn hơn.

3.1.3 PicoRV32

PicoRV32 là một lõi RISC-V 32-bit mã nguồn mở, được thiết kế với tiêu chí tối giản và hiệu quả, nhằm mục đích hoạt động như một bộ xử lý phụ trợ trong các thiết kế FPGA và ASIC.

Một số đặc trưng:

- Khả năng cấu hình linh hoạt: Hỗ trợ các tập lệnh RV32E, RV32I, RV32IC,

RV32IM và RV32IMC cho phép người dùng lựa chọn cấu hình phù hợp với nhu cầu.

- Kích thước nhỏ: PicoRV32 chỉ sử dụng từ 750 đến 2000 LUTs trên FPGA Xilinx 7-Series giúp tiết kiệm diện tích silicon và giảm thiểu tiêu thụ năng lượng.
- Tần số hoạt động cao: Đạt được tần số hoạt động từ 250 đến 450 MHz trên FPGA Xilinx 7-Series đảm bảo hiệu năng xử lý tốt cho các tác vụ đơn giản.
- Giao tiếp linh hoạt: Cung cấp lựa chọn sử dụng giao thức bộ nhớ riêng hoặc AXI4-Lite giúp dễ dàng tích hợp với các hệ thống khác.
- Hỗ trợ ngắt và bộ đồng xử lý: Hỗ trợ xử lý ngắt (IRQ) và giao tiếp với bộ đồng xử lý thông qua Pico Co-Processor Interface (PCPI) mở rộng khả năng ứng dụng.

Nhận xét: Ưu điểm nổi bật là mã nguồn mở, kích thước nhỏ gọn và tần số hoạt động cao. Tuy nhiên, PicoRV32 chỉ hỗ trợ tập lệnh cơ bản, không có khối quản lí bộ nhớ, hiệu năng hạn chế nên không phù hợp với ứng dụng phức tạp, đòi hỏi tính toán cao.

3.1.4 CV32E40P

CV32E40P là một lõi xử lý RISC-V 32-bit mã nguồn mở, nổi bật với kiến trúc 4 giai đoạn (pipeline) giúp tăng tốc độ xử lý và hiệu năng.

Một số đặc trưng:

- Tập lệnh mở rộng: Không chỉ hỗ trợ tập lệnh cơ bản RV32I mà còn tích hợp các tập lệnh mở rộng như RV32C, RV32M và một phần của RV32F.
- Lệnh đặc thù PULP: Bao gồm các lệnh Post-Incrementing load and stores, Multiply-Accumulate extensions, ALU extensions, Hardware Loops.
- Hỗ trợ đa dạng: Hỗ trợ các tính năng như debug module, quản lý năng lượng, xử lý ngắt và hệ điều hành thời gian thực FreeRTOS giúp việc phát triển và tích hợp hệ thống trở nên dễ dàng hơn.

Nhận xét: Phù hợp với các ứng dụng có quy mô từ trung bình đến lớn, yêu cầu hiệu năng xử lý cao và tiết kiệm năng lượng. Tuy nhiên, lõi này có thể đòi hỏi kiến

thức chuyên sâu để triển khai do tài liệu hướng dẫn còn hạn chế và cấu hình phức tạp.

3.1.5 CVA6

CVA6 là CPU RISC-V 64-bit, đơn luồng, được thiết kế với kiến trúc 6 giai đoạn (pipeline) và hỗ trợ đầy đủ hệ điều hành kiểu Unix.

Một số đặc trưng:

- Hỗ trợ hệ điều hành: thực hiện ba cấp độ đặc quyền (M, S, U) theo tiêu chuẩn RISC-V cho phép chạy các hệ điều hành phức tạp như Linux.
- Tập lệnh mở rộng: hỗ trợ đầy đủ tập lệnh cơ bản RV64I, cùng với các tập lệnh mở rộng quan trọng như RV64M và RV64C. Ngoài ra, lõi còn bao gồm một số tập lệnh đặc thù để tăng tốc các tác vụ cụ thể.
- Quản lý bộ nhớ nâng cao: được trang bị các tính năng quản lý bộ nhớ tiên tiến, bao gồm TLB (Translation Lookaside Buffer) riêng biệt, phần cứng PTW (Page Table Walker) để tăng tốc độ dịch địa chỉ và hỗ trợ dự đoán nhánh để tối ưu hóa luồng thực thi lệnh.

Nhận xét: Với khả năng tính toán mạnh mẽ và hỗ trợ phần cứng chuyên dụng, CVA6 là một lõi xử lý lý tưởng cho các ứng dụng đòi hỏi hiệu năng cao.

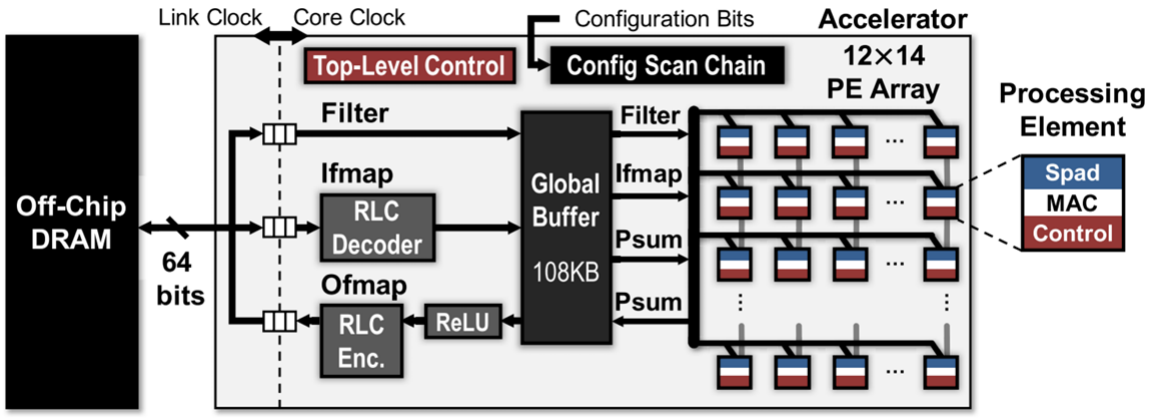
3.2 Khảo sát về tăng tốc xử lí cnn-lstm trên phần cứng

3.2.1 Eyeriss: An Energy Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks

Ý tưởng:

- Giảm sự dịch chuyển dữ liệu giữa các thành phần trong chip
- Khai thác tính thống kê của dữ liệu

Kiến trúc tổng quát:

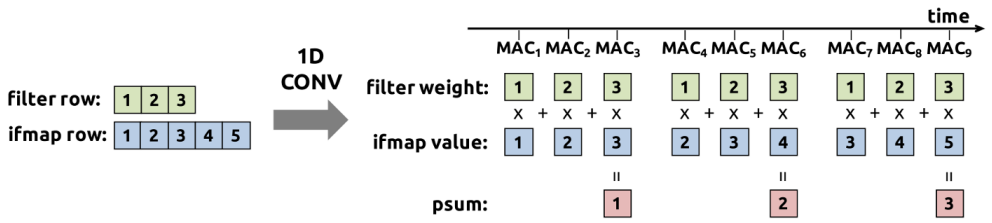


Hình 3.21. Kiến trúc của eyeriss

Theo như kiến trúc ở Hình 3.21 , Eyeriss có hai miền clock: miền core clock nhằm phục vụ việc xử lý các tính toán và miền link clock để giao tiếp với DRAM bên ngoài chip thông qua bus dữ liệu hai chiều 64 bit. Hai miền clock này chạy độc lập và giao tiếp với nhau thông qua một interface FIFO không đồng bộ.

Bên trong miền core clock là một array 2D gồm 168 PE có dạng hình chữ nhật 12×14, một GLB với không gian lưu trữ 108-kB, một RLC CODEC và một module ReLU. Nhằm truyền dữ liệu phục vụ việc tính toán, mỗi PE có thể giao tiếp với các PE xung quanh hoặc GLB (thông qua NoC) hoặc truy cập vào bộ nhớ cục bộ của PE được gọi là spad. Miền link clock phục vụ cho giao tiếp với DRAM bên ngoài chip.

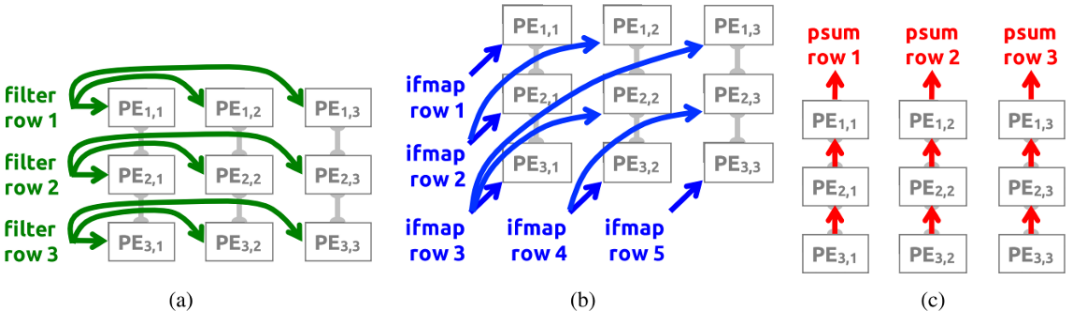
- 1-D Convolution Primitive: RS Dataflow đầu tiên chia quá trình tính toán thành các 1-D Convolution Primitive để các thành phần này có thể chạy song song với nhau. Mỗi tính toán primitive ở trên 1 hàng của kernel weight và 1 hàng của input feature map và tạo ra 1 hàng các tổng bán phần. Các tổng này sau đó được cộng tích lũy giữa các PE để tạo thành output feature map cuối cùng. Bằng cách ánh xạ mỗi primitive cho 1 PE, quá trình tính toán của từng cặp hàng ổn định ở trong PE (row pair stays stationary).



Hình 3.22. Quá trình ánh xạ 1-D convolution primitive vào một PE [13]

- 2-D Convolution PE Set: Một 2-D Convolution Set là tập hợp của nhiều 1-D Convolution primitive và việc tính toán của nó là chia sẻ cùng hàng của kernel hoặc input feature map giữa các 1-D Convoluton primitive và cộng tích lũy giá trị tổng bán phần ở nhiều primitive với nhau.

  Trong một PE Set, mỗi hàng filter được tái sử dụng theo chiều ngang, mỗi hàng input feature map được tái sử dụng theo đường chéo và các hàng psum được tích lũy theo chiều dọc. Kích thước của PE Set được xác định bởi filter và kích thước output feature map của một layer nhất định. Cụ thể, chiều cao và chiều rộng của PE Set lần lượt bằng số hàng filter (R) và hàng output feature map (E). Hình 3.23 thể hiện một PE Set được gom nhóm để chạy 2-D convolution và khai thác việc tái sử dụng tích chập interprimitive và tích lũy psum, giúp tránh truy cập dữ liệu từ GLB và DRAM.



