

ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA ĐIỆN – ĐIỆN TỬ
BỘ MÔN ĐIỆN TỬ

-----o0o-----



ĐỒ ÁN TỐT NGHIỆP ĐẠI HỌC

**ĐÈN GIAO THÔNG CÓ PHÁT HIỆN KẾT XE TẠI GIAO LỘ BẰNG MÔ HÌNH
MÁY HỌC TRÊN FPGA**

GVHD: ThS. Trần Hoàng Quân

SVTH: Nguyễn Thanh Toàn

MSSV: 2014777

TP. HỒ CHÍ MINH, THÁNG 03 NĂM 2025

ĐẠI HỌC QUỐC GIA TP.HỒ CHÍ MINH CỘNG HÒA XÃ HỘI CHỦ NGHĨA VIỆT NAM
TRƯỜNG ĐẠI HỌC BÁCH KHOA Độc lập – Tự do – Hạnh phúc.

-----☆-----
Số: _____/BKĐT
Khoa: **Điện – Điện tử**
Bộ Môn: **Điện Tử**

-----☆-----

NHIỆM VỤ LUẬN VĂN TỐT NGHIỆP

1. HỌ VÀ TÊN: NGUYỄN THANH TOÀN MSSV: 2014777
 2. NGÀNH: **ĐIỆN TỬ - VIỄN THÔNG** LỚP : DD20DV1
 3. Đề tài: Đèn giao thông có phát hiện kẹt xe tại giao lộ bằng mô hình máy học trên FPGA
 4. Nhiệm vụ (Yêu cầu về nội dung và số liệu ban đầu):
 - Thiết kế mô hình mobile net có khả năng phân biệt kẹt xe và không kẹt xe từ bức ảnh đầu vào.
 - Nghiên cứu thực hiện mô hình mobile net trên FPGA, cụ thể kit DE0-nano.

.....

.....

.....

.....
 5. Ngày giao nhiệm vụ luận văn: 10/02/2025.
 6. Ngày hoàn thành nhiệm vụ:
 7. Họ và tên người hướng dẫn: Phần hướng dẫn
.....
.....
- Nội dung và yêu cầu LVTN đã được thông qua Bộ Môn.

Tp.HCM, ngày..... tháng..... năm 20

CHỦ NHIỆM BỘ MÔN

NGƯỜI HƯỚNG DẪN CHÍNH

PHẦN DÀNH CHO KHOA, BỘ MÔN:

Người duyệt (chấm sơ bộ):.....
Đơn vị:.....
Ngày bảo vệ :
Điểm tổng kết:
Nơi lưu trữ luận văn:

LỜI CẢM ƠN

Lời đầu tiên, em xin trân trọng cảm ơn giảng viên thầy Trần Hoàng Quân - Người đã trực tiếp hướng dẫn em trong quá trình hoàn thành đồ án. Em cũng xin được gửi lời cảm ơn đến quý thầy, cô giáo trường đại học Bách Khoa Tp.Hồ Chí Minh, đặc biệt là các thầy, cô khoa Kỹ thuật Điện tử, những người đã truyền lửa và giảng dạy kiến thức cho em suốt thời gian qua. Em đã cố gắng vận dụng những kiến thức đã học được và tìm tòi thêm nhiều thông tin để hoàn thành đồ án. Tuy nhiên, do kiến thức còn hạn chế và không có nhiều kinh nghiệm trên thực tiễn nên khó tránh khỏi những thiếu sót trong bài làm. Rất kính mong quý thầy, cô cho em thêm những góp ý để kết quả của em được hoàn thiện hơn. Cuối cùng em xin cảm ơn đến gia đình, bạn bè đã luôn chia sẻ, ủng hộ, động viên và giúp đỡ trong suốt quá trình học tập của bản thân. Em xin trân trọng cảm ơn!

Tp. Hồ Chí Minh, ngày 25 tháng 03 năm 2025

Sinh viên

TÓM TẮT LUẬN VĂN

Trong bối cảnh phát triển nhanh chóng của các công nghệ xử lý tín hiệu và trí tuệ nhân tạo, việc ứng dụng các mô hình học sâu vào các hệ thống nhúng đang trở thành xu hướng thiết yếu trong các ứng dụng thực tiễn. Luận văn này trình bày quá trình nghiên cứu và triển khai mô hình MobileNet trên nền tảng FPGA với bộ kit DE0-Nano, hướng đến giải pháp phát hiện kẹt xe tại các vị trí giao lộ – một vấn đề đang được quan tâm trong quản lý giao thông đô thị.

Phần nghiên cứu tập trung vào việc thực hiện mô hình MobileNet phù hợp với kiến trúc FPGA, qua đó đảm bảo khả năng thực thi nhanh chóng và tiêu thụ điện năng thấp. Quá trình chuyển đổi mô hình từ môi trường phần mềm sang phần cứng được thực hiện thông qua các kỹ thuật giảm độ chính xác (quantization) và tối ưu hóa cấu trúc mạch số, cố gắng thực hiện mô hình tính toán trên nguồn tài nguyên bị hạn chế.

Kết quả Khẳng định tiềm năng ứng dụng của FPGA trong việc triển khai các mô hình học sâu cho các bài toán nhận dạng trong thời gian thực, đồng thời mở ra hướng phát triển mới cho các giải pháp thông minh trong lĩnh vực quản lý giao thông. Luận văn không chỉ góp phần đẩy mạnh ứng dụng công nghệ hiện đại vào thực tiễn mà còn là cơ sở nghiên cứu cho các hệ thống nhúng hiệu năng cao trong tương lai.

Luận văn này trình bày về tổng quan lý thuyết về máy học (các mô hình dự đoán, phân loại hình ảnh, đặc biệt là mô hình mobile net phù hợp cho những phần cứng có nguồn tài nguyên hạn chế). Chi tiết về xây dựng bộ dữ liệu, xây dựng mô hình, chạy thử nghiệm mô hình trên máy tính. Chi tiết về thiết kế phần cứng hệ thống trên FPGA để thực hiện được mô hình trên FPGA.

MỤC LỤC

1. GIỚI THIỆU	1
1.1 Tổng quan	1
1.2 Tình hình nghiên cứu trong và ngoài nước	1
1.3 Nhiệm vụ luận văn.....	2
2. LÝ THUYẾT VỀ MÔ HÌNH PHÂN LOẠI ẢNH.	4
2.1 Tìm hiểu về mô hình CNN.....	4
2.2 Những hạn chế của mô hình CNN	6
2.3 Giới thiệu mô hình Mobile Net.....	7
3. CHUẨN BỊ DỮ LIỆU	10
3.1 Bộ dữ liệu training và testing	10
3.2 Thêm dữ liệu cho testing phù hợp với giao thông Việt Nam.....	11
4. THIẾT KẾ VÀ XÂY DỰNG MÔ HÌNH TRÊN MÁY TÍNH.....	15
5. KẾT QUẢ THỰC HIỆN MÔ HÌNH TRÊN MÁY TÍNH.	22
6. THỰC HIỆN PHẦN CỨNG.....	24
6.1 Kiến trúc tổng quan.....	24
6.2 Khối Camera control.....	24
6.2.1 Camera control read	25
6.2.2 Control write to FIFO	26
6.2.3 control store data to sdram	27
6.3 Khối main mobile net	30
6.3.2 Ram nguồn (source ram) và Ram đích (destination ram)	33
6.3.4 Khối thuật toán Average	41
6.3.5 Write back destiantion ram to SDRAM	46
6.4 Module master control	48
7. VERIFICATION	54

7.1 Kiểm tra khối camera read và lưu dữ liệu vào FIFO.....	54
7.2 Testbench module fetching memory	56
7.3 Testbench module write back	57
8. SYNTHESIS.....	59
8.1 Tìm hiểu về kit DE0-Nano.....	59
8.2 Synthesis project.....	60
9. KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN	65
9.1 Kết luận.....	65
9.2 Hướng phát triển	65
10. TÀI LIỆU THAM KHẢO.....	66
11. PHỤ LỤC.....	67

DANH SÁCH HÌNH MINH HỌA

Hình 1: Minh họa lớp tích chập của mô hình CNN.....	4
Hình 2: Minh họa lớp max pooling thường được sử dụng nhằm giảm kích thước trong mô hình CNN	5
Hình 3: Minh họa lớp liên kết đầy đủ (Fully connected) trong mô hình CNN	6
Hình 4: Minh họa kiến trúc, tính toán trọng số của mô hình CNN truyền thống.....	8
Hình 5: Minh họa kiến trúc Depthwise Separable Convolution.....	9
Hình 6: Minh họa tổ chức dữ liệu của bộ dataset Mendely	10
Hình 7: Minh họa bộ label của dataset.....	11
Hình 8: Minh họa ứng dụng camera giao thông Việt Nam	12
Hình 9: Hình ảnh "thô" chưa qua xử lý cho bộ data dùng để kiểm thử mô hình trên giao thông Việt Nam	12
Hình 10: Lưu đồ xử lý hình ảnh đầu vào.	13
Hình 11: Hình ảnh minh họa dữ liệu hình ảnh kiểm thử sau khi đã xử lý.	13
Hình 12: Minh họa lớp tích chập 2D.	16
Hình 13: Minh họa hàm ReLU	16
Hình 14: Minh họa lớp sử dụng kiến trúc depthwise.....	17
Hình 15: Hình ảnh minh họa lớp global average pooling 2D.'	17
Hình 16: Lưu đồ xây dựng và kiểm thử mô hình trên máy tính.....	18
Hình 17: Kiến trúc mô hình sau khi xây dựng trên mô hình máy tính.	19
Hình 18: Kiến trúc thực tế của mô hình.	20
Hình 19: Kết quả huấn luyện mô hình trên máy tính.....	22
Hình 20: Kiểm thử mô hình trên ảnh của giao thông Việt Nam 1.....	23
Hình 21: Kiểm thử mô hình với ảnh giao thông Việt Nam 2.....	23
Hình 22: Sơ đồ khối camera control	25
Hình 23: Sơ đồ mô hình Mobile Net.....	31

Hình 24: Kiến trúc module conv	35
Hình 25: Minh họa thêm module chức năng padding cho module convolution	37
Hình 26: Minh họa khối control convolution giao tiếp với khối convolution	39
Hình 27: Kiến trúc khối thực hiện chức năng tính average.....	45
Hình 28: Kiến trúc phần tính average bao gồm luôn phần fullyconnected và drive led.....	46
Hình 29: Kiến trúc khối chức năng write back.....	47
Hình 30: Minh họa giao tiếp module master control và các module control tiến trình chức năng thành phần.....	49
Hình 31: Ảnh xạ địa chỉ feature 1.....	52
Hình 32: Ảnh xạ địa chỉ feature 2.....	53
Hình 33: Chạy testbench cho khối camera read	54
Hình 34: Testbench cho module control write to FIFO case 1.....	55
Hình 35: Testbench cho module control write to FIFO case2.....	55
Hình 36: Testbench cho module control write to FIFO case 3.....	56
Hình 37: Testbench cho module fetching memory.....	57
Hình 38: Hình ảnh thực tế Kit DE0-nano	59
Hình 39: Kết quả synthesis module RAM	61
Hình 40: Kết quả synthesis khối convolution.....	62
Hình 41: Kết quả synthesis khối aveage register.....	62
Hình 42: Kết quả systhesis module weight generate	63
Hình 43: Kết quả synthesis module master control	64

DANH SÁCH BẢNG SỐ LIỆU

Bảng 1: Bảng chuyển trạng thái khối cameraRead.....	26
Bảng 2: Bảng chuyển trạng thái khối control write to FIFO.....	27
Bảng 3: Bảng ngõ ra máy trạng thái khối control write to FIFO.....	27
Bảng 4: Bảng mô tả danh sách ngõ vào khối control read cam to SDRAM.....	27
Bảng 5: Bảng mô tả danh sách ngõ ra khối control read cam to SDRAM	28
Bảng 6: Bảng chuyển trạng thái khối control read cam to sdram.....	28
Bảng 7: Bảng ngõ ra máy trạng thái khối control read camera to SDRAM	29
Bảng 8: Danh sách input port của module memory fetching	31
Bảng 9: Danh sách output port của module fetching memory	32
Bảng 10: Bảng chuyển trạng thái module memory fetching.....	32
Bảng 11: Danh sách input output port của khối điều khiển RAM.....	34
Bảng 12: Danh sách input/output module convCore	36
Bảng 13: Danh sách input output module padding control.....	37
Bảng 14: Danh sách input output port module convolution control.....	39
Bảng 15: Bảng chuyển trạng thái khối convolution control.....	40
Bảng 16: Bảng output của máy trạng thái module convolution control	40
Bảng 17: Danh sách input output của module average.....	41
Bảng 18: Danh sách input output của module averageControl	42
Bảng 19: Bảng chuyển trạng thái module average control	42
Bảng 20: Danh sách input, output của module write back control.....	47
Bảng 21: Bảng chuyển trạng thái module write back control	48
Bảng 22: Danh sách input output của module master control.....	49
Bảng 23: Bảng chuyển trạng thái master control.....	50
Bảng 24: Thông số tài nguyên của Altera Cyclone IV EP4CE22F17C6N.....	59
Bảng 25: Thông số kỹ thuật SDRAM trên kit DE0-Nano	60

1. GIỚI THIỆU

1.1 Tổng quan

Trong những năm gần đây, với sự bùng nổ của công nghệ trí tuệ nhân tạo và học sâu (deep learning), việc ứng dụng các mô hình mạng nơ-ron tích chập (CNN) vào các hệ thống nhúng đã mở ra nhiều hướng nghiên cứu mới, đặc biệt là trong lĩnh vực xử lý ảnh và nhận dạng đối tượng. MobileNet, với thiết kế nhẹ và tối ưu cho các thiết bị có tài nguyên hạn chế, là một trong những mô hình CNN phổ biến, cho phép triển khai trên các nền tảng nhúng như FPGA.

FPGA (Field Programmable Gate Array) mang lại những ưu điểm vượt trội như khả năng xử lý song song, tốc độ thực thi cao và tiêu thụ năng lượng thấp, điều này rất phù hợp cho các ứng dụng thời gian thực như nhận diện tình trạng giao thông và phát hiện kẹt xe tại các giao lộ. Tuy nhiên, việc chuyển đổi một mô hình được huấn luyện trên môi trường phần mềm sang một nền tảng phần cứng như FPGA đòi hỏi phải tiến hành các bước tối ưu hóa như giảm độ chính xác (quantization), tối ưu hóa cấu trúc mạch số và đảm bảo tính ổn định của hệ thống trong điều kiện hoạt động thực tế.

Trong bối cảnh đô thị hiện nay, vấn đề kẹt xe không chỉ gây ảnh hưởng đến hiệu quả giao thông mà còn tác động đến môi trường và chất lượng cuộc sống. Việc phát triển một hệ thống nhận diện giao thông dựa trên FPGA với mô hình MobileNet không chỉ giúp cải thiện hiệu quả giám sát và quản lý giao thông mà còn mở ra hướng nghiên cứu ứng dụng trí tuệ nhân tạo vào các bài toán nhúng thời gian thực.

1.2 Tình hình nghiên cứu trong và ngoài nước

Hiện tại có dự án đã thành công thực hiện mô hình mobilenet trên FPGA. Trong đó nổi bật là: Nhận diện chữ viết tay trên FPGA sử dụng camera OV7670, Nhúng mobile net vào kit DE0-Nano để thực hiện tính toán. Dùng màn hình TFT để hiển thị kết quả. Hệ thống hoạt động theo nguyên tắc xây dựng hệ khối xử lý tính toán bên trong FPGA thực hiện tính toán trực tiếp trên bộ nhớ RAM được thiết kế trực tiếp trên FPGA vì mô hình nhỏ với dữ liệu đầu vào 28x28x1. Dự án này đã thành công nhận diện chữ

viết bằng mô hình mobile net thực hiện trên FPGA. Nguyên tắc cơ bản của dự án: Thực hiện mô hình bằng phần mềm máy tính, Huấn luyện mô hình trên máy tính để được bộ trọng số. Sử dụng bộ trọng số này cho bộ thực hiện mobile net trên FPGA để hệ thống hoạt động. Vì vậy hệ thống trên FPGA sẽ không có khả năng học nữa. Mà sử dụng trọng số có sẵn từ mô hình trên máy tính và thực hiện tính toán để cho ra kết quả. Điều này sẽ phù hợp bởi vì đối với một hệ thống nhúng có tài nguyên hạn chế, để thực hiện khả năng thực hiện việc học “training” sẽ làm hệ thống trở nên phức tạp hơn rất nhiều và tốn nhiều tài nguyên phần cứng hơn. Và đa phần các ứng dụng sẽ không yêu cầu hệ thống học thêm nữa. Mà chỉ cần dựa vào dữ liệu đầu vào, thực hiện tính vào và đưa ra kết quả ở đầu ra từ bộ trọng số đã có sẵn từ việc “học” (training) trên máy tính – nơi có tài nguyên về bộ nhớ dồi dào hơn và lập trình dễ hơn vì được điều khiển bằng phần mềm.

1.3 Nhiệm vụ luận văn

Nội dung 1: Tìm hiểu lý thuyết về mô hình học máy trên máy tính, đặc biệt là mô hình mobile net (Mô hình tối ưu sử dụng trên hệ thống có tài nguyên hạn chế).

Đã có rất nhiều bài nghiên cứu về mô hình máy học để nhận diện hoặc phân loại hình ảnh. Ví dụ như mô hình CNN, tuy nhiên mô hình CNN thuần túy đòi hỏi khá lớn về tài nguyên tính toán, tài nguyên bộ nhớ cần sử dụng, Vì vậy ta sẽ tập trung chủ yếu vào mô hình Mobile net.

Nội dung 2: Chuẩn bị dataset, xây dựng mô hình trên máy tính, và tiến hành training mô hình để được bộ trọng số.

Bởi vì nhiệm vụ của mô hình là phân loại tình trạng giao thông trong một bức ảnh có bị tắc nghẽn hay không. Vì vậy nguồn dữ liệu ta có thể cân nhắc: Nguồn dữ liệu có sẵn trên internet, open camera giao thông được chiếu trực tiếp trên các nền tảng web hoặc ứng dụng. Xây dựng mô hình mobile net trên máy tính ta sẽ sử dụng thư viện tensorflow bằng ngôn ngữ python nhằm nhanh chóng và dễ dàng xây dựng mô hình hay training mô hình.

Nội dung 3: Nghiên cứu phần cứng FPGA được sử dụng – Kit DE0-NANO.

Tài liệu tham khảo chính dùng để hiểu và sử dụng kit DE0-nano sẽ là DE0-Nano user manual. Tập trung chủ yếu vào việc tìm hiểu các ngoại vi nằm trên kit DE0-Nano,

tài nguyên trên Kit (SD-RAM, Led, button, ...) Tài nguyên trên FPGA (LUT, memory block, DSP block,...)

Nội dung 4: Nghiệm cứu thực thi mô hình mobile net trên Kit đã lựa chọn.

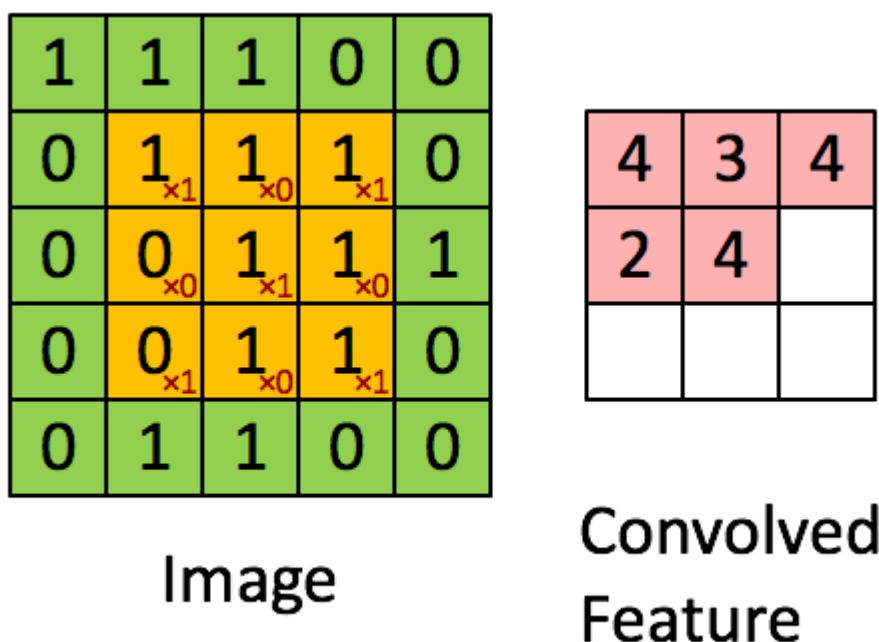
Dựa vào thuật toán của mô hình mobile net trên máy tính (computer vision). Cố gắng thực hiện lại thuật toán đó trên FPGA thông qua sự phối hợp hoạt động của các khối chức năng. Sẽ được trình bày cụ thể trong phần 3. Thiết kế và thực hiện phần cứng.

2. LÝ THUYẾT VỀ MÔ HÌNH PHÂN LOẠI ẢNH.

2.1 Tìm hiểu về mô hình CNN

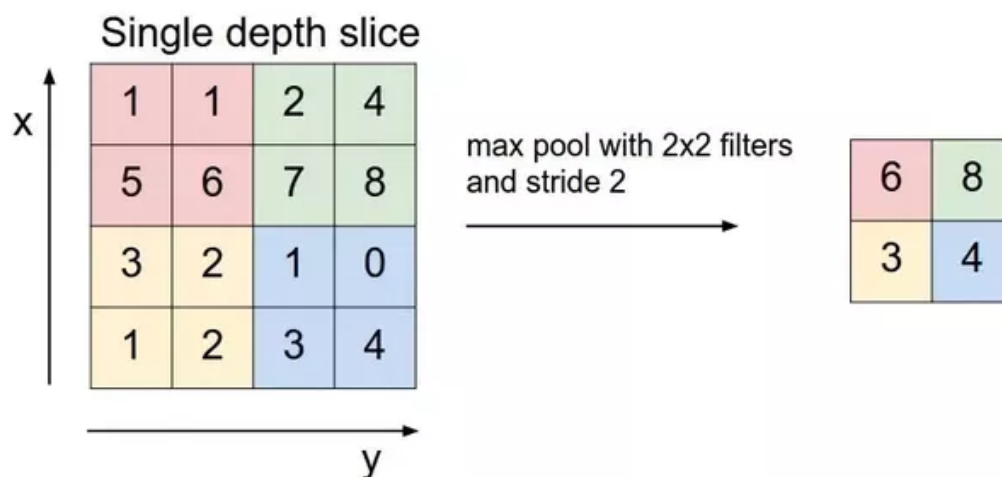
Mạng nơ-ron tích chập (CNN) là một trong những kiến trúc học sâu được sử dụng phổ biến trong xử lý ảnh và nhận dạng đối tượng. Những đặc điểm cơ bản của CNN bao gồm:

Tầng tích chập (Convolutional Layers): Đây là tầng cốt lõi của CNN, nơi mà các phép tích chập được thực hiện giữa bộ lọc (filter) hay các parameter (kernel) và một vùng nhỏ của ảnh đầu vào có kích thước thường tương ứng với kích thước của filter (kernel). Mỗi bộ lọc có khả năng trích xuất các đặc trưng (features) như cạnh, góc cạnh, kết cấu từ ảnh,... Qua đó, các đặc trưng này được học một cách tự động qua quá trình huấn luyện. Trong quá trình huấn luyện trọng số của bộ lọc (filter) hay kernel sẽ được điều chỉnh nhằm thu được kết quả tốt nhất. Sau khi thực hiện xong trên một vùng ảnh, bộ lọc (filter) hay kernel này được dịch và di chuyển để thực hiện tích chập tiếp theo trên một vùng ảnh khác cho đến khi đã thực hiện trên toàn bộ bức ảnh hay dữ liệu đầu vào.



Hình 1: Minh họa lớp tích chập của mô hình CNN

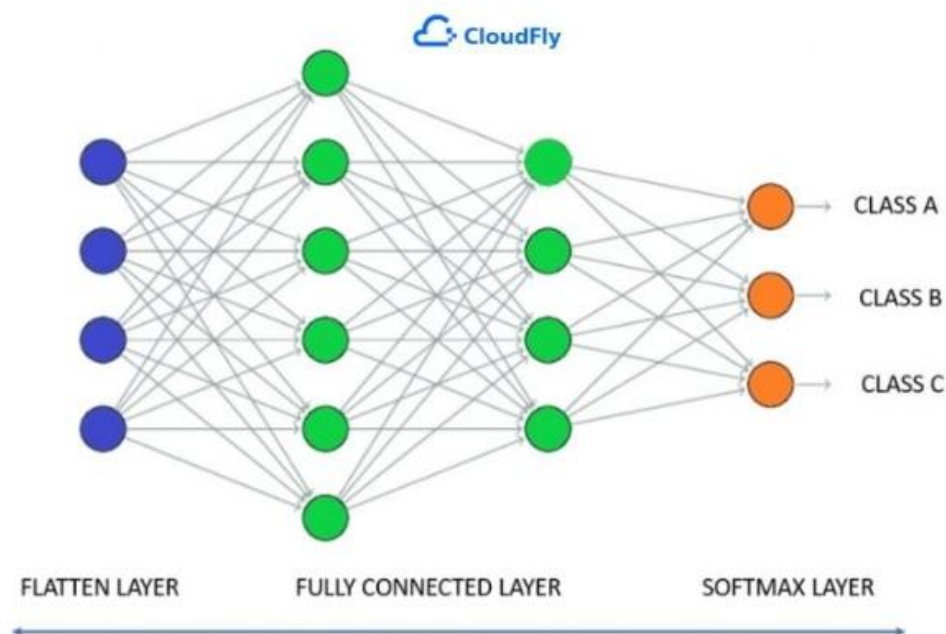
Tầng pooling (Pooling Layers): Lớp pooling sẽ giảm bớt số lượng tham số khi hình ảnh quá lớn. Không gian pooling còn được gọi là lấy mẫu con hoặc lấy mẫu xuống làm giảm kích thước của mỗi map nhưng vẫn giữ lại thông tin quan trọng. Các pooling có thể có nhiều loại khác nhau: Max Pooling, average pooling, sum pooling.¹



Hình 2: Minh họa lớp max pooling thường được sử dụng để giảm kích thước trong mô hình CNN

Tầng kết nối đầy đủ (Fully Connected Layers): Sau các lớp tích chập và pooling, các đặc trưng được chuyển thành dạng vector và đưa vào tầng kết nối đầy đủ. Tại đây, mạng sẽ thực hiện các phép biến đổi tuyến tính và phi tuyến để đưa ra kết quả cuối cùng (phân loại, nhận dạng đối tượng, ...).

