

ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN
KHOA KỸ THUẬT MÁY TÍNH

TRẦN MINH TUẤN
DƯƠNG THANH TÙNG

KHÓA LUẬN TỐT NGHIỆP

ỨNG DỤNG THUẬT TOÁN PHÁT HIỆN ĐỐI TƯỢNG TRÊN
SOC-FPGA SỬ DỤNG OPENCL

Application Of Object Detection Algorithm On Soc-Fpga Using
Opencl

KỸ SƯ KỸ THUẬT MÁY TÍNH

TP. HỒ CHÍ MINH, 2022

ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN
KHOA KỸ THUẬT MÁY TÍNH

TRẦN MINH TUẤN - 18521608

DƯƠNG THANH TÙNG - 18521613

KHÓA LUẬN TỐT NGHIỆP

**ỨNG DỤNG THUẬT TOÁN PHÁT HIỆN ĐỐI TƯỢNG TRÊN
SOC-FPGA SỬ DỤNG OPENCL**

**Application Of Object Detection Algorithm On Soc-Fpga Using
Opencl**

KỸ SƯ KỸ THUẬT MÁY TÍNH

GIẢNG VIÊN HƯỚNG DẪN

ThS. NGÔ HIẾU TRƯỜNG

TP. HỒ CHÍ MINH, 2022

THÔNG TIN HỘI ĐỒNG CHẤM KHÓA LUẬN TỐT NGHIỆP

Hội đồng chấm khóa luận tốt nghiệp, thành lập theo Quyết định số 526/QĐ-ĐHCNTT ngày 19 tháng 07 năm 2022 của Hiệu trưởng Trường Đại học Công nghệ Thông tin.

LỜI CẢM ƠN

Lời đầu tiên chúng em xin được bày tỏ lòng biết ơn tới đội ngũ giảng viên, nhân viên của trường Đại học Công nghệ Thông tin - ĐHQG TP.HCM nói chung và khoa Kỹ thuật Máy tính nói riêng đã tận tình giảng dạy, giúp đỡ, tạo mọi điều kiện cho chúng em trong quá trình học tập và nghiên cứu tại trường để chúng em có thể đủ điều kiện để thực hiện khóa luận tốt nghiệp.

Đồng thời chúng em xin gửi lời cảm ơn chân thành tới giảng viên Thạc sĩ Ngô Hiếu Trường, người đã luôn theo sát tụi em trong thời gian nghiên cứu và thực hiện khóa luận tốt nghiệp để hỗ trợ và hướng dẫn kịp thời để tụi em có thể đạt được kết quả ngày hôm nay.

Chúc quý thầy cô khoa Kỹ thuật Máy tính – trường Đại học Công nghệ Thông tin luôn đi đầu trong công tác giáo dục đại học ở trường và là người định hướng tận tâm cho nhiều thế hệ sinh viên. Cảm ơn thầy cô đã luôn đồng hành và dẫn dắt sinh viên chúng em từ những ngày mới bước chân vào giảng đường đại học cho đến nay. Đó là cả một hành trình dài và nhiều bài học quý giá mà chúng em học được.

Cuối cùng, chúng em xin một lần nữa gửi quý thầy cô lời biết ơn chân thành nơi chúng em và đặc biệt tới thầy Trường (Ths. Ngô Hiếu Trường). Chúc thầy gặt hái nhiều thành công hơn nữa trong công tác giảng dạy tại khoa và gia đình nhiều sức khỏe!

TP.Hồ Chí Minh, tháng 06 năm 2022

Sinh viên thực hiện

Trần Minh Tuấn, Dương Thanh Tùng

MỤC LỤC

Chương 1.	TỔNG QUAN ĐỀ TÀI	2
1.1.	Tổng quan	2
1.2.	Lý do thực hiện đề tài	2
1.3.	Mục tiêu thực hiện	3
1.4.	Nội dung chính	3
1.5.	Giới hạn đề tài	3
Chương 2.	CƠ SỞ LÝ THUYẾT	4
2.1.	Boot linux SD Card image trên board DE1-SoC.....	4
2.2.	Số dấu chấm động (Floating point)	5
2.2.1.	Khái niệm số dấu chấm động	5
2.2.2.	Chuẩn half-precision floating-point ieee-754-2008	5
2.3.	Convolutional Neural Network – Mạng Noron tích chập	7
2.4.	Convolutional Layer – Lớp tích chập.....	8
2.5.	Pooling Layer – Lớp tổng hợp.....	9
2.6.	Fully Connected Layer – Lớp kết nối đầy đủ.....	10
2.7.	Activation function - Các hàm kích hoạt.....	11
2.7.1.	ReLU	11
2.7.2.	Leaky ReLU	12
2.8.	YOLOv1	12
2.8.1.	Ý tưởng.....	13
2.8.2.	Hàm tính IOU	15
2.8.3.	Nhược điểm của YOLOv1	15
2.8.4.	Tiny-YOLO	16

2.9.	YOLOv2	17
2.9.1.	Anchor box	17
2.9.2.	Dự đoán vị trí trực tiếp	18
2.9.3.	Batch Normalization.....	19
2.9.4.	Bộ phân loại độ phân giải cao.	20
2.9.5.	Fine-Grained Features	20
2.9.6.	Multi-Scale Training	21
2.9.7.	Light-weight backbone.....	21
2.10.	Field Programmable Gate Array (FPGA)	22
2.11.	Tổng quan về board DE1-SoC	23
2.12.	Intel® FPGA SDK for OpenCL™	25
2.12.1.	Giới thiệu.....	25
2.12.2.	Quy trình programing FPGA sử dụng Intel® FPGA SDK for OpenCL™	26
Chương 3.	XÂY DỰNG HỆ ĐIỀU HÀNH TÙY BIẾN LINUX CHO BOARD DE1-SOC	28
3.1.	Tổng quan về hệ điều hành tùy biến linux cho board DE1-SoC	28
3.2.	Quy trình xây dựng linux SD Card image cho board DE1-SoC	28
3.2.1.	Các công cụ cần thiết.....	28
3.2.2.	Tạo rbf (Raw Binary File).....	28
3.2.3.	Tạo file Preloader - preloader-mkpimage.bin	29
3.2.4.	Tạo file u-boot - u-boot.img.....	30
3.2.5.	Tạo zImage	30
3.2.6.	Tạo dtb (Device Tree Blob File)	31

3.2.7.	Xây dựng hệ thống file root (root filesystem)	31
3.2.8.	Xây dựng OpenCL driver	31
3.2.9.	Xây dựng file U-Boot Script	32
3.2.10.	Tạo SD Card image	32
Chương 4.	THIẾT KẾ VÀ HIỆN THỰC MÔ HÌNH CUSTOM YOLO TRÊN DE1-SOC SỬ DỤNG OPENCL	34
4.1.	Hệ thống SoC-FPGA của đề tài	34
4.1.1.	Tổng quan hệ thống	34
4.1.2.	Luồng hiện thực hệ thống SoC trong nhận diện đối tượng	35
4.2.	Thiết kế mô hình YOLO tùy biến trên HPS portion	35
4.3.	Thiết kế bộ nhân ma trận (General matrix multiply) trên FPGA portion....	37
4.4.	Huấn luyện mô hình custom yolo	39
4.4.1.	Chuẩn bị dữ liệu	39
4.4.2.	Huấn luyện mô hình	41
Chương 5.	KẾT QUẢ VÀ ĐÁNH GIÁ	43
5.1.	Kết quả training mô hình Custom YOLO	43
5.2.	Kết quả thực nghiệm mô hình	46
5.2.1.	Kết quả chạy mô hình với ảnh	46
5.2.2.	Kết quả chạy mô hình với video	49
5.3.	Kết quả và đánh giá	53
5.3.1.	Kết quả mong đợi	53
5.3.2.	Kết quả thực tế	53
5.3.3.	Đánh giá kết quả	54
Chương 6.	KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN	55