Hình 3.23. Ánh xạ 2-D Convolution PE Set [13]

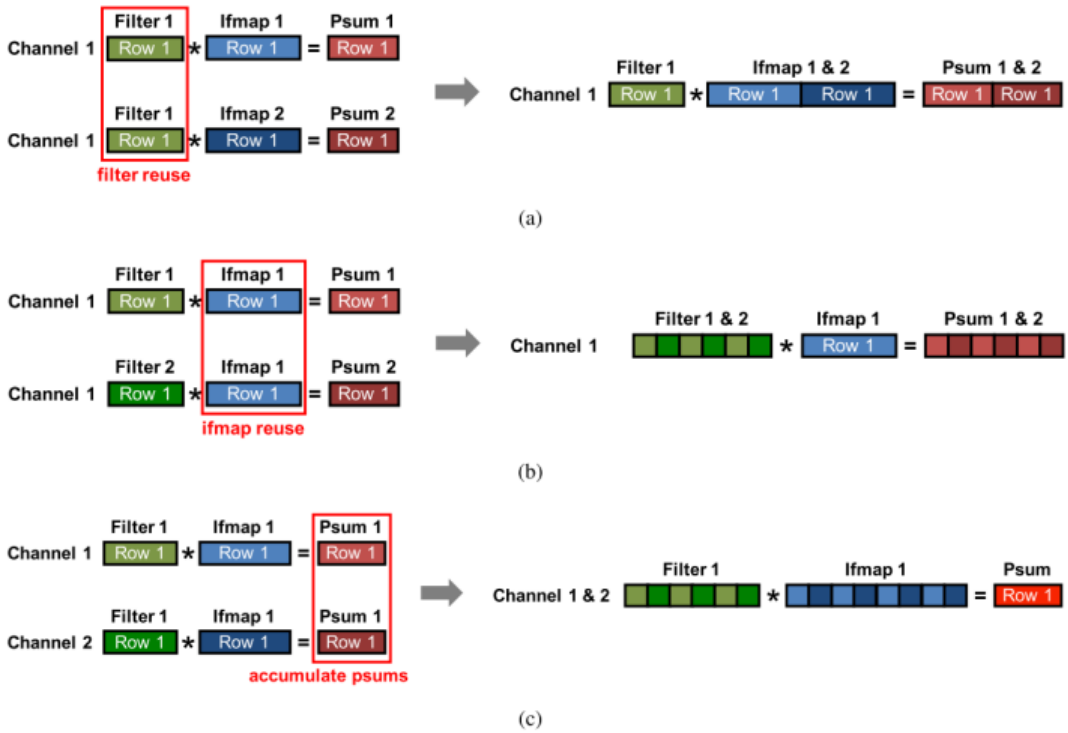
- PE Set Mapping: Kích thước của PEset là hàm số thể hiện hình dạng của một layer và độc lập với kích thước vật lý của PE array. Một PE Set có thể ánh xạ vào bất kỳ nhóm nào của PE array mà có cùng kích thước để tận dụng lợi thế của việc chia sẻ dữ liệu cục bộ và tích lũy psum. Tuy nhiên, không thể áp dụng cho các trường hợp sau:

  – PE set có nhiều hơn 168 thành phần.

  - PE set có ít hơn 168 thành phần, nhưng có kích thước chiều dài lớn hơn 14 hoặc chiều rộng lớn hơn 12.

- Dimensions Beyond 2-D in PE Array: cần phải xử lý nhiều tích chập 2-D để hoàn thành việc tính toán dựa trên ba yếu tố: batch size, số lượng channel và số lượng kernel weight. Giả sử mỗi lần chỉ thay đổi 1-D và cố định phần còn lại giống nhau, hai 2-D convolution sử dụng:
  - Các input feature map khác nhau sử dụng lại cùng một filter (filter reuse).
  - Các filter khác nhau sử dụng lại cùng một input feature map (input feature map reuse).
  - Các filter và input feature map từ các channel khác nhau có thể cộng tích lũy psum của chúng lại với nhau (partial sum accumulate).

  Việc tái sử dụng filter có thể được khai thác một cách đơn giản bằng cách truyền các input feature map khác nhau thông qua cùng một PE Set một cách tuần tự như hình 3.24.



Hình 3.24. Xử lý kích thước ngoài 2-D trong mỗi PE [13]

Cơ hội tái sử dụng input feature map và psum accumulation cũng có thể được khai thác bằng cách sử dụng các spad hoặc tính song song không gian của PE

array, do đó tình trạng truy cập vào DRAM và GLB để lấy dữ liệu sẽ giảm thiểu hơn nữa.

- Multiple 2-D Convolutions in a PE Set: Mỗi PE set được chạy nhiều tác vụ 2-D convolution ở filter và channel khác nhau.
- Multiple PE Sets in the PE Array: PE array có thể chạy nhiều hơn một PE set nếu PE set đủ nhỏ.

Đánh giá:

Thực hiện thí nghiệm với hai mô hình CNN: AlexNet và VGG-16.

TABLE V
PERFORMANCE BREAKDOWN OF THE FIVE CONV LAYERS IN AlexNet AT 1 V. BATCH SIZE (N) IS 4. THE CORE AND LINK CLOCKS RUN AT 200 AND 60 MHz, RESPECTIVELY

Layer	Power (mW)	Total Latency (ms)	Processing Latency (ms)	Num. of MACs	Num. of Active PEs	Zeros in Ifmaps (%)	Global Buff. Accesses	DRAM Accesses
CONV1	332	20.9	16.5	0.42G	154 (92%)	0.01%	18.5 MB	5.0 MB
CONV2	288	41.9	39.2	0.90G	135 (80%)	38.7%	77.6 MB	4.0 MB
CONV3	266	23.6	21.8	0.60G	156 (93%)	72.5%	50.2 MB	3.0 MB
CONV4	235	18.4	16.0	0.45G	156 (93%)	79.3%	37.4 MB	2.1 MB
CONV5	236	10.5	10.0	0.30G	156 (93%)	77.6%	24.9 MB	1.3 MB
Total	**278**	**115.3**	**103.5**	**2.66G**	**148 (88%)**	**57.53%**	**208.5 MB**	**15.4 MB**

Hình 3.25. Bảng mô tả hiệu suất đo lường của năm lớp CONV trong mô hình AlexNet với điện áp 1V [13]

Bảng hiệu suất (Hình 3.25) cho thấy chip Eyeriss đạt hiệu năng ấn tượng khi xử lý AlexNet trên điện áp 1V. Đặc biệt, nhờ kỹ thuật data gating tận dụng hiệu quả các giá trị 0 trong input feature maps, mức tiêu thụ năng lượng giảm dần qua các lớp, điều này góp phần nâng cao hiệu suất năng lượng tổng thể của hệ thống. Chip đạt tốc độ khung hình trung bình 34.7 fps, tương đương thông lượng xử lý 23.1 GMACS. Công suất tiêu thụ đo được là 278 mW, dẫn đến hiệu suất năng lượng đạt 83.1 GMACS/W.

TABLE VI

PERFORMANCE BREAKDOWN OF THE 13 CONV LAYERS IN VGG-16 AT 1 V. BATCH SIZE (N) IS 3. THE CORE AND LINK CLOCKS RUN AT 200 AND 60 MHz, RESPECTIVELY

Layer	Power (mW)	Total Latency (ms)	Processing Latency (ms)	Num. of MACs	Num. of Active PEs	Zeros in Ifmaps (%)	Global Buff. Accesses	DRAM Accesses
CONV1-1	247	76.2	38.0	0.26G	156 (93%)	1.6%	112.6 MB	15.4 MB
CONV1-2	218	910.3	810.6	5.55G	156 (93%)	47.7%	2402.8 MB	54.0 MB
CONV2-1	242	470.3	405.3	2.77G	156 (93%)	24.8%	1201.4 MB	33.4 MB
CONV2-2	231	894.3	810.8	5.55G	156 (93%)	38.7%	2402.8 MB	48.5 MB
CONV3-1	254	241.1	204.0	2.77G	156 (93%)	39.7%	607.4 MB	20.2 MB
CONV3-2	235	460.9	408.1	5.55G	156 (93%)	58.1%	1214.8 MB	32.2 MB
CONV3-3	233	457.7	408.1	5.55G	156 (93%)	58.7%	1214.8 MB	30.8 MB
CONV4-1	278	135.8	105.1	2.77G	168 (100%)	64.3%	321.8 MB	17.8 MB
CONV4-2	261	254.8	210.0	5.55G	168 (100%)	74.7%	643.7 MB	28.6 MB
CONV4-3	240	246.3	210.0	5.55G	168 (100%)	85.4%	643.7 MB	22.8 MB
CONV5-1	258	54.3	48.3	1.39G	168 (100%)	79.4%	90.0 MB	6.3 MB
CONV5-2	236	53.7	48.5	1.39G	168 (100%)	87.4%	90.0 MB	5.7 MB
CONV5-3	230	53.7	48.5	1.39G	168 (100%)	88.5%	90.0 MB	5.6 MB
Total	**236**	**4309.5**	**3755.2**	**46.04G**	**158 (94%)**	**58.6%**	**11035.8 MB**	**321.1 MB**

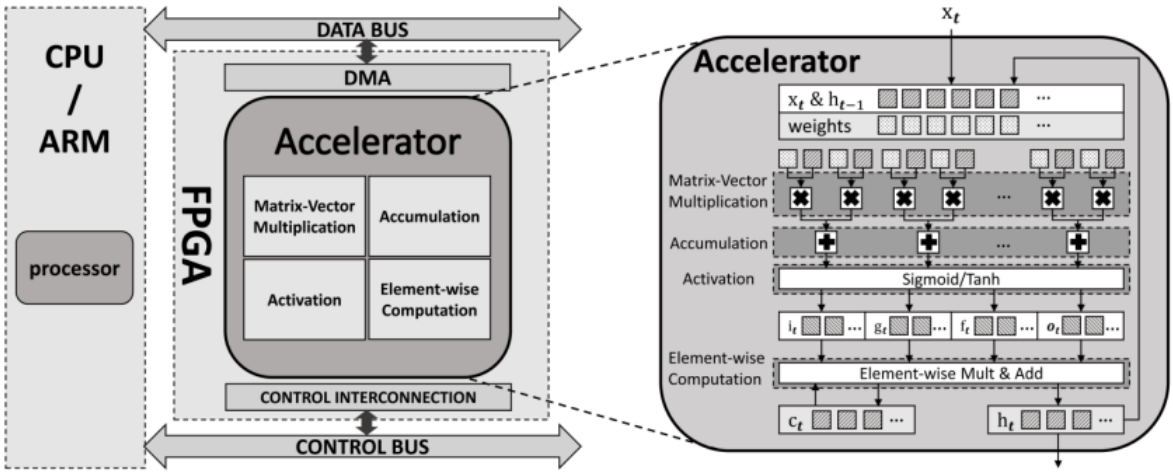
Hình 3.26. Bảng mô tả hiệu suất đo lường của 13 layer CONV trong mô hình VGG- 16 ở điện áp 1V [13]

Bảng hiệu suất (Hình 3.26) cung cấp thông tin chi tiết về hiệu suất của 13 lớp CONV trong mô hình VGG-16 khi chip hoạt động ở điện áp 1V. Tốc độ khung hình trung bình đạt 0.7 fps với mức tiêu thụ năng lượng là 236 mW. So với AlexNet, tốc độ khung hình của VGG-16 thấp hơn đáng kể do khối lượng tính toán lớn hơn gấp 23 lần.

3.2.2 Implementation and Optimization of the Accelerator Based on FPGA Hardware for LSTM Network

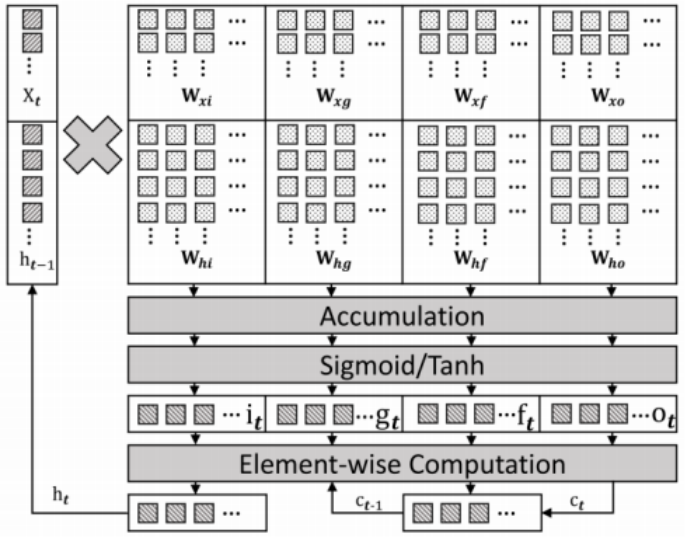
Ý tưởng: Nghiên cứu này tập trung vào xây dựng phần cứng tăng tốc xử lý mô hình RNN, cụ thể là loại LSTM. Kiến trúc này kết hợp cả phần mềm và phần cứng để tối ưu hóa hiệu năng xử lý.