¹ VBLO, [Deep Learning] Tìm hiểu về mạng tích chập (CNN), (10/12/2020), truy cập tại <https://viblo.asia/p/deep-learning-tim-hieu-ve-mang-tich-chap-cnn-maGK73bOKj2>



Hình 3: Minh họa lớp liên kết đầy đủ (Fully connected) trong mô hình CNN

2

Tính chất trích xuất đặc trưng tự động: CNN được thiết kế để tự động học các đặc trưng từ dữ liệu đầu vào thông qua quá trình huấn luyện, giảm thiểu sự phụ thuộc vào các đặc trưng thủ công và cải thiện khả năng tổng quát hóa của mô hình.

2.2 Những hạn chế của mô hình CNN

Mặc dù các mô hình CNN truyền thống đã đạt được thành công đáng kể trong các bài toán nhận dạng ảnh, nhưng chúng lại gặp một số hạn chế khi triển khai trên các thiết bị có tài nguyên hạn chế:

Số lượng tham số lớn: Tuy so với mô hình neuron network truyền thống nhất thì mô hình CNN sử dụng các filter đã giảm được rất nhiều các tham số, tuy nhiên khi ảnh đầu vào là 3 chiều, các đặc trưng được trích xuất nhiều hơn thì các mô hình CNN truyền thống này vẫn có số lượng tham số khổng lồ, dẫn đến yêu cầu bộ nhớ và tính toán cao. Vì vậy mô hình CNN truyền thống có yêu cầu rất lớn về tài nguyên bộ nhớ (Lưu trữ trọng số của các filter, trọng số của lớp fully

² Cloufy, Thuật Toán CNN Là Gì? Tìm Hiểu Tất Tần Tật Về CNN, (18/9/2024), truy cập tại <https://cloudfly.vn/docs/tin-cong-nghe/thuot-toan-cnn-la-gi-tim-hieu-tat-tan-tat-ve-cnn>

connected, chứa dữ liệu hình ảnh, chứa kết quả của các lớp – các đặc trưng được trích xuất qua các filter)

Tốc độ xử lý chậm: Vì có nhiều trọng số vì vậy mô hình CNN truyền thống cũng có số lượng phép tính lớn, việc triển khai trên các thiết bị nhúng (embedded systems) hoặc FPGA gặp khó khăn do hạn chế về tài nguyên tính toán và độ phức tạp của hệ thống khi thiết kế.

Tiêu thụ năng lượng cao: Với yêu cầu tài nguyên bộ nhớ và tài nguyên tính toán lớn, thực hiện nhiều phép toán cũng đồng nghĩa với các mô hình CNN truyền thống sẽ thường tiêu thụ nhiều năng lượng, không phù hợp với các ứng dụng thời gian thực cần tiết kiệm điện năng.

2.3 Giới thiệu mô hình Mobile Net

Các khái niệm cơ bản về CNN đã cung cấp nền tảng vững chắc cho việc phát triển các kiến trúc tiên tiến như MobileNet. Các yếu tố quan trọng trong việc thiết kế MobileNet có thể được xem là sự tinh chỉnh và tối ưu hóa dựa trên các nguyên tắc cơ bản của mô hình CNN:

Từ việc trích xuất đặc trưng thông qua các tầng tích chập truyền thống, MobileNet tối ưu bằng cách áp dụng depthwise separable convolutions nhằm giảm chi phí tính toán.

Các tầng pooling và fully connected vẫn giữ vai trò quan trọng nhưng được thiết kế lại để phù hợp với khả năng xử lý song song và giới hạn về tài nguyên của các thiết bị nhúng.

MobileNet không chỉ giảm thiểu số lượng tham số mà còn tối ưu hóa kiến trúc để đảm bảo rằng hệ thống có thể được thực hiện trên FPGA với độ phức tạp của hệ thống có thể chấp nhận được.

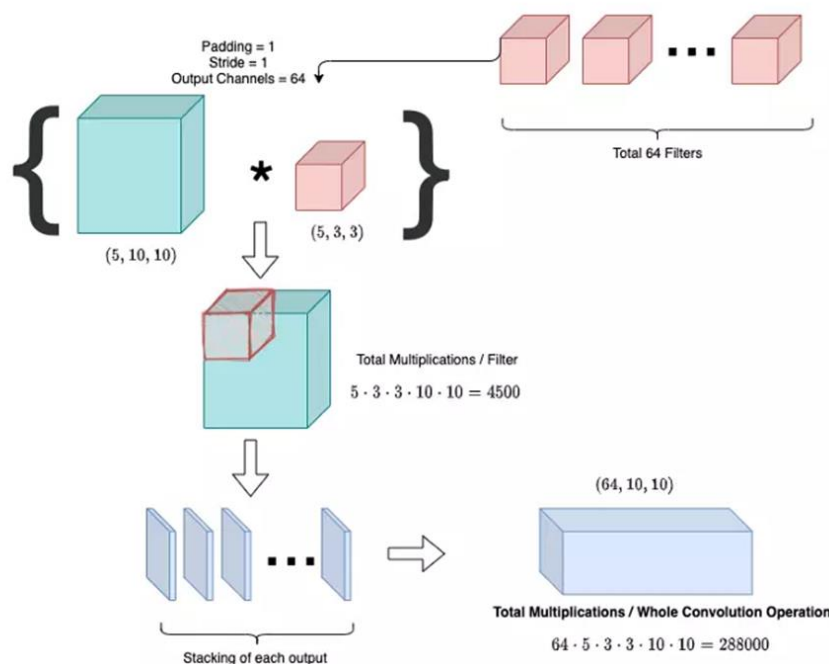
Kiến trúc Depthwise Separable Convolution: MobileNet thay thế các phép tích chập chuẩn bằng phép tích chập phân tách theo chiều sâu (depthwise separable convolution). Quá trình này được chia thành hai bước:

Depthwise Convolution: Thực hiện phép tích chập riêng lẻ cho mỗi kênh của ảnh đầu vào, giúp trích xuất đặc trưng một cách riêng biệt.

Pointwise Convolution: Sau đó, kết hợp các kênh thông qua một phép tích chập 1×1 để tạo ra đầu ra mong muốn.

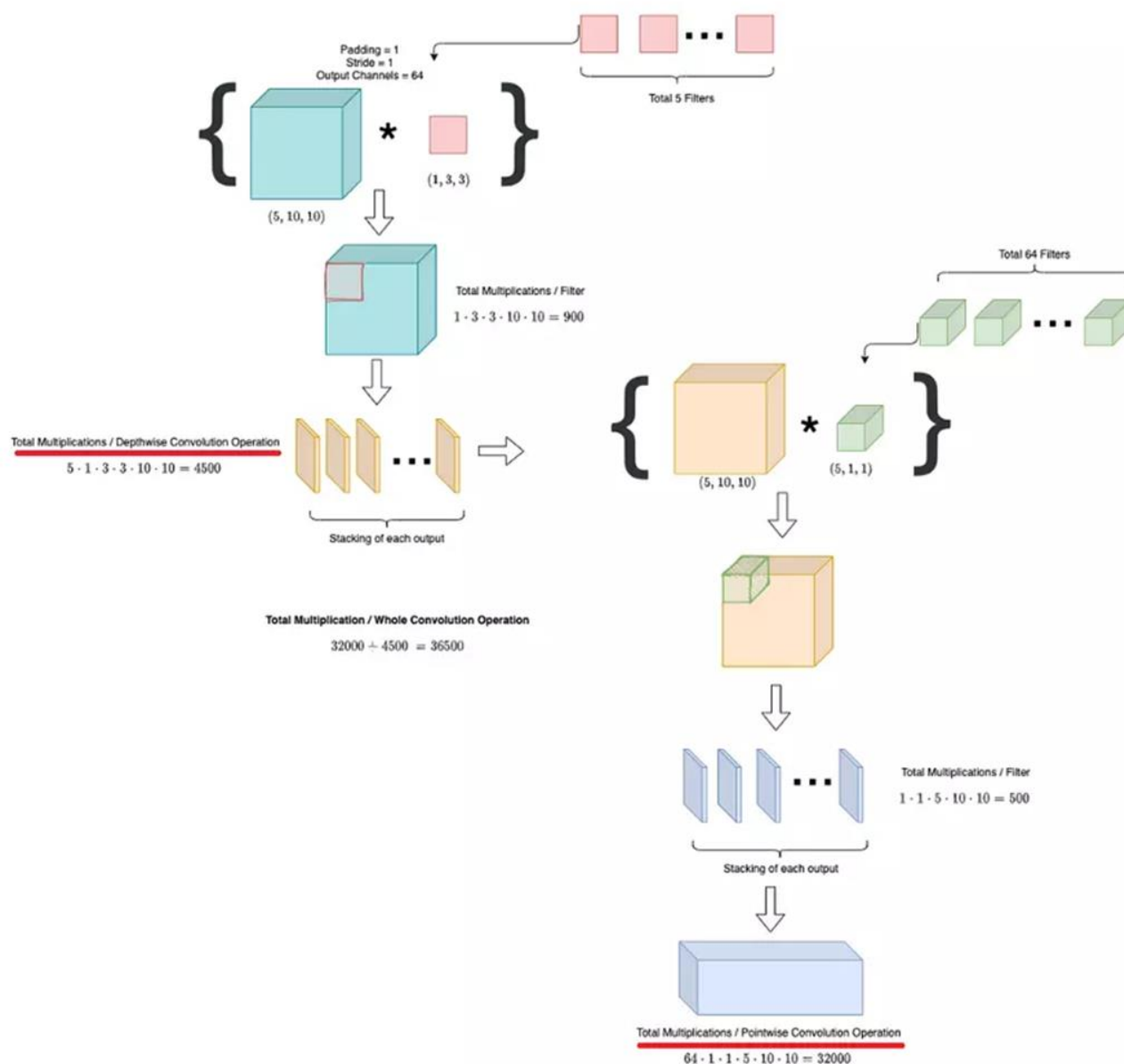
Việc tách nhỏ phép tính này giúp giảm số lượng phép toán và tham số xuống đáng kể mà vẫn duy trì được chất lượng trích xuất đặc trưng.

Để hiểu rõ hơn về kiến trúc Depthwise Separable Convolution của mô hình mobile net, Ta xem xét kiến trúc mô hình ví dụ sau:



Hình 4: Minh họa kiến trúc, tính toán trọng số của mô hình CNN truyền thống

Đây là ví dụ về kiến trúc của lớp tích chập trong mô hình CNN truyền thống. Input có kích thước là $10 \times 10 \times 5$ và output có kích thước là $10 \times 10 \times 64$. Như tính toán đã được trình bày trong ảnh. Ta cần tổng cộng 64 filter kích thước $3 \times 3 \times 5$. Như vậy tổng số trọng số là: $64 \times 3 \times 3 \times 5 = 2880$ tham số. Tổng số phép tính cần thực hiện là: $64 \times 3 \times 3 \times 5 \times 10 \times 10 = 288000$ phép tính. Chưa nói đến các phép cộng liên quan.



Hình 5: Minh họa kiến trúc Depthwise Separable Convolution

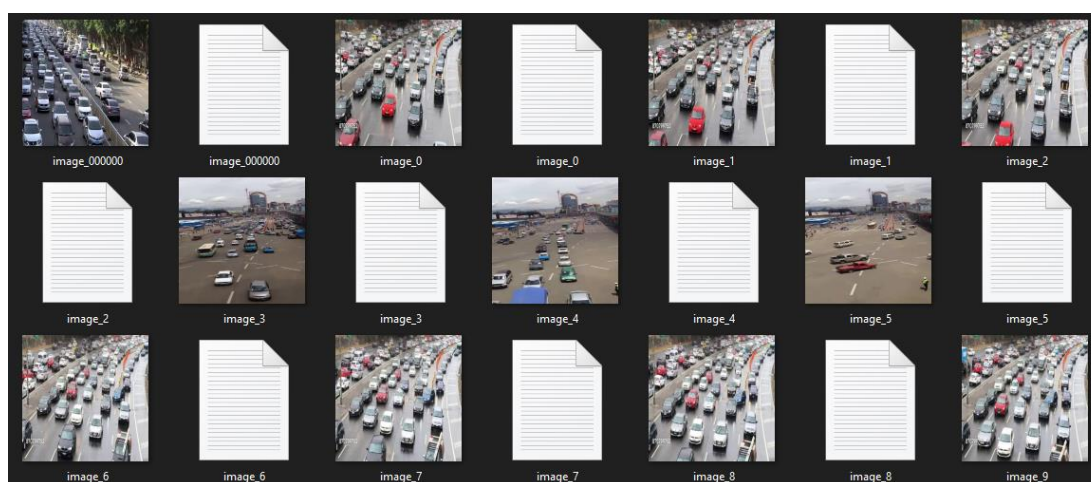
Đây là ví dụ về kiến trúc Depthwise Seprable Convolution được sử dụng trong mô hình mobile net, tương tự như ví dụ trong convolution truyền thống được sử dụng trong mô hình CNN ta có input có kích thước là $10 \times 10 \times 5$ và output mô hình là $64 \times 10 \times 10$. Tổng số tham số của kiến trúc là: $5 \times 3 \times 3 + 64 \times 5 = 365$ (tham số). Tổng số phép tính: $5 \times 3 \times 3 \times 10 \times 10 + 64 \times 5 \times 10 \times 10 = 36500$ (phép nhân)

3. CHUẨN BỊ DỮ LIỆU

3.1 Bộ dữ liệu training và testing

Bởi vì nhiệm vụ của đề tài là phân loại tình hình giao thông trong một bức ảnh có hiện tượng tắc nghẽn hay không? Vì vậy ta cần có nguồn dữ liệu đáp ứng nhu cầu: đầu vào định dạng hình ảnh (nên là các format ảnh phổ biến như .jpg, .png,... để dễ dàng cho việc xử lý) Phải có nhãn đánh dấu ảnh là “Kẹt xe” hay “Không có kẹt xe” hoặc bộ dữ liệu được phân loại sẵn thành 2 thư mục riêng biệt.

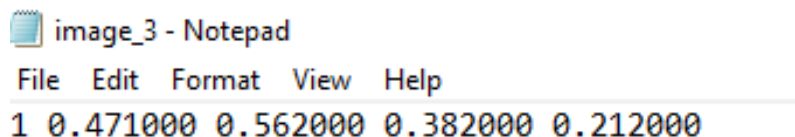
Ở đây ta sử dụng bộ dataset của Mendely data³. “Mục đích chính của tập dữ liệu này là cho phép phát hiện tình trạng tắc nghẽn giao thông từ camera giám sát bằng cách sử dụng các bộ phát hiện vật thể một giai đoạn. Tập dữ liệu chứa các cảnh giao thông tắc nghẽn và không tắc nghẽn với các nhãn tương ứng của chúng. Tập dữ liệu này được thu thập từ các cảnh quay video của các camera giám sát khác nhau. Để chuẩn bị cho tập dữ liệu, các khung hình được trích xuất từ các nguồn video và được thay đổi kích thước thành kích thước 500 x 500 với định dạng hình ảnh .jpg. Để chú thích, phần mềm LabelImg đã sử dụng hình ảnh. Định dạng của nhãn là .txt có cùng tên với hình ảnh. Tập dữ liệu chủ yếu được chuẩn bị cho các Mô hình YOLO nhưng có thể được chuyển đổi sang định dạng mô hình khác.”⁴



Hình 6: Minh họa tổ chức dữ liệu của bộ dataset Mendely

³ Mendelely Data, Traffic congestion Dataset, (03/9/2020), truy cập tại <https://data.mendeley.com/datasets/wtp4ssmwsd/1>

Như vậy bộ dữ liệu được tổ chức dạng riêng lẻ thành những tấm ảnh, định dạng .jpg, Mỗi ảnh sẽ đi kèm với một tập tin .txt có cùng tên với ảnh chứa thông tin về lable của ảnh.



Hình 7: Minh họa bộ lable của dataset

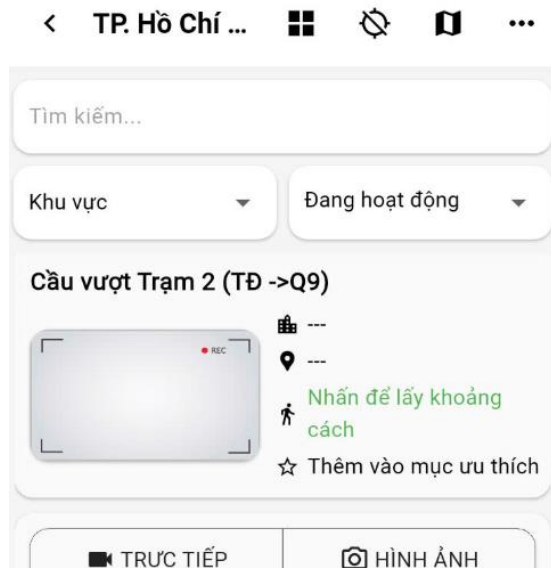
Mặc dù bộ dữ liệu được thiết kế dành cho mô hình Yolo. Tuy nhiên trong phần lable của một ảnh ta có thông số đầu tiên 1 hoặc 0, biểu thị cho việc có xuất hiện trình trạng kẹt xe trong bức ảnh hay không. Như vậy ta có thể dùng bộ dataset này để training mô hình.

3.2 Thêm dữ liệu cho testing phù hợp với giao thông Việt Nam

Trong trường hợp muốn có độ chính xác cao nhất, phù hợp nhất cho tính chất giao thông ở Việt Nam thì ta nên xây dựng một bộ dataset cho giao thông Việt Nam. Tuy nhiên vì hạn chế về thời gian thực hiện đồ án nên ta chỉ có thể thu thập một số hình ảnh giao thông ở Việt Nam để kiểm thử mô hình có hoạt động được và ổn định với giao thông Việt Nam không ?

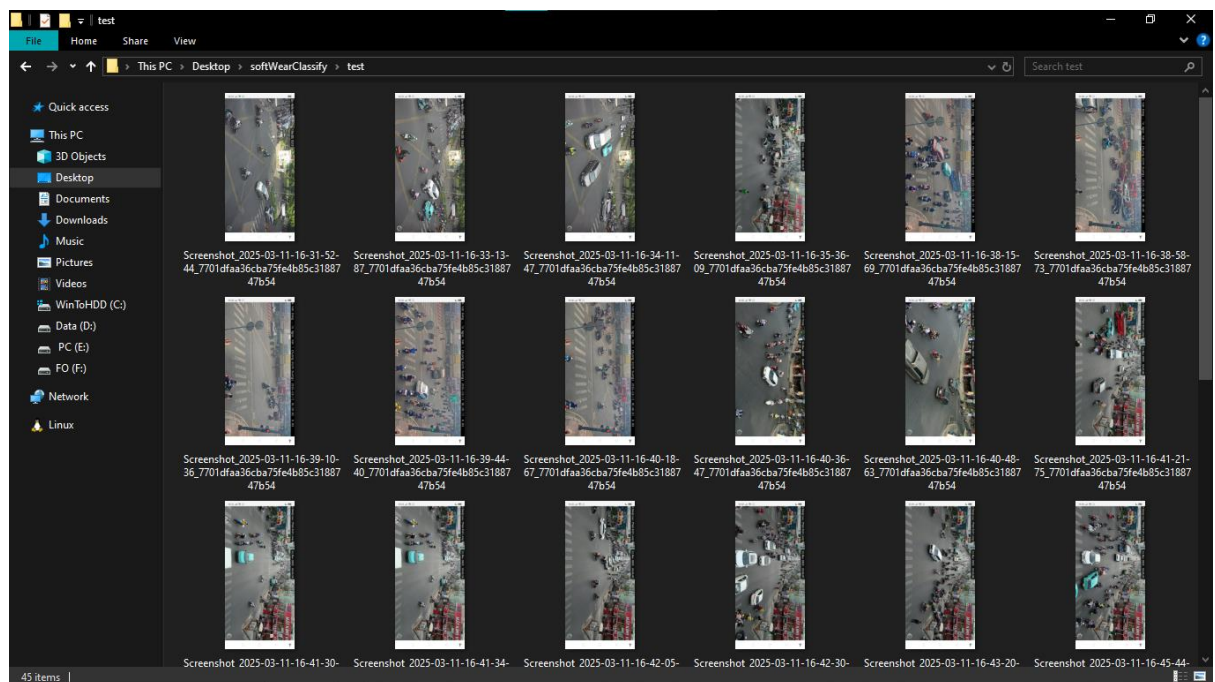
Ta có sử dụng ứng dụng xem trực tiếp giao thông tại Việt Nam để lấy dữ liệu hình ảnh. Ứng dụng “Camera Giao Thông Việt Nam”⁵

⁵ Google play, Camera Giao Thông Việt Nam, (12/03/2024), truy cập tại https://play.google.com/store/apps/details?id=com.freeapp.camtraffic.traffic_cam&hl=vi&pli=1



Hình 8: Minh họa ứng dụng camera giao thông Việt Nam

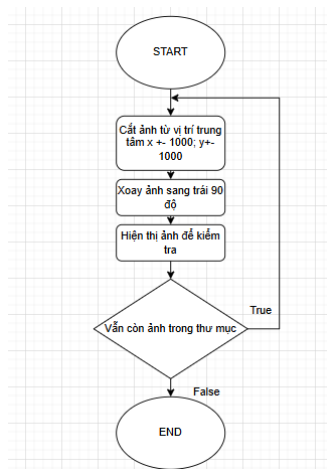
Chúng ta sẽ sử dụng hình ảnh từ camera để tạo hình ảnh để kiểm thử. Tuy nhiên hình ảnh lấy được từ ứng dụng sẽ có tỉ lệ và kích thước không phù hợp với mô hình.



Hình 9: Hình ảnh "thô" chưa qua xử lý cho bộ data dùng để kiểm thử mô hình trên giao thông Việt Nam

Vì vậy ta cần xử lý hình ảnh về hình ảnh phù hợp hơn cho mô hình. Các tính chất ảnh ta cần đạt được để phù hợp với mô hình: Lật ảnh sang trái 90 độ, biến ảnh thành kích thước 500px x 500px hoặc ít nhất biến ảnh thành tỉ lệ 1:1 và dùng hàm resize của

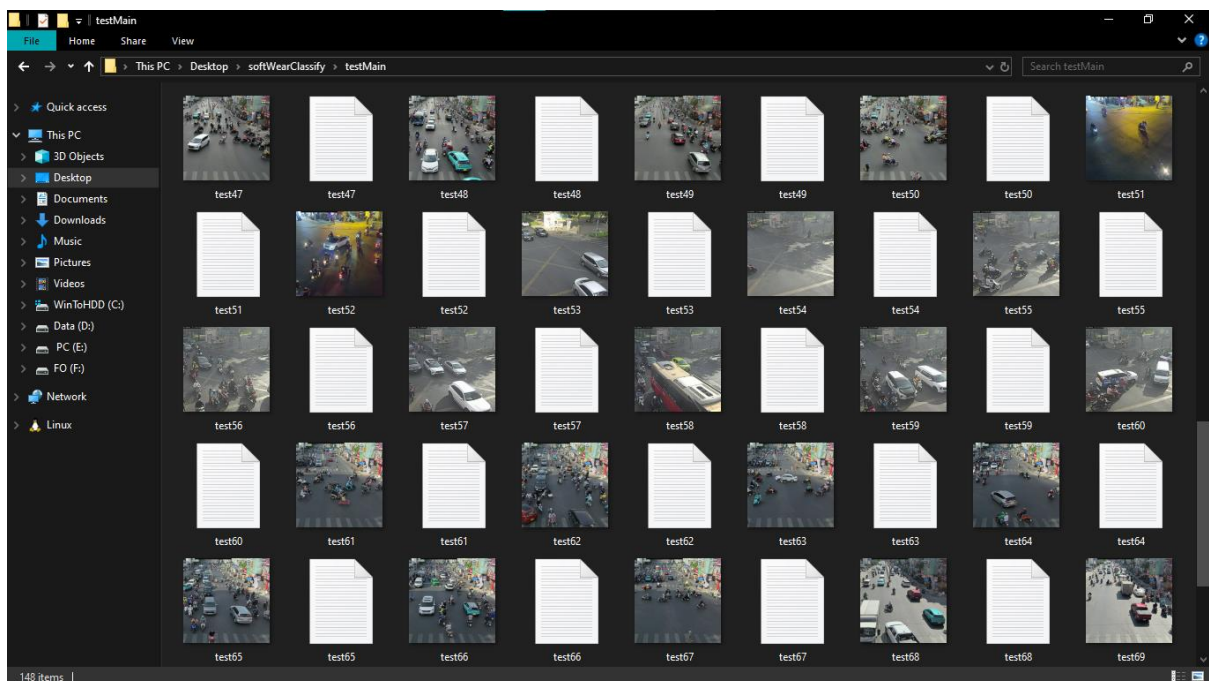
openCV để chuyển ảnh về kích thước 500x500px, Để thực hiện được các biến đổi trên ta sử dụng thư viện openCV và script python



Hình 10: Lưu đồ xử lý hình ảnh đầu vào.

Script python chi tiết tại phần phụ lục.

Sau khi thực hiện giải thuật để xử lý hình ảnh đầu vào. Ta sẽ được bộ data kiểm thử như sau.



Hình 11: Hình ảnh minh họa dữ liệu hình ảnh kiểm thử sau khi đã xử lý.