6.1. Kết luận.....	55
6.1.1. Những gì nhóm đã đạt được	55
6.1.2. Khó khăn	55
6.2. Hướng phát triển.....	56
TÀI LIỆU THAM KHẢO.....	57

DANH MỤC HÌNH ẢNH

Hình 2.1: Layout của SD card dùng để boot linux vào board DE1-SoC	4
Hình 2.2: Định dạng số half precision tuân theo tiêu chuẩn IEEE 754-2008	6
Hình 2.3 Mô hình VGG-16	7
Hình 2.4: Ví dụ về tích chập 2 chiều Stride 1, Valid Padding	8
Hình 2.5: Lóp Pooling khi Stride bằng 2	9
Hình 2.6: Lóp kết nối đầy đủ với 7 nôt đầu vào và 5 nôt đầu ra.	10
Hình 2.7: Đồ thị hàm ReLU	11
Hình 2.8: Đồ thị hàm Leaky ReLU	12
Hình 2.9: Sơ đồ kiến trúc mạng YOLOv1	13
Hình 2.10: Ví dụ về cách thức mô hình YOLO hoạt động	14
Hình 2.11: Công thức của hàm IOU	15
Hình 2.12: Kiến trúc mô hình tiny-YOLO	16
Hình 2.13: Dự đoán bounding box trong trong YOLOv2	19
Hình 2.14: Kiến trúc mô hình YOLOv2	21
Hình 2.15: Kiến trúc chung của FPGA	22
Hình 2.16: Cấu trúc của khối logic có thể định cấu hình (CLB)	23
Hình 2.17: Layout thực của board DE1-SoC	24
Hình 2.18: Block diagram của board DE1-SoC	24
Hình 2.19: Quy trình thiết kế và biên dịch của một ứng dụng sử dụng sử dụng Intel® FPGA SDK for OpenCL™	26
Hình 3.1: BSP(Board Support Package) for Altera SDK OpenCL tại trang chủ Terasic	29
Hình 4.1: Kiến trúc hệ thống SoC-FPGA	34
Hình 4.2: Luồng thiết kế của mô hình chạy trên board DE1-SoC	35
Hình 4.3: Hình ảnh kiến trúc mô hình custom YOLO	36
Hình 4.4: Tensor convolution	38
Hình 4.5: Thuật toán GEMM	39
Hình 4.6: Một số labels trong tập train COCO dataset	41

Hình 5.3: Kết quả chạy mô hình trên Intel CPU	47
Hình 5.4: Kết quả chạy mô hình trên Board DE1-SoC.....	48
Hình 5.5: Kết quả dự đoán video trên board DE1-SoC	53

DANH MỤC BẢNG

Bảng 2.1: Chi tiết các file hệ thống trong mỗi partition	4
Bảng 2.2: Giá trị thể hiện của số dấu chấm động	6
Bảng 4.1: Các thông số của mạng custom YOLO	37
Bảng 4.2: Thể hiện số lượng nhãn trong tập dữ liệu VOC và COCO	40
Bảng 5.1: Thể hiện độ chính xác AP của từng labels	43
Bảng 5.2: Hiệu suất mô hình Tiny YOLO tập dữ liệu COCO.....	46
Bảng 5.3: mAP và FLOPS của Tiny Yolov2 và Custom Yolo.....	46
Bảng 5.4: IoU và Recall của Tiny-Yolov2 vs Custom-Yolo	47
Bảng 5.5: Tốc độ trung bình giữa CPU Intel và DE1-SoC.....	49
Bảng 5.6: Tài nguyên sử dụng trên FPGA của kernel.	54

DANH MỤC TỪ VIẾT TẮT

Ký hiệu hoặc viết tắt	Chú thích
mAP	Mean Average Precision
FPS	Frame per second
FPGA	Field-programmable gate array
CNN	Convolutional Neural Network
AI	Artificial intelligence
SDK	Software Development Kit
YOLO	You only look once
ReLU	Rectified Linear Unit
IOU	Intersection Over Union
NN	Neural Network
HPS	Hardware Processor System
rbf	Raw Binary File
dtb	Device Tree Blob
CLB	Configurable Logic Block

TÓM TẮT KHÓA LUẬN

Trong lĩnh vực computer vision, có rất nhiều chủ đề, bài toán mà con người vẫn đang cố gắng tối ưu: classification, object detection/recognition, semantic segmentation, style transfer... Trong đó object detection là bài toán rất quan trọng và phổ biến, ứng dụng rộng rãi. Một trong những cách dùng để cải thiện tốc độ tính toán là hiện thực mạng CNN (Convolutional Neural Network) trên FPGA (Field-programmable gate array). Tuy nhiên việc sử dụng FPGA gặp một trở ngại đó là việc phải sử dụng ngôn ngữ đặc tả phần cứng để thiết kế mạng CNN khá phức tạp và tốn nhiều thời gian.

Để giải quyết khó khăn trên, trong đề tài này, nhóm đề xuất một hướng đi mới trong việc thiết kế phần cứng là sử dụng công nghệ Intel® FPGA SDK (Software Development Kit) for OpenCL™ để thiết kế phần cứng bằng ngôn ngữ C với framework OpenCL. Từ đó hiện thực mạng CNN trên board DE1-SoC dùng để giải quyết bài toán Object detection với độ chính xác trung bình (Mean Average Precision – mAP) là 13.54% và tốc độ xử lý đạt trung bình là 3~4 FPS (Frame per second).

Chương 1. TỔNG QUAN ĐỀ TÀI

1.1. Tổng quan

Trong thời đại ngày nay, AI (artificial intelligence) mà cụ thể là lĩnh vực computer vision đã đang là một trong những lĩnh vực được quan tâm và đầu tư phát triển nhất trên hầu hết các quốc gia. Và một trong những lĩnh vực nghiên cứu nhiều nhất hiện nay là việc làm sao để tăng tốc cho quá trình tính toán của mạng CNN. Một trong những phương pháp có thể nói đang là xu hướng hiện nay đó là việc hiện thực mô hình CNN lên FPGA do ưu điểm là khả năng tái lập trình và tốc độ tính toán nhanh.

Theo như phương pháp truyền thống, ta sẽ sử dụng ngôn ngữ mô tả phần cứng như Verilog hay VHDL để thiết kế mạng CNN rồi sau đó sẽ nạp lên FPGA. Tuy nhiên nhược điểm của phương pháp này đó là ngôn ngữ mô tả phần cứng khá là phức tạp. Việc thiết kế mô hình với các ngôn ngữ trên sẽ tốn rất nhiều thời gian và thậm chí với những mô hình lớn việc thực hiện có thể nói là bất khả thi. Để giải quyết vấn đề trên, Intel phát triển một công nghệ phần mềm gọi là Intel® FPGA SDK for OpenCL™ cung cấp các công cụ giúp chúng ta tổng hợp phần cứng từ ngôn ngữ cấp cao là C với framework OpenCL nạp lên FPGA.