Kiến trúc tổng quát:



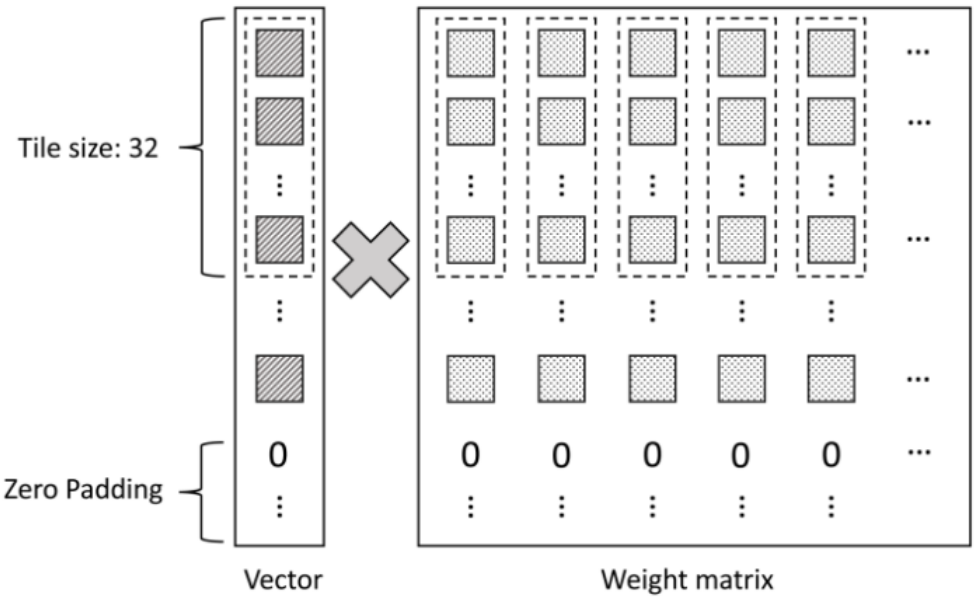
Hình 3.27. Kiến trúc tổng quát cho bộ tăng tốc dựa trên FPGA cho mạng LSTM [14]

Hình 3.27 minh họa kiến trúc tổng quát của bộ tăng tốc mạng LSTM được triển khai trên FPGA. CPU/ARM đóng vai trò điều khiển hệ thống, quản lý luồng xử lý và điều phối các mô-đun phần cứng. Bộ tăng tốc sử dụng DMA để đọc dữ liệu cần thiết từ bộ nhớ. Đầu tiên, nó đọc trọng số của lớp mạng LSTM và lưu trữ các tham số trong BRAM. Sau đó, bộ tăng tốc đọc các vectơ đầu vào và xuất ra các vectơ kết quả thông qua DMA.



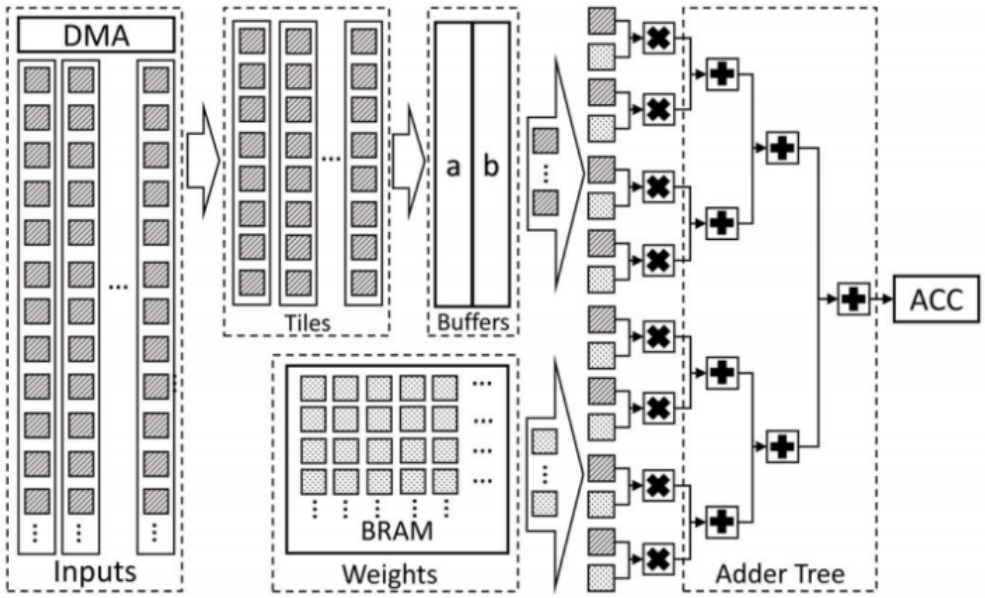
Hình 3.28. Toàn bộ quá trình thực hiện tính toán trong mạng LSTM [14]

Bộ tăng tốc thực hiện các phép tính chính của mạng LSTM, bao gồm: Nhân ma trận-vectơ, Tích lũy (Accumulation), Hàm kích hoạt (activation), Tính toán theo phần tử (element-wise).



Hình 3.29. Phép nhân tile ma trận-vectơ [14]

Trong quá trình nhân ma trận-vectơ, các vectơ đầu vào và đầu ra được gộp lại thành một vectơ duy nhất, đồng thời các ma trận trọng số riêng lẻ được kết hợp thành một ma trận lớn. Cách tiếp cận này tận dụng tính độc lập trong phép nhân ma trận-vectơ của LSTM. Mặc dù phép nhân vectơ-ma trận có thể được tăng tốc bằng cách tính toán song song, nhưng tài nguyên FPGA hạn chế đặt ra thách thức cho việc triển khai mạng LSTM quy mô lớn. Để giải quyết vấn đề này, vectơ và ma trận được chia thành nhiều tile nhỏ hơn. Cụ thể, vectơ và mỗi cột của ma trận được chia thành các tile có kích thước 32 phần tử. Các tile này sau đó được nhân với nhau một cách độc lập. Nếu kích thước của vectơ hoặc cột ma trận không phải là bội số của 32, kỹ thuật zero-padding được sử dụng để thêm các giá trị 0 vào cuối, đảm bảo mỗi tile đều có kích thước 32.

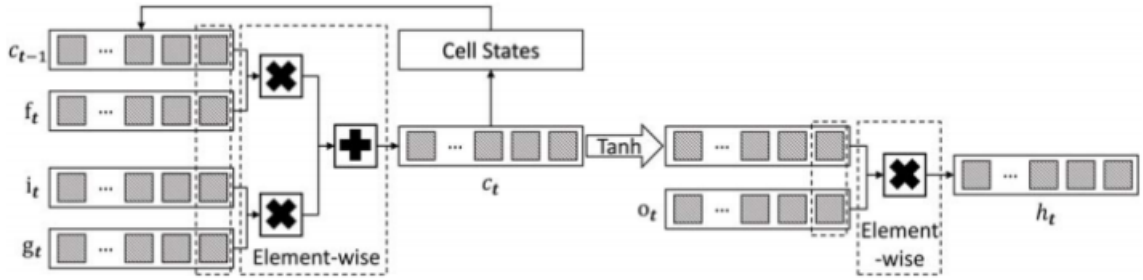


Hình 3.30. Thực hiện toàn bộ quá trình tính toán ma trận-vectơ [14]

Trong quá trình tích lũy, thay vì sử dụng phương pháp vòng lặp truyền thống, nhóm tác giả đã áp dụng cây cộng nhị phân để tận dụng khả năng tính toán song song của FPGA. Cây cộng này được thiết kế với chiều cao 5, tương ứng với kích thước tile là 32, cho phép giảm độ phức tạp thời gian từ O(32) xuống O(5).

Để đảm bảo tính toán diễn ra liên tục, hai bộ đệm được sử dụng luân phiên. Trong mỗi lần lặp, một bộ đệm cung cấp dữ liệu đã được đọc từ lần lặp trước, trong khi bộ đệm còn lại đồng thời đọc dữ liệu mới cho lần lặp tiếp theo. Cách tiếp cận này loại bỏ thời gian chờ đợi truy cập dữ liệu, cho phép tạo ra kết quả cuối cùng một cách liên tục mà không bị gián đoạn.

Theo công thức của mạng nơ-ron LSTM, cần các hàm kích hoạt (bao gồm hàm sigmoid và Tanh) để xử lý giá trị trung gian. Phải sử dụng xấp xỉ tuyến tính từng đoạn để hiện thực các hàm kích hoạt. Nội suy tuyến tính từng đoạn có thể được biểu diễn dưới dạng $f(x) = a_i * x + b_i, \quad x \in [x_i, x_{i+1})$. Khi khoảng cách k giữa x_i và x_{i+1} đủ nhỏ, nội suy tuyến tính từng đoạn có thể thực hiện bất kỳ hàm kích hoạt nào với độ chính xác bị mất không đáng kể.



Hình 3.31. Tính toán theo phần tử trong mạng LSTM [14]

Vectơ kết quả trung gian thu được bằng phép nhân vectơ-ma trận, phép toán tích lũy và hàm kích hoạt. Khi đó vectơ được chia thành 4 phần, biểu diễn vectơ của cổng i, g, f, o. Bốn phần này liên quan đến phép nhân và phép cộng theo từng phần tử để có được vectơ trạng thái c_t và vectơ đầu ra h_t thể hiện trong hình 3.31.

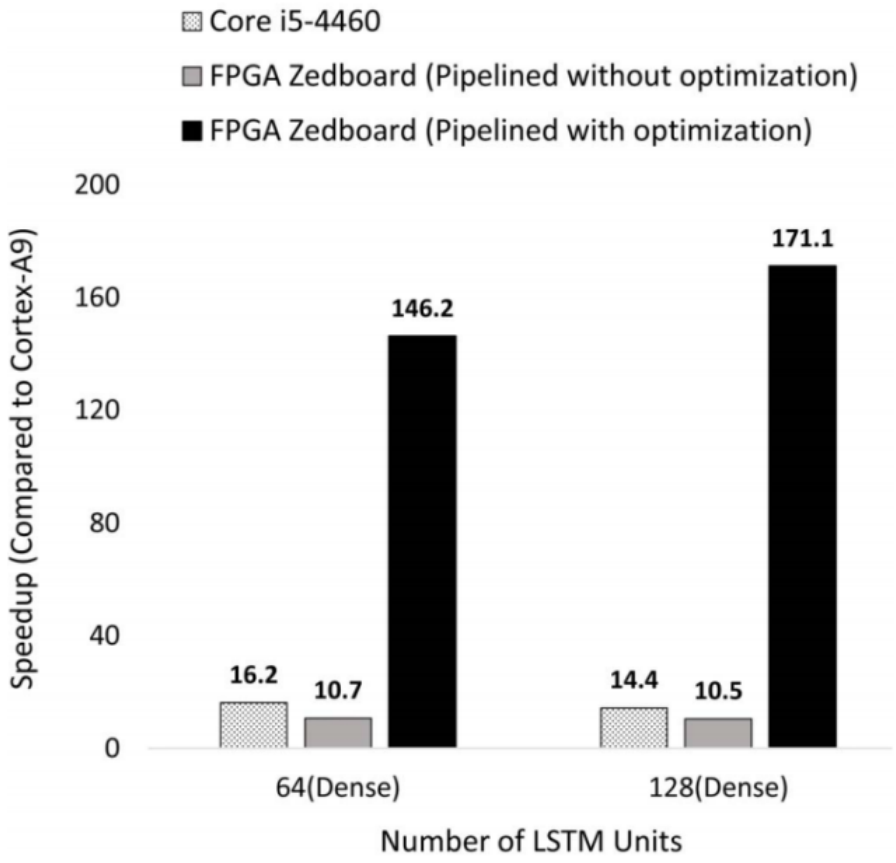
Đánh giá:

Kết quả được đánh giá dựa trên nền tảng như sau:

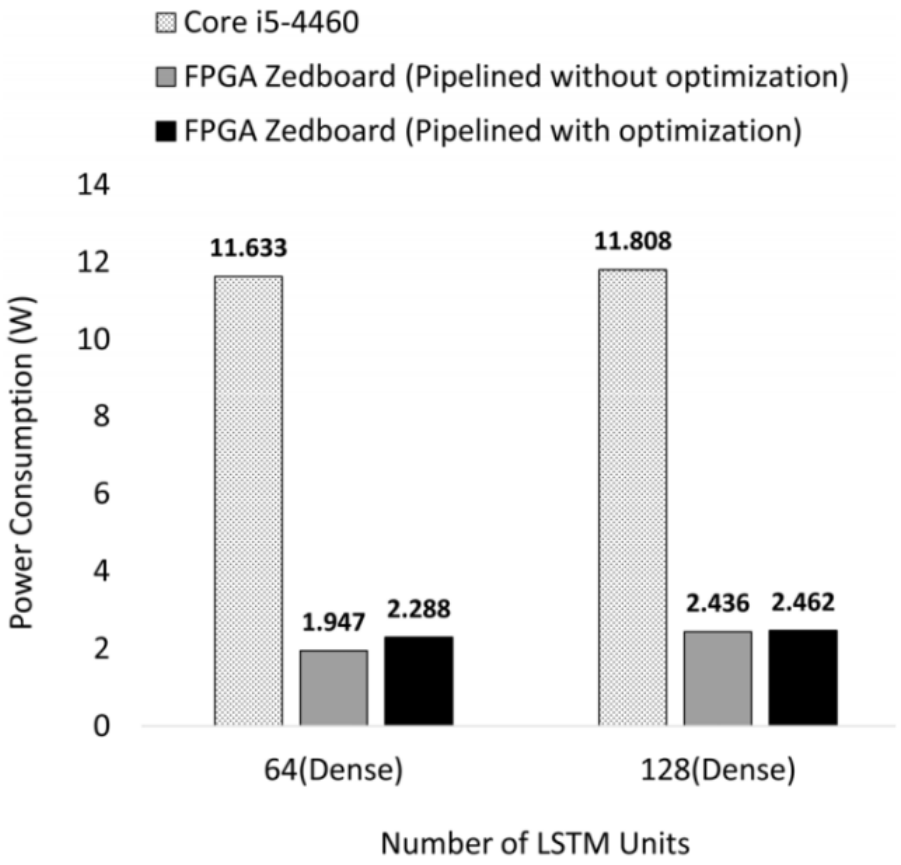
- Nền tảng FPGA: Bo mạch phát triển Zedboard để triển khai LSTM (Bộ xử lý của Zedboard là ARM Cortex A9, tốc độ 667 MHz), bộ đếm thời gian AXI để đếm tổng số chu kỳ đồng hồ đang chạy.
- Nền tảng CPU: Triển khai logic tính toán chuyển tiếp của lớp mạng thần kinh LSTM bằng ngôn ngữ C và chạy chương trình trên bộ xử lý ARM Cortex A9 và Intel Core i5-4460 (tốc độ 3,2 GHz) tương ứng.
- Công cụ triển khai trên FPGA: Sử dụng công cụ Vivado để mô phỏng chu trình hoạt động, mức tiêu thụ điện năng, sử dụng tài nguyên và các thông tin khác của FPGA.

Layer Size		BRAM	DSP48E	FF	LUT
64 (Without optimization)	Utilization	76	52	7845	15001
	Utilization(%)	27	23	7	28
64 (With optimization)	Utilization	72	200	26179	46705
	Utilization(%)	25	90	24	87
128 (Without optimization)	Utilization	264	52	7876	15083
	Utilization(%)	95	23	7	28
128 (With optimization)	Utilization	264	200	26212	46834
	Utilization(%)	95	90	24	88

Hình 3.32. Tài nguyên sử dụng trên FPGA [14]



Hình 3.33. Speedup (higher is better) [14]



Hình 3.34. Power Consumption (lower is better) [14]

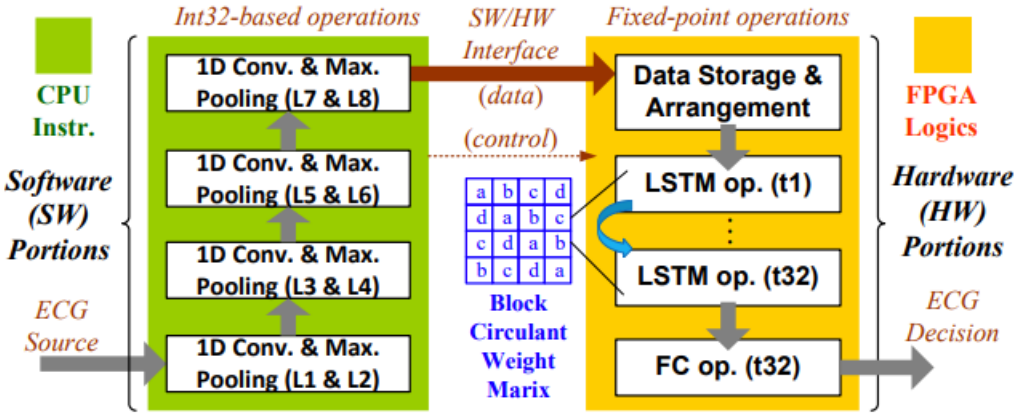
- Sự khác biệt rất lớn giữa tài nguyên sử dụng khi layer có 64 unit và layer có 128 unit: Tài nguyên sử dụng chênh lệch gần gấp 4 lần và layer có 128 unit chiếm tới 95% tài nguyên trền bo mạch. Vì vậy, cần đặc biệt lưu ý về tài nguyên khi xử lý những mạng có kích thước lớn.

- Độ tăng tốc: Nhanh hơn ≈146 lần so với Cortex-A9 và 9 lần so với Core i5 khi xử lý lớp LSTM 64 units. Nhanh hơn ≈ 171 lần so với Cortex-A9 và ≈11 lần so với Core i5 khi xử lý lớp LSTM 128 units. Các phương pháp tối ưu hóa, đặc biệt là giảm khoảng pipeline, đã đóng góp đáng kể vào sự tăng tốc này. Bộ tăng tốc chưa được tối ưu hóa chỉ đạt tốc độ gấp ≈10 lần Cortex-A9 và kém hơn Core i5 do khoảng pipeline dài.

- Công suất tiêu thụ: Tiêu thụ năng lượng trung bình thấp hơn ≈5 lần so với Core i5-4460 khi chạy cùng một lớp mạng LSTM. Mức tiêu thụ năng lượng của bộ tăng tốc được tối ưu hóa chỉ nhỉnh hơn một chút so với bản chưa tối ưu hóa, cho thấy việc tối ưu hóa đạt được hiệu năng cao hơn đáng kể mà không gây tốn kém năng lượng quá nhiều.

3.2.3 Hybrid CNN-LSTM Network for ECG Classification and Its Software-Hardware Co-Design Approach

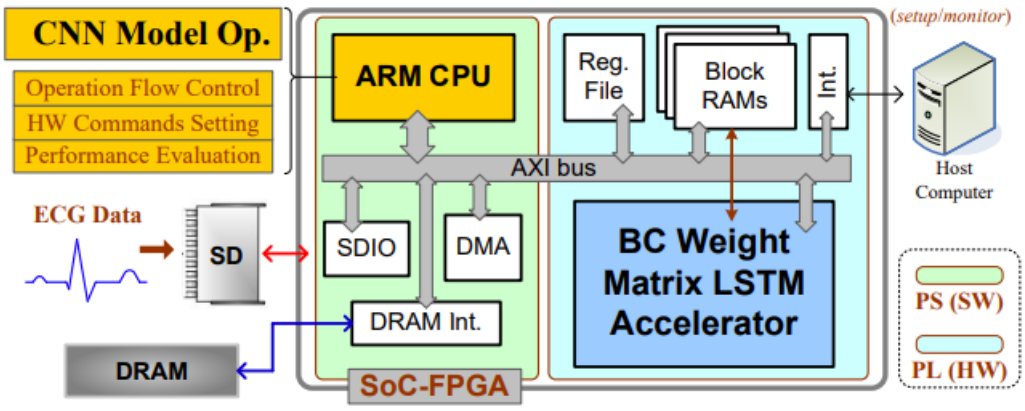
Ý tưởng:



Hình 3.35. Sơ đồ thiết kế cơ bản [15]

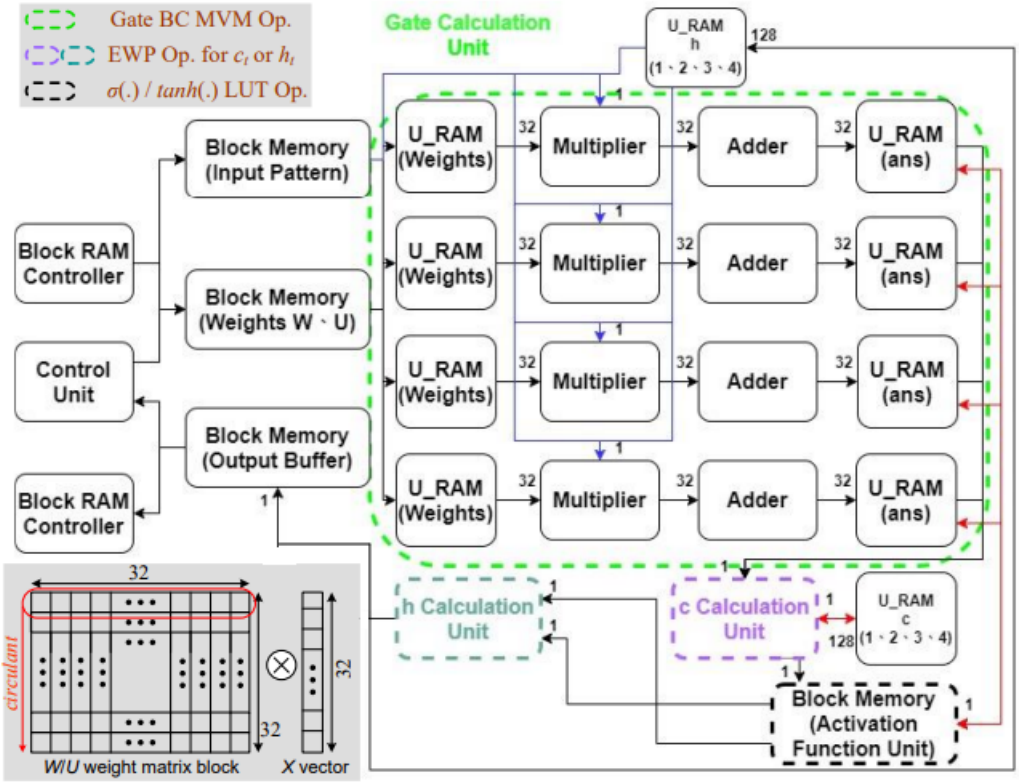
- Sử dụng CNN để trích xuất các đặc trưng không gian từ tín hiệu điện tâm đồ (ECG) và LSTM để nắm bắt các phụ thuộc thời gian trong dữ liệu chuỗi thời gian.
- Triển khai mô hình lai trên nền tảng SoC-FPGA, trong đó CNN được thực hiện bằng phần mềm và LSTM được tăng tốc bằng phần cứng FPGA.
- Sử dụng ma trận trọng số tuần hoàn khối (Block Circulant) trong mô hình LSTM để giảm độ phức tạp tính toán và cho phép triển khai hiệu quả trên phần cứng.

Kiến trúc tổng quát:



Hình 3.36. Sơ đồ thiết kế đồng thời phần mềm/phần cứng [15]

Hình 3.36 minh họa sơ đồ thiết kế trên nền tảng SoC-FPGA. Dữ liệu ECG được lưu trữ trong bộ nhớ ngoài (DRAM) và được truyền đến FPGA thông qua DMA. Các phép toán CNN được thực hiện trên CPU ARM và kết quả được gửi đến bộ tăng tốc LSTM trên FPGA. Bộ tăng tốc LSTM được thiết kế để thực hiện hiệu quả các phép toán LSTM bằng cách sử dụng ma trận trọng số tuần hoàn khối, cho phép lưu trữ dữ liệu trọng số trong bộ nhớ Block RAMs trên FPGA.



Hình 3.37. Kiến trúc bộ tăng tốc LSTM [15]

Bộ tăng tốc này sử dụng ma trận trọng số tuần hoàn khối với kích thước khối là 32, giúp giảm băng thông bộ nhớ và tăng tốc độ xử lý tổng thể, một yếu tố then chốt trong các ứng dụng thời gian thực như phân loại ECG. Bộ tăng tốc LSTM bao gồm các thành phần chính sau:

- Gate Calculation Unit: Thực hiện các phép nhân ma trận-vectơ (MVM) giữa ma trận trọng số BC ($W_{f/i/c/o}$, $U_{f/i/c/o}$) và vectơ đầu vào/trạng thái ẩn (X_t, h_{t-1}). Nhờ sử dụng 4 bộ mô-đun nhân-tích lũy (MAC) 32 đường, đơn vị này có thể thực hiện song song 4 khối MVM kích thước 32, tăng tốc đáng kể các phép toán MVM, vốn là phép toán tốn kém nhất trong LSTM.
- c-or-h Calculation Unit: Thực hiện các phép toán tích theo phần tử (EWP) giữa các vectơ trung gian.