Như ta thấy ngoài việc chuyển ảnh về tỉ lệ, góc độ phù hợp với mô hình thì script python còn tạo ra file .txt nhằm chứa label của hình ảnh. Điều này tuy không trực tiếp

sử dụng trong đồ án này tuy nhiên có thể là tiền đề (một công cụ hỗ trợ) để thành lập bộ data cho riêng giao thông Việt Nam nhằm nâng cao độ chính xác của mô hình khi dữ liệu đầu vào là hình ảnh giao thông Việt Nam

4. THIẾT KẾ VÀ XÂY DỰNG MÔ HÌNH TRÊN MÁY TÍNH.

Kiến trúc TensorFlow hoạt động được chia thành 3 phần:

- Tiền xử lý dữ liệu
- Dựng model
- Train và ước tính model⁶

Trong đó tiền xử lý dữ liệu: Thư viện TensorFlow sẽ cung cấp các hàm, công cụ để biến đổi dữ liệu, làm sạch dữ liệu (scale, resize,..)

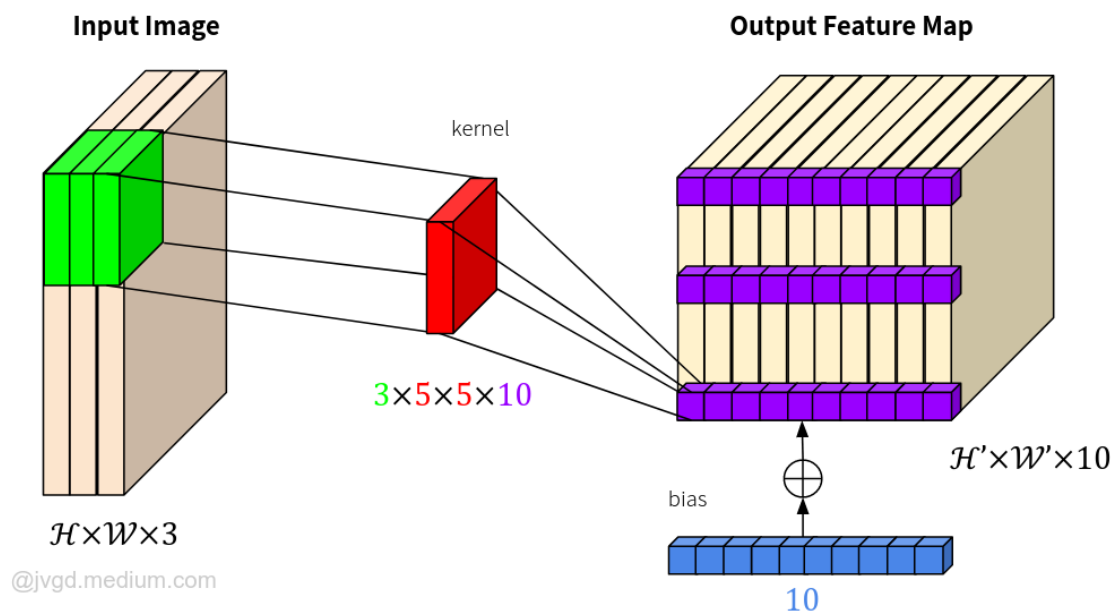
Dựng model: Thư viện TensorFlow cung cấp các câu lệnh để ta xây dựng mà ta có thể tùy chỉnh được các node, số lượng node có trong một lớp, số lượng lớp có trong mô hình. Thư viện TensorFlow cung cấp nhiều kiến trúc của các lớp khác nhau như: fully connected (Lớp liên kết đầy đủ), lớp Flatten (Lớp trải phẳng) – thường dùng khi chuyển đổi từ dạng 2D (một bức ảnh) sang 1D (dạng vector), Lớp Conv2D (Lớp tính tích chập truyền thống) – thường dùng trong mô hình CNN, Đặc biệt là lớp DepthwiseConv2D (Depthwise Seprable Convolution) – được dùng trong mô hình mobile net để tiết kiệm tài nguyên tính toán và tài nguyên bộ nhớ và cũng là lớp mà ta sẽ sử dụng chủ yếu trong mô hình mà ta sẽ thiết kế.

Các hàm cơ bản để thực hiện mô hình:

`inputs = Input(shape=(64, 64,3))` : Tạo ra lớp input cho mô hình có kích thước 64x64x3 – bức ảnh có chiều dài 64px, chiều rộng là 64px, chiều sâu – kênh màu là 3 kênh (RGB)

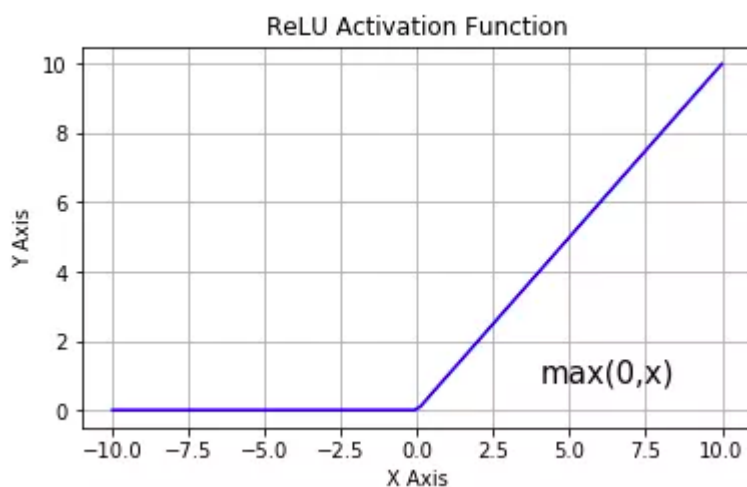
`x = Conv2D(16, kernel_size=3, strides=1, padding='same', use_bias=False)(inputs)`: Tạo ra lớp tích chập 2D nhằm trích xuất các đặc trưng của hình ảnh đầu vào. 16 là số filter được dùng đồng nghĩa với việc trích xuất 16 đặc trưng từ bức ảnh. Kernal_size là kích thước của bộ lọc 3x3 tương ứng với mỗi bộ lọc có 9 tham số, strides độ dịch của bộ lọc sau mỗi lần tính (Dịch 1 pixel sau mỗi lần tích chập), padding = 'same' giữ nguyên kích thước của đặc trưng đầu ra so với ảnh đầu vào, use_bias là hệ số bias thường dùng trong lớp fully connected, trong conv2d thường không sử dụng hệ số bias.

⁶ Top-Dev, TensorFlow là gì? Tìm hiểu về TensorFlow từ A đến Z, (23/04/2025), truy cập tại <https://topdev.vn/blog/tensorflow-la-gi/>



Hình 12: Minh họa lớp tích chập 2D.⁷

$x = \text{ReLU}(x)$ Thực hiện hàm activation. Nhằm tạo ra hàm phi tuyến cho mô hình.

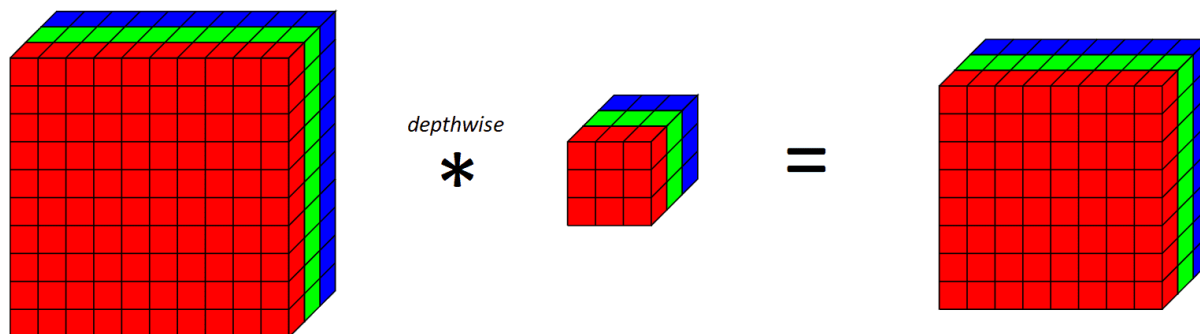


Hình 13: Minh họa hàm ReLU

$x = \text{DepthwiseConv2D}(\text{kernel_size}=3, \text{strides}=1, \text{padding}='same', \text{use_bias}=\text{False})(x)$: Tạo ra lớp tích chập với kiến trúc Depthwise. Tương tự như lớp conv2D: kernel_size kích thước của bộ lọc, strides độ dịch chuyển của bộ lọc sau mỗi lần tích chập, padding = 'same' giữ nguyên kích thước của đầu ra so với đầu vào, trong

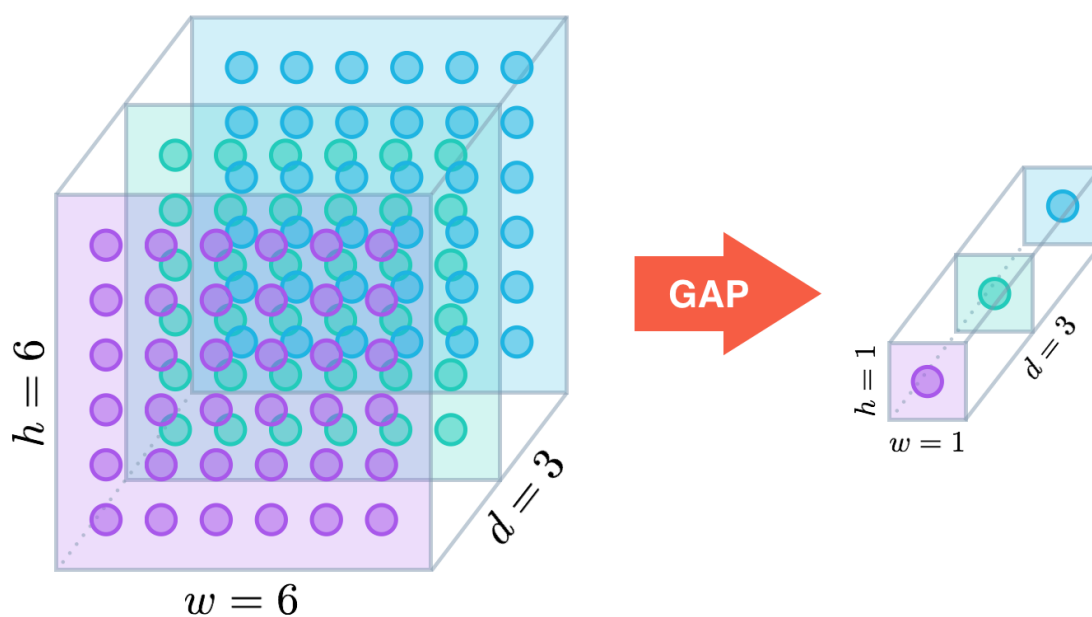
⁷ Medium, Pytorch Conv2d Weights Explained, (26/11/2021), truy cập tại <https://medium.com/data-science/pytorch-conv2d-weights-explained-ff7f68f652eb>

lớp sử dụng kiến trúc Depthwise thường cũng không sử dụng hệ số bias, hệ số bias thường sẽ thường sử dụng trong lớp fully connected.



Hình 14: Minh họa lớp sử dụng kiến trúc depthwise⁸

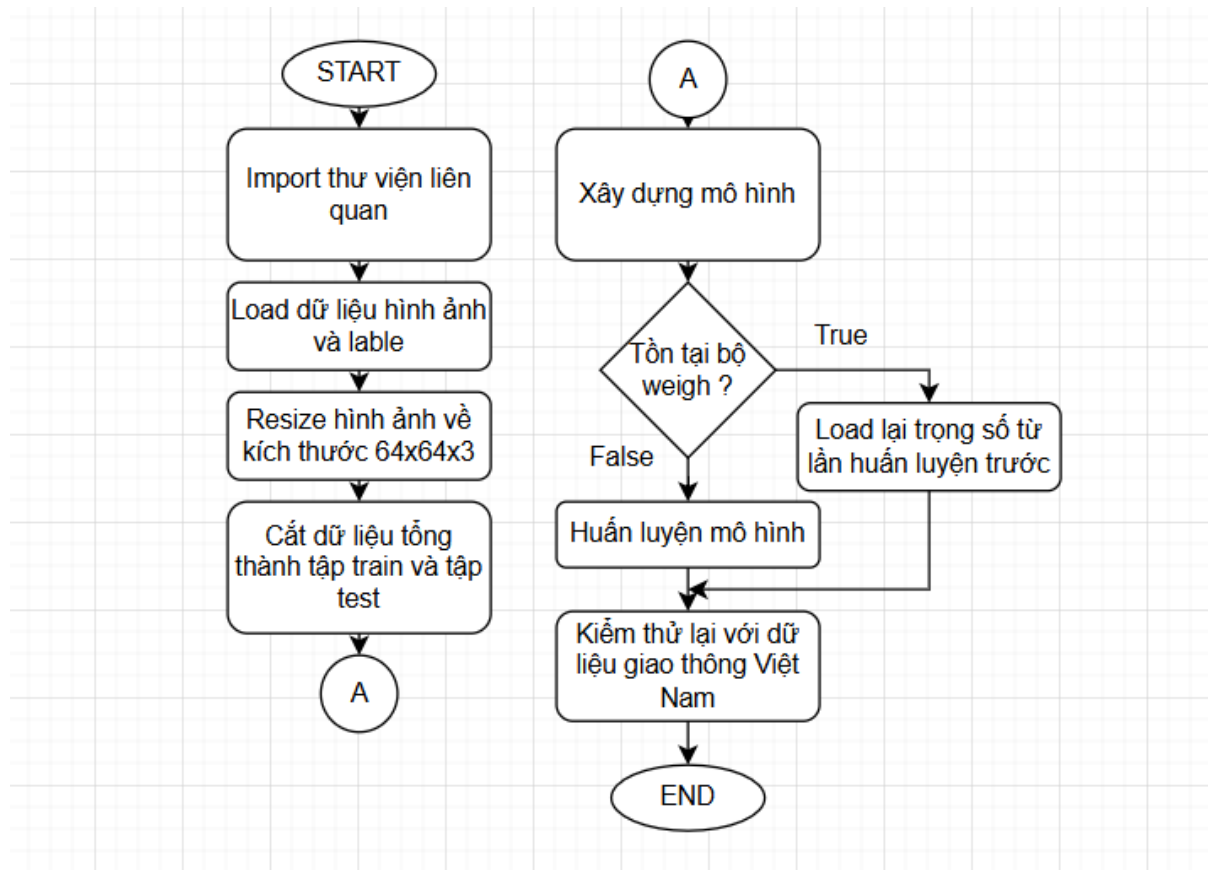
$x = \text{GlobalAveragePooling2D}()(x)$ Hàm của thư viện TensorFlow dùng để tạo ra lớp tính trung bình, nhằm giảm kích thước của dữ liệu giúp tiết kiệm tài nguyên tính toán và tài nguyên bộ nhớ.



Hình 15: Hình ảnh minh họa lớp global average pooling 2D.⁸

`model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=25, batch_size=64)` Hàm của thư viện TensorFlow được sử dụng để training mô hình từ tập X_{train} – chứa dữ liệu ảnh và tập Y_{train} – chứa label.

⁸ Machine Learning Mastery, Using Depthwise Separable Convolutions in Tensorflow, (10/08/2022), truy cập tại <https://machinelearningmastery.com/using-depthwise-separable-convolutions-in-tensorflow/>



Hình 16: Lưu đồ xây dựng và kiểm thử mô hình trên máy tính

Chi tiết script python để tạo lập mô hình:

```

# Define a simple lightweight model
inputs = Input(shape=(64, 64,3))
# Initial Conv Layer
x = Conv2D(16, kernel_size=3, strides=1, padding='same', use_bias=False)(inputs)
x = ReLU()(x)
# Depthwise Convolution Block 1
x = DepthwiseConv2D(kernel_size=3, strides=1, padding='same', use_bias=False)(x)
x = ReLU()(x)
# Depthwise Convolution Block 2
x = DepthwiseConv2D(kernel_size=3, strides=1, padding='same', use_bias=False)(x)
x = ReLU()(x)
# Depthwise Convolution Block 3
x = DepthwiseConv2D(kernel_size=3, strides=1, padding='same', use_bias=False)(x)
x = ReLU()(x)
# Global Average Pooling and Output
x = GlobalAveragePooling2D()(x)
  
```

```

outputs = Dense(1, activation='sigmoid')(x) # Binary output: Traffic Jam (1) or Not (0)
# Build model
model = Model(inputs, outputs)

```

Kiến trúc của mô hình sau khi xây dựng như sau:

Model: "functional"

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 64, 64, 3)	0
conv2d (Conv2D)	(None, 64, 64, 16)	432
re_lu (ReLU)	(None, 64, 64, 16)	0
depthwise_conv2d (DepthwiseConv2D)	(None, 64, 64, 16)	144
re_lu_1 (ReLU)	(None, 64, 64, 16)	0
depthwise_conv2d_1 (DepthwiseConv2D)	(None, 64, 64, 16)	144
re_lu_2 (ReLU)	(None, 64, 64, 16)	0
depthwise_conv2d_2 (DepthwiseConv2D)	(None, 64, 64, 16)	144
re_lu_3 (ReLU)	(None, 64, 64, 16)	0
global_average_pooling2d (GlobalAveragePooling2D)	(None, 16)	0
dense (Dense)	(None, 1)	17

Hình 17: Kiến trúc mô hình sau khi xây dựng trên mô hình máy tính.

Như vậy mô hình có kiến trúc 11 lớp: trong đó bao gồm 1 lớp input, 1 lớp output, 4 lớp activation ReLU.

Lớp 1: Input layer – Lớp này chịu trách nhiệm cho phần input của mô hình có kích thước 64x64x3 cũng chính là kích thước ảnh đầu vào của mô hình.

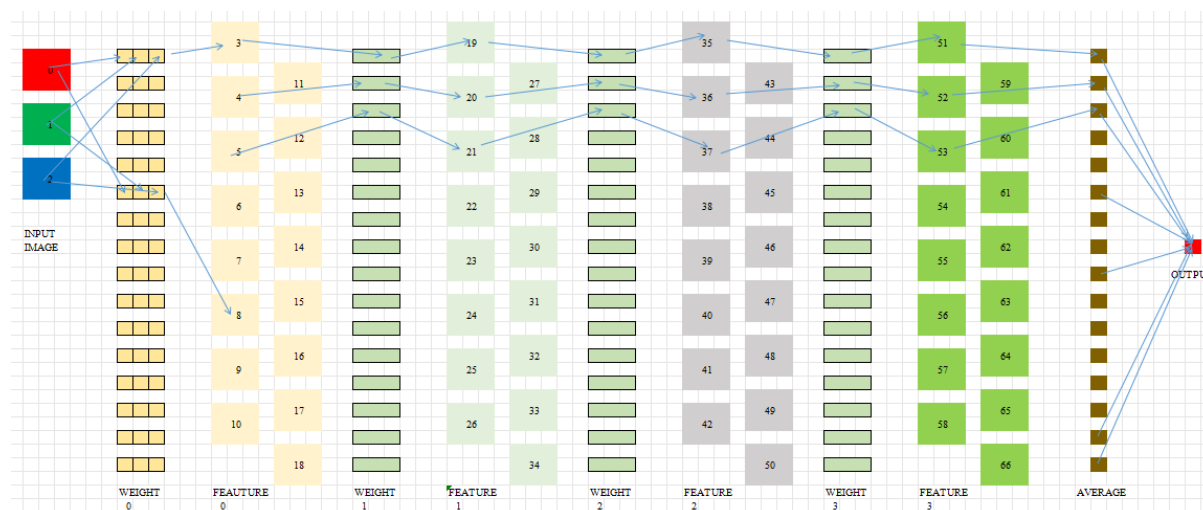
Lớp 2: Conv2D – Lớp này chịu trách nhiệm trích xuất các đặc trưng của hình ảnh đầu vào, vì ảnh đầu vào có chiều sâu là 3 kênh ảnh vì vậy lớp Conv2D sẽ phù hợp hơn tại vị trí này.

Lớp 3: ReLU – Lớp này đóng vai trò là lớp activation function, nhằm tạo ra tính phi tuyến cho mô hình. Sở dĩ việc chọn activation function là ReLU vì đây là hàm rất đơn giản $\text{Max}(y,0)$ nên sẽ dễ thiết kế và thực hiện hơn khi thực thi trên FPGA.

Lớp 4: Depthwise_conv2d – Lớp này đóng vai trò hidden layer, lớp theo kiến trúc Depthwise Seprable Convolution nhằm tiết kiệm tài nguyên tính toán và tài nguyên bộ nhớ.

Các lớp còn lại tương tự như các tương tự như các lớp phía trên.

Ở đây kích thước các đặc trưng của ảnh (các hidden layer) được giữ nguyên với mục đích nhằm đơn giản hóa mô hình. Vì khi các lớp tương tự nhau, các phép toán tương tự nhau sẽ tạo ra lợi thế khi thiết kế mô hình trên FPGA giúp đơn giản hóa thuật toán, đơn giản hóa các khối chức năng bằng cách cố gắng tương đồng hóa các lớp ẩn với nhau, với phương pháp này ta có thể thực hiện các lớp ẩn tương tự nhau giúp kiến trúc của các khối điều khiển, các khối chức năng được đơn giản hơn, không quá phức tạp góp phần khả thi hơn khi thiết kế trên FPGA.



Hình 18: Kiến trúc thực tế của mô hình.

Trong đó lớp đầu tiên (INPUT IMAGE) chính là 3 kênh màu của hình ảnh đầu vào.

WEIGHT0, WEIGHT1, WEIGHT2, WEIGHT3: Là các trọng số mô hình qua từng lớp. Ví dụ tích chập của lớp Input Image và Weight0 ta sẽ được lớp FEATURE0 (Trích xuất đặc trưng của ảnh đầu vào).

FEATURE0, FEATURE1, FEATURE2, FEATURE3: Là các lớp chứa các đặc trưng của ảnh đầu vào. Ví dụ lớp Feature0 là kết quả của trích xuất đặc trưng của lớp input image với trọng số là lớp Weight0.

AVERAGE: Là lớp chứa giá trị trung bình của một Feature. Đối mô hình CNN truyền thống ở lớp này thường là lớp Flatten (lớp trải phẳng) Tuy nhiên nếu trải phẳng sẽ sinh ra rất nhiều tham số cho phần fully connected. Vì vậy ở đây chúng ta dùng lớp trung bình nhằm tiết kiệm tài nguyên phần cứng.

Các quá trình (loại tính toán) bao gồm:

Từ INPUT IMAGE đến FEATURE0: Thực hiện thuật toán convolution 2D truyền thống.

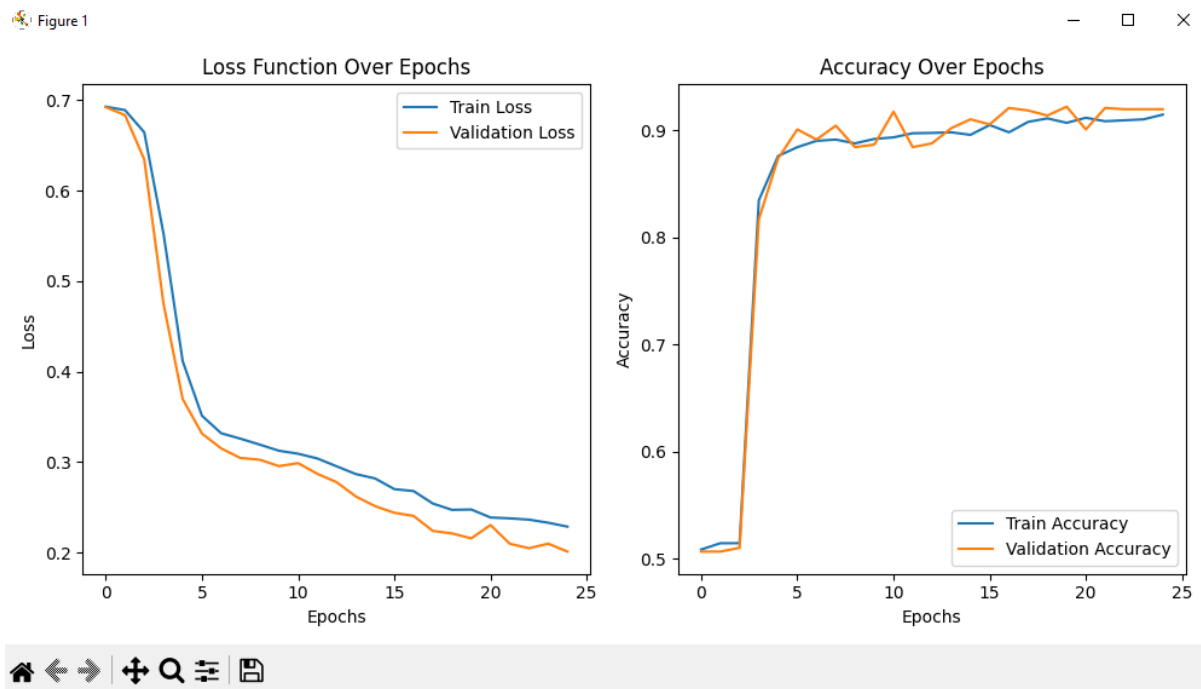
Từ FEATURE0 đến FEATURE1, từ FEATURE1 đến FEATURE2, từ FEATURE2 đến FEATURE3: Thực hiện thuật toán depthwise convolution 2D nhằm tiết kiệm tài nguyên.

Từ FEATURE3 đến AVERAGE: thực hiện thuật toán global average pooling 2D.

Từ AVERAGE đến OUTPUT: Thực hiện thuật toán fully connected (dense).