Dựa vào công nghệ trên của Intel, đề tài **ỨNG DỤNG THUẬT TOÁN PHÁT HIỆN ĐỐI TƯỢNG TRÊN SOC-FPGA SỬ DỤNG OPENCL** là một hướng đi mới để hiện thực mạng CNN trên FPGA để giải quyết các bài toán về phát hiện đối tượng (object detection).

1.2. Lý do thực hiện đề tài

Theo kết quả của nghiên cứu và tìm hiểu của nhóm thì hiện tại các bài báo về đề tài hiện thực mạng CNN lên FPGA thì đã được thực hiện khá nhiều tuy nhiên việc sử dụng công nghệ Intel® FPGA SDK (Software Development Kit) for OpenCL™ của Intel để hiện thực mạng CNN thì chưa có bài báo hay tạp chí nào đề cập đến. Tuy nhiên việc hiện thực CNN bằng công nghệ Intel® FPGA SDK for OpenCL™ ở trên thế giới đã được khá nhiều nhóm nghiên cứu thực hiện và đã có những bài báo viết về đề tài này. Do đó nhóm quyết định thực hiện đề tài này để có thể giúp các nhóm

nguyên cứu sau có thể tiếp cận một hướng đi mới trong việc hiện thực mạng CNN trên FPGA.

1.3. Mục tiêu thực hiện

Mục tiêu chính của đề tài là hiện thực mô hình custom YOLO bằng cách sử dụng OpenCL để tối ưu hóa quá trình thực thi. Từ đó phát triển hệ thống phát hiện nhiều loại đối tượng đa dạng chính xác trong thời gian thực. Kết hợp xây dựng hệ điều hành tùy biến linux để tối ưu thuật toán trên board.

1.4. Nội dung chính

Những nội dung chính trong việc thực hiện đề tài này là:

- Xây dựng hệ điều hành linux cho board DE1-SoC.
- Thiết kế mô hình custom YOLO (You only look once) trên DE1-SoC sử dụng OpenCL.
- Hiện thực mô hình custom YOLO trên DE1-SoC và kiểm thử kết quả.

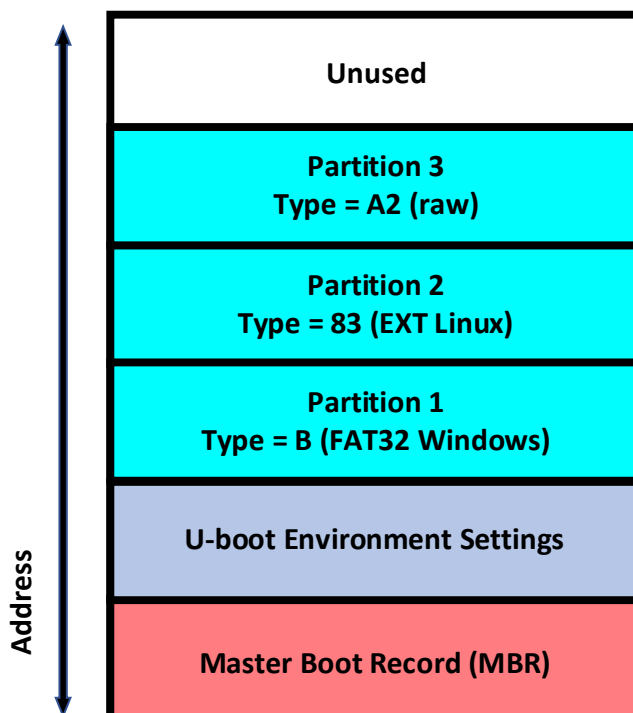
1.5. Giới hạn đề tài

Với framework của nhóm đang sử dụng nhóm thử nghiệm với mô hình Tiny-Yolov2 với 9 lớp convolution việc thực hiện phép toán trên board DE1-SoC là không đủ tài nguyên vì vậy với các tham số đó nhóm quyết định custom mô hình darknet bao gồm 8 lớp để phát hiện 80 đối tượng phổ biến cần phát hiện cho xu hướng về lĩnh vực thị giác và robot hiện nay để giảm lượng tài nguyên và tốc độ thực thi tốt hơn với hiệu suất sẽ giảm nhưng không đáng kể.

Chương 2. CƠ SỞ LÝ THUYẾT

2.1. Boot linux SD Card image trên board DE1-SoC

Để có thể boot linux lên board DE1-SoC bằng SD Card cần phải tạo trên SD Card 3 partition chứa các file hệ thống được mô tả ở Bảng 2.1 và có layout như Hình 2.1.



Hình 2.1: Layout của SD card dùng để boot linux vào board DE1-SoC
Trong mỗi partition cần chứa các file theo Bảng 2.1 dưới đây.

Bảng 2.1: Chi tiết các file hệ thống trong mỗi partition

Vị trí	Tên file	Mô tả
Partition 1	zImage	Compressed Linux kernel image file
	soc_system.dtb	Linux Device Tree Blob file
	opencl.rbf	Compressed FPGA configuration file
	uboot.img	Uboot Image
	uboot.scr	Uboot Script
Partition 2	rootfs	Linux root filesystem
Partition 3	preloader-mkpmimage.bin	preloader image

2.2. Số dấu chấm động (Floating point)

2.2.1. Khái niệm số dấu chấm động

Trong tính toán, số dấu chấm động là dạng biểu diễn gần đúng của số thực với sự thỏa hiệp giữa độ chính xác và khoảng giá trị. Dạng biểu diễn số dấu chấm động bao gồm một phần định trị (significand), số mũ (exponent) và cơ số (base). Cơ số có thể là 2, 10 hoặc 16. Dạng biểu diễn của số dấu chấm động như sau:

$$\text{Significand} \times \text{base}^{\text{exponent}}$$

Ví dụ ta biểu diễn số thực có giá trị 1.2345 ở dạng số dấu chấm động như sau:

$$1.2345 = 12345 \times 10^{-4}$$

Thuật ngữ dấu chấm động chỉ rằng dấu chấm cơ số có thể di động, nghĩa là dấu chấm cơ số có thể đặt tại bất kỳ vị trí nào trong phần định trị của số. Vị trí trên số của dấu chấm này được xác định bởi phần mũ và từ đó biểu diễn số dấu chấm động có thể xem là một dạng của biểu diễn khoa học.