- LUT Unit: Thực hiện các phép toán kích hoạt phi tuyến $\sigma(.)$ và $TanH(.)$ bằng cách tra cứu giá trị từ bảng LUT. Việc sử dụng LUT giúp đơn giản hóa phần cứng và tăng tốc độ xử lý.
- Block Memory: Lưu trữ dữ liệu đầu vào, trọng số và kết quả trung gian. Các khối bộ nhớ U-RAM được sử dụng để lưu trữ tạm thời dữ liệu, trong khi Block RAM được sử dụng để lưu trữ ma trận trọng số BC.
- Control Unit: Điều khiển luồng dữ liệu và hoạt động của các thành phần khác trong bộ tăng tốc.

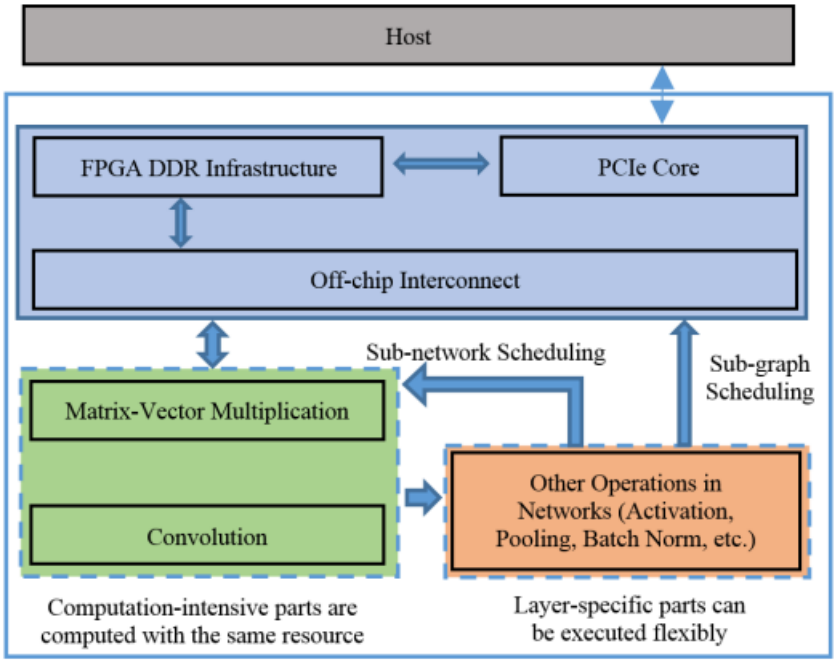
Đánh giá:

1D-CNN SW Ops.	LSTM HW resources (Utilization)				Inference Accuracy*	Execution Time
	LUT	FF	DSP	BRAM		
ARM CPU	36708	36176	130	63	98.63 %	208.2 *ms*

Hình 3.38. Hiệu năng của hệ thống CNN-LSTM trên SoC-FPGA [15]

Nền tảng SoC-FPGA sử dụng thiết bị Xilinx Zynq-7000. Phương pháp này mất khoảng 208 ms để phân loại mỗi đoạn ECG. Giả sử nhịp tim tối đa là 200-bpm, thời gian khả dụng để xử lý mỗi nhịp tim là 300 ms. Điều này có nghĩa là phương pháp được đề xuất đủ nhanh để xử lý tín hiệu ECG trong thời gian thực, ngay cả khi nhịp tim của bệnh nhân cao. Đây là một yếu tố quan trọng để đảm bảo tính ứng dụng của phương pháp trong các thiết bị y tế thực tế. Mô hình có khả năng phân loại 15 loại ECG với độ chính xác đạt 98.63%.

3.2.4 An OpenCL-Based hybrid cnn-rnn Inference Accelerator On FPGA

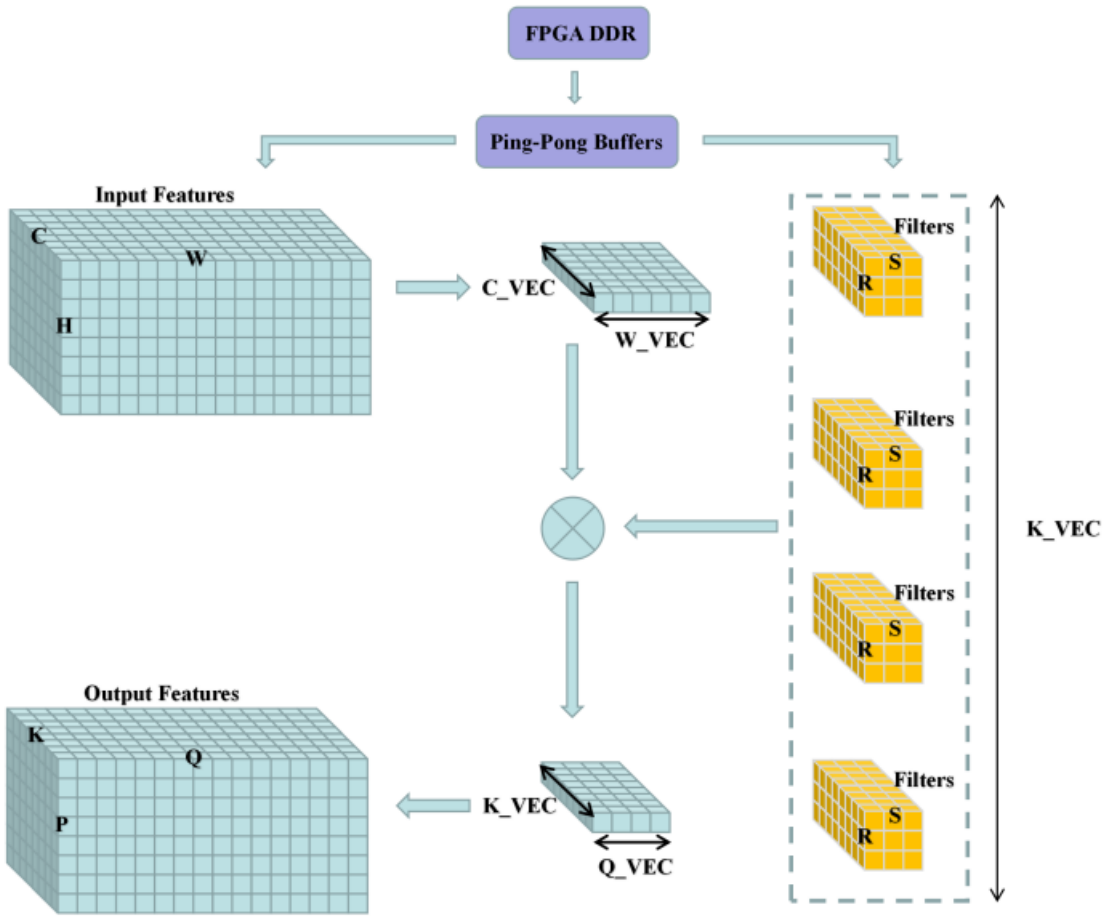


Hình 3.39. Kiến trúc tổng thể của bộ tăng tốc CNN-RNN [16]

Ý tưởng: các phép tính như nhân ma trận-vectơ trong RNN và tích chập trong CNN thực hiện trên cùng trên cùng một tài nguyên phần cứng mà không làm giảm hiệu suất.

Kiến trúc tổng quát:

Mặc dù cấu trúc liên kết và độ phức tạp của DNN có thể khác nhau, nhưng tất cả chúng đều được cấu tạo bằng cách xếp chồng các lớp khác nhau, chẳng hạn như lớp tích chập, lớp hồi quy, fully connected layers, pooling layers, activation layers,... Trong số các lớp này, lớp tích chập và lớp hồi quy là những lớp tốn nhiều tài nguyên và tính toán nhất chiếm phần lớn thời gian thực thi mạng.



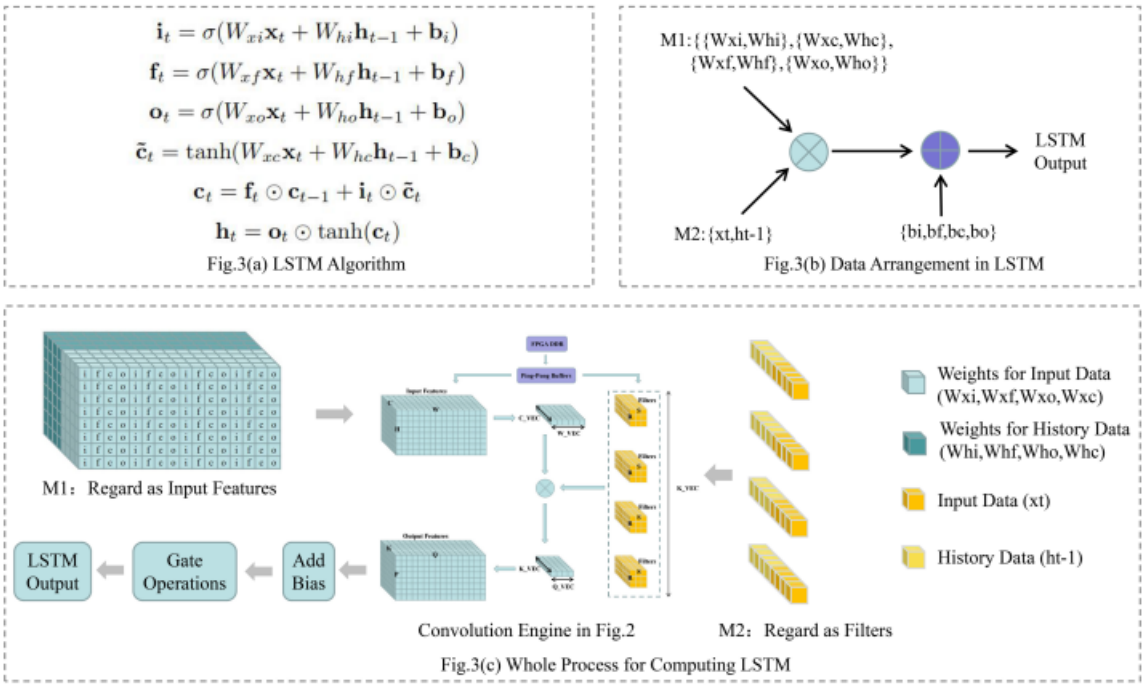
Hình 3.40. Nguyên lý của Convolution Engine [16]

Hình 3.40 minh họa nguyên lý hoạt động của engine tích chập được sử dụng trong bộ tăng tốc CNN-RNN. Engine này được thiết kế để đạt được thông lượng cao và hiệu quả DSP tối ưu bằng cách áp dụng kỹ thuật vector hóa đa chiều. Các bước hoạt động:

- Nạp dữ liệu: Dữ liệu đầu vào (Input Features) và bộ lọc (Filters) được nạp từ bộ nhớ FPGA DDR vào bộ đệm Ping-Pong.
- Vector hóa: Dữ liệu đầu vào và bộ lọc được vector hóa theo nhiều chiều (C_VEC, W_VEC, K_VEC).
- Tính toán: Mỗi phần tử xử lý (PE) trong engine tích chập sẽ nhận $W\_VEC * C\_VEC$ dữ liệu đầu vào và thực hiện phép tính tích chập, tạo ra Q_VEC đặc trưng đầu ra trung gian.
- Tích lũy: Các kết quả trung gian Q_VEC được tích lũy cho đến khi hoàn thành tích chập trên cả hai chiều R và C.

- Xuất kết quả: Đặc trưng đầu ra (Output Features) được gửi ra khỏi engine.

Hiệu quả: Bằng cách vector hóa, engine có thể tận dụng tối đa các khối DSP (Digital Signal Processor) có sẵn trên FPGA. Dữ liệu được đưa vào engine theo luồng, giúp DSP luôn bận rộn và không có chu kỳ nhàn rỗi.



Hình 3.41. Ánh xạ hoạt động LSTM vào Convolution Engine [16]

Hình 3.41 minh họa cách thức ánh xạ các phép toán LSTM lên engine tích chập, cho phép tận dụng engine này để tăng tốc cả CNN và RNN.

- Biểu diễn LSTM: Hình 3(a) trình bày thuật toán LSTM tiêu chuẩn với 4 cổng (input, forget, update, output) và các phương trình tính toán tương ứng.
- Sắp xếp dữ liệu: Hình 3(b) minh họa cách sắp xếp dữ liệu cho LSTM. Các trọng số của 4 cổng được gộp thành một ma trận lớn M1. Dữ liệu đầu vào (x_t) và trạng thái ẩn trước đó (h_{t-1}) được gộp thành ma trận M2.
- Ánh xạ lên engine tích chập: Hình 3(c) mô tả toàn bộ quá trình tính toán LSTM bằng engine tích chập. M1 được coi là dữ liệu đầu vào của phép tích chập, M2 được coi là bộ lọc. Engine tích chập thực hiện phép nhân ma trận-vector giữa M1 và M2, tạo ra song song kết quả cho 4 cổng LSTM. Các phép toán cộng bias và kích hoạt được thực hiện sau đó.