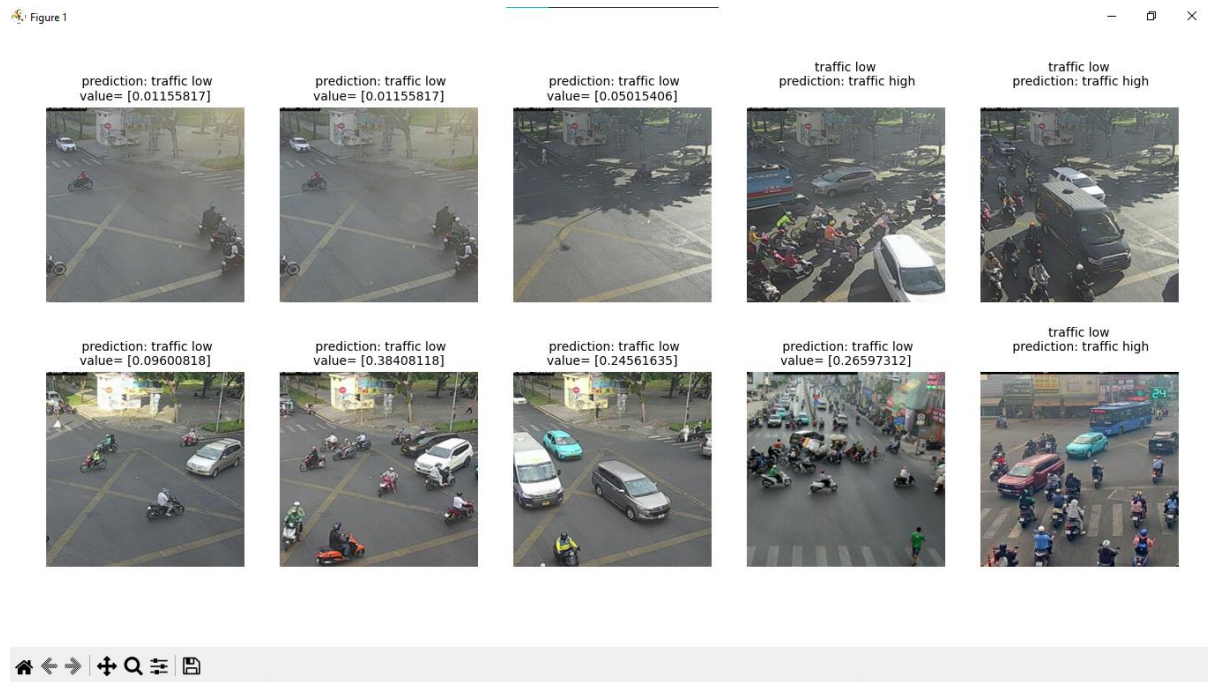
5. KẾT QUẢ THỰC HIỆN MÔ HÌNH TRÊN MÁY TÍNH.

Kết quả của quá trình huấn luyện mô hình trên máy tính: Loss sẽ giảm dần qua những lần học, Độ chính xác của mô hình sẽ tăng lên sau mỗi lần học và đạt bão hòa vào khoảng 0.9 (90%) trên tập test.

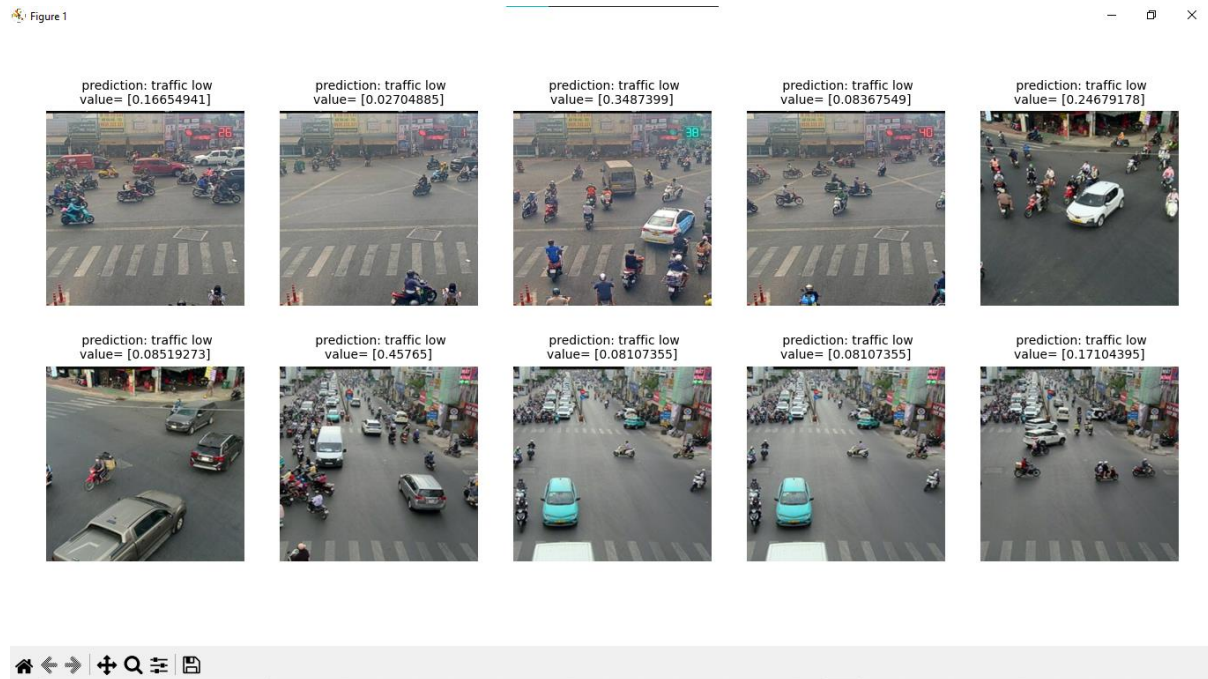


Hình 19: Kết quả huấn luyện mô hình trên máy tính.

Kết quả kiểm tra trên tập dữ liệu test của giao thông Việt Nam:



Hình 20: Kiểm thử mô hình trên ảnh của giao thông Việt Nam 1



Hình 21: Kiểm thử mô hình với ảnh giao thông Việt Nam 2

Ta có thể nhận thấy rằng độ chính xác của mô hình đã bị giảm, do nguồn dữ liệu của tập training không thật sự phù hợp với giao thông Việt Nam (tỉ lệ số lượng xe máy, xe ô tô, góc độ của camera, ánh sáng, hình dạng cung đường, giao lộ)

6. THỰC HIỆN PHẦN CỨNG

6.1 Kiến trúc tổng quan

Hệ thống triển khai MobileNet trên FPGA được tổ chức thành ba khối xử lý dữ liệu chính và quan trọng nhất, nhằm thực hiện tuần tự các công việc từ thu nhận ảnh đầu vào xử lý ảnh đầu vào qua các thực toán, đến xuất kết quả.

Phần 1: Camera control

Chức năng chính: Điều khiển đọc tín hiệu VSYNC, HREF, PCLK từ camera OV7670, lưu từng pixel vào FIFO. Xử lý chia dữ liệu thô từ camera thành 3 kênh màu, điều khiển, điều phối thực hiện ghi dữ liệu từ FIFO vào SDRAM.

Phần 2: Main MobileNet

Chức năng chính: Đây là khối xử lý cốt lõi trong hệ thống mô hình MobileNet thực thi trên FPGA trong vấn đề thực thi tính toán. Chức năng chính của nó là thực hiện các hoạt động xử lý dữ liệu trung tâm của kiến trúc MobileNet sau khi hình ảnh đầu vào được lấy từ camera và được lưu trữ trong SDRAM.

2.1 Fetching memory: Lấy dữ liệu từ SDRAM (các feature – đặc trưng) lưu vào source RAM, nhằm tăng tốc quá trình tính toán cho hệ thống.

2.2 Convolution algorithm: Lấy dữ liệu từ source RAM và weight generation thực hiện tính tích chập và lưu vào destination RAM.

2.3 Average algorithm: Lấy dữ liệu từ source RAM, thực hiện tính trung bình, lưu vào thanh ghi chứa giá trị trung bình.

2.4 Fully connected algorithm: Lấy dữ liệu từ thanh ghi chứa dữ liệu trung bình. Thực hiện phép toán tích chập với trọng số. Lưu giá trị vào drive và điều khiển LED báo kết quả.

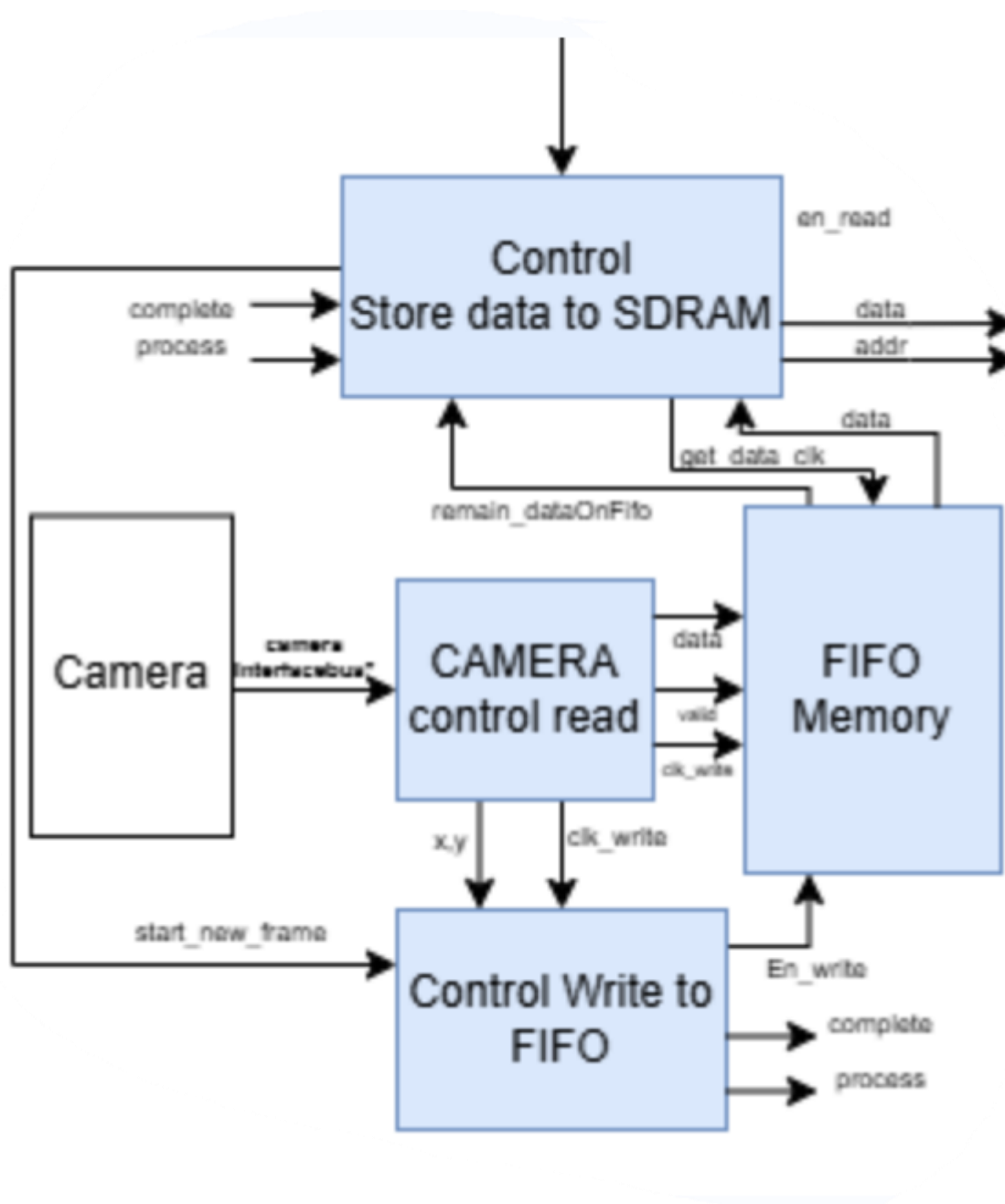
2.5 Write back memory: Sau khi các khối tính toán xong và lưu vào RAM đích, khối này sẽ thực hiện chức năng lưu dữ liệu từ destination RAM về lại SDRAM nhằm giữ không gian cho việc tính toán feature tiếp theo.

Phần 3: MASTER Control:

Chức năng chính: Đây là khối có vai trò quan trọng cốt lõi trong việc điều khiển, điều hòa, điều tiếp hoạt động của các nguồn tài nguyên như SDRAM, camera, source RAM, destination RAM,...

6.2 Khối Camera control

Phần này có chức năng nhận tín hiệu điều khiển từ bộ master control. Nhận dữ liệu thô từ camera lưu vào FIFO. Xử lý dữ liệu thô từ FIFO thành dữ liệu 3 kênh RGB điều tiết, lưu vào bộ SDRAM.



Hình 22: Sơ đồ khối camera control

Phần này gồm 3 module chính bao gồm: Camera control read, control write to FIFO, control store data to SDRAM và một bộ nhớ fifo memory.

6.2.1 Camera control read

Module này có chứa năng nhận tín hiệu từ camera điều khiển ghi dữ liệu thô vào FIFO.

Input port:

PCLK, VSYNC, HREF: đồng bộ xung pixel, khung và dòng.

D[7:0]: dữ liệu màu/điểm ảnh.

Output port:

pixel_data[15:0]: dữ liệu pixel 16 bit hợp lệ khi pixel_valid = 1.

pixel_valid: cao trong chu kỳ PCLK nếu dữ liệu của pixel là hợp lệ.

X_index[8:0]: số chỉ vị trí của pixel theo trục x.

Y_index[8:0]: số chỉ vị trí của pixel theo trục y.

FSM:

State	Mô tả	Điều khiển chuyển trạng thái
Idle	Chờ tín hiệu VSYNC lên mức cao khi bắt đầu một khung. Reset y_index về lại 0.	VSYNC = 1, trạng thái kế tiếp Row read
ReadFirstByte	Ghi byte đầu tiên	Chuyển sang ReadLastByte
ReadLastByte	Ghi byte cuối của một pixel. Cộng x_index tiếp theo lên 1.	Nếu HREF = 1 chuyển lại về ReadFirstByte, Nếu HREF chuyển về 0 chuyển sang endRow
Endrow	Chờ HREF lên lại 1 cho hàng pixel tiếp theo, y_index tăng lên 1.	Nếu HREF = 1 chuyển đến ReadFirstByte, Nếu VSYNC = 1, chuyển về trạng thái Idle.

Bảng 1: Bảng chuyển trạng thái khối cameraRead

6.2.2 Control write to FIFO

Chức năng của khối này là nhận tín hiệu đọc frame từ bộ control store to sdram để bắt đầu nhận dữ liệu từ bộ camera read và ghi dữ liệu pixel từ cameraRead vào FIFO.

Module này điều khiển việc ghi dữ liệu từ camera vào FIFO dựa trên các tín hiệu i_xIndex, i_yIndex, và i_get. Nó sử dụng một máy trạng thái (FSM) với 3 trạng thái: idle, process, finish.

Bảng chuyển trạng thái:

Trạng thái hiện tại	Điều kiện chuyển trạng thái	Trạng thái kế tiếp	Mô tả
---------------------	-----------------------------	--------------------	-------

Idle	X_index = 0; y_index =0; i_get = 1	Process	Trạng thái nghỉ chờ tín hiệu bắt đầu và chờ đồng bộ frame
Idle	Không thỏa điều kiện trên	Idle	Chờ
Process	x_index = 63, y_index =63	Finish	Kết thúc việc ghi vào fifo vì đã đủ 64x64 pixel.
Process	Không thỏa điều kiện trên	Process	Tiếp tục xử lý
Finish	Bắc kỳ	Idle	Bắc cờ hoàn thành.

Bảng 2: Bảng chuyển trạng thái khối control write to FIFO

Bảng output tương ứng với trạng thái:

Bảng 3: Bảng ngõ ra máy trạng thái khối control write to FIFO

Trạng thái	masterE	O_complete	O_process	O_eWriteFIFO
Idle	064	1	0	0
Process	1	0	1	1(nếu x < 64, y <)
Finish	0	1	0	0

6.2.3 control store data to sdram

Module này điều khiển việc đọc dữ liệu từ FIFO (được ghi từ camera) và ghi vào SDRAM. Nó xử lý dữ liệu RGB, chia thành các thành phần Red, Green, Blue, và ghi vào các vùng nhớ khác nhau trên SDRAM.

Input port list:

Bảng 4: Bảng mô tả danh sách ngõ vào khối control read cam to SDRAM

Signal	Width	Mô tả
I_clk	1	Clock signal input
I_reset	1	Reset signal active low
I_start	1	Start signal to start module
I_remainOnFifo	10	Remain word data on FIFO
I_process	1	Signal báo bộ điều khiển ghi vào FIFO vẫn còn hoạt động
I_complete	1	Signal báo bộ điều khiển ghi vào FIFO đã hoàn thành, sẽ không còn word nào được ghi thêm vào FIFO.
I_dataFIFO	16	Data đọc từ FIFO (RGB565)

Output port list:

Bảng 5: Bảng mô tả danh sách ngõ ra khỏi control read cam to SDRAM

Signal	Width	Mô tả
o_get	1	Enable và tín hiệu start cho bộ control write fifo bắt đầu hoạt động.
o_EnReadFifo	1	Tín hiệu xin phép đọc FIFO
o_RdClkFifo	1	Tín hiệu clock read FIFO
o_dataSdram	16	Data đến SDRAM
o_addressToSdram	19	SDRAM address
o_wrSdram	1	Tín hiệu xin phép ghi vào SDRAM
o_finish	1	Cờ báo đã lưu được một frame vào SDRAM

Bảng chuyển trạng thái:

Bảng 6: Bảng chuyển trạng thái khỏi control read cam to sdram

Trạng thái hiện tại	Điều kiện	Trạng thái tiếp theo	Mô tả
Idle	i_start & i_reset	Start	Trạng thái chờ
Idle	Không thỏa điều kiện trên	Idle	
Start	I_process	Suppend	Bật cờ khởi động cho khối control write to FIFO.
Start	!I_process	Start	Tiếp tục chờ đến khi khối control write to FIFO khởi động
Suppend	I_remainOnFifo > 16 Hoặc i_complete	readFifo0	Sẵn sàng chuyển data từ fifo sang SDRAM
Suppend	Không thỏa điều kiện trên	Suppend	Tiếp tục chờ FIFO đủ dữ liệu
readFifo0	Luôn luôn chuyển	readFifo	Tạo cạnh lên cho clock đọc dữ liệu fifo
readFifo1	Luôn luôn chuyển	setRed	Kéo clock read fifo về lại 0

setRed	Luôn luôn chuyển	Wait0	Đưa cổng output data thành dữ liệu lệnh đỏ, chuẩn bị address, bậc chờ cho phép ghi vào SDRAM
Wait0	I_sdramRead	setGreen	Chờ SDRAM hoàn thành việc ghi.
Wait0	Không thỏa điều kiện trên	Wait0	Tiếp tục chờ SDRAM hoàn thành ghi.
setGreen	Luôn luôn chuyển	Wait1	Chuẩn bị ghi kênh Green
Wait1	I_sdramReady	setBlue	Chờ hoàn thành ghi
Wait1	Không thỏa điều kiện trên	Wait1	
setBlue	Luôn luôn chuyển	Wait2	Chuẩn bị ghi kênh Blue
Wait2	I_sdramReady	Update	Chờ hoàn thành ghi
Wait2	Không thỏa điều kiện trên	Wait2	
Update	!i_complete i_remain > 0	Suspend	Cập nhật địa chỉ local, Kiểm tra nếu chưa hoàn thành frame.
Update	I_complete & i_remain = 0	Finish	Cập nhật địa chỉ local, kiểm tra nếu đã hoàn thành frame
Finish	Luôn luôn chuyển	Idle	Bậc chờ hoàn thành

Bảng output của máy trạng thái:

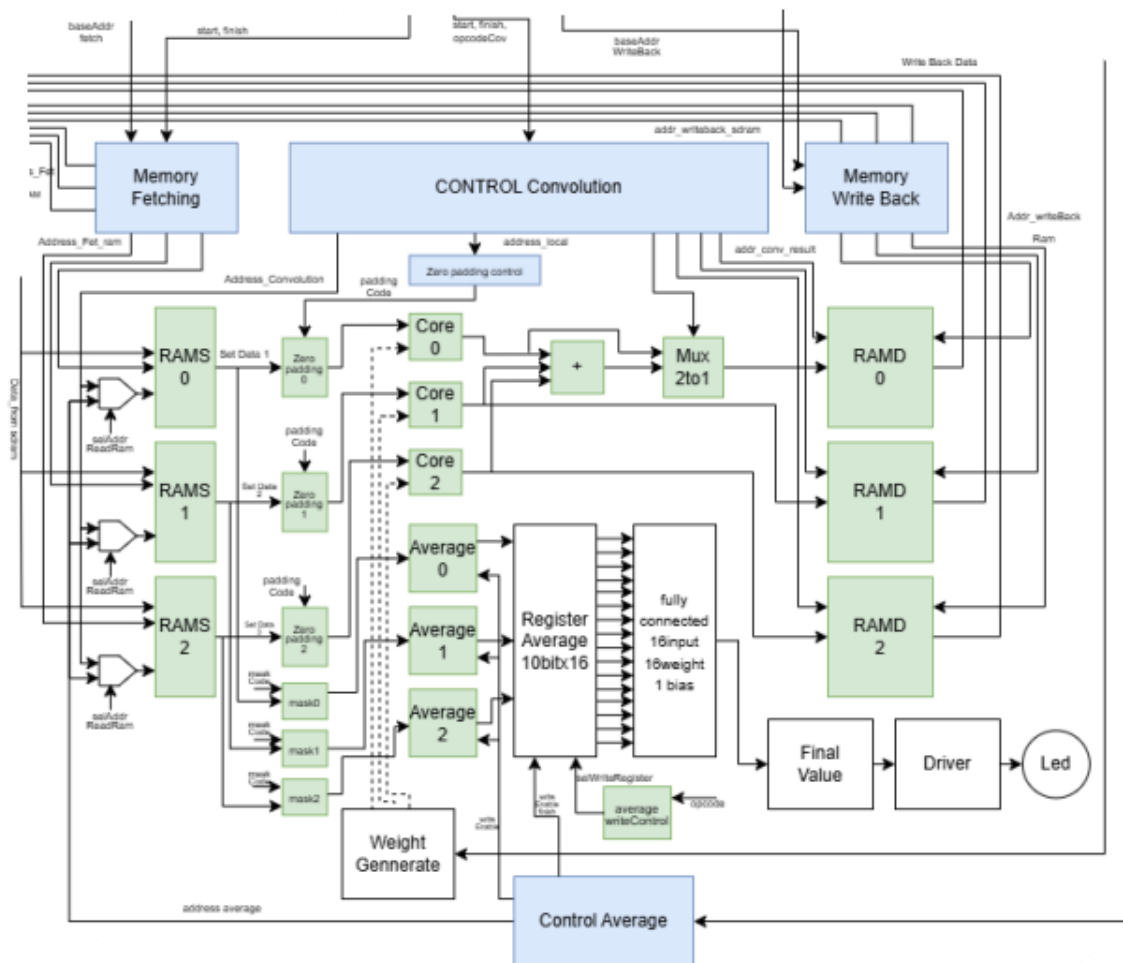
Bảng 7: Bảng ngõ ra máy trạng thái khối control read camera to SDRAM

Trạng thái	O_get	O_En Read FIFO	O_Rd Clk Fifo	O_wr Sdram	O_finish	Mô tả hành vi
Idle	0	0	0	0	0	Chờ tín hiệu i_start
Start	1	0	0	0	0	Bậc chờ o_get để khởi động khối control write to FIFO

Suppend	0	1	0	0	0	Bậc chờ xin phép đọc fifo, chờ FIFO đủ data
readFifo0	0	1	1	0	0	Tạo rasing read clock fifo
readFifo1	0	1	0	0	0	Reset clock read fifo trở lại 0, tắt tín hiệu xin phép đọc fifo.
setRed	0	0	0	1	0	Bậc tín hiệu xin phép ghi vào SDRAM, chuẩn bị data kênh đỏ, address kênh đỏ. $Addr = addrLocal + 0$
Wait0	0	0	0	0	0	Chờ sdram ghi xong
SetGreen	0	0	0	1	0	Chuẩn bị ghi kênh xanh lá. $Addr = addrLocal + 4096$
Wait1	0	0	0	0	0	Chờ sdram ghi xong
setBlue	0	0	0	1	0	Chuẩn bị ghi kênh xanh dương $Addr = addrLocal + 8192$
Wait2	0	0	0	0	0	Chờ SDRAM ghi xong
Update	0	0	0	0	0	Tăng addrLocal lên 1, kiểm tra fifo còn data không.
Finish	0	0	0	0	1	Bậc chờ finish, reset addressLocal về lại 0

6.3 Khối main mobile net

Khối Main MobileNet là trung tâm xử lý chính trong hệ thống, thực hiện toàn bộ các bước tính toán của mô hình MobileNet từ khi nhận dữ liệu ảnh cho đến khi đưa ra kết quả. Bao gồm các giai đoạn: đọc dữ liệu (fetch), tính toán tích chập (convolution), tính trung bình cộng (average pooling), lớp kết nối đầy đủ (fully connected) và ghi lại dữ liệu (write back). Mỗi bước đều được thiết kế tối ưu để tận dụng tài nguyên phần cứng của FPGA, đồng thời đảm bảo khả năng xử lý song song và tuần tự phù hợp với kiến trúc MobileNet ban đầu (3 core xử lý song song). Việc tổ chức độc lập các khối chức năng và dùng chung các khối RAM nguồn (source ram) và RAM đích (destination ram) nhằm tiết kiệm tài nguyên bộ nhớ cho FPGA.



Hình 23: Sơ đồ mô hình Mobile Net

6.3.1 Memory fetching

Module này có chức năng lấy dữ liệu từ SDRAM và lưu vào bộ nhớ RAM nguồn (source ram) nhằm chuẩn bị cho việc tính toán.

Input port của module fetching memory.

Bảng 8: Danh sách input port của module memory fetching

Tên tín hiệu	Độ rộng	Mô tả
I_baseAddr0	19	Địa chỉ cơ sở cho SDRAM vùng 0
I_baseAddr1	19	Địa chỉ cơ sở cho SDRAM vùng 1
I_baseAddr2	19	Địa chỉ cơ sở cho SDRAM vùng 2
I_clk	1	Clock
I_reset	1	Reset signal

I_sdramReady	1	Tín hiệu báo SDRAM sẵn sàng
I_start	1	Tín hiệu khởi động module fetching

Ouput port của module fetching memory.

Bảng 9: Danh sách output port của module fetching memory

Tín hiệu	Độ rộng	Mô tả
O_rdSdram	1	Tín hiệu xin phép đọc SDRAM
O_addrToSdram	19	Address đến SDRAM
O_finish	1	Tín hiệu báo hoàn thành fetching
O_wrRam0	1	Tín hiệu xin phép ghi vào RAM0
O_wrRam1	1	Tín hiệu xin phép ghi vào RAM1
O_wrRam2	1	Tín hiệu xin phép ghi vào RAM2
O_addrToRam	12	Địa chỉ cho việc ghi vào RAM. Dùng chung cho 3 vùng ram.