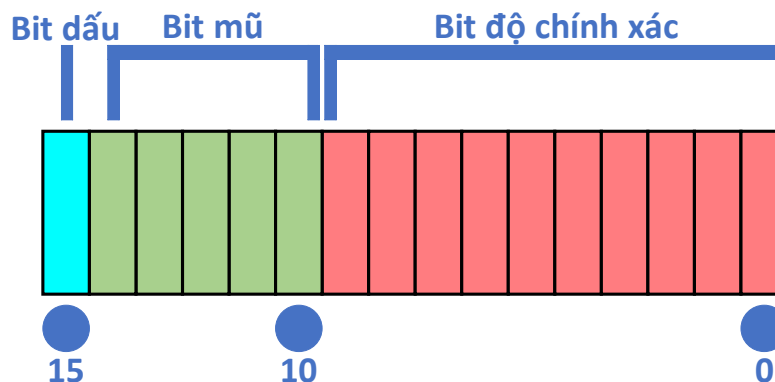
Có nhiều dạng biểu diễn số dấu chấm động được sử dụng trong hệ thống máy tính. Tuy nhiên từ thập kỷ 1990, dạng biểu diễn số dấu chấm động được sử dụng nhiều nhất là dạng biểu diễn được định nghĩa bởi chuẩn IEEE 754.

Tốc độ của phép tính dấu chấm động thường được đo bởi thuật ngữ FLOPS là một đặc trưng quan trọng trong hệ thống máy tính.

2.2.2. Chuẩn half-precision floating-point ieee-754-2008

Trong máy tính, half precision (đôi khi được gọi là FP16) là một định dạng số máy tính dấu chấm động có 16-bit (2-byte trong máy tính hiện đại) trong bộ nhớ máy tính. Nó được thiết kế để lưu trữ các giá trị dấu chấm động trong các ứng dụng không cần độ chính xác cao hơn, đặc biệt là hình ảnh đồ họa máy tính và mạng nơ-ron.

Hầu hết các ứng dụng sử dụng số half precision đều tuân theo tiêu chuẩn IEEE 754-2008 có định dạng như Hình 2.2.



Hình 2.2: Định dạng số half precision tuân theo tiêu chuẩn IEEE 754-2008

Trong đó:

- Số bit dấu: 1 bit
- Số bit mũ: 5 bits
- Số bit độ chính xác: 10 bits

Định dạng trên có thể biểu thị các giá trị trong phạm vi $\pm 65,504$, với giá trị nhỏ nhất lớn hơn 1 là: $1 + 1/1024$.

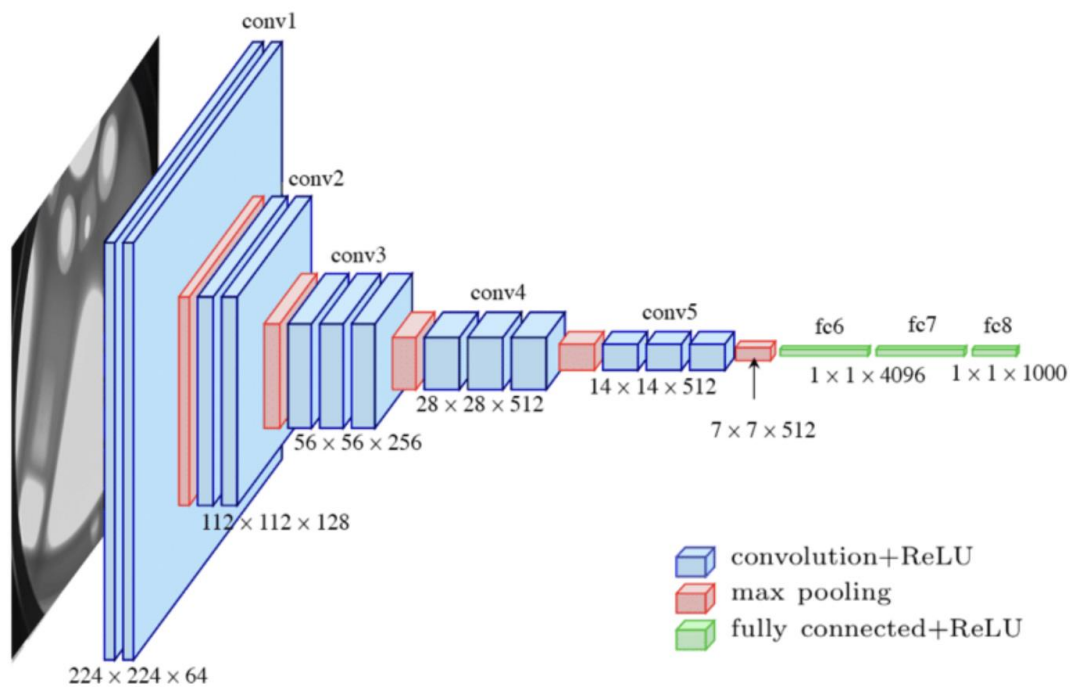
Chuẩn IEEE 754 cho số dấu chấm động độ chính xác đơn còn quy định các giá trị đặc biệt như số vô cùng (infinity), số không, số không chuẩn hóa (denormalized), Not a Number (NaN). Bảng 2.2 thể hiện các giá trị thể hiện của số dấu chấm động theo số mũ và phần định trị được quy định trong chuẩn IEEE 754.

Bảng 2.2: Giá trị thể hiện của số dấu chấm động

Số mũ	Bit dấu	Giá trị thể hiện
0	0	0
0	Khác không	Số không chuẩn hóa
1 đến 254	Giá trị bất kỳ	Số dấu chấm động
255	0	Infinity
255	Khác không	NaN

2.3. Convolutional Neural Network – Mạng Nơron tích chập

Mạng nơron tích chập (CNN) được lấy cảm hứng từ vỏ não thị giác. Mỗi khi chúng ta nhìn thấy một vật, một loạt các lớp tế bào thần kinh được kích hoạt, và mỗi lớp thần kinh sẽ phát hiện một tập hợp các đặc trưng như đường thẳng, cạnh, màu sắc, ... của đối tượng, lớp thần kinh càng cao sẽ phát hiện các đặc trưng phức tạp hơn để nhận ra những gì chúng ta đã thấy. Mạng nơron tích chập là mạng bao gồm một lớp đầu vào và một lớp đầu ra, ngoài ra, ở giữa còn có nhiều lớp ẩn (hidden layers). Các lớp ẩn của mạng thường bao gồm nhiều lớp tích chập (convolutional), lớp tổng hợp (pooling), lớp kết nối đầy đủ (Fully Connected) và theo sau các lớp ẩn là các hàm kích hoạt (ReLU, Softmax, ...). Hình 2.3 dưới đây là mô hình VGG-16 một mô hình CNN hoàn chỉnh.



Hình 2.3 Mô hình VGG-16

Các mạng nơron tích chập nổi bật như là LeNet - một trong những mạng CNN lâu đời nổi tiếng nhất được Yann LeCun phát triển vào năm 1998, AlexNet - mạng CNN đã dành chiến thắng trong cuộc thi ImageNet LSVRC-2012 năm 2012 với large margin (15.3% VS 26.2% error rates), VGGNet (Hình 2.3) - có tỉ lệ sai (error rate) nhỏ hơn AlexNet trong ImageNet Large Scale Visual Recognition Challenge

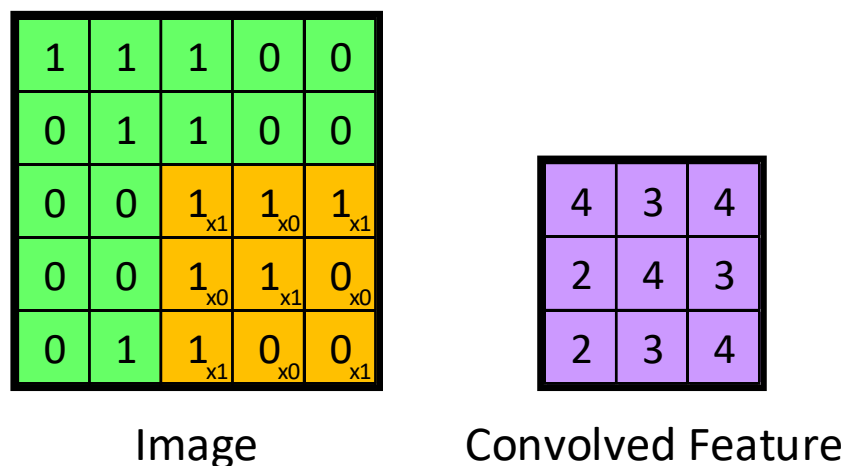
(ILSVRC) năm 2014, ResNets phát triển bởi Microsoft vào năm 2015, thắng giải ImageNet ILSVRC competition 2015 với tỉ lệ sai 3.57%.