Bên cạnh các lớp tích chập và hồi quy, một số lớp khác trong cấu trúc liên kết DNN cũng có thể được hỗ trợ trong thiết kế. Các lớp fully connected có thể được ánh xạ lên convolution engine theo phương pháp được đề cập trong Intel DLA [17]. Batch normalization cũng có thể được tích hợp vào convolution. Đối với các lớp không tương thích về mặt số học với convolution engine, như pooling và activation, các lớp này được xử lý riêng biệt sau engine tích chập. Chúng được kết nối với engine thông qua kênh FIFO (First-In, First-Out) và được thực thi theo kiểu streaming bằng OpenCL API.

Đánh giá:

Kết quả được đánh giá dựa trên nền tảng bao gồm:

- Host: CPU Intel(R) Core(TM) i9-7900X 3.3 GHz, chạy hệ điều hành Ubuntu 16.04.3 LTS.
- Device: FPGA Intel Arria 10 1150 GX, kết nối với host thông qua khe cắm PCIe Gen 3x8.

Nền tảng này sử dụng OpenCL để lập trình và điều khiển FPGA. Kernel OpenCL được biên dịch bằng Intel OpenCL SDK for FPGA Version 17.0, sau đó được tổng hợp thành bitstream phần cứng và cấu hình lên board FPGA.

Model	Data Type	Device	Frequency	Design Method	Throughput in GOPS
Pruned LRCN [12]	16bit &12bit	Virtex-7 VC709	100 MHz	HLS	36.25
CRNN (this)	16bit	Arria 10 1150 GX	268 MHz	OpenCL	646
VGG + LSTM [10]	16bit	Stratix V GSMD5	150 MHz	RTL-HLS	364 for CNN 315 for RNN
LRCN [13]	8bit &12bit	Xilinx ZU5EG	200 MHz	RTL	690 (20% Sparse)

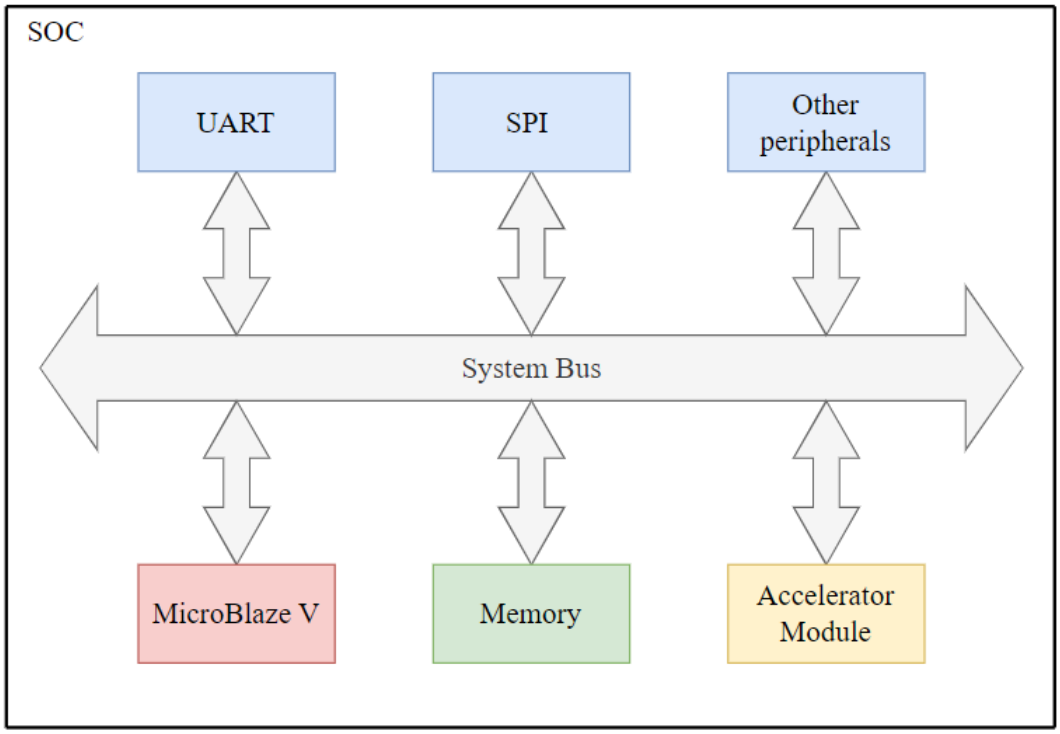
Hình 3.42. Ánh xạ hoạt động LSTM vào Convolution Engine [16]

Hiệu năng của bộ tăng tốc được so sánh với các bộ tăng tốc FPGA khác dành cho mạng lai (hybrid networks). Triển khai mạng CRNN, bao gồm cấu trúc CNN giống VGG và hai lớp LSTM hai chiều. Do chưa sử dụng kỹ thuật lượng tử hóa để giảm độ chính xác biểu diễn tham số LSTM, kích thước hidden lớn nhất mà bộ tăng tốc hỗ trợ là 512. Tuy nhiên, với hạn chế này, bộ tăng tốc vẫn đạt được thông lượng 646 GOPS, vượt xa hiệu năng của VGG + LSTM [19] và gần với phiên bản RTL của LRCN [20].

CHƯƠNG 4

ĐỀ XUẤT HỆ THỐNG

4.1 Hệ thống



Hình 4.43. Tổng quan hệ thống

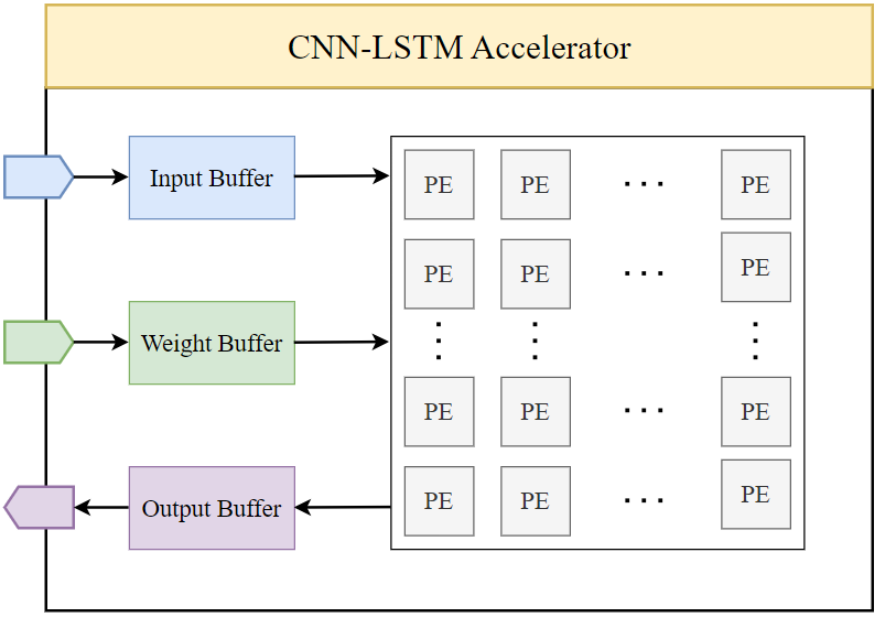
Hệ thống bao gồm:

- MicroBlaze V: Lõi MicroBlaze V đóng vai trò chính trong việc điều khiển hoạt động của SoC bao gồm việc thực hiện các tính toán thông thường, quản lý bộ nhớ và các điều khiển hoạt động giao tiếp của ngoại vi.
- Memory: Là bộ nhớ trong phục vụ cho hoạt động của lõi MicroBlaze V. Có thể là SRAM hoặc DRAM hoặc bất kỳ loại bộ nhớ nào phù hợp.
- CNN-LSTM Accelerator: Một module chuyên dụng nhằm thực hiện các tính toán đặc thù dành riêng cho mô hình CNN-LSTM, có khả năng điều phối dataflow linh động tùy thuộc vào trạng thái layer đang tính toán để cho ra kết quả tối ưu.

- UART, SPI và các ngoại vi khác đóng vai trò như giao diện giao tiếp với các thành phần kết nối với SoC như cảm biến, thiết bị điều khiển,..
- System Bus: Đóng vai trò kết nối toàn bộ hệ thống SoC. System Bus có thể được thay đổi thành các dạng bus phức tạp như AXI,...

4.2 Bộ tăng tốc - CNN-LSTM Accelerator

Nhóm đề xuất kiến trúc Accelerator sẽ nhận vai trò hoạt động ở cấp layer.



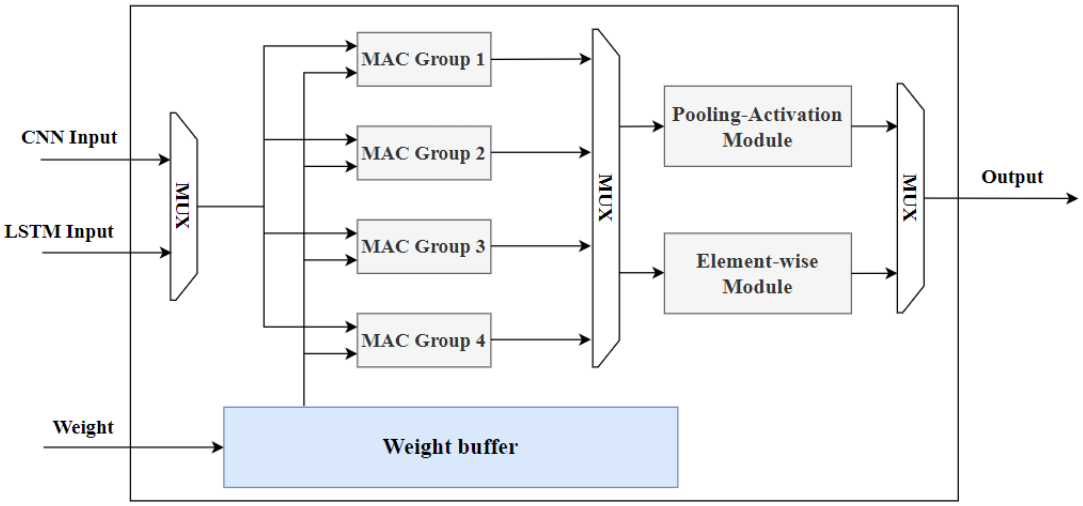
Hình 4.44. Kiến trúc bộ tăng tốc

- Input buffer: Bộ nhớ đệm chứa các đầu vào cho mô hình cnn-lstm.
- Output buffer: Bộ nhớ đệm chứa đầu ra là các kết quả tính toán của khối tăng tốc.
- Weight buffer: Dùng để lưu các trọng số cần thiết trong công việc tính toán.
- Các khối PE (Processing Element): Các khối này sẽ được hoạt động song song để đạt tốc độ tốt nhất.

4.2.1 PE

Để đạt được các phép tính song song với N PE, chúng ta cần cấu hình hợp lý dữ liệu đầu vào và trọng số, phù hợp với các đặc điểm tính toán của mạng CNN và

LSTM. Đầu tiên, các trọng số được lưu trữ trong BRAM bên trong từng PE trước khi tính toán. Tiếp theo, dữ liệu đầu vào được đưa vào tất cả các PE cùng lúc, cho phép các PE hoạt động song song.



Hình 4.45. Kiến trúc của khối PE

Mỗi PE bao gồm một BRAM được gọi là bộ đệm trọng số (weight buffer), bốn mô-đun nhóm MAC (MAC Group), các mô-đun pooling-Activation, và các mô-đun tính toán theo từng phần tử (element-wise modules). Kiến trúc của đơn vị PE được hiển thị trong hình 4.45.