Bảng chuyển trạng thái của khối memory fetching

Bảng 10: Bảng chuyển trạng thái module memory fetching

Trạng thái hiện tại	Điều kiện chuyển	Trạng thái kế tiếp	Mô tả
Idle	I_start	setSdram0	Chờ tín hiệu khởi động bộ fet
Idle	!i_start	Idle	
setSdram0	Luôn luôn	waitSdram0	Chuẩn bị address đọc dữ liệu từ SDRAM
waitSdram0	I_sdramReady	setRam0	Chờ dữ sdram hoàn thành đọc dữ liệu
waitSdram0	!i_sdramReady	waitSdram0	
setRam0	Luôn luôn	setSdram1	Ghi dữ liệu vừa đọc được từ sdram vào block ram 0
setSdram1	Luôn luôn	waitSdram1	Chuẩn bị address đọc dữ liệu từ sdram cho block ram 1
waitSdram1	i_sdsramReady	setRam1	Chờ sdram hoàn thành đọc
WaitSdram1	!i_sdramReady	waitSdram1	

setRam1	Luôn luôn	setSdram2	Ghi dữ liệu vừa đọc được từ sdram vào block ram 1
setSdram2	Luôn luôn	waitSdram2	Chuẩn bị địa chỉ đọc dữ liệu sdram cho block ram 2
waitSdram2	i_sdramReady	setRam2	Chờ Sdram hoàn thành đọc
waitSdram2	!i_sdramReady	waitSdram2	
setRam2	Luôn luôn	Update	Ghi dữ liệu vừa đọc được từ sdram vào block ram 2
Update	addrLocal = 4095	Finish	Cập nhật addrLocal và kiểm tra nếu đây là dữ liệu cuối cùng của feature
Update	addrLocal!= 4095	setSdram0	
Finish	Luôn luôn	Idle	Bậc chờ báo fetching hoàn thành

Output máy trạng thái module memory fetching

Trạng thái	O_rdSdram	O_wr Ram0	O_wr Ram1	O_wr Ram2	O_addrTo Sdram	O_finish
Idle	0	0	0	0	0	0
setSdram0	1	0	0	0	I_baseAddr0 +addrLocal	0
waitSdram0	0	0	0	0	I_baseAddr0 +addrLocal	0
setRam0	0	1	0	0	I_baseAddr0 +addrLocal	0
setSdram1	1	0	0	0	I_baseAddr1 +addrLocal	0
waitSdram1	0	0	0	0	I_baseAddr1 +addrLocal	0
setRam1	0	0	1	0	I_baseAddr1 +addrLocal	0
setSdram2	1	0	0	0	I_baseAddr2 +addrLocal	0
waitSdram2	0	0	0	0	I_baseAddr2 +addrLocal	0
setRam2	0	0	0	1	I_baseAddr2 +addrLocal	0
Update	0	0	0	0	I_baseAddr0 +addrLocal	0
Finish	0	0	0	0	0	1

6.3.2 Ram nguồn (source ram) và Ram đích (destination ram)

Module Ram này có cấu trúc gồm một block Ram (convRam) và một bộ điều khiển đọc ghi tuần tự (ramControl)

Module ramControl quản lý việc đọc tuần tự từ RAM (ramConv), tổng hợp dữ liệu thành bộ đệm 90-bit và cung cấp giao diện đọc nhanh.

Module convRam là loại RAM một cổng (4096x10-bit) với hoạt động đọc/ghi đồng bộ.

Như vậy ta có thể thấy rằng module RAM mặc dù cần xuất 9 dữ liệu qua 9 cổng output (10 bit) để thực hiện tích chập với kernel. Tuy nhiên thiết kế vẫn sử dụng module RAM một cổng ngõ ra 10 bit đồng bộ. Điều này sẽ có hạn chế là thay vì RAM có 9 port data output sẽ đọc được 9 mẫu dữ liệu trong một chu kỳ, Còn khi thực hiện đọc tuần tự như trên sẽ cần ít nhất 9 chu kỳ để có thể đọc được 9 dữ liệu ở 9 địa chỉ khác nhau. Tuy nhiên ta vẫn phải thực hiện điều này do các nguyên nhân sau đây: Tối Ưu Tài Nguyên Phần Cứng, Đọc song song 9x10-bit yêu cầu 9 bộ đọc độc lập, 9 đường địa chỉ, và 9 đường dữ liệu. Chiếm dụng nhiều LUTs, FFs, và BRAMs trên FPGA, đặc biệt với kích thước RAM 4096x10-bit. Đọc tuần tự chỉ cần 1 bộ đọc và 1 đường địa chỉ/dữ liệu, tái sử dụng qua các chu kỳ tiết kiệm tài nguyên phần cứng của FPGA. Quan trọng hơn là khi dùng ram 1 port dữ liệu ra sẽ được quartus synthesis sử dụng embedded ram trên FPGA. Nếu ta sử dụng 9 port song song quartus buộc phải dùng đến memory element phân tán (distribute memory elements) Điều này sẽ tiêu tốn rất nhiều tài nguyên của FPGA dẫn đến không thể synthesis được do thiếu tài nguyên.

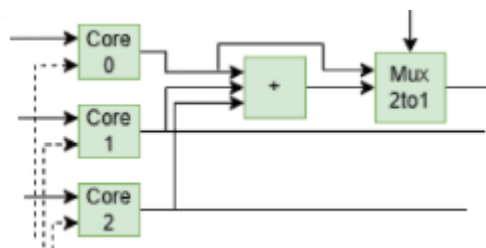
Input/output port module ramControl

Bảng 11: Danh sách input output port của khối điều khiển RAM.

Port name	In/out	Width	Description
I_addrOut	Input	108	9x12bit địa chỉ đọc dữ liệu 9 port
I_addrIn	Input	12	Địa chỉ ghi vào RAM
I_dataIn	Input	10	Dữ liệu ghi vào RAM
I_wrEnable	Input	1	Tín hiệu xin phép ghi vào bộ RAM
I_quickGet	Input	1	Cờ kích hoạt chế độ đọc nhanh (Chỉ đọc 1 dữ liệu, mà không cần phải chờ 9 chu

			kỳ để đọc 9 bộ dữ liệu)
I_addrOutQuick	Input	12	Địa chỉ cho việc đọc nhanh
I_clk	Input	1	Xung clock
I_reset	Input	1	Tín hiệu reset (tích cực thấp)
I_start	Input	1	Tín hiệu bắt đầu đọc tuần tự
O_dataOut	Output	90	Dữ liệu đọc được từ mode đọc tuần tự
O_quickData	Output	10	Dữ liệu đọc được từ mode đọc nhanh 1 dữ liệu
O_valid	Output	1	Cờ báo hiệu hợp lệ cho mode đọc tuần tự
O_ready	Output	1	Cờ báo bộ RAM đang ở chế độ rảnh, sẵn sàng nhận lệnh

6.3.3 Module tính toán convolution



Hình 24: Kiến trúc module conv

Module coreConv là một khối xử lý tích chập 9 mẫu: Nhận vào hai bus 90 bit (i_data và i_weight), mỗi bus chứa 9 phần tử 10 bit liên tiếp, sau đó giải nén thành mảng $data[8:0]$ và $weight[8:0]$. Chín cặp $data[i]-weight[i]$ được đưa song song vào chín khối multi để tính tích, sinh ra mảng $resultMulti[9:0]$. Cuối cùng, toàn bộ tám tích số này được cộng lại trong một khối plus9Para đa ngõ (9-input adder) để cho ra kết quả 10 bit duy nhất trên đầu ra o_data .

Trên nền tảng đó, module conv đóng vai trò top-level, tái sử dụng ba module coreConv hoạt động song song: Mỗi instantiation có bus dữ liệu ($i_busData0/1/2$) và bus trọng số ($i_busWeight0/1/2$) riêng biệt dài 90 bit, cho phép thực hiện đồng thời ba phép tích chập khác nhau. Kết quả 10 bit của từng kênh được xuất ra qua

o_data0, o_data1 và o_data2. Tham số i_opcode 1 bit dùng để điều khiển chế độ hoạt động (0: thực hiện mode tích chập 3 feature thành 1 kết quả, 1: thực hiện mode tích chập 1 feature được 1 kết quả) cho cả ba khối con, giúp linh hoạt trong các ứng dụng yêu cầu các phép tính tích chập có 2 mode điều chỉnh.

Danh sách ngõ ra, ngõ vào module convCore:

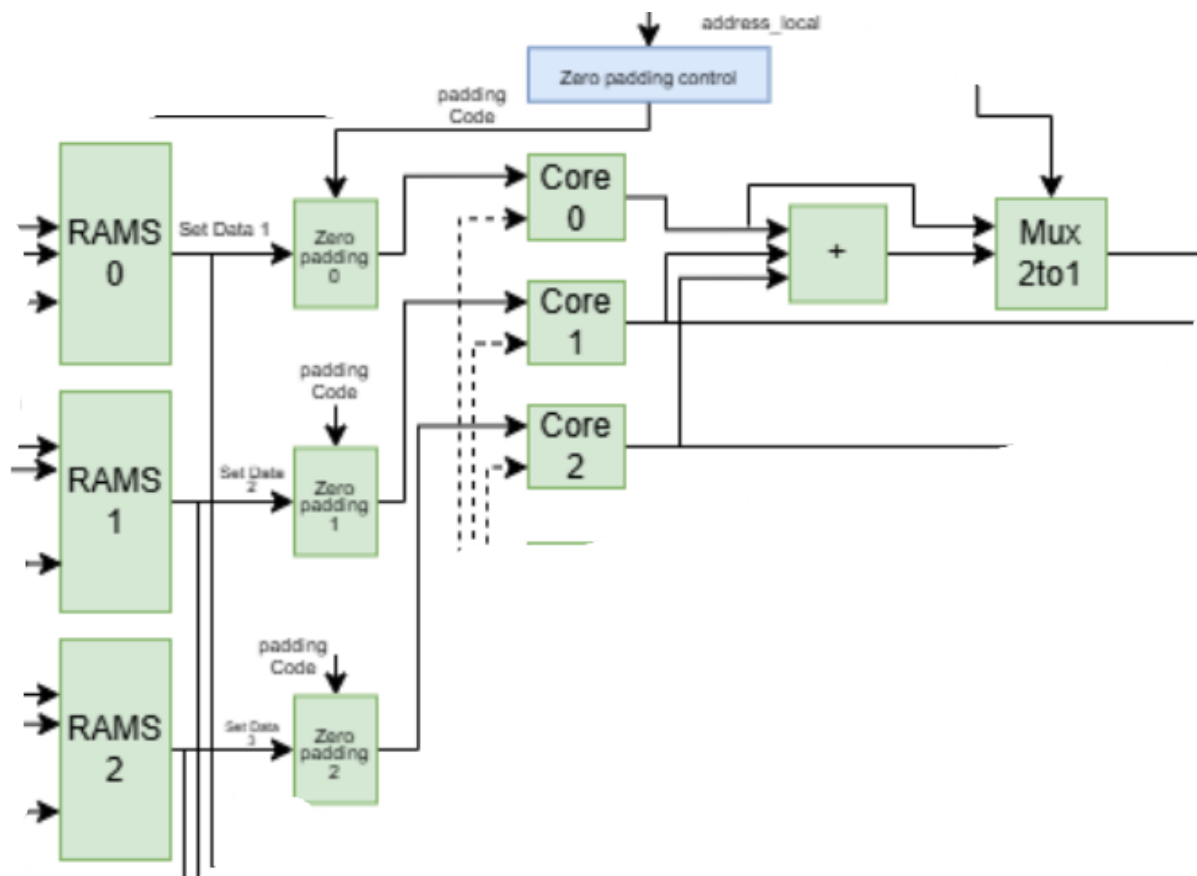
Bảng 12: Danh sách input/output module convCore

Port name	In/out	Width	Discription
I_data	Input	90	Dữ liệu đầu vào 9x10bit
I_weight	Input	90	Weight đầu vào 9x10bit
O_data	Output	10	Kết quả phép tính chập (Tổng 9 phép nhân data và trọng số)

Danh sách ngõ ra, vào khối conv:

Port name	In/out	Width	Discription
I_busData0	Input	90	Dữ liệu vào luồng 0, 9x10bit
I_busData1	Input	90	Dữ liệu vào luồng 1, 9x10bit
I_busData1	Input	90	Dữ liệu vào luồng 2, 9x10bit
I_busWeight0	Input	90	Weight vào luồng 0, 9x10bit
I_busWeight1	Input	90	Weight vào luồng 1, 9x10bit
I_busWeight2	Input	90	Weight vào luồng 2, 9x10bit
I_opcode	Input	90	Mode tích chập 3 feature hoặc 1 feature
O_data0	Output	10	Kết quả tích chập luồng 0
O_data1	Output	10	Kết quả tích chập luồng 1
O_data2	Output	10	Kết quả tích chập luồng 2

Ngoài ra khi tính toán đôi khi ra không lấy hoàn toàn 9 bộ dữ liệu từ RAM mà đôi khi phải chèn 0 vào một số mẫu. Lý do khi thực hiện tích chập ở các cạnh của feature ta cần thêm 0 vào những vị trí không có mẫu (ví dụ khi thực hiện tích chập tại vị trí có tọa độ $x=12, y=0$ – địa chỉ mẫu ngay hàng đầu tiên của feature, khi này ta cần thêm 3 mẫu 0 vào hàng trên của cửa sổ.)



Hình 25: Minh họa thêm module chức năng padding cho module convolution

Như vậy ta có thêm khối Zero padding và khối padding control. Khối Zero padding có chức năng thêm các mẫu 0 vào bộ data với sự điều khiển từ khối padding control. Khối padding control có chức năng nhận tín hiệu vị trí tham chiếu của vị trí đang được tính tích chập để đưa ra tín hiệu điều khiển padding.

Khối padding control port list:

Bảng 13: Danh sách input output module padding control

Port name	In/out	Width	Discription
I_localAddr	Input	12	Địa chỉ pixel trong ma trận (kích thước 64x64)

O_sel[0:8]	Output	1	Tín hiệu chọn để xác định vị trí padding
------------	--------	---	--

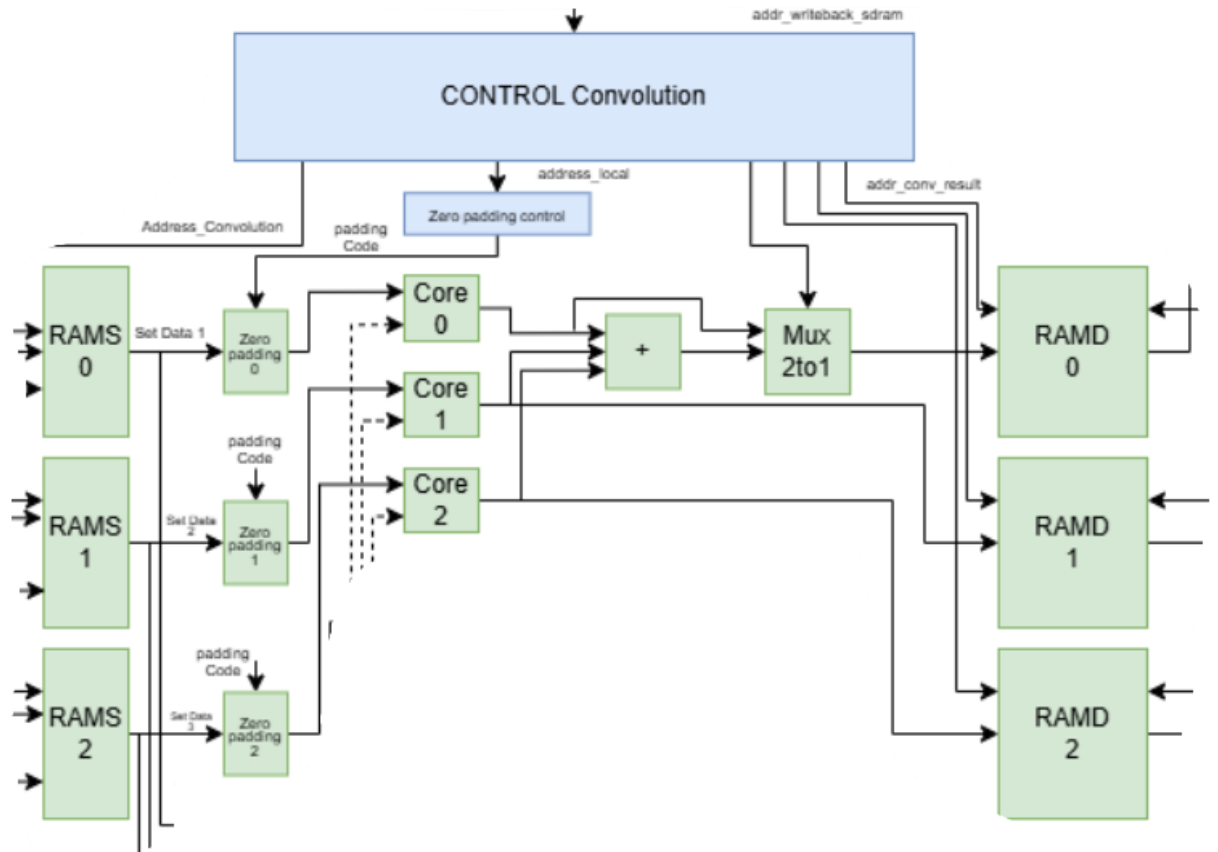
Logic: Phân tích i_localAddr thành 2 phần: hHigh/hLow: Kiểm tra 6 bit cao (địa chỉ hàng). hHigh = 1 khi tất cả 6 bit cao là 1 → Hàng cuối cùng. hLow = 1 khi tất cả 6 bit cao là 0 → Hàng đầu tiên. lHigh/lLow: Kiểm tra 6 bit thấp (địa chỉ cột). lHigh = 1 khi tất cả 6 bit thấp là 1 → Cột cuối cùng. lLow = 1 khi tất cả 6 bit thấp là 0 → Cột đầu tiên.

Xác định vị trí padding: Các tín hiệu o_sel0–o_sel8 tương ứng với 9 vị trí trong kernel 3x3.

Khởi zero padding: Đơn giản hơn rất nhiều chỉ cần thực hiện phép toán AND tương ứng data và tín hiệu select. Vd: o_data[9:0] = i_sel0 & i_data[9:0] tương tự với các data còn lại.

Tiếp theo là khối rất quan trọng đóng vai trò điều khiển hoạt động của bộ convolution – Module controlConv

Module convolutionControl điều khiển quá trình thực hiện tích chập: từ việc phát lệnh khởi động, đọc dữ liệu từ source RAM, chờ source RAM trả dữ liệu hợp lệ, ghi kết quả ngược lại vào destination RAM rồi tự động cập nhật con trỏ địa chỉ cho đến khi hoàn thành duyệt hết vùng dữ liệu (đến địa chỉ 4095). Toàn bộ quá trình này được cài đặt dưới dạng FSM, sử dụng các tín hiệu điều khiển i_start, i_reset, i_validRam và i_opcode để đồng bộ và phân luồng truy cập RAM, đồng thời sinh ra các tín hiệu điều khiển đọc/ghi và báo hiệu hoàn thành.



Hình 26: Minh họa khối control convolution giao tiếp với khối convolution

Danh sách ngõ ra, ngõ vào khối convolutionControl:

Bảng 14: Danh sách input output port module convolution control

Port name	In/out	Width	Discription
I_clk	Input	1	Xung clock
I_reset	Input	1	Reset signal (active low)
I_start	Input	1	Tín hiệu khởi động quá trình tích chập
I_opcode	Input	1	Tín hiệu điều khiển mode tích chập (3 feature hoặc 1 feature)
I_validRam	Input	1	Tín hiệu xác nhận dữ liệu đọc từ source ram đã sẵn sàng và hợp lệ
O_addrRead0	Output	108	Địa chỉ đọc dữ liệu cho luồng 0 (9x12bit)
O_addrRead1	Output	108	Địa chỉ đọc dữ liệu cho luồng 1 (9x12bit)
O_addrRead2	Output	108	Địa chỉ đọc dữ liệu cho luồng 2 (9x12bit)
O_startRam	Output	1	Tín hiệu khởi động source RAM bắt đầu đọc
O_selRamD0	Output	1	Tín hiệu chọn opcode cho convolution
O_addrWrite0	Output	12	Địa chỉ ghi kết quả trên destination ram luồng 0
O_addrWrite1	Output	12	Địa chỉ ghi kết quả trên destination ram luồng 1

O_addrWrite2	Output	12	Địa chỉ ghi kết quả trên destination ram luồng 2
O_wrEnable	Output	1	Tín hiệu write enable cho destination ram
O_finish	Output	1	Cờ báo hiệu feature đã được tích chập xong, kết quả đã được lưu vào destination ram
O_localAddr	Output	12	Đại chỉ pixel hiện tại đang được xử lý

Bảng chuyển trạng thái:

Bảng 15: Bảng chuyển trạng thái khối convolution control

Trạng thái hiện tại	Điều kiện chuyển trạng thái	Trạng thái kế tiếp
Idle	I_start & i_reset	readRam
Idle	Không thỏa điều kiện trên	Idle
readRam	Luôn luôn chuyển	waitDone
waitDone	I_validRam	setWrite
waitDone	!i_validRam	waitDone
setWrite	Luôn luôn chuyển	Update
Update	addrLocal = 4095	Finish
Update	Không thỏa điều kiện trên	readRam
Finish	Luôn luôn chuyển	Idle

Bảng output của mã trạng thái:

Bảng 16: Bảng output của máy trạng thái module convolution control

Trạng thái	O_startRam	O_wrEnable	O_finish	Mô tả hành vi
Idle	0	0	0	Chờ tín hiệu i_start. Reset địa chỉ và tín hiệu điều khiển.
readRam	1	0	0	Kích hoạt đọc RAM. Tính toán địa chỉ đọc (9 vị trí xung quanh addrLocal_q).
waitDone	0	0	0	Chờ tín hiệu i_validRam để xác nhận dữ liệu đã sẵn sàng.
setWrite	0	1	0	Kích hoạt ghi kết quả vào RAM tại địa chỉ addrLocal_q.
Update	0	0	0	Tăng addrLocal_q. Nếu đạt 4095, chuyển sang finish; ngược lại, tiếp tục đọc.
Finish	0	0	1	Báo hiệu hoàn thành và reset trạng thái.

Module tính toán 9 địa chỉ đọc xung quanh vị trí hiện tại (addrLocal_q) để lấy dữ liệu cho kernel 3x3:

$\text{addrRead0} = \text{addrLocal_q} - 65$ (góc trên trái)
 $\text{addrRead1} = \text{addrLocal_q} - 64$ (trên giữa)
 $\text{addrRead2} = \text{addrLocal_q} - 63$ (góc trên phải)
 $\text{addrRead3} = \text{addrLocal_q} - 1$ (giữa trái)
 $\text{addrRead4} = \text{addrLocal_q}$ (trung tâm)
 $\text{addrRead5} = \text{addrLocal_q} + 1$ (giữa phải)
 $\text{addrRead6} = \text{addrLocal_q} + 63$ (góc dưới trái)
 $\text{addrRead7} = \text{addrLocal_q} + 64$ (dưới giữa)
 $\text{addrRead8} = \text{addrLocal_q} + 65$ (góc dưới phải)

6.3.4 Khối thuật toán Average

Average trong kiến trúc MobileNet đảm nhiệm chức năng pooling trung bình (“average pooling”) để gom thông tin không gian từ feature-map thành một vector giá trị đặc trưng cho mỗi kênh, giảm độ phân giải không gian mà vẫn giữ lại đặc trưng chung. Phần Average gồm một số module chính như sau: AverageControl: một FSM điều khiển tuần tự đọc tuần tự các pixel trong feature. Average: mạch chính thực hiện phép cộng tích lũy các giá trị đã đọc, rồi chia (hoặc dịch bit) theo số phần tử để tính trung bình. AverageWriteControl: module sinh tín hiệu select ghi và tín hiệu wrEnable để xuất kết quả trung bình vào bộ register chứa giá trị trung bình.

Module average: Có chức năng tính trung bình bằng cách cộng dồn dữ liệu, mỗi lần cộng dồn là 9 dữ liệu đầu vào.

Bảng 17: Dánh sách input output của module average

Port name	In/out	Width	Discription
I_data	Input	90	Dữ liệu đầu vào 9x10bit
I_clk	Input	1	Xung clock
I_reset	Input	1	Reset signal
I_writeAdd	Input	1	Tín hiệu cộng dồn
O_data	Output	10	Giá trị trung bình đầu ra

Simple logic: Bất kì khi nào có clock thì module sẽ kiểm tra nếu i_writeAdd bằng 1 thì thanh ghi tính tổng sẽ cộng dồn i_data vào. Và khi hoàn thành, để tính

trung bình 64 phần tử ta cần chia giá trị tổng cho tổng số lượng phần tử ở đây là 64x64 phần tử hay 4096 đúng bằng 2 lũy thừa 12. Vì vậy ta chỉ cần dịch sang phải 12 bit là được tương ứng với chia 4096.

Như ta có thể thấy rằng module average không thể tính trung bình một lượt 64x64 phần tử vì Ram không thể xuất cùng lúc 64x64 phần tử. Đồng thời nếu làm vậy sẽ gây áp lực tính toán lớn hơn cho tài nguyên trên FPGA. Do vậy average đã được thiết kế để tính cộng dồn tích lũy mỗi lần 9 phần tử đến khi hoàn thành. Vì vậy ta cần một module để điều khiển việc load dữ liệu từ RAM, thực hiện cộng dồn, ghi dữ liệu vào register.