2.4. Convolutional Layer – Lớp tích chập

Mỗi lớp tích chập trong mạng nơron sẽ có đầu vào là một tensor với kích thước (Số ảnh đầu vào) x (chiều rộng ảnh) x (chiều cao ảnh) x (chiều sâu của ảnh). Lớp tích chập sẽ thực hiện nhân chập với một kernel (ma trận lọc), chiều dài và rộng của kernel sẽ được chọn theo dạng tham số, còn chiều sâu thì phải bằng với chiều sâu của ảnh, sau đó, kết quả của lớp này sẽ được truyền sang lớp tiếp theo.

Để tính tích chập ta cần quan tâm đến hai thông số là Stride và Padding. Kernel sẽ được trượt qua ảnh, với mỗi lần trượt ta sẽ tính kết quả tại vị trí đầu ra. Số bước ở mỗi lần trượt kernel được gọi là Stride, thông thường, trong tích chập, Stride bằng 1. Nếu tích chập Same Padding, sẽ tạo các điểm ảnh bằng 0 ở rìa của ảnh, đặc trưng đầu ra sẽ có kích thước không đổi so với ảnh đầu vào. Còn Valid Padding, đặc trưng đầu ra sẽ giảm $K - 1$ so với ảnh đầu vào, với K là kích thước kernel.

Hình 2.4 là ví dụ về nhân tích chập 2 chiều trên ảnh kích thước 5x5, kích thước kernel là 3x3, stride bằng 1 và Valid Padding. Sau đó, đặc trưng thu được có kích thước 3x3.



Hình 2.4: Ví dụ về tích chập 2 chiều Stride 1, Valid Padding

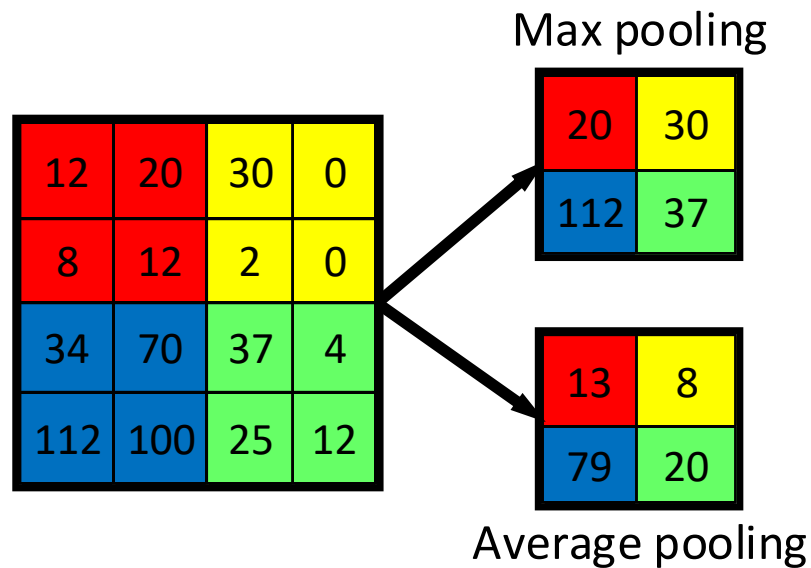
2.5. Pooling Layer – Lớp tổng hợp

Lớp tổng hợp chịu trách nhiệm giảm kích thước không gian của đặc trưng sau khi tích chập. Nhờ đó, lớp tổng hợp giúp giảm số lượng tính toán của dữ liệu. Hơn nữa, lớp này rất hữu ích trong việc lọc ra những đặc trưng “trội”, giúp cho model được huấn luyện tốt và dễ hơn. Lớp tổng hợp cũng có hai thông số Stride và Padding, Stride được định nghĩa giống với Stride ở lớp tích chập. Với Padding Same, kích thước đầu ra sẽ được xác định với công thức (2.1), với Padding Valid, kích thước đầu ra sẽ được xác định với công thức (2.2), gọi H, W là kích thước đầu vào, H', W' là kích thước đầu ra, K là kích thước của cửa sổ trượt.

$$\begin{cases} H' = \text{Ceil}(\frac{H}{K}) \\ W' = \text{Ceil}(\frac{W}{K}) \end{cases} \quad (2.1)$$

$$\begin{cases} H' = \frac{H}{K} \\ W' = \frac{W}{K} \end{cases} \quad (2.2)$$

Có hai loại Pooling là Max Pooling (trả về giá trị lớn nhất trong cửa sổ trượt) và Average Pooling (trả về giá trị trung bình trong cửa sổ trượt). Hình 2.5 là ví dụ của Max Pooling và Average Pooling với cửa sổ trượt kích thước 2×2 , đầu vào có kích thước 4×4 , stride bằng 2.



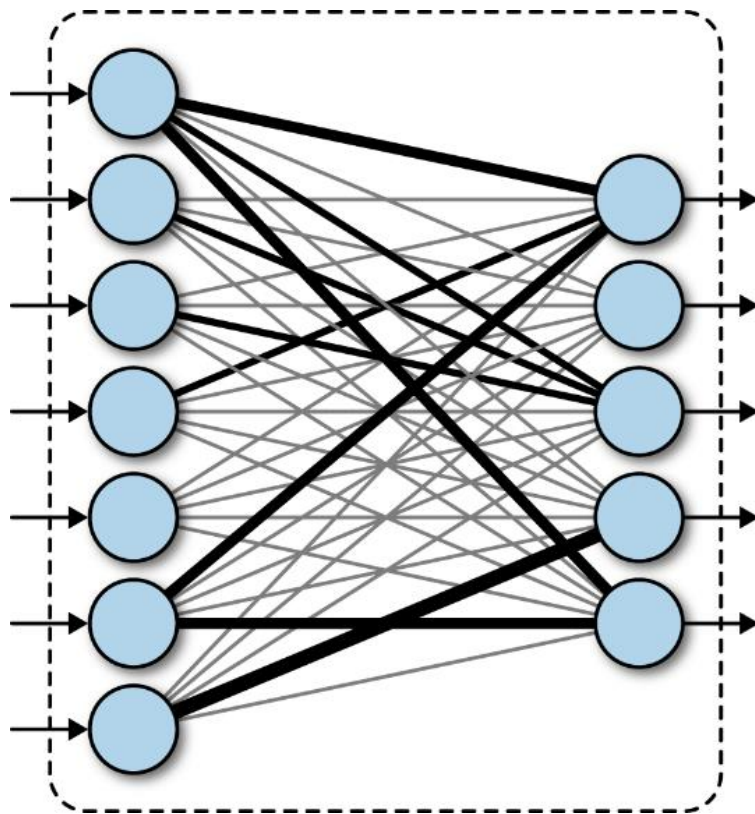
Hình 2.5: Lớp Pooling khi Stride bằng 2

2.6. Fully Connected Layer – Lớp kết nối đầy đủ

Sau khi thực hiện các lớp tích chập và tổng hợp, mạng CNN thường kết thúc bằng lớp kết nối đầy đủ - kết nối tất cả nôt của lớp này với tất cả nôt của lớp khác. Lớp này thường dùng trong mục đích để phân loại ảnh và để dữ liệu dữ đoán có thể hiểu được, hàm kích hoạt Softmax sẽ được áp dụng để đưa dữ liệu về dạng xác suất.

Đầu vào của lớp sẽ được “làm phẳng” thành ma trận một chiều, tất cả các nôt của ma trận sẽ được nhân với một trọng số để tạo ra một nôt đầu ra. Lớp này được tính bằng công thức (1), với N là số nôt đầu ra. Hình 2.6 mô tả lớp kết nối đầy đủ với 7 nôt đầu vào và 5 nôt đầu ra.

$$y_i = \sum_{k=0}^{W*H-1} x_k * w_k + b_i, 0 \leq i < N \quad (1)$$



Hình 2.6: Lớp kết nối đầy đủ với 7 nôt đầu vào và 5 nôt đầu ra.

2.7. Activation function - Các hàm kích hoạt

Hàm kích hoạt (activation function) mô phỏng tỷ lệ truyền xung qua axon của một neuron thần kinh. Trong một mạng nơ-ron nhân tạo, hàm kích hoạt đóng vai trò là thành phần phi tuyến tại output của các nơ-ron. Mục đích của các hàm kích hoạt là áp dụng tính chất phi tuyến tính vào mạng. Nếu không có hàm kích hoạt phi tuyến tính trong mạng thì cho dù nó có bao nhiêu lớp, mạng sẽ hoạt động giống như một tri giác một lớp, bởi vì việc tổng hợp các lớp này sẽ cung cấp cho bạn một hàm tuyến tính khác.

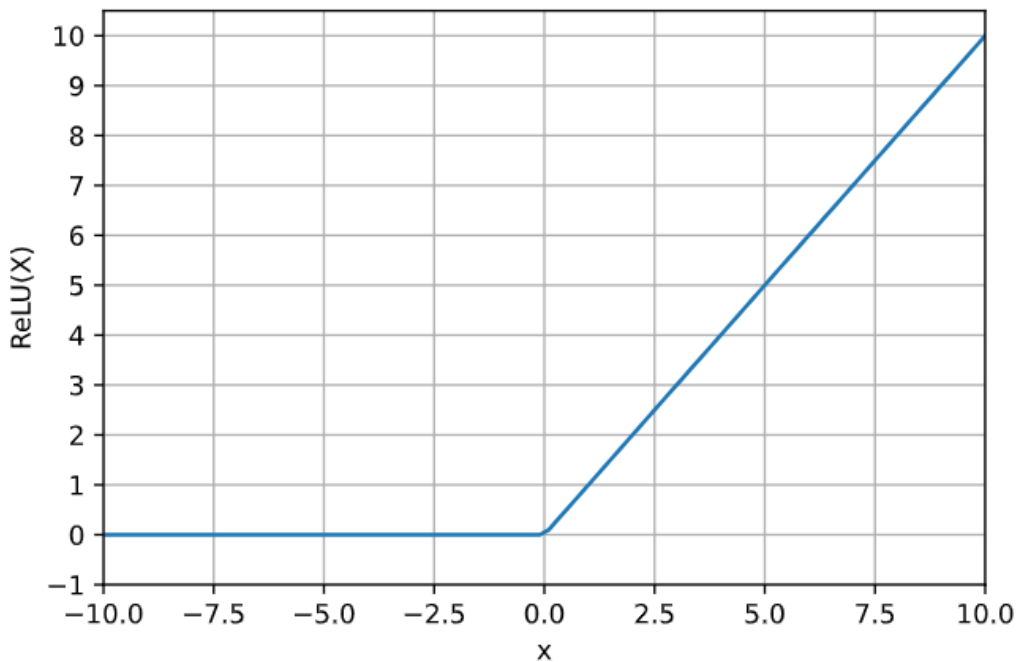
Dưới đây là các hàm kích được dùng trong đề tài của nhóm.

2.7.1. ReLU

Công thức

$$f(x) = \max(0, x) \quad (2.3)$$

ReLU, đầy đủ là Rectified Linear Unit, đang được sử dụng khá nhiều trong những năm gần đây khi huấn luyện các mạng neuron. ReLU đơn giản lọc các giá trị < 0 . Nhìn công thức (2.4) và đồ thị (Hình 2.7) chúng ta dễ dàng hiểu được cách hoạt động của nó.



Hình 2.7: Đồ thị hàm ReLU

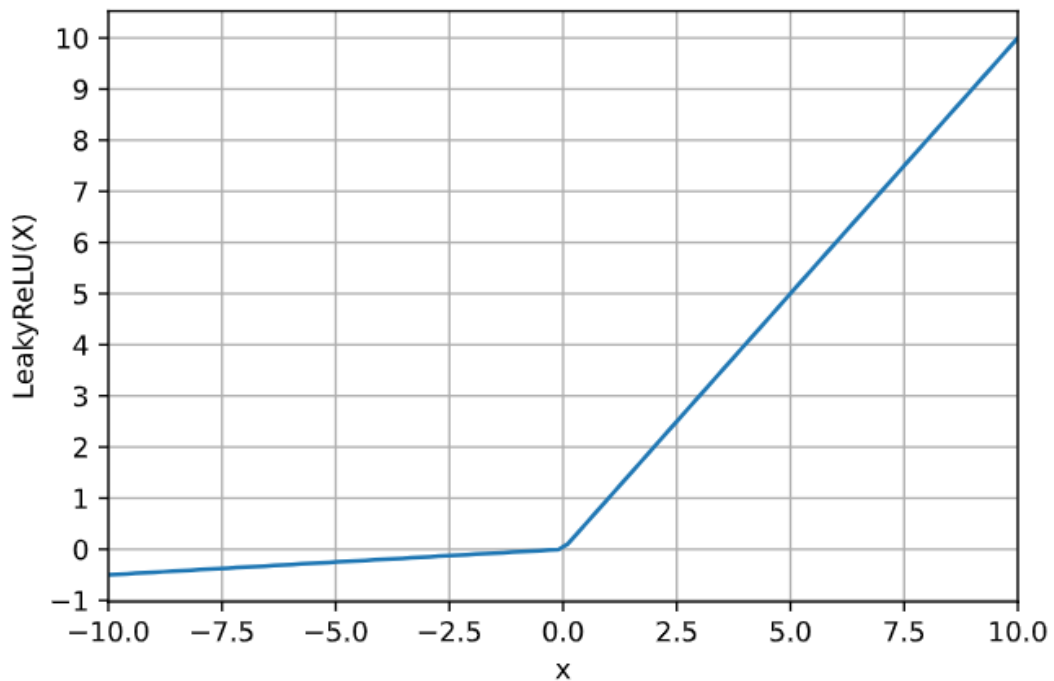
2.7.2. Leaky ReLU

Công thức

$$f(x) = (x < 0)(\alpha x) + 1(x \geq 0)(x)$$

Với α là hằng số nhỏ

Leaky ReLU là một cố gắng trong việc loại bỏ "dying ReLU". Thay vì trả về giá trị 0 với các đầu vào < 0 thì Leaky ReLU tạo ra một đường xiên có độ dốc nhỏ (xem đồ thị - Hình 2.8). Có nhiều báo cáo về việc hiệu Leaky ReLU có hiệu quả tốt hơn ReLU, nhưng hiệu quả này vẫn chưa rõ ràng và nhất quán.