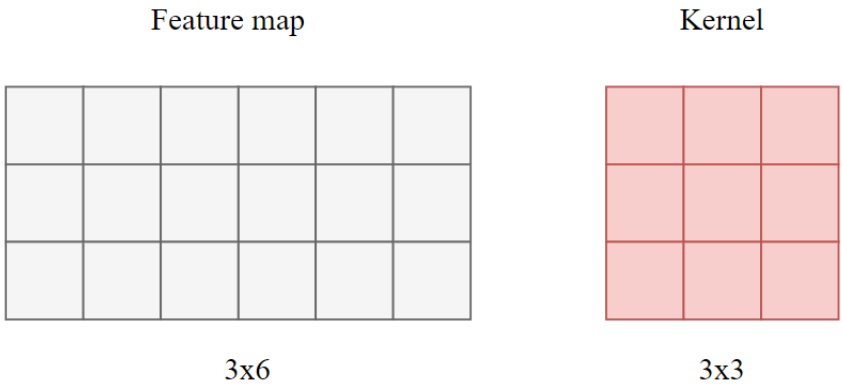
Mô-đun PE có khả năng tái cấu hình, cho phép tăng tốc tính toán trong cả hai chế độ CNN và LSTM. Ngoài ra, bộ đệm trọng số và bốn mô-đun nhóm MAC được sử dụng chung trong cả hai chế độ.

Phân bổ dữ liệu đầu vào và trọng số cho hai chế độ tính toán là khác nhau.

- Trong chế độ CNN, bộ đệm trọng số (weight buffer) của một PE lưu trữ toàn bộ các tham số của một kernel tích chập trong lớp CNN hiện tại, và bốn nhóm MAC (MAC Groups) thực thi song song. Kết quả từ các nhóm MAC được lưu vào thanh ghi (register) và lần lượt được đưa vào mô-đun pooling-Activation để hoàn thành quá trình pooling và kích hoạt (activation).
  - Nhóm MAC đầu tiên: Tương ứng với tính toán tích chập của kernel tại vị trí ban đầu trên bản đồ đặc trưng (feature map).
  - Nhóm MAC thứ hai: Tương ứng với tính toán tích chập khi kernel trượt một bước nhất định (stride) theo hướng một chiều trên bản đồ đặc trưng.

– Tương tự, các nhóm MAC còn lại thực hiện tích chập tại các vị trí khác nhau trên bản đồ đặc trưng cùng lúc. Điều này có nghĩa là bốn nhóm DSP có thể đồng thời thực thi các phép tính tích chập tại bốn vị trí khác nhau trên bản đồ đặc trưng.

– Ví dụ ta có một bản đồ đặc trưng (feature map) và hạt nhân (kernel) hay còn gọi là bộ lọc như hình 4.46.

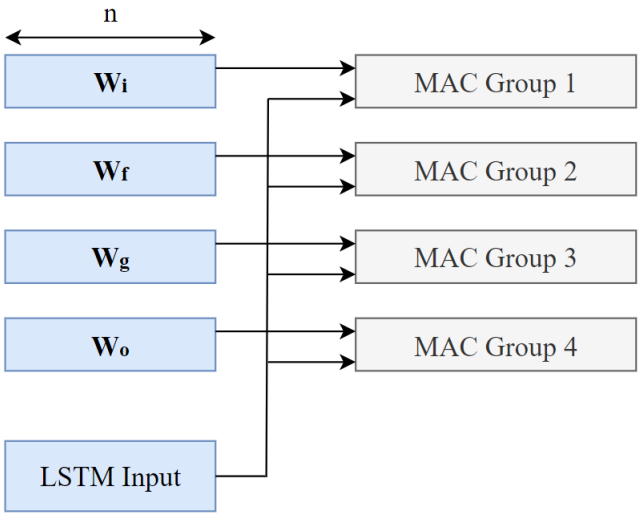


Hình 4.46. Feature map và kernel

Ta sẽ trượt bộ lọc với bước nhảy là 1 trên bản đồ đặc trưng. Mỗi khu vực được bộ lọc trượt qua sẽ được chuyển tới từng MAC Group khác nhau để tính toán.

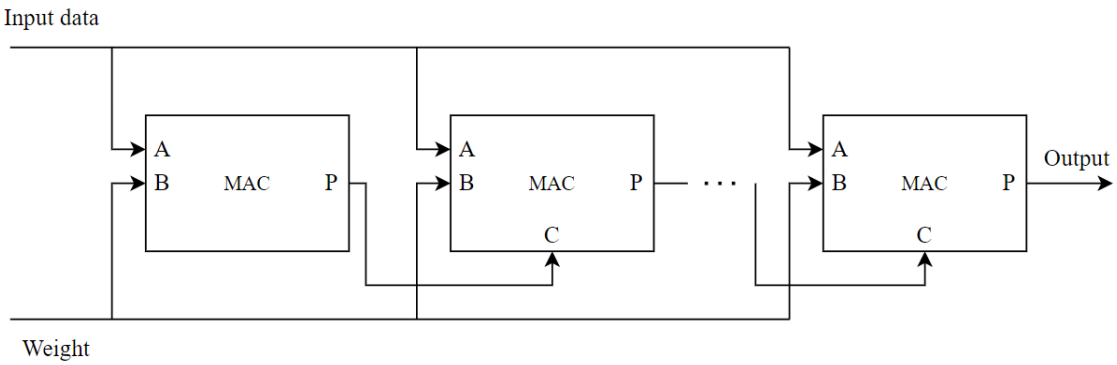
Hình 4.47. Phân phối dữ liệu vào các MAC Group trong chế độ CNN

- Trong chế độ LSTM, bộ đệm trọng số (weight buffer) của PE lưu trữ tất cả các trọng số của một nút trong lớp ẩn của lớp LSTM hiện tại. Theo đặc điểm của cấu trúc mạng LSTM, chúng tôi kết hợp đầu vào x_t và h_{t-1} thành một vector dài một chiều. Các trọng số của cổng quên f_t, cổng vào i_t, cổng ra o_t, và g_t, cũng được kết hợp thành một ma trận trọng số. Bước tiếp theo, ma trận trọng số được chia thành bốn vector dài một chiều: w_i, w_f, w_g và w_o. Bốn Nhóm MAC được sử dụng để thực hiện các phép tính nhân tích lũy của dữ liệu đầu vào và trọng số chứa w_i, w_f, w_g và w_o.



Hình 4.48. Phân phối dữ liệu vào các MAC Group trong chế độ LSTM

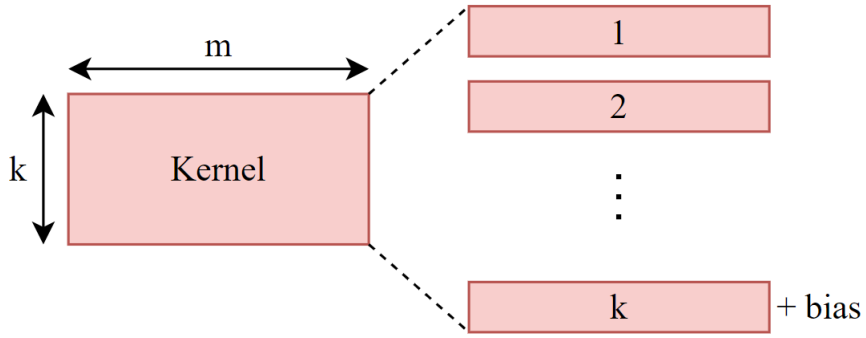
4.2.2 MAC Group



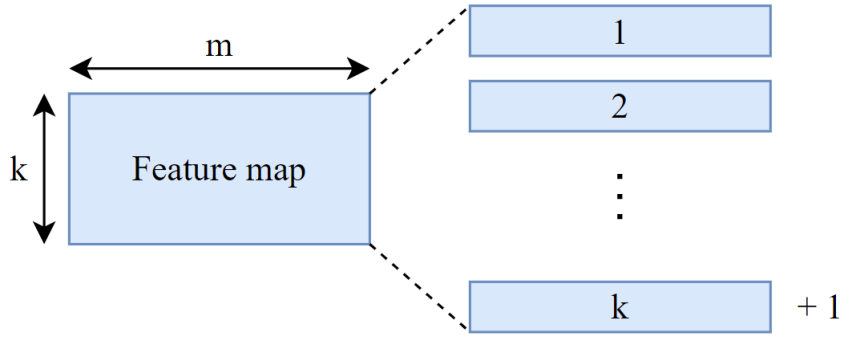
Hình 4.49. Kiến trúc của MAC Group

Mỗi nhóm MAC (MAC Group) bao gồm nhiều nhiều khối MAC nhỏ như hình 4.49. Số lượng của khối MAC trong MAC Group sẽ bằng số hàng của bộ lọc (kernel). Mặc dù nhóm MAC được sử dụng theo cách giống nhau trong cả hai chế độ, nhưng phương pháp phân bổ dữ liệu đầu vào là khác nhau. Nhưng trong cả 2 chế độ khi dữ liệu đầu vào và trọng số đã được phân phối vào các MAC Group thì sẽ có kích thước giống nhau.

- Ở chế độ **CNN** cả bản đồ đặc trưng và bộ lọc sẽ có cùng kích thước với nhau $k \times m$ trong đó k là số hàng, còn m là số cột. Chúng được tách ra thành k vector một chiều có kích thước $1 \times m$. Sau đó ở cuối vecto thứ k trong dữ liệu đầu vào thì sẽ được thêm vào giá trị 1. Còn ở cuối vector thứ k của bộ lọc sẽ được thêm giá trị bias.

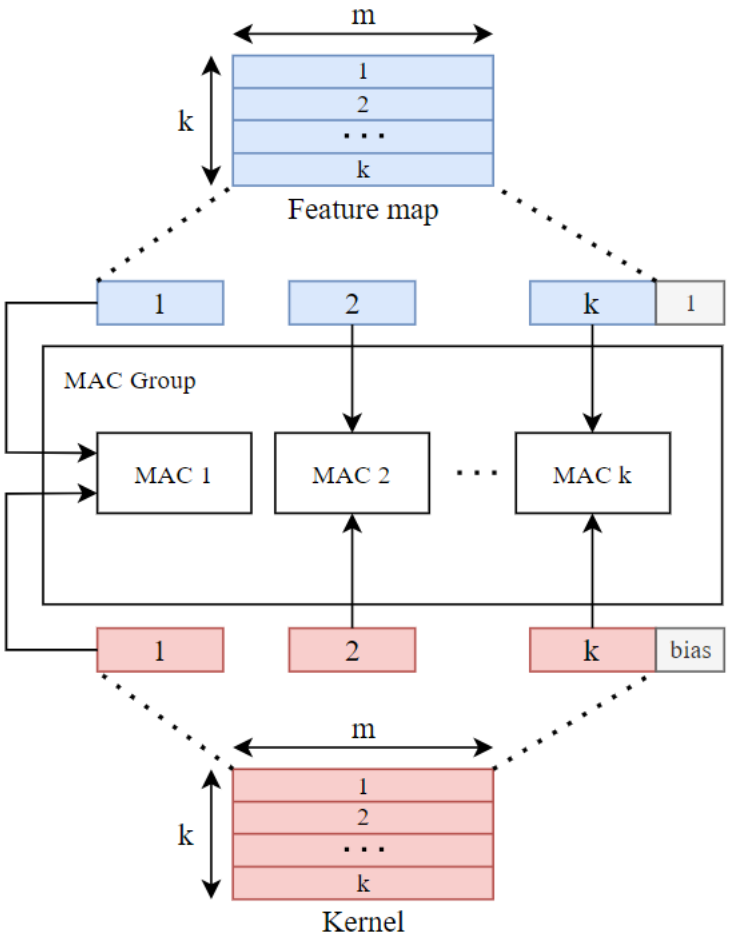


Hình 4.50. Phân rã dữ liệu bộ lọc để đưa vào các khối MAC nhỏ



Hình 4.51. Phân rã bản đồ đặc trưng để đưa vào các khối MAC nhỏ

Từ đó ta đưa từng vector $1 \times m$ của bộ lọc và bản đô đặc trưng tương ứng từ hàng 1 đến k vào các khối MAC nhỏ để tính toán phép nhân ma trận vì thế nên trong mỗi MAC Group sẽ có k khối MAC nhỏ

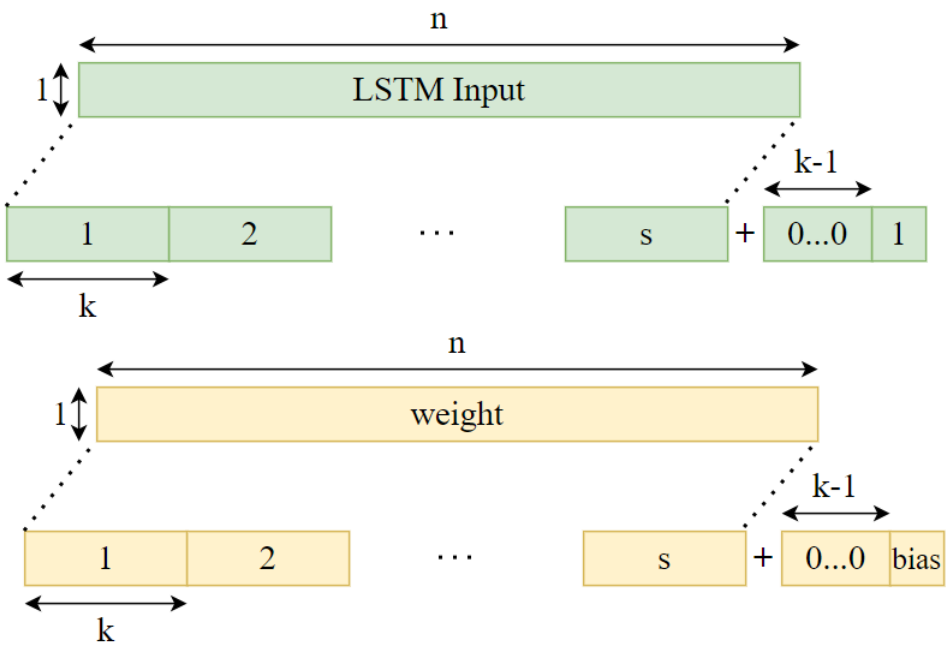


Hình 4.52. Phân phối dữ liệu của CNN vào các khối MAC nhỏ

- Trong chế độ **LSTM**, dữ liệu đầu vào và trọng số đều được biểu diễn dưới dạng các vector dài một chiều kích thước $1 \times n$ Chúng được chia thành nhiều vector con kích thước $1 \times k$ và số lượng vector con là s. Nếu n không chia hết cho k thì các vị trí còn lại sẽ điền thêm số 0.

  Ngoài ra phần cuối dữ liệu đầu vào và trọng số cần phải được xử lý để phù hợp với các phép tính. Để bổ sung thêm, một vector con một chiều có kích thước $1 \times k$ được thêm vào cuối của vector LSTM Input và vector weight. Trong đó phần tử cuối cùng của vector được thêm vào sẽ là 1 nếu là vector LSTM Input và sẽ là bias nếu là vector weight.

  Mỗi lần, cả dữ liệu đầu vào và trọng số sẽ lấy ra một vector con $1 \times k$ (bởi vì có k khối MAC nhỏ trong MAC Group), sau đó được gửi riêng biệt đến các khối MAC nhỏ khác nhau cho đến khi tính toán tương ứng hoàn tất.



Hình 4.53. Phân phối dữ liệu của LSTM vào các khối MAC nhỏ

4.2.3 Khối Pooling-Activation

Lớp pooling sử dụng phương pháp max pooling, trong đó so sánh từng giá trị dữ liệu đầu vào để tìm giá trị lớn nhất.

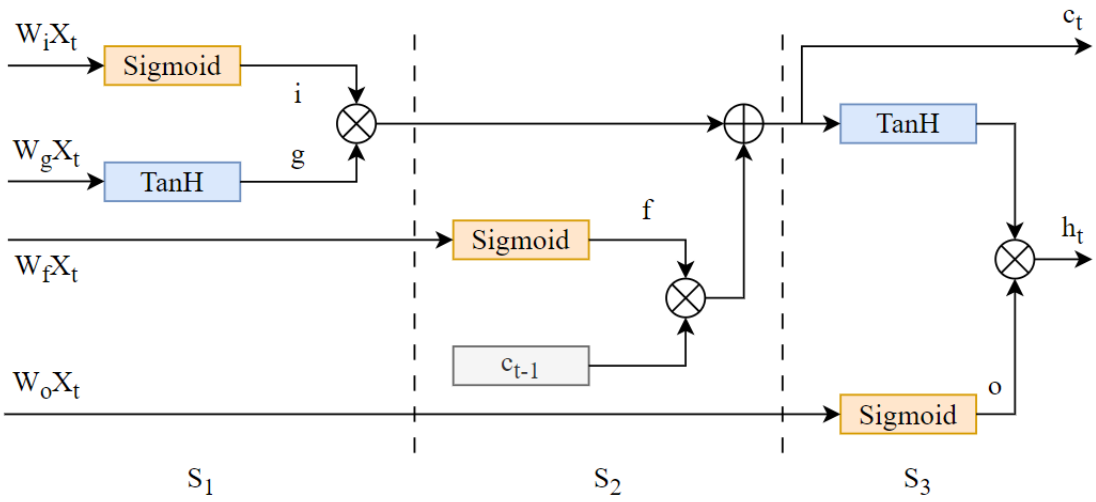
Sau đó, một số hàm kích hoạt được kích hoạt phi tuyến.

4.2.4 Khối Element Wise

Trong thiết kế này, khối element_wise sử dụng chiến lược time-sharing multiplexing, chia các phép tính thành ba chu kỳ.

Time-Sharing Multiplexing là một giải pháp hiệu quả cho các hệ thống yêu cầu chia sẻ tài nguyên vật lý nhưng vẫn đảm bảo hoạt động ổn định và khả năng phục vụ nhiều người dùng.

Module này thực chất chỉ bao gồm một khối sigmoid, một khối tanh, một bộ cộng và một bộ nhân. Cấu trúc của module được minh họa trong Hình 4.54



Hình 4.54. Cấu trúc của mô-đun Element-wise.

CHƯƠNG 5

KẾT LUẬN

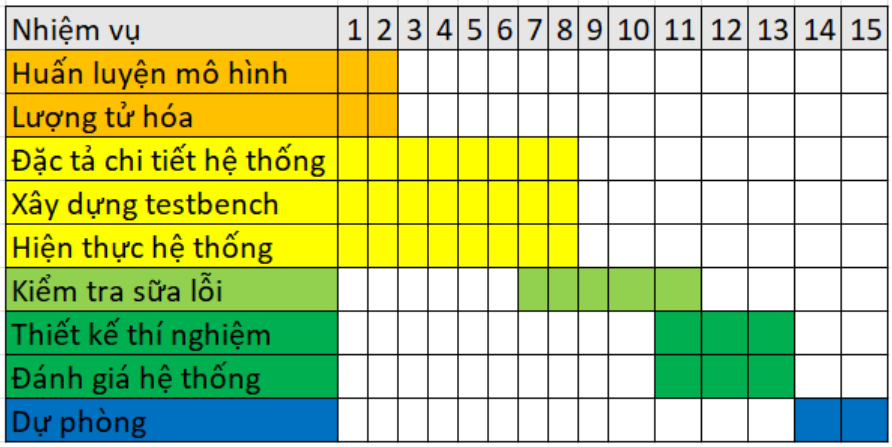
5.1 Mục tiêu

- Định hướng thiết kế, đặc tả chi tiết và hiện thực source RTL cho module CNN-LSTM Accelerator tích hợp vào SoC Microblaze V.
- Xây dựng và hiện thực bộ testbench kiểm thử cho toàn bộ module và hệ thống.
- Xây dựng và hiện thực hướng tối ưu cấp phần mềm (sử dụng một hoặc cả hai kĩ thuật Quantization và Pruning) từ các ví dụ mô hình mẫu trên nền tảng TensorFlow.

5.2 Kế hoạch thực hiện

Đồ án được thực hiện tiếp trong giai đoạn sau với thời gian là 15 tuần. Nhóm dự định phân bổ thời gian như sau:

- Tuần 1 - 2: Huấn luyện và lượng tử hóa mô hình trên TensorFlow
- Tuần 1 - 8: Đặc tả chi tiết cho các module, hệ thống và hiện thực.
- Tuần 7 - 11: Kiểm tra sữa lỗi.
- Tuần 11 - 13: Thiết kế, thực hiện các thí nghiệm, đánh giá hiệu năng hệ thống
- Tuần 14 - 15: Thời gian dự trù cho các sự cố phát sinh.



Hình 5.55. Bảng tiến độ

Tài liệu tham khảo

[1] RISC-V Technical Specifications.

Available: https://lf-riscv.atlassian.net/wiki/spaces/HOME/pages/16154769/RISC

[2] AMD MicroBlaze™ V Processor

Available: https://www.amd.com/en/products/software/adaptive-socs-and-fpgas/microblaze.html

[3] Thuật toán CNN là gì? Tìm hiểu về Convolutional Neural Network

Available: https://vietnix.vn/cnn-la-gi/

[4] Multiple Input and Multiple Output Channels

Available: https://classic.d2l.ai/chapter_convolutional-neural-networks/channels.html

[5] Dr. Thanh-Sach LE, Artificial Neural Networks and Deep Learning Slides, GVLab: Graphics and Vision Laboratory, Faculty of Computer Science and Engineering, HCMUT

[6] Lý thuyết về mạng LSTM

Available: https://phamdinhkhanh.github.io/2019/04/22/Ly_thuyet_ve_mang_LSTM.html

[7] L. Li, W. Xie, and X. Zhou: "Cooperative spectrum sensing based on LSTM-CNN combination network in cognitive radio system"

Available: https://ieeexplore.ieee.org/document/10217825

[8] Xin Zhou, Wei Xie, Han Zhou, Yongjing Cheng, Ximing Wang, Yun Ren: "An Accelerated FPGA-Based Parallel CNN-LSTM Computing Device"

Available: https://ieeexplore.ieee.org/document/10621060

[9] lowRisc, Ibex Reference Guide/Pipeline Details, 2018.

Available: https://ibex-core.readthedocs.io/en/latest/03_reference/pipeline_details.html

[10] PicoRV32 - A Size-Optimized RISC-V CPU

Available: https://github.com/YosysHQ/picorv32

[11] OpenHW Group, Introduction to CV32E40P

Available: https://docs.openhwgroup.org/projects/cv32e40p-user-manual/en/latest/intro.html

[12] OpenHW Group, CVA6 Requirement Specification

Available: https://docs.openhwgroup.org/projects/cva6-user-manual/02_cva6_requirements/cva6_requirements_specification.html#introduction

[13] Jinmook Lee, Changhyeon Kim, Sanghoon Kang, Dongjoo Shin, Sangyeob Kim, Hoi-Jun Yoo: "UNPU: An Energy Efficient Deep Neural Network Accelerator With Fully Variable Weight Bit Precision" pp.173 - 185, October 4, 2018

Available: https://ieeexplore.ieee.org/document/8481682

[14] Yiwei Zhang, Chao Wang, Lei Gong, Yuntao Lu, Fan Sun, Chongchong Xu: "Implementation and Optimization of the Accelerator Based on FPGA Hardware for LSTM Network"

Available: https://ieeexplore.ieee.org/document/8367328

[15] Song-Nien Tang, Yuan-Ho Chen, Yu-Wei Chang, Yu-Ting Chen, Shuo-Hung Chou: "Hybrid CNN-LSTM Network for ECG Classification and Its Software-Hardware Co-Design Approach"

Available: https://ieeexplore.ieee.org/document/10396448

[16] Yunfei Sun, Brian Liu, Xianchao Xu: "An OpenCL-Based Hybrid CNN-RNN Inference Accelerator On FPGA"

Available: https://ieeexplore.ieee.org/document/8977882

[17] Mohamed S. Abdelfattah, David Han, Andrew Bitar, Roberto DiCecco, Shane O'Connell, Nitika Shanker: "DLA: Compiler and fpga overlay for neural network inference acceleration"

Available: https://ieeexplore.ieee.org/document/8533533

[18] Ying Yang, Fen Ge, Danfeng Qiu, Xin Yue, Ziyu Li, Fang Zhou: "Implementation of Reconfigurable CNN-LSTM Accelerator Based on FPGA"

Available: https://ieeexplore.ieee.org/document/9657920

[19] Jinmook Lee, Changhyeon Kim, Sanghoon Kang, Dongjoo Shin, Sangyeob Kim, Hoi-Jun Yoo: "FP-DNN: An Automated Framework for Mapping Deep Neural Networks onto FPGAs with RTLHLS Hybrid Templates"

Available: https://ieeexplore.ieee.org/document/7966671

[20] Shulin Zeng, Kaiyuan Guo, Shaoxia Fang, Junlong Kang, Dongliang Xie, Yi Shan: "An Efficient Reconfigurable Framework for General Purpose CNN-RNN Models on FPGAs"

Available: https://ieeexplore.ieee.org/document/8631880