Bảng 18: Danh sách input output của module averageControl

Port name	In/out	Width	Discription
I_clk	In	1	Xung clock
I_reset	In	1	Reset signal (active low)
I_start	In	1	Tín hiệu khởi động quá trình tính trung bình của một feature
I_validRam	In	1	Tín hiệu báo hiệu RAM đọc hoàn tất và dữ liệu được sẵn sàng
O_addrRead0 - 2	Output	108	Địa chỉ đọc dữ liệu RAM 9x12bit, 3 core
O_writeEnable	Output	1	Tín hiệu xin phép ghi kết quả tính trung bình vào register
O_reserAverage	Output	1	Tín hiệu để reset giá trị cộng tích lũy về lại 0
O_mask	Output	1	Che dữ liệu khi thực hiện chu kỳ cuối cùng, feature không còn đủ 9 data để tính
O_startRam	Output	1	Tín hiệu khởi động đọc RAM
O_finish	Output	1	Cờ báo hoàn thành tính trung bình của một feature

Như vậy ta cần thiết kế một máy trạng thái để thực hiện module average control:

Bảng 19: Bảng chuyển trạng thái module average control

Trạng thái hiện tại	Điều kiện chuyển	Trạng thái kế tiếp	Mô tả hành vi
Idle	I_start & i_reset	Reset	Chờ tín hiệu khởi động
Reset	Luôn luôn chuyển	Buffer	Bậc tín hiệu reset để xóa giá

			trị tổng tích lũy cũ
Buffer	Luôn luôn chuyển	setAddr	Xóa tín hiệu reset
setAddr	Luôn luôn chuyển	waitDone	Thiết lập địa chỉ đọc dữ liệu cho bộ RAM, bậc chờ khởi động đọc
waitDone	I_validRam	writeAdd	Chờ RAM hoàn thành việc đọc dữ liệu
writeAdd	Luôn luôn chuyển	Update	Bậc chờ cho phép cộng tích lũy
Update	addrLocal = 4095	Finish	Cập nhập địa chỉ addrLocal += 9, kiểm tra nếu đây là pixel cuối của feature.
Update	Không thỏa điều kiện trên	setAddr	
Finish	Luôn luôn chuyển	Idle	Bậc chờ hoàn thành và bậc chờ ghi vào register.

Vì dữ liệu sau khi tích trung bình được lưu vào một bộ register. Không phải là RAM đích (destinaiton ram) như module convolution. Vì vậy ta cần một module nữa để điều khiển việc ghi vào register file. Khối này có chức năng tương tự như khối write back của khối convolution.

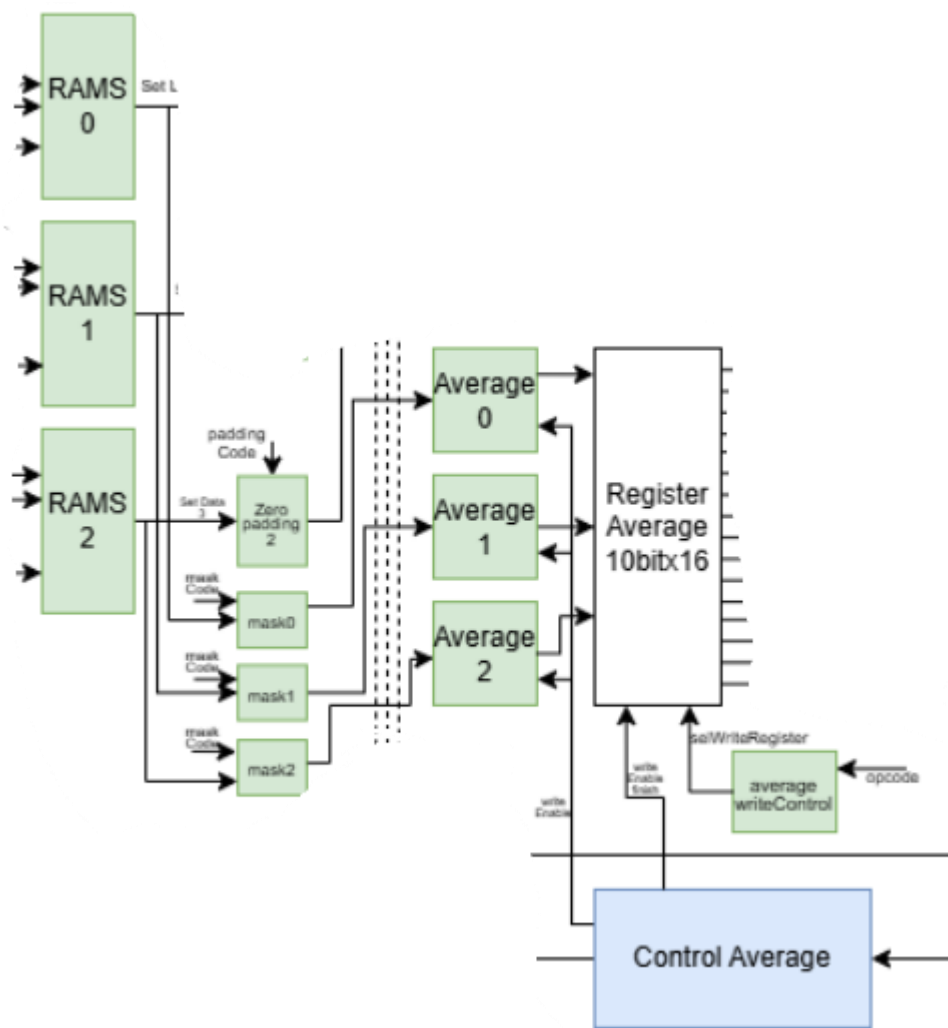
Port name	In/out	Width	Discription
I_opcode	Input	6	Mã lệnh xác định feature đang được tính trung bình
O_selWrite	Output	16	Một mặt nạ 16 bit dùng làm tín hiệu write enable cho register file

Module registerAverage được thiết kế để quản lý một bộ gồm 16 thanh ghi, mỗi thanh ghi có độ rộng 10 bit, dùng để lưu trữ dữ liệu và xử lý sau này. Module hoạt

đồng đồng bộ với tín hiệu xung nhịp và có tín hiệu reset tích cực mức thấp, giúp khởi tạo tất cả các thanh ghi về giá trị 0. Việc ghi dữ liệu được điều khiển thông qua tín hiệu cho phép ghi (`i_enableWrite`) và đầu vào chọn 16 bit (`i_selWrite`), xác định thanh ghi cụ thể nào sẽ được cập nhật. Khi ghi được cho phép, module sẽ ghi ba dữ liệu đầu vào 10 bit (`i_data0`, `i_data1`, hoặc `i_data2`) vào thanh ghi được chọn, dựa trên điều kiện khớp trong câu lệnh case. Module đã có phân xử lý việc nhập và lưu trữ dữ liệu.

Bởi vì average sẽ tính trung bình tích lũy theo từng 9 mẫu data trong khi một feature có 4096 data. Vì vậy ở lần cộng tích lũy cuối cùng ta chỉ còn 1 data thứ 4095 duy nhất. Vì vậy ta cần một mast để xóa đi những dữ liệu trên mà RAM cho ra khi cố gắng đọc 9 phần tử.

Như vậy kiến trúc average sẽ như sau:



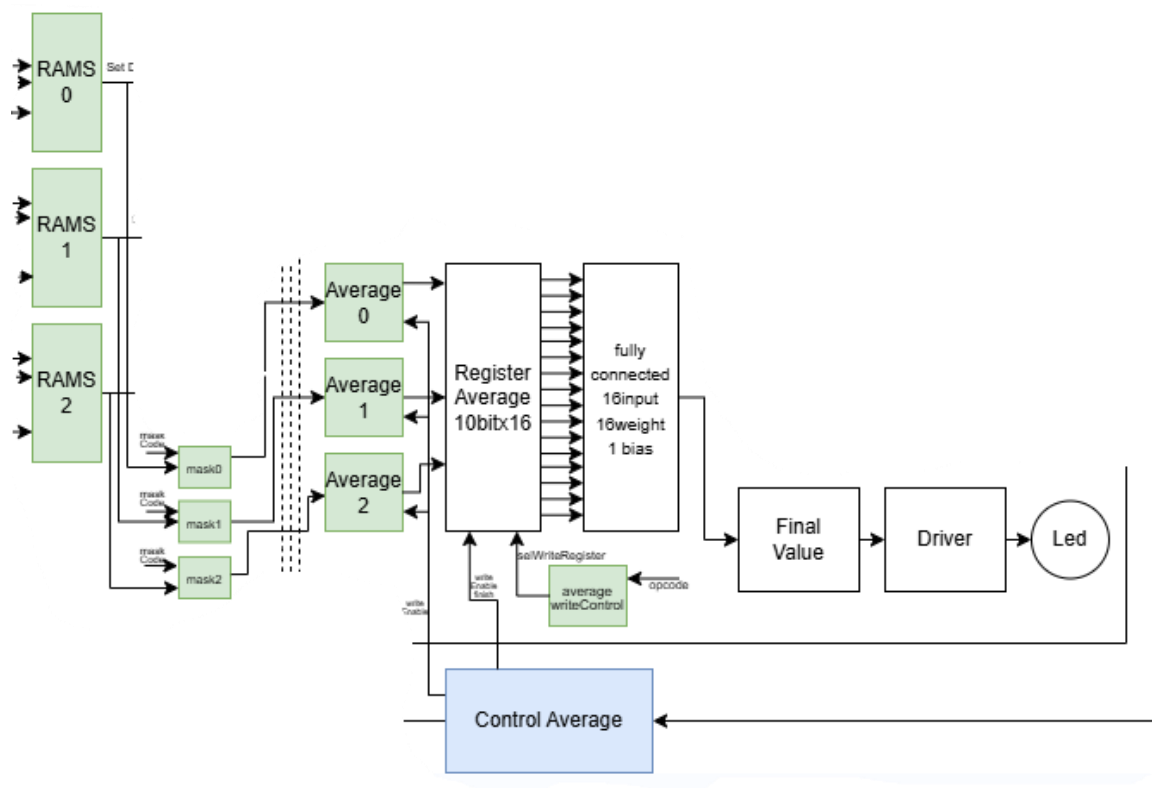
Hình 27: Kiến trúc khối thực hiện chức năng tính average

Tuy nhiên việc hoàn thành thực hiện tính toán trung bình vẫn chưa hoàn thành mô hình mobile net. Bước cuối của việc này là tính tích chập 16 giá trị trung bình này với 16 weight và 1 bias để được 1 giá trị output của mô hình. Và ta sẽ dùng giá trị output này để drive led nhằm hiển thị kết quả.

Module fullyConnected này triển khai một lớp mạng neural fully connected đơn giản, nhận vào 16 giá trị đầu vào (mỗi giá trị rộng 10 bit), nhân từng giá trị với 16 trọng số nội bộ, cộng chúng theo từng cặp, và cuối cùng cộng thêm giá trị bias để tính ra kết quả đầu ra. Các đầu vào được đóng gói vào một bus rộng 160 bit. Trong quá trình hoạt động, khi i_enable ở mức cao, module sẽ thực hiện phép tính: mỗi đầu vào được nhân với trọng số tương ứng (ví dụ: $in0 * w0$, $in1 * w1$, v.v...), sau đó các kết quả được cộng theo cặp (như $in0*w0 + in1*w1$, rồi $in2*w2 + in3*w3$, v.v...).

Cấu trúc nhóm này giúp giảm độ phức tạp của phép tính trong phần cứng. Kết quả cuối cùng, bao gồm cả bias, được lưu vào o_result sau mỗi chu kỳ xung nhịp.

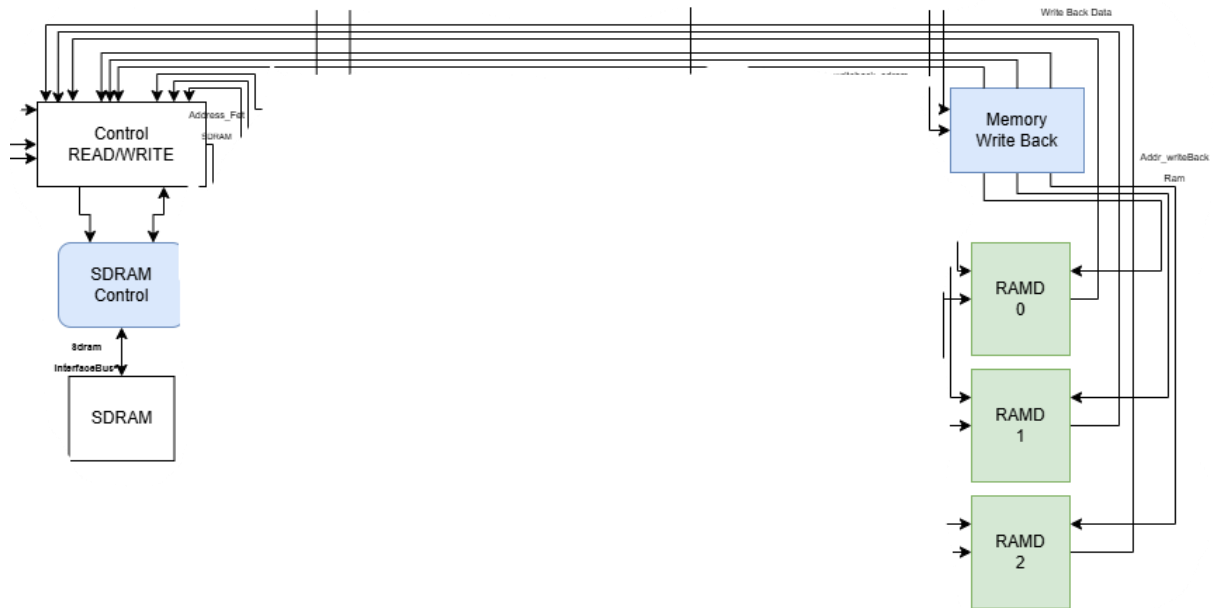
Như vậy kiến trúc của phần average và fully connected sẽ có dạng như sau:



Hình 28: Kiến trúc phần tính average bao gồm luôn phần fullyconnected và drive led

6.3.5 Write back destination ram to SDRAM

Mô-đun Verilog này được thiết kế để kiểm soát quá trình ghi dữ liệu lại từ RAM đích (destination ram) đến SDRAM ngoài. Nó đảm bảo rằng dữ liệu đã xử lý được lưu trữ trong destination RAM bên trong có thể được chuyển trở lại bộ nhớ SDRAM một cách hiệu quả và chính xác. Mô-đun này có khả năng quản lý việc tạo địa chỉ, xác thực dữ liệu và tín hiệu “hand shake” cần thiết để truyền dữ liệu an toàn. Nó hoạt động theo trình tự được kiểm soát, cho phép ghi dữ liệu vào SDRAM từng word một hoặc theo từng đợt. Ta buộc phải thực hiện việc write back mà không lưu lại kết quả ở destination ram là vì ta cần giải phóng không gian cho việc thực hiện convolution cho các feature tiếp theo. Và SDRAM trên DE0-Nano có dung lượng rất lớn lên đến 32MB nên ta không cần phải lo lắng về việc ghi lại kết quả vào SDRAM.



Hình 29: Kiến trúc khối chức năng write back

Ta có thể thấy trong khối chức năng write back. Phần quan trọng nhất là khối Memory write back control.

Bảng 20: Danh sách input, output của module write back control

Port name	In/out	Width	Discription
I_clk	In	1	Xung clock
I_reset	In	1	Tín hiệu reset (active low, asynchorus)
I_sdramReady	In	1	Tín hiệu báo sdram đang rảnh
I_baseAddr0	In	19	Base address point to first SDRAM region
I_baseAddr1	In	19	Base address point to second SDRAM region
I_baseAddr2	In	19	Base address point to 3th SDRAM region
I_start	In	1	Tín hiệu bắt đầu quá trình write back
O_addrToRam0	Out	12	Address to get data from RAM0
O_addrToRam1	Out	12	Address to get data from RAM1
O_addrToRam2	Out	12	Address to get data from RAM2
O_quickRam	Out	1	Tín hiệu chọn mode quick read of RAM
O_addrToSdram	Out	19	Address to SDRAM
O_wrSdram	Out	1	Tín hiệu xin phép ghi vào SDRAM
O_selData	Out	2	Chọn data để ghi vào SDRAM
O_finish	Out	1	Cờ báo hoàn thành quá trình write back

Bảng 21: Bảng chuyển trạng thái module write back control

Trạng thái hiện tại	Điều kiện	Trạng thái tiếp theo	O_finish	O_Quick Ram	O_selData	O_wr SDRAM
Idle	I_reset & i_start	setRam	0	0	0	0
setRam	Luôn luôn	setSdram0	0	1	0	0
setSdram0	Luôn luôn	waitDone0	0	0	0	1
waitDone0	I_sdramReady	setSdram1	0	0	0	0
setSdram1	Luôn luôn	waitDone1	0	0	1	1
waitDone1	I_sdramReady	setSdram2	0	0	1	0
setSdram2	Luôn luôn	waitDone2	0	0	2	1
waitDone2	I_sdramReady	Update	0	0	2	0
Update	addrLocal = 4095	Finish	0	0	0	0
Update	Không thỏa trường hợp trên	setRam	0	0	0	0

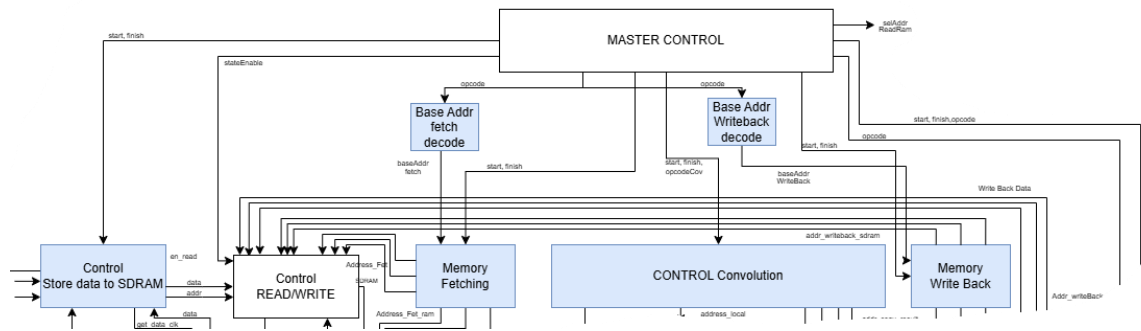
6.4 Module master control

Module Master Control đóng vai trò là bộ điều phối trung tâm cho một hệ thống nhiều module (Bao gồm những khối chính về fetching memory, store data camera to SDRAM, convolution, average compute, write back). Nó điều khiển kích hoạt tuần tự các module con chẳng hạn như ghi nhận dữ liệu từ camera, thực hiện phép tích chập (conv), tính trung bình (average, averageControl), và ghi kết quả về bộ nhớ (writeBackControl)—nhằm đảm bảo sự đồng bộ và luồng dữ liệu đúng đắn và tránh xung đột nhau khi sử dụng chung một loại tài nguyên ví dụ như SDRAM, source RAM, destination Ram,....

Thông qua việc giải mã opcode và quản lý các địa chỉ cơ sở (thông qua baseAddrFetDecode và baseAddrWriteBackDecode), module này định tuyến dữ liệu giữa RAM, SDRAM và các khối xử lý, đồng thời xử lý các ràng buộc về thời gian. Bộ điều khiển sử dụng máy trạng thái (FSM) để kích hoạt các thao tác như lấy ảnh từ camera, lọc tích chập, tính trung bình, và lưu trữ kết quả; đồng thời giám sát các tín hiệu handshake (ví dụ: i_sdramReady, i_validRam, o_finish) để tránh xung đột và đảm bảo tính hoàn tất của quá trình xử lý.

Giao diện điều khiển của khối master control lên các module tiến trình thành phần dựa trên kỹ thuật handshake rút gọn gồm tín hiệu bắt đầu (start) – hay tín hiệu

khởi động được cấp từ master control và tín hiệu hoàn thành (finish) – hay tín hiệu kết thúc tiến trình từ module chức năng thành phần.



Hình 30: Minh họa giao tiếp module master control và các module control tiến trình chức năng thành phần

Bảng 22: Danh sách input output của module master control

Port name	In/out	Width	Discription
I_clk	In	1	Xung clock
I_reset	In	1	Reset signal active low
I_finish_cam	In	1	Tín hiệu báo hoàn thành lưu frame từ camera
I_finish_fet	In	1	Tín hiệu báo hoàn thành fet
I_finish_conv	In	1	Tín hiệu báo hoàn thành conv
I_finish_writeBack	In	1	Tín hiệu báo hoàn thành write back
I_finish_ave	In	1	Tín hiệu báo hoàn thành tính trung bình
O_startCam	Out	1	Tín hiệu khởi động khối đọc camera lưu vào SDRAM
O_startFet	Out	1	Tín hiệu khởi động khối đọc Fet
O_startConvolution	Out	1	Tín hiệu khởi động khối convolution
O_startAve	Out	1	Tín hiệu khởi động khối tính trung bình AVE
O_startWriteBack	Out	1	Tín hiệu khởi động khối write back

O_wrActiveCam	Out	1	Tín hiệu ưu tiên luồng dữ liệu ghi vào SDRAM
O_opConv	Out	1	Tín hiệu điều khiển loại convolution 3 feature hoặc 1 feature
O_opcode	Out	6	Opcode tín hiệu xác định feature mục tiêu.

Bảng 23: Bảng chuyển trạng thái master control

Trạng thái hiện tại	Điều kiện	Trạng thái kế tiếp	Mô tả
Idle	Luôn luôn	startCam	Trạng thái nghỉ
startCam	Luôn luôn	waitdoneCam	Bậc chờ khởi động readCam
waitDoneCam	I_finish_cam	Fetch1	Chờ cam xong
Fetch1	Luôn luôn	waitDoneFet1	Bậc chờ khởi động fetch
waitdoneFet1	I_fnish_fet	Convolution0	Chờ fet xong
Convolution0	Luôn luôn	waitDoneConv0	Bậc chờ khởi động convolution 3 feature
waitDoneConv0	I_finish_conv	writeBack1	Chờ convolution 3 feature xong
Writeback1	Luôn luôn	waitDoneWriteback1	Bậc chờ writeback
waitDoneWriteBack1	I_finish_writeBack	Update1	Chờ write back xong
Update 1	Opcode = 15	Fet2	Cập nhật opcode kiểm tra chuyển sang tiến trình khác

Update 1	Opcode < 15	Convolution0	Trở lại thực hiện feature tiếp theo
Fet2	Luôn luôn	waitdoneFet2	Khởi động tiến trình fet
waitDoneFet2	I_finish_fet	Convolution1	Chờ hoàn thành fet
Convolution1	Luôn luôn	waitDoneConvolution1	Khởi động khối convolution 1 feature
waitdoneConvolution1	I_finish_conv	writeBack2	Chờ tiến trình convolution 1 feature hoàn thành
writeBack2	Luôn luôn	waitDoneWriteback2	Khởi động khối write back
waitDoneWriteBack2	I_finish_writeback	Update2	Chờ tiến trình writeback hoàn thành
Update2	Opcode = 31	Fet3	Cập nhật opcode kiểm tra chuyển đến tiến trình khác
Update2	Opcode < 31	Fet2	Cập nhật opcode chuyển về và thực hiện feature tiếp theo
Fet3	Luôn luôn	waitDoneFet3	Khởi động tiến trình fet
waitDoneFet3	I_finish_fet	Average	Chờ fet xong
Average	Luôn luôn	waitDoneAve	Khởi động tiến trình tính trung bình
waitDoneAve	I_finish_ave	Update3	Chờ hoàn thành tính average

Update3	Opcode = 37	Idle	Hoàn thành tính toán một frame
Update	Opcode < 37	Fet3	Thực hiện tiếp feature tiếp theo

Ảnh xạ địa chỉ của feature được tổ chức như sau:

ADDRESS IMAGE AND FEATURE ON SDRAM		
On SDRAM		
stt/feature_Index	start address	stopAddr
0	0	4095
1	4096	8191
2	8192	12287
3	12288	16383
4	16384	20479
5	20480	24575
6	24576	28671
7	28672	32767
8	32768	36863
9	36864	40959
10	40960	45055
11	45056	49151
12	49152	53247
13	53248	57343
14	57344	61439
15	61440	65535
16	65536	69631
17	69632	73727
18	73728	77823
19	77824	81919
20	81920	86015
21	86016	90111
22	90112	94207
23	94208	98303
24	98304	102399
25	102400	106495
26	106496	110591
27	110592	114687
28	114688	118783
29	118784	122879
30	122880	126975
31	126976	131071

Hình 31: Ảnh xạ địa chỉ feature 1

31	126976	131071		
32	131072	135167		
33	135168	139263		
34	139264	143359		
35	143360	147455		
36	147456	151551		
37	151552	155647		
38	155648	159743		
39	159744	163839		
40	163840	167935		
41	167936	172031		
42	172032	176127		
43	176128	180223		
44	180224	184319		
45	184320	188415		
46	188416	192511		
47	192512	196607		
48	196608	200703		
49	200704	204799		
50	204800	208895		
51	208896	212991		
52	212992	217087		
53	217088	221183		
54	221184	225279		
55	225280	229375		
56	229376	233471		
57	233472	237567		
58	237568	241663		
59	241664	245759		
60	245760	249855		
61	249856	253951		
62	253952	258047		
63	258048	262143		
64	262144	266239		
65	266240	270335		
66	270336	274431		

Hình 32: Ảnh xạ địa chỉ feature 2

Trong phần khối master control, có một khối phụ là khối weight generate. Mặc dù là khối phụ nhưng đây là khối rất quan trọng tạo ra các trọng số cho khối convolution hoạt động. Module này nhận vào giá trị opcode của khối master control (6bit) Tra ROM và xuất weight qua 3 ngõ ra 90 bit.