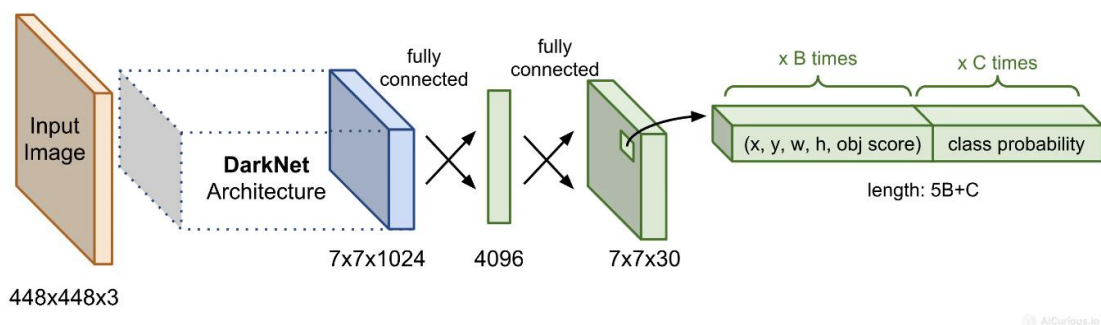


Hình 2.8: Đồ thị hàm Leaky ReLU

2.8. YOLOv1

YOLO là một mô hình mạng CNN cho việc phát hiện, nhận dạng, phân loại đối tượng. YOLO có rất nhiều ứng dụng trong đời thực như: Hệ thống theo dõi xe tự lái, hệ thống theo dõi người...

Yolo được tạo ra từ việc kết hợp giữa các lớp convolution và các lớp fully connected. Trong đó các lớp convolution sẽ trích xuất ra các feature của ảnh, còn các lớp fully connected sẽ dự đoán ra xác suất đó và tọa độ của đối tượng.



Hình 2.9: Sơ đồ kiến trúc mạng YOLOv1

Hình 2.9 mô tả sơ đồ kiến trúc mạng YOLOv1. Thành phần Darknet Architecture được gọi là base network có tác dụng trích xuất đặc trưng. Output của base network là một feature map có kích thước 7x7x1024 sẽ được sử dụng làm input cho các Extra layers có tác dụng dự đoán nhãn và tọa độ bounding box của vật thể.

Đầu vào của mô hình YOLO là một ảnh, mô hình sẽ nhận dạng ảnh đó có đối tượng nào hay không, sau đó sẽ xác định tọa độ của đối tượng trong bức ảnh. Các phương pháp trước YOLOv1 thường sử dụng 2 bước: bước (1) thường sử dụng sliding window để lấy các vùng khác nhau của bức ảnh, hoặc sử dụng một thuật toán lựa chọn các vùng ứng viên (có thể chứa vật), tiếp theo đó, bước (2) sẽ phân loại các vị trí này xem vật đó thuộc lớp nào. Các cách tiếp cận này có nhược điểm là yêu cầu một lượng tính toán lớn, và bị phân nhỏ thành nhiều bước, khó có thể tối ưu về mặt tốc độ. Kiến trúc YOLOv1 coi bài toán phát hiện vật như một bài toán regression. Từ input là ảnh đầu vào, qua một mạng gồm các lớp convolution, pooling và fully connected là có thể ra được output. Kiến trúc này có thể được tối ưu để chạy trên GPU với một lần forward pass, và vì thế đạt được tốc độ rất cao.

2.8.1. Ý tưởng

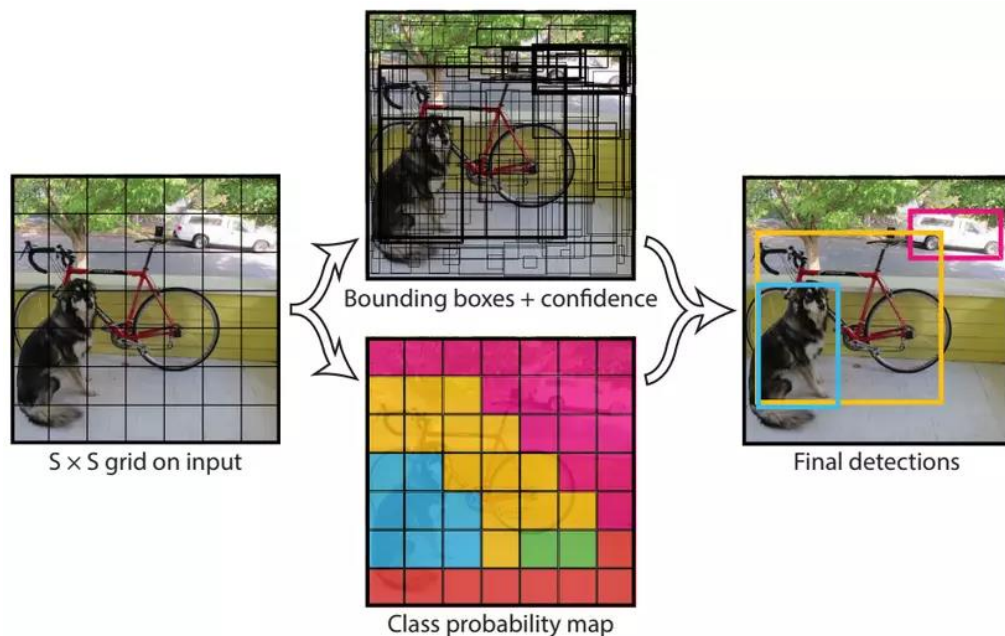
Ý tưởng chính của YOLOv1 là chia ảnh thành một lưới các ô (grid cell) với kích thước SxS thường thì sẽ là 3x3, 7x7, 9x9, ... Với mỗi grid cell, mô hình sẽ đưa ra dự đoán cho B bounding box. Ứng với mỗi box trong B bounding box này sẽ là 5 tham số x, y, w, h, confidence, lần lượt là tọa độ tâm (x, y), chiều rộng, chiều cao và độ tự

tin của dự đoán. Với grid cell trong lưới SxS kia, mô hình cũng dự đoán xác suất rơi vào mỗi class.

Độ tự tin của dự đoán ứng với mỗi bounding box được định nghĩa là $p(Object) \times IOU_{pred}^{truth}$ trong đó $p(Object)$ là xác suất có vật trong cell và IOU_{pred}^{truth} (intersection over union – được trình bày ở mục 2.8.2 là hàm đánh giá độ chính xác của object detector trên tập dữ liệu cụ thể).

Xác suất rơi vào mỗi class cho một grid cell được ký hiệu $p(Class_i|Object)$. Các giá trị xác suất cho C class sẽ tạo ra C output cho mỗi grid cell. Lưu ý là B bounding box của cùng một grid cell sẽ chia sẻ chung một tập các dự đoán về class của vật, đồng nghĩa với việc tất cả các bounding box trong cùng một grid cell sẽ chỉ có chung một class.

Với Input là 1 ảnh, output của mô hình là một ma trận 3 chiều có kích S x S x (5 x B + C) với số lượng tham số mỗi ô là (5 x B + C) với B và C lần lượt là số lượng Box và Class mà mỗi ô cần dự đoán.

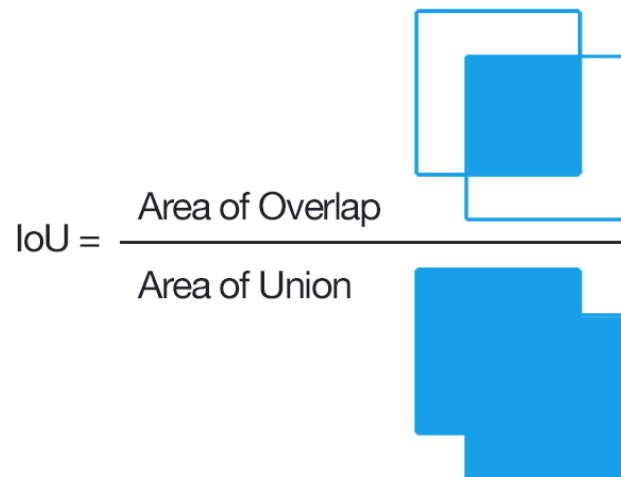


Hình 2.10: Ví dụ về cách thức mô hình YOLO hoạt động

Như Hình 2.10 mô hình chia bức ảnh đầu vào thành thành 7 x 7 ô. Sau đó nó sẽ phát hiện tất cả các khối có các đối tượng và phân loại mỗi ô thành các nhóm màu. Màu xanh là chó, màu vàng là xe đạp và màu hồng là ô tô.

2.8.2. Hàm tính IOU

Như có đề cập ở trên IOU (Intersection Over Union) là hàm đánh giá độ chính xác của object detector trên tập dữ liệu cụ thể. IOU được theo công thức ở Hình 2.11.


$$\text{IoU} = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$

Hình 2.11: Công thức của hàm IOU

Trong đó Area of Overlap là diện tích phần giao nhau giữa predicted bounding box (bounding box được model sinh ra) với ground-truth bounding box (bounding box đúng của đối tượng, ví dụ như bounding box của đối tượng được khoanh vùng và đánh nhãn bằng tay sử dụng trong tập test), còn Area of Union là diện tích phần hợp giữa predicted bounding box với ground-truth bounding box. Nếu $\text{IOU} > 0.5$ thì prediction được đánh giá là tốt.

2.8.3. Nhược điểm của YOLOv1

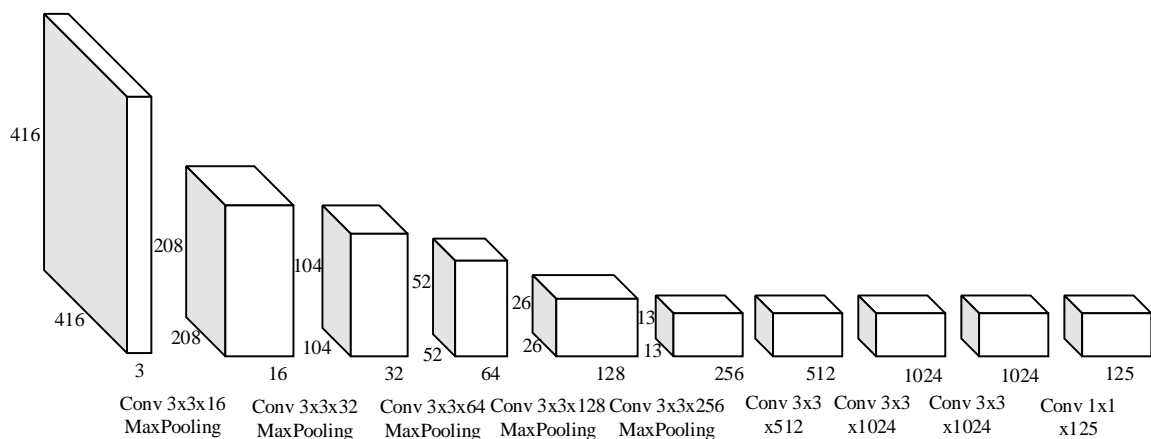
Tuy tốc độ của YOLO là cao tuy nhiên mô hình vẫn còn tồn tại một số nhược điểm cần khắc phục:

- Độ chính xác vẫn còn kém so với các Region-based detector.

- Do thiết kế output của YOLO v1, ta chỉ có thể dự đoán tối đa một object trong 1 grid cell (vì chỉ có một class distribution cho mỗi cell). Điều đó có nghĩa rằng nếu ta chọn $S \times S = 7 \times 7$, số lượng object tối đa chỉ bằng 49. Với những trường hợp nhiều vật cùng nằm trong 1 cell, YOLO sẽ kém hiệu quả.
- (x,y,w,h) được predict ra trực tiếp \rightarrow giá trị tự do. Trong khi đó trong hầu hết các dataset, các bounding box có kích cỡ không quá đa dạng mà tuân theo những tỷ lệ nhất định.

2.8.4. Tiny-YOLO

Tiny-YOLO là một biến thể của mô hình YOLO được Joseph Redmon đề xuất trong bài báo You Only Look Once: Unified, Real-Time Object Detection vào năm 2016.



Hình 2.12: Kiến trúc mô hình tiny-YOLO

Như trong Hình 2.12, kiến trúc của tiny-YOLO sử dụng 9 lớp convolution thay vì 24 như kiến trúc của YOLOv1. Điều này khiến tốc độ xử lý của tiny-YOLO cao hơn khoảng 442% so với YOLOv1 tức có thể đạt 244 FPS trên một GPU. Đồng thời kích thước mô hình giảm xuống dưới 50MB. Tuy nhiên độ chính xác của mô hình chỉ đạt được là 23.7% mAP giảm gần một nửa so với mô hình YOLOv1 (khoảng 51-57%). Tuy nhiên 23.7% vẫn đủ để sử dụng trong một vài ứng dụng.

Với kích thước nhỏ và tốc độ tính toán nhanh, tiny-YOLO rất phù hợp để hiện thực trên các máy tính vision/deep learning nhưng có dung lượng bộ nhớ không quá lớn như Raspberry Pi, Google Coral, and NVIDIA Jetson Nano.

2.9. YOLOv2

YOLOv2 hay YOLO9000 đã được Joseph Redmon và Ali Farhadi công bố vào cuối năm 2016 YOLO9000: Better, Faster, Stronger và có mặt trong 2017 CVPR. Việc tăng performance trong Computer Vision thường xoay quanh việc tăng kích thước