7. VERIFICATION

7.1 Kiểm tra khối camera read và lưu dữ liệu vào FIFO

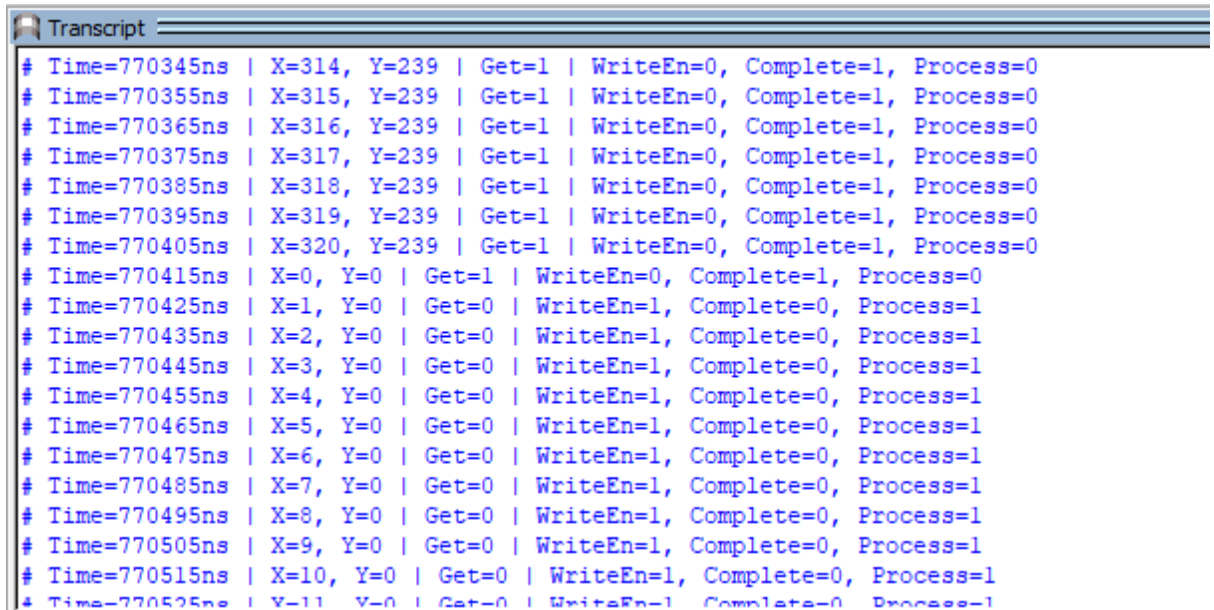
Khối camera read sẽ đưa ra giá trị pixel 16 bit, địa chỉ X_index, Y_index. Ta thực hiện testbench theo luận lý. Giá trị pixel bằng X_index của pixel - 1. Valid pixel bằng tín hiệu valid. Chỉ ghi và xét dữ liệu khi tín hiệu Valid = 1

PROBLEMS	OUTPUT	DEBUG CONSOLE	TERMINAL	PORTS
# Time = 27060 ns		X=0, Y=1	Pixel= 319 (Valid=0)	
# Time = 27100 ns		X=0, Y=1	Pixel= 319 (Valid=0)	
# Time = 27140 ns		X=0, Y=1	Pixel= 319 (Valid=0)	
# Time = 27180 ns		X=0, Y=1	Pixel= 319 (Valid=0)	
# Time = 27220 ns		X=0, Y=1	Pixel= 63 (Valid=0)	
# Time = 27260 ns		X=1, Y=1	Pixel= 0 (Valid=1)	
# Time = 27300 ns		X=1, Y=1	Pixel= 0 (Valid=0)	
# Time = 27340 ns		X=2, Y=1	Pixel= 1 (Valid=1)	
# Time = 27380 ns		X=2, Y=1	Pixel= 1 (Valid=0)	
# Time = 27420 ns		X=3, Y=1	Pixel= 2 (Valid=1)	
# Time = 27460 ns		X=3, Y=1	Pixel= 2 (Valid=0)	
# Time = 27500 ns		X=4, Y=1	Pixel= 3 (Valid=1)	
# Time = 27540 ns		X=4, Y=1	Pixel= 3 (Valid=0)	
# Time = 27580 ns		X=5, Y=1	Pixel= 4 (Valid=1)	
# Time = 27620 ns		X=5, Y=1	Pixel= 4 (Valid=0)	
# Time = 27660 ns		X=6, Y=1	Pixel= 5 (Valid=1)	
# Time = 27700 ns		X=6, Y=1	Pixel= 5 (Valid=0)	
# Time = 27740 ns		X=7, Y=1	Pixel= 6 (Valid=1)	
# Time = 27780 ns		X=7, Y=1	Pixel= 6 (Valid=0)	
# Time = 27820 ns		X=8, Y=1	Pixel= 7 (Valid=1)	
# Time = 27860 ns		X=8, Y=1	Pixel= 7 (Valid=0)	
# Time = 27900 ns		X=9, Y=1	Pixel= 8 (Valid=1)	
# Time = 27940 ns		X=9, Y=1	Pixel= 8 (Valid=0)	
# Time = 27980 ns		X=10, Y=1	Pixel= 9 (Valid=1)	
# Time = 28020 ns		X=10, Y=1	Pixel= 9 (Valid=0)	
# Time = 28060 ns		X=11, Y=1	Pixel= 10 (Valid=1)	

Hình 33: Chạy testbench cho khối camera read

Nhận xét: module hoạt động chính xác giá trị pixel valid và có giá trị bằng X_index-1.

Testbench khối control write to FIFO: Dữ liệu chỉ có thể ghi vào FIFO khi pixel nằm trong kích thước 64x64. Khi i_get = 1 và x_index = 0, y_index = 0 thì module sẽ tiến vào trạng thái process. Chỉ khi nào i_index và j_index thuộc trong kích thước 64x64 thì wrFIFO mới được phép bậc 1, ta sẽ dùng những ràng buộc này thực hiện kiểm tra hoạt động của khối.



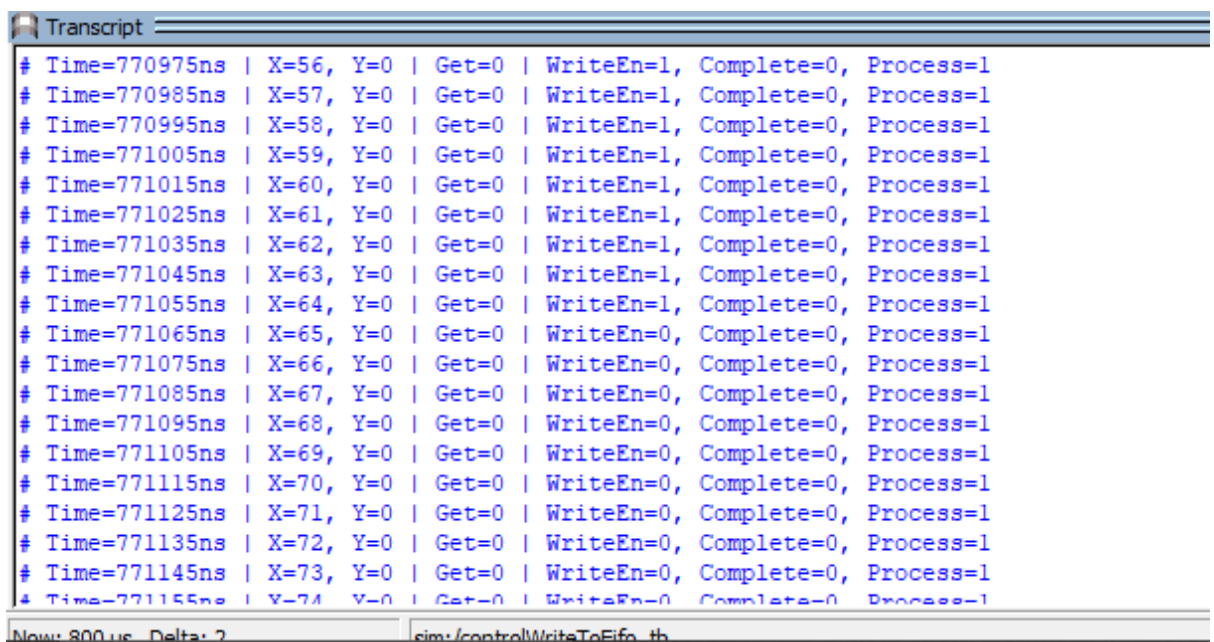
```

# Time=770345ns | X=314, Y=239 | Get=1 | WriteEn=0, Complete=1, Process=0
# Time=770355ns | X=315, Y=239 | Get=1 | WriteEn=0, Complete=1, Process=0
# Time=770365ns | X=316, Y=239 | Get=1 | WriteEn=0, Complete=1, Process=0
# Time=770375ns | X=317, Y=239 | Get=1 | WriteEn=0, Complete=1, Process=0
# Time=770385ns | X=318, Y=239 | Get=1 | WriteEn=0, Complete=1, Process=0
# Time=770395ns | X=319, Y=239 | Get=1 | WriteEn=0, Complete=1, Process=0
# Time=770405ns | X=320, Y=239 | Get=1 | WriteEn=0, Complete=1, Process=0
# Time=770415ns | X=0, Y=0 | Get=1 | WriteEn=0, Complete=1, Process=0
# Time=770425ns | X=1, Y=0 | Get=0 | WriteEn=1, Complete=0, Process=1
# Time=770435ns | X=2, Y=0 | Get=0 | WriteEn=1, Complete=0, Process=1
# Time=770445ns | X=3, Y=0 | Get=0 | WriteEn=1, Complete=0, Process=1
# Time=770455ns | X=4, Y=0 | Get=0 | WriteEn=1, Complete=0, Process=1
# Time=770465ns | X=5, Y=0 | Get=0 | WriteEn=1, Complete=0, Process=1
# Time=770475ns | X=6, Y=0 | Get=0 | WriteEn=1, Complete=0, Process=1
# Time=770485ns | X=7, Y=0 | Get=0 | WriteEn=1, Complete=0, Process=1
# Time=770495ns | X=8, Y=0 | Get=0 | WriteEn=1, Complete=0, Process=1
# Time=770505ns | X=9, Y=0 | Get=0 | WriteEn=1, Complete=0, Process=1
# Time=770515ns | X=10, Y=0 | Get=0 | WriteEn=1, Complete=0, Process=1
# Time=770525ns | X=11, Y=0 | Get=0 | WriteEn=1, Complete=0, Process=1

```

Hình 34: Testbench cho module control write to FIFO case 1

Nhận xét: Module hoạt động đúng trong case khởi động. Chỉ tiến vào process khi i_get bậc 1 và $x_index = 0$, $y_index = 0$. Và $writeENfifo$ bậc 1 khi pixel thuộc 64×64 .



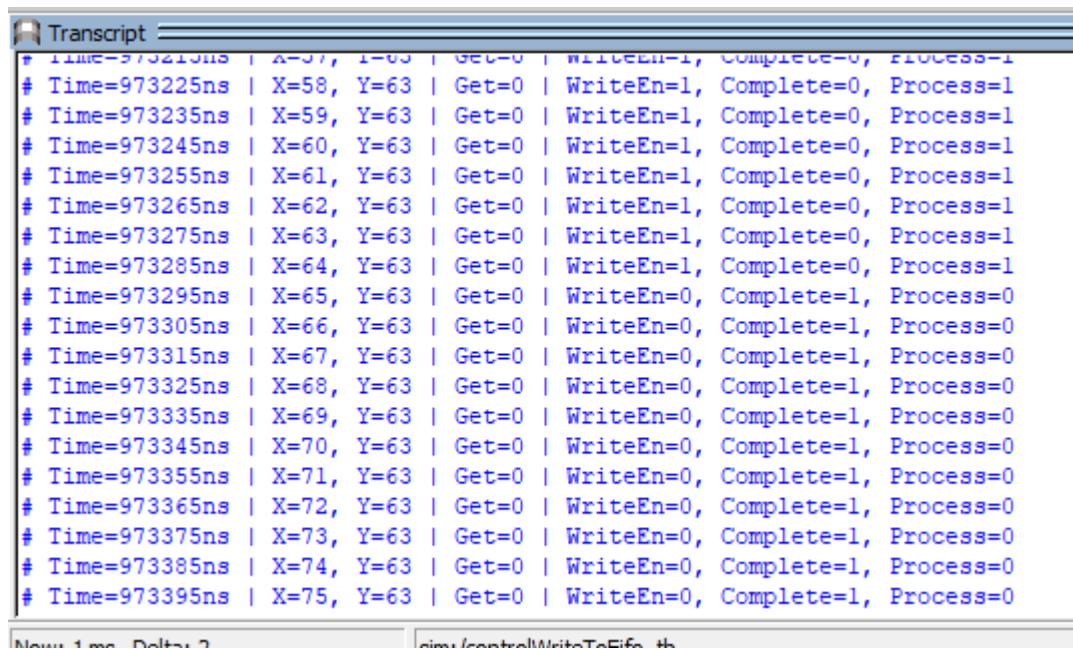
```

# Time=770975ns | X=56, Y=0 | Get=0 | WriteEn=1, Complete=0, Process=1
# Time=770985ns | X=57, Y=0 | Get=0 | WriteEn=1, Complete=0, Process=1
# Time=770995ns | X=58, Y=0 | Get=0 | WriteEn=1, Complete=0, Process=1
# Time=771005ns | X=59, Y=0 | Get=0 | WriteEn=1, Complete=0, Process=1
# Time=771015ns | X=60, Y=0 | Get=0 | WriteEn=1, Complete=0, Process=1
# Time=771025ns | X=61, Y=0 | Get=0 | WriteEn=1, Complete=0, Process=1
# Time=771035ns | X=62, Y=0 | Get=0 | WriteEn=1, Complete=0, Process=1
# Time=771045ns | X=63, Y=0 | Get=0 | WriteEn=1, Complete=0, Process=1
# Time=771055ns | X=64, Y=0 | Get=0 | WriteEn=1, Complete=0, Process=1
# Time=771065ns | X=65, Y=0 | Get=0 | WriteEn=0, Complete=0, Process=1
# Time=771075ns | X=66, Y=0 | Get=0 | WriteEn=0, Complete=0, Process=1
# Time=771085ns | X=67, Y=0 | Get=0 | WriteEn=0, Complete=0, Process=1
# Time=771095ns | X=68, Y=0 | Get=0 | WriteEn=0, Complete=0, Process=1
# Time=771105ns | X=69, Y=0 | Get=0 | WriteEn=0, Complete=0, Process=1
# Time=771115ns | X=70, Y=0 | Get=0 | WriteEn=0, Complete=0, Process=1
# Time=771125ns | X=71, Y=0 | Get=0 | WriteEn=0, Complete=0, Process=1
# Time=771135ns | X=72, Y=0 | Get=0 | WriteEn=0, Complete=0, Process=1
# Time=771145ns | X=73, Y=0 | Get=0 | WriteEn=0, Complete=0, Process=1
# Time=771155ns | X=74, Y=0 | Get=0 | WriteEn=0, Complete=0, Process=1

```

Hình 35: Testbench cho module control write to FIFO case2

Nhận xét: Module hoạt động đúng, Pixel x khi out khỏi 64×64 . WriteEn đã được tắt đúng như kỳ vọng.



```

Transcript
# Time=973210ns | X=57, Y=63 | Get=0 | WriteEn=1, Complete=0, Process=1
# Time=973225ns | X=58, Y=63 | Get=0 | WriteEn=1, Complete=0, Process=1
# Time=973235ns | X=59, Y=63 | Get=0 | WriteEn=1, Complete=0, Process=1
# Time=973245ns | X=60, Y=63 | Get=0 | WriteEn=1, Complete=0, Process=1
# Time=973255ns | X=61, Y=63 | Get=0 | WriteEn=1, Complete=0, Process=1
# Time=973265ns | X=62, Y=63 | Get=0 | WriteEn=1, Complete=0, Process=1
# Time=973275ns | X=63, Y=63 | Get=0 | WriteEn=1, Complete=0, Process=1
# Time=973285ns | X=64, Y=63 | Get=0 | WriteEn=1, Complete=0, Process=1
# Time=973295ns | X=65, Y=63 | Get=0 | WriteEn=0, Complete=1, Process=0
# Time=973305ns | X=66, Y=63 | Get=0 | WriteEn=0, Complete=1, Process=0
# Time=973315ns | X=67, Y=63 | Get=0 | WriteEn=0, Complete=1, Process=0
# Time=973325ns | X=68, Y=63 | Get=0 | WriteEn=0, Complete=1, Process=0
# Time=973335ns | X=69, Y=63 | Get=0 | WriteEn=0, Complete=1, Process=0
# Time=973345ns | X=70, Y=63 | Get=0 | WriteEn=0, Complete=1, Process=0
# Time=973355ns | X=71, Y=63 | Get=0 | WriteEn=0, Complete=1, Process=0
# Time=973365ns | X=72, Y=63 | Get=0 | WriteEn=0, Complete=1, Process=0
# Time=973375ns | X=73, Y=63 | Get=0 | WriteEn=0, Complete=1, Process=0
# Time=973385ns | X=74, Y=63 | Get=0 | WriteEn=0, Complete=1, Process=0
# Time=973395ns | X=75, Y=63 | Get=0 | WriteEn=0, Complete=1, Process=0

```

Hình 36: Testbench cho module control write to FIFO case 3

Nhận xét: Module hoạt động đúng. Khi y đạt 63 hàng cuối trong 64x64 và x bằng 64 thì module chuyển sang trạng thái complete, ngắt writeEn.

7.2 Testbench module fetching memory

Trong testbench module fetching memory. Ta sẽ thực hiện theo luận lý mô phỏng latency của SDRAM read operation. Cho đầu vào base address lần lượt là 0, 10000, 20000. Ta bắt tín hiệu rdSdram, wrRam để xuất hành vi hệ thống. Mô phỏng tín hiệu i_start để khởi động tiến trình, o_finish báo hiệu hoàn thành tiến trình.

```
# Time=655135ns | [READ SDRAM] | address SDRAM = 14094
# Time=655175ns | [Write RAM 1] | address = 4094
# Time=655185ns | [READ SDRAM] | address SDRAM = 24094
# Time=655225ns | [Write RAM 2] | address = 4094
# Time=655245ns | [READ SDRAM] | address SDRAM = 4095
# Time=655285ns | [Write RAM 0] | address = 4095
# Time=655295ns | [READ SDRAM] | address SDRAM = 14095
# Time=655335ns | [Write RAM 1] | address = 4095
# Time=655345ns | [READ SDRAM] | address SDRAM = 24095
# Time=655385ns | [Write RAM 2] | address = 4095
# Simulation completed.
# ** Note: $finish : ./fetchingControl_tb.v(77)
# Time: 655495 ns Iteration: 0 Instance: /fetchingControl_tb
# End time: 00:02:54 on Apr 25,2025, Elapsed time: 0:00:10
# Errors: 0, Warnings: 0
PS C:\Users\ntkhi\Desktop\DATN\Implement\sim>
```

Hình 37: Testbench cho module fetching memory.

Nhận xét: Module hoạt động chính xác theo thứ tự thực hiện kỳ vọng. Đọc sdram cho cho Ram đầu tiên, ghi vào RAM 0, Đọc SDRAM cho Ram thứ 2, ghi vào Ram2, đọc SDRAM cho ram 3, ghi vào RAM3. Sau khi hoàn thành pixel cuối (base address 4095) thì kết thúc tiến trình.

7.3 Testbench module write back

Thực hiện luận lý testbench giống như testbench của khối fetching memory.

```

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

# Time=572735ns | [write RAM 2 to SDRAM] | at address = 24090
# Time=572785ns | [read quick RAM0, 1, 2] | at address0 = 4091 | addr1 = 4091 | addr2 = 4091
# Time=572795ns | [write RAM 0 to SDRAM] | at address = 4091
# Time=572835ns | [write RAM 1 to SDRAM] | at address = 14091
# Time=572875ns | [write RAM 2 to SDRAM] | at address = 24091
# Time=572925ns | [read quick RAM0, 1, 2] | at address0 = 4092 | addr1 = 4092 | addr2 = 4092
# Time=572935ns | [write RAM 0 to SDRAM] | at address = 4092
# Time=572975ns | [write RAM 1 to SDRAM] | at address = 14092
# Time=573015ns | [write RAM 2 to SDRAM] | at address = 24092
# Time=573065ns | [read quick RAM0, 1, 2] | at address0 = 4093 | addr1 = 4093 | addr2 = 4093
# Time=573075ns | [write RAM 0 to SDRAM] | at address = 4093
# Time=573115ns | [write RAM 1 to SDRAM] | at address = 14093
# Time=573155ns | [write RAM 2 to SDRAM] | at address = 24093
# Time=573205ns | [read quick RAM0, 1, 2] | at address0 = 4094 | addr1 = 4094 | addr2 = 4094
# Time=573215ns | [write RAM 0 to SDRAM] | at address = 4094
# Time=573255ns | [write RAM 1 to SDRAM] | at address = 14094
# Time=573295ns | [write RAM 2 to SDRAM] | at address = 24094
# Time=573345ns | [read quick RAM0, 1, 2] | at address0 = 4095 | addr1 = 4095 | addr2 = 4095
# Time=573355ns | [write RAM 0 to SDRAM] | at address = 4095
# Time=573395ns | [write RAM 1 to SDRAM] | at address = 14095
# Time=573435ns | [write RAM 2 to SDRAM] | at address = 24095
# Simulation completed.
# ** Note: $finish : ./writeBackControl_tb.v(82)
# Time: 573485 ps Iteration: 0 Instance: /writeBackControl_tb
# End time: 00:41:34 on Apr 25,2025, Elapsed time: 0:00:09
# Errors: 0, Warnings: 0

```

Nhận xét: Module hoạt động đúng thứ tự kỳ vọng là đọc dữ liệu từ 3 ram trong cùng một chu kỳ với mode quick read. Thực hiện ghi vào sdrام cho ram 0 trước, tiếp theo là ram 1 và cuối cùng là Ram2. Pixel cuối cùng là 4095, sau khi hoàn thành pixel này module sẽ bậc cờ finish và hoàn thành tiến trình về lại trạng thái idle.

Các module khác ta thực hiện testbench tương tự.

8. SYNTHESIS

8.1 Tìm hiểu về kit DE0-Nano

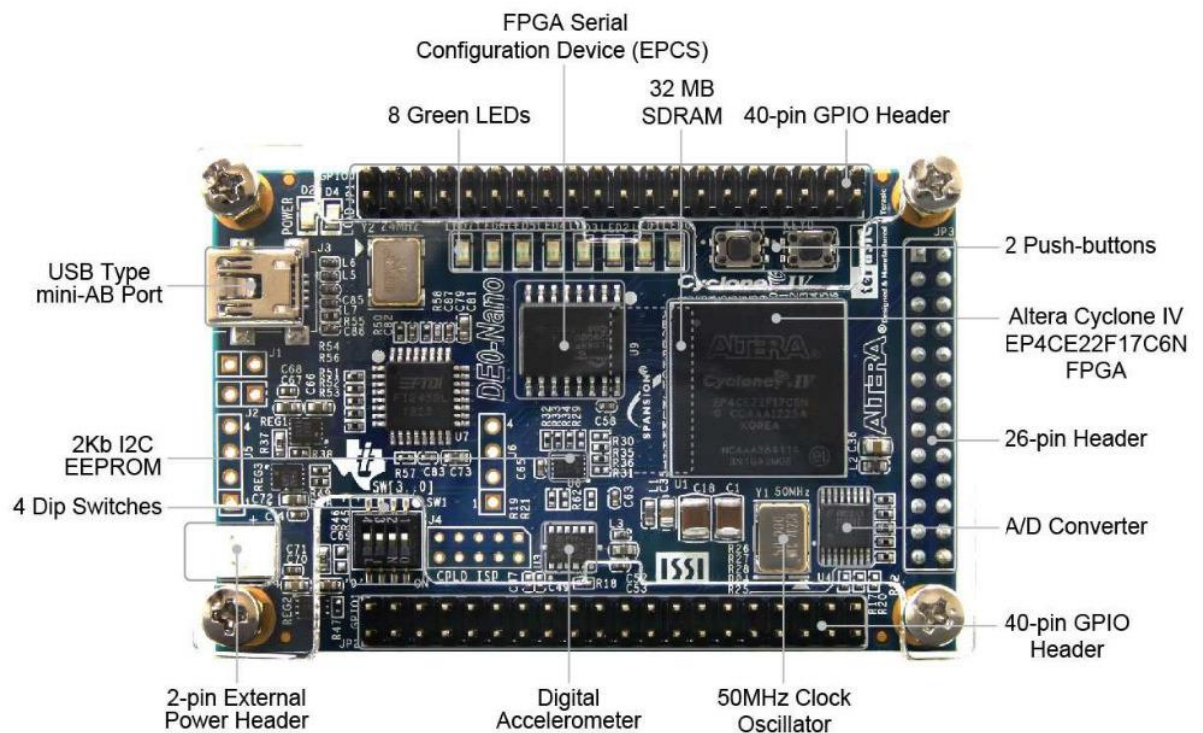
FPGA Chip được sử dụng trên kit DE0-Nano: Altera Cyclone IV

EP4CE22F17C6N

Các thông số quan trọng nhất liên quan trực tiếp đến project bao gồm:

Bảng 24: Thông số tài nguyên của Altera Cyclone IV EP4CE22F17C6N

Resource	Available
Logic Elements (LEs)	22,320
Logic Register	22.320
Memory Bits (Block RAM)	594,432 bit
Embedded Multipliers (18x18)	66
PLL (Phase-Locked Loops)	4
Maximum User I/O pin	150



Hình 38: Hình ảnh thực tế Kit DE0-nano

Như ta có thể thấy Kit có kích thước tương đối nhỏ so với một số loại KIT FPGA khác nên sẽ phù hợp với các ứng dụng mang tính di động, ưu tiên sự gọn nhẹ.

Project của chúng ta còn một thông số tài nguyên rất quan trọng không nằm trên CHIP FPGA. Đó chính là SDRAM – bộ nhớ lưu trữ toàn bộ feature của mô hình mobile net. Trên kit DE0-nano được kèm sẵn chip SDRAM 32Mb onboard có thông số như sau:

Bảng 25: Thông số kỹ thuật SDRAM trên kit DE0-Nano

Feature	Specification
Chip Part Number	ISSI IS42S16320F
Memory Type	Synchronous DRAM (SDRAM)
Memory Size	32Mbyte (256Mbit = 2Mx16x8)
Data Bus Width	16 bit
Address Bus Width	13 bit
Row Address	A0-A12 (13bit)
Column Address	A0-A8 (9bit)
Band Address	BA0-BA1: 4 banks
Max Clock Frequency	133Mhz
CAS Latency	2 or 3
Interface	Standard parallel SDRAM interface

8.2 Synthesis project

Bởi vì project vẫn chưa hoàn thiện đầy đủ. Vì vậy chỉ có thể synthesis từng module đã thiết kế và thực hiện testbench thành công để đánh giá mức độ khả thi khi thực hiện mô hình mobile net trên FPGA (kit DE0-nano)

Flow Summary	
<<Filter>>	
Flow Status	Flow Failed - Fri Apr 25 01:16:19 2025
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	mobileNet
Top-level Entity Name	ramControl
Family	Cyclone IV E
Device	EP4CE22F17C6
Timing Models	Final
Total logic elements	181 / 22,320 (< 1 %)
Total registers	101
Total pins	249 / 154 (162 %)
Total virtual pins	0
Total memory bits	40,960 / 608,256 (7 %)
Embedded Multiplier 9-bit elements	0 / 132 (0 %)
Total PLLs	0 / 4 (0 %)

Hình 39: Kết quả synthesis module RAM

Khối RAM là khối sử dụng nhiều tài nguyên bộ nhớ nhất nên ta cần quan tâm đến tổng memory bits tài nguyên đã sử dụng. Ta nhận thấy rằng 1 block RAM sẽ sử dụng khoảng 7%. Theo diagram đã thiết kế hệ thống cần dùng 6 RAM tương tự nên tài nguyên bộ nhớ tiêu thụ cho tổng bộ RAM là: $6 \times 7\% = 42\%$ Hoàn toàn khả thi về tài nguyên bộ nhớ.

Flow Summary	
<<Filter>>	
Flow Status	Flow Failed - Fri Apr 25 01:23:08 2025
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	mobileNet
Top-level Entity Name	conv
Family	Cyclone IV E
Device	EP4CE22F17C6
Timing Models	Final
Total logic elements	511 / 22,320 (2 %)
Total registers	0
Total pins	571 / 154 (371 %)
Total virtual pins	0
Total memory bits	0 / 608,256 (0 %)
Embedded Multiplier 9-bit elements	54 / 132 (41 %)
Total PLLs	0 / 4 (0 %)

Hình 40: Kết quả synthesis khối convolution

Khối convolution được synthesis khả thi và tiêu thụ tài nguyên tính toán Embedded Multiplier tương ứng 41%

Flow Summary	
<<Filter>>	
Flow Status	Flow Failed - Fri Apr 25 01:29:07 2025
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	mobileNet
Top-level Entity Name	registerAverage
Family	Cyclone IV E
Device	EP4CE22F17C6
Timing Models	Final
Total logic elements	179 / 22,320 (< 1 %)
Total registers	160
Total pins	209 / 154 (136 %)
Total virtual pins	0
Total memory bits	0 / 608,256 (0 %)
Embedded Multiplier 9-bit elements	0 / 132 (0 %)
Total PLLs	0 / 4 (0 %)

Hình 41: Kết quả synthesis khối aveage register

Module này không tiêu tốn thêm memory bit. Chỉ tiêu tốn thanh ghi với số lượng ít. Nên hoàn toàn khả thi.

Flow Summary	
<<Filter>>	
Flow Status	Flow Failed - Fri Apr 25 01:32:41 2025
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	mobileNet
Top-level Entity Name	weightGen
Family	Cyclone IV E
Device	EP4CE22F17C6
Timing Models	Final
Total logic elements	1,005 / 22,320 (5 %)
Total registers	0
Total pins	276 / 154 (179 %)
Total virtual pins	0
Total memory bits	0 / 608,256 (0 %)
Embedded Multiplier 9-bit elements	0 / 132 (0 %)
Total PLLs	0 / 4 (0 %)

Hình 42: Kết quả synthesis module weight generate

Module weight generate cũng hoàn toàn khả thi tuy có số lượng weight tương đối nhưng module không tiếp tốn thêm memory bit, Chỉ dùng logic elements khoảng 5%. Hoàn toàn khả thi trên FPGA.

Flow Summary	
<<Filter>>	
Flow Status	Successful - Fri Apr 25 01:38:14 2025
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	mobileNet
Top-level Entity Name	masterControl
Family	Cyclone IV E
Device	EP4CE22F17C6
Timing Models	Final
Total logic elements	39 / 22,320 (< 1 %)
Total registers	28
Total pins	20 / 154 (13 %)
Total virtual pins	0
Total memory bits	0 / 608,256 (0 %)
Embedded Multiplier 9-bit elements	0 / 132 (0 %)
Total PLLs	0 / 4 (0 %)

Hình 43: Kết quả synthesis module master control

Ta có thể nhận thấy rằng module master control có độ phức tạp cao nhất, nhiều state nhất. Tuy nhiên module cũng không tiêu tốn quá nhiều tài nguyên FPGA. Vì vậy việc thực hiện mô hình mobile net trên FPGA (DE0-Nano) là hoàn toàn khả thi. Tuy nhiên vì độ phức tạp của hệ thống nên đòi hỏi khả năng thiết kế tương đối vững.

9. KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

9.1 Kết luận

Tuy đồ án chưa thể hoàn thành hoàn chỉnh toàn bộ mô hình mobile net trên FPGA, tuy nhiên với việc hoàn thành tất cả những khối chức năng, khối điều khiển và có mô phỏng kiểm tra hoạt động của các module. Đã chứng minh được tính khả thi của việc thiết kế mô hình mobile net trên FPGA.

9.2 Hướng phát triển

Đồ án này còn rất nhiều thiếu sót và còn rất nhiều hướng để phát triển hơn nữa nếu có cơ hội. Như tiếp tục hoàn thành hoàn thiện hệ thống.

Tối ưu hóa hệ thống về tốc độ: Hiện tại có một số khối điều khiển đang được viết các state chưa được tối ưu, có thể điều chỉnh lại để giảm số lượng state, giúp giảm số chu kỳ xung clock cần thiết để hoàn thành một tiến trình.

Giao tiếp với SDRAM của đồ án đang chỉ sử dụng mode single word. Mode này đọc và ghi dữ liệu trên SDRAM rất chậm, phải chịu latency không pipeline. (cần nhiều chu kỳ xung clock để hoàn thành lệnh đọc hoặc ghi mà không thời gian đó không thể thực hiện lệnh khác). Trong standard parallel SDRAM interface có mode đọc, ghi burst cho phép thực hiện lệnh đọc và ghi liên tiếp các word cạnh nhau một cách liên tục giúp giảm chu kỳ xung clock phải chờ SDRAM hoàn thành lệnh đọc hoặc ghi.

10. TÀI LIỆU THAM KHẢO

- [1] Machine Learning Mastery, Using Depthwise Separable Convolutions in Tensorflow, <https://machinelearningmastery.com/using-depthwise-separable-convolutions-in-tensorflow/>
- [2] Ti ,DE0-NanoUserNanual(Terasis/Altera), <https://www.ti.com/lit/ug/tidu737/tidu737.pdf>
- [3] ISSI Corp, IS42R86400F/16320F, IS45R86400F/16320F IS42S86400F/16320F, IS45S86400F/16320F, https://www.issi.com/WW/pdf/42-45R-S_86400F-16320F.pdf
- [4] Medium, Pytorch Conv2d Weights Explained, <https://medium.com/data-science/pytorch-conv2d-weights-explained-ff7f68f652eb>.
- [5] TopDev, TensorFlow là gì? Tìm hiểu về TensoFlow từ A đến Z, <https://topdev.vn/blog/tensorflow-la-gi/>

11. PHỤ LỤC

Mã nguồn RTL design verilog.

Master control:

```
module masterControl(  
    input i_clk,  
    input i_reset,  
    input i_finish_cam,  
    input i_finish_fet,  
    input i_finish_conv,  
    input i_finish_writeBack,  
    input i_finish_ave,  
    output reg o_startCam,  
    output reg o_startFet,  
    output reg o_startConvolution,  
    output reg o_startAve,  
    output reg o_startWriteBack,  
    output reg o_wrActiveCam,  
    output reg o_opConv,  
    output [5:0] o_opcode  
);  
localparam [4:0] idle = 5'd0,  
                startCam = 5'd1,  
                waitDoneCam = 5'd2,  
                fet1 = 5'd3,  
                waitDoneFet1 = 5'd4,  
                convolution0 = 5'd5,  
                waitDoneConv0 = 5'd6,  
                writeBack1 = 5'd7,  
                waitDoneWriteBack1 = 5'd8,  
                update1 = 5'd9,  
                fet2 = 5'd10,  
                waitDoneFet2 = 5'd11,  
                convolution1 = 5'd12,  
                waitDoneConv1 = 5'd13,  
                writeBack2 = 5'd14,  
                waitDoneWriteBack2 = 5'd15,  
                update2 = 5'd16,  
                fet3 = 5'd17,  
                waitDoneFet3 = 5'd18,  
                average = 5'd19,  
                waitDoneAve = 5'd20,  
                update3 = 5'd21;  
reg [4:0] state_q, state_d;
```

```
reg [5:0] opcode_q, opcode_d;

always @(posedge i_clk or negedge i_reset) begin
    if(!i_reset) begin
        state_q <= idle;
        opcode_q <= 6'd0;
    end else begin
        state_q <= state_d;
        opcode_q <= opcode_d;
    end
end

always @(*) begin
    case (state_q)
        idle: begin
            state_d = startCam;
            o_startCam = 0;
            o_startFet = 0;
            o_startConvolution = 0;
            o_startAve = 0;
            o_startWriteBack = 0;
            o_wrActiveCam = 0;
            o_opConv = 0;
            opcode_d = 6'd0;
        end
        startCam: begin
            state_d = waitDoneCam;
            o_startCam = 1;
            o_startFet = 0;
            o_startConvolution = 0;
            o_startAve = 0;
            o_startWriteBack = 0;
            o_wrActiveCam = 0;
            o_opConv = 0;
            opcode_d = opcode_q;
        end
        waitDoneCam: begin
            if(i_finish_cam) begin
                state_d = fet1;
            end else begin
                state_d = waitDoneCam;
            end
            o_startCam = 0;
            o_startFet = 0;
            o_startConvolution = 0;
            o_startAve = 0;
        end
    end
end
```

```
    o_startWriteBack = 0;
    o_wrActiveCam = 0;
    o_opConv = 0;
    opcode_d = opcode_q;
end
fet1: begin
    state_d = waitDoneFet1;
    o_startCam = 0;
    o_startFet = 1;
    o_startConvolution = 0;
    o_startAve = 0;
    o_startWriteBack = 0;
    o_wrActiveCam = 0;
    o_opConv = 0;
    opcode_d = opcode_q;
end
waitDoneFet1: begin
    if(i_finish_fet) begin
        state_d = convolution0;
    end else begin
        state_d = waitDoneFet1;
    end
    o_startCam = 0;
    o_startFet = 0;
    o_startConvolution = 0;
    o_startAve = 0;
    o_startWriteBack = 0;
    o_wrActiveCam = 0;
    o_opConv = 0;
    opcode_d = opcode_q;
end
convolution0: begin
    state_d = waitDoneConv0;
    o_startCam = 0;
    o_startFet = 0;
    o_startConvolution = 1;
    o_startAve = 0;
    o_startWriteBack = 0;
    o_wrActiveCam = 0;
    o_opConv = 0;
    opcode_d = opcode_q;
end
waitDoneConv0: begin
    if(i_finish_conv) begin
        state_d = writeBack1;
    end else begin
```

```
        state_d = waitDoneConv0;
    end
    o_startCam = 0;
    o_startFet = 0;
    o_startConvolution = 0;
    o_startAve = 0;
    o_startWriteBack = 0;
    o_wrActiveCam = 0;
    o_opConv = 0;
    opcode_d = opcode_q;
end
writeBack1: begin
    state_d = waitDoneWriteBack1;
    o_startCam = 0;
    o_startFet = 0;
    o_startConvolution = 0;
    o_startAve = 0;
    o_startWriteBack = 1;
    o_wrActiveCam = 0;
    o_opConv = 0;
    opcode_d = opcode_q;
end
waitDoneWriteBack1: begin
    if(i_finish_writeBack) begin
        state_d = update1;
    end else begin
        state_d = waitDoneWriteBack1;
    end
    o_startCam = 0;
    o_startFet = 0;
    o_startConvolution = 0;
    o_startAve = 0;
    o_startWriteBack = 0;
    o_wrActiveCam = 0;
    o_opConv = 0;
    opcode_d = opcode_q;
end
update1: begin
    if(opcode_q == 6'd15) begin
        state_d = fet2;
    end else begin
        state_d = convolution0;
    end
    o_startCam = 0;
    o_startFet = 0;
    o_startConvolution = 0;
```

```
    o_startAve = 0;
    o_startWriteBack = 0;
    o_wrActiveCam = 0;
    o_opConv = 0;
    opcode_d = opcode_q + 6'd1;
end
fet2: begin
    state_d = waitDoneFet2;
    o_startCam = 0;
    o_startFet = 1;
    o_startConvolution = 0;
    o_startAve = 0;
    o_startWriteBack = 0;
    o_wrActiveCam = 0;
    o_opConv = 1;
    opcode_d = opcode_q;
end
waitDoneFet2: begin
    if(i_finish_fet) begin
        state_d = convolution1;
    end else begin
        state_d = waitDoneFet2;
    end
    o_startCam = 0;
    o_startFet = 0;
    o_startConvolution = 0;
    o_startAve = 0;
    o_startWriteBack = 0;
    o_wrActiveCam = 0;
    o_opConv = 1;
    opcode_d = opcode_q;
end
convolution1: begin
    state_d = waitDoneConv1;
    o_startCam = 0;
    o_startFet = 0;
    o_startConvolution = 1;
    o_startAve = 0;
    o_startWriteBack = 0;
    o_wrActiveCam = 0;
    o_opConv = 1;
    opcode_d = opcode_q;
end
waitDoneConv1: begin
    if(i_finish_conv) begin
        state_d = writeBack2;
```



```
    end else begin
        state_d = waitDoneConv1;
    end
    o_startCam = 0;
    o_startFet = 0;
    o_startConvolution = 0;
    o_startAve = 0;
    o_startWriteBack = 0;
    o_wrActiveCam = 0;
    o_opConv = 1;
    opcode_d = opcode_q;
end
writeBack2: begin
    state_d = waitDoneWriteBack2;
    o_startCam = 0;
    o_startFet = 0;
    o_startConvolution = 0;
    o_startAve = 0;
    o_startWriteBack = 1;
    o_wrActiveCam = 0;
    o_opConv = 1;
    opcode_d = opcode_q;
end
waitDoneWriteBack2: begin
    if(i_finish_writeBack) begin
        state_d = update2;
    end else begin
        state_d = waitDoneWriteBack2;
    end
    o_startCam = 0;
    o_startFet = 0;
    o_startConvolution = 0;
    o_startAve = 0;
    o_startWriteBack = 0;
    o_wrActiveCam = 0;
    o_opConv = 1;
    opcode_d = opcode_q;
end
update2: begin
    if(opcode_q == 6'd31) begin
        state_d = fet3;
    end else begin
        state_d = fet2;
    end
    o_startCam = 0;
    o_startFet = 0;
```

```
    o_startConvolution = 0;
    o_startAve = 0;
    o_startWriteBack = 0;
    o_wrActiveCam = 0;
    o_opConv = 1;
    opcode_d = opcode_q + 6'd1;
end
fet3: begin
    state_d = waitDoneFet3;
    o_startCam = 0;
    o_startFet = 1;
    o_startConvolution = 0;
    o_startAve = 0;
    o_startWriteBack = 0;
    o_wrActiveCam = 0;
    o_opConv = 0;
    opcode_d = opcode_q;
end
waitDoneFet3: begin
    if(i_finish_fet) begin
        state_d = average;
    end else begin
        state_d = waitDoneFet3;
    end
    o_startCam = 0;
    o_startFet = 0;
    o_startConvolution = 0;
    o_startAve = 0;
    o_startWriteBack = 0;
    o_wrActiveCam = 0;
    o_opConv = 0;
    opcode_d = opcode_q;
end
average: begin
    state_d = waitDoneAve;
    o_startCam = 0;
    o_startFet = 0;
    o_startConvolution = 0;
    o_startAve = 1;
    o_startWriteBack = 0;
    o_wrActiveCam = 0;
    o_opConv = 0;
    opcode_d = opcode_q;
end
waitDoneAve: begin
    if(i_finish_ave) begin
```

```

        state_d = update3;
    end else begin
        state_d = waitDoneAve;
    end
    o_startCam = 0;
    o_startFet = 0;
    o_startConvolution = 0;
    o_startAve = 0;
    o_startWriteBack = 0;
    o_wrActiveCam = 0;
    o_opConv = 0;
    opcode_d = opcode_q;
end
update3: begin
    if(opcode_q == 6'd37) begin
        state_d = idle;
    end else begin
        state_d = fet3;
    end
    o_startCam = 0;
    o_startFet = 0;
    o_startConvolution = 0;
    o_startAve = 0;
    o_startWriteBack = 0;
    o_wrActiveCam = 0;
    o_opConv = 0;
    opcode_d = opcode_q + 6'd1;
end
default: begin
    state_d = idle;
    o_startCam = 0;
    o_startFet = 0;
    o_startConvolution = 0;
    o_startAve = 0;
    o_startWriteBack = 0;
    o_wrActiveCam = 0;
    o_opConv = 0;
    opcode_d = 6'd0;
end
endcase
end
assign o_opcode = opcode_q;
endmodule

```

Convolution Block:

```
module conv(
    input [89:0] i_busData0,
    input [89:0] i_busData1,
    input [89:0] i_busData2,
    input [89:0] i_busWeight0,
    input [89:0] i_busWeight1,
    input [89:0] i_busWeight2,
    input      i_opcode,
    output [9:0] o_data0,
    output [9:0] o_data1,
    output [9:0] o_data2
);
//internal wire connect result of coreConv block
wire [9:0] result [2:0];
//architechture
coreConv coreBlock0(.i_data(i_busData0), .i_weight(i_busWeight0),
.o_data(result[0]));
coreConv coreBlock1(.i_data(i_busData1), .i_weight(i_busWeight1),
.o_data(result[1]));
coreConv coreBlock2(.i_data(i_busData2), .i_weight(i_busWeight2),
.o_data(result[2]));
//plus to handel CONV operation
wire [9:0] resultConv;
plus3para plus3paraBlock(.i_data0(result[0]), .i_data1(result[1]),
.i_data2(result[2]), .o_data(resultConv));
//assign output
mux2to1 mux2to1Block(.i_dataA(resultConv), .i_dataB(result[0]),
.i_sel(i_opcode), .o_data(o_data0));
assign o_data1 = result[1];
assign o_data2 = result[2];
endmodule

module plus3para(
    input [9:0] i_data0,
    input [9:0] i_data1,
    input [9:0] i_data2,
    output [9:0] o_data
);
//for high performance
//state 0
wire [9:0] data01;
assign data01 = i_data1 + i_data0;
//final state
assign o_data = data01 + i_data2;
endmodule
```

StoreCamControl:

```
module storeCamControl(
    input        i_clk,
    input        i_reset,
    input        i_start,
    input [9:0]   i_remainOnFifo,
    input        i_process,
    input        i_sdramReady,
    input        i_complete, //complete signal of controlWriteFifo
    input [15:0]  i_dataFifo,
    output reg    o_get,
    output reg    o_EnReadFifo,
    output reg    o_RdClkFifo,
    output [15:0] o_dataSdram,
    output reg [18:0] o_addressToSdram,
    output reg    o_wrSdram,
    output reg    o_finish
);

localparam [3:0] idle = 4'd0,
                start = 4'd1,
                suspend = 4'd2,
                readFifo0 = 4'd3,
                readFifo1 = 4'd4,
                setRed = 4'd5,
                wait0 = 4'd6,
                setGreen = 4'd7,
                wait1 = 4'd8,
                setBlue = 4'd9,
                wait2 = 4'd10,
                update = 4'd11,
                finish = 4'd12;

reg [3:0] state_q, state_d;
reg [18:0] addrLocal_q, addrLocal_d;
reg [7:0] dataToSdram8;

always @(posedge i_clk or negedge i_reset) begin //sequential circuit update
    state, addr local
        if(!i_reset) begin
            state_q <= idle;
            addrLocal_q <= 19'd0;
        end else begin
            state_q <= state_d;
            addrLocal_q <= addrLocal_d;
        end
    end
end
```

```
always @(*) begin
  case (state_q)
    idle: begin
      if(i_start & i_reset) begin
        state_d = start;
      end else begin
        state_d = idle;
      end
      o_get = 0;
      o_EnReadFifo = 0;
      o_RdClkFifo = 0;
      dataToSdram8 = 8'd0;
      o_addressToSdram = 19'd0;
      o_wrSdram = 0;
      o_finish = 0;
      addrLocal_d = 19'd0;
    end
    start: begin
      if(i_process) begin
        state_d = suspend;
      end else begin
        state_d = start;
      end
      o_get = 1;
      o_EnReadFifo = 0;
      o_RdClkFifo = 0;
      dataToSdram8 = 8'd0;
      o_addressToSdram = 19'd0;
      o_wrSdram = 0;
      o_finish = 0;
      addrLocal_d = 19'd0;
    end
    suspend: begin
      if((i_remainOnFifo > 10'd16) | (i_complete)) begin
        state_d = readFifo0;
      end else begin
        state_d = suspend;
      end
      o_get = 0;
      o_EnReadFifo = 1;
      o_RdClkFifo = 0;
      dataToSdram8 = 8'd0;
      o_addressToSdram = 19'd0;
      o_wrSdram = 0;
      o_finish = 0;
    end
  end case
end
```

```
        addrLocal_d = addrLocal_q;
    end
    readFifo0: begin
        state_d = readFifo1;
        o_get = 0;
        o_EnReadFifo = 1;
        o_RdClkFifo = 1;
        dataToSdram8 = 8'd0;
        o_addressToSdram = 19'd0;
        o_wrSdram = 0;
        o_finish = 0;
        addrLocal_d = addrLocal_q;
    end
    readFifo1: begin
        state_d = setRed;
        o_get = 0;
        o_EnReadFifo = 1;
        o_RdClkFifo = 0;
        dataToSdram8 = 8'd0;
        o_addressToSdram = 19'd0;
        o_wrSdram = 0;
        o_finish = 0;
        addrLocal_d = addrLocal_q;
    end
    setRed: begin
        state_d = wait0;
        o_get = 0;
        o_EnReadFifo = 0;
        o_RdClkFifo = 0;
        dataToSdram8 = {2'd0,i_dataFifo[15:11], 1'd0};
        o_addressToSdram = addrLocal_q;
        o_wrSdram = 1;
        o_finish = 0;
        addrLocal_d = addrLocal_q;
    end
    end
    wait0: begin
        if(i_sdramReady) begin
            state_d = setGreen;
        end else begin
            state_d = wait0;
        end
        end
        o_get = 0;
        o_EnReadFifo = 0;
        o_RdClkFifo = 0;
        dataToSdram8 = {2'd0,i_dataFifo[15:11], 1'd0};
        o_addressToSdram = addrLocal_q;
```

```
        o_wrSdram = 0;
        o_finish = 0;
        addrLocal_d = addrLocal_q;
    end
    setGreen: begin
        state_d = wait1;
        o_get = 0;
        o_EnReadFifo = 0;
        o_RdClkFifo = 0;
        dataToSdram8 = {2'd0,i_dataFifo[10:5]};
        o_addressToSdram = addrLocal_q + 19'd4096; //sdram region 2 store
    feature Green
        o_wrSdram = 1;
        o_finish = 0;
        addrLocal_d = addrLocal_q;
    end
    wait1: begin
        if(i_sdramReady) begin
            state_d = setBlue;
        end else begin
            state_d = wait1;
        end
        o_get = 0;
        o_EnReadFifo = 0;
        o_RdClkFifo = 0;
        dataToSdram8 = {2'd0,i_dataFifo[10:5]};
        o_addressToSdram = addrLocal_q + 19'd4096;
        o_wrSdram = 0;
        o_finish = 0;
        addrLocal_d = addrLocal_q;
    end
    setBlue: begin
        state_d = wait2;
        o_get = 0;
        o_EnReadFifo = 0;
        o_RdClkFifo = 0;
        dataToSdram8 = {2'd0, i_dataFifo[4:0], 1'd0};
        o_addressToSdram = addrLocal_q + 19'd8192; //sdram region 2 store
    feature Green
        o_wrSdram = 1;
        o_finish = 0;
        addrLocal_d = addrLocal_q;
    end
    wait2: begin
        if(i_sdramReady) begin
            state_d = update;
```



```
        end else begin
            state_d = wait2;
        end
        o_get = 0;
        o_EnReadFifo = 0;
        o_RdClkFifo = 0;
        dataToSdram8 = {2'd0, i_dataFifo[4:0], 1'd0};
        o_addressToSdram = addrLocal_q + 19'd8192;
        o_wrSdram = 0;
        o_finish = 0;
        addrLocal_d = addrLocal_q;
    end
    update: begin
        if((~i_complete) | (i_remainOnFifo > 10'd0)) begin
            state_d = suspend;
        end else begin
            state_d = finish;
        end
        o_get = 0;
        o_EnReadFifo = 0;
        o_RdClkFifo = 0;
        dataToSdram8 = 8'd0;
        o_addressToSdram = 19'd0;
        o_wrSdram = 0;
        o_finish = 0;
        addrLocal_d = addrLocal_q + 19'd1; //update local address
    end
    finish: begin
        state_d = idle;
        o_get = 0;
        o_EnReadFifo = 0;
        o_RdClkFifo = 0;
        dataToSdram8 = 8'd0;
        o_addressToSdram = 19'd0;
        o_wrSdram = 0;
        o_finish = 1;
        addrLocal_d = 19'd0;
    end
    default: begin
        state_d = idle;
        o_get = 0;
        o_EnReadFifo = 0;
        o_RdClkFifo = 0;
        dataToSdram8 = 8'd0;
        o_addressToSdram = 19'd0;
        o_wrSdram = 0;
```

```

        o_finish = 0;
        addrLocal_d = 19'd0;
    end
endcase
end

assign o_dataSdram = {8'd0, dataToSdram8}; //assign to 16 bit for SDRAM
(Extend unsigned)

endmodule

```

Phần mã nguồn còn lại được lưu trữ tại: <https://github.com/nttoan-khiem/MobileNetOnFPGA>

Sơ đồ hệ thống đầy đủ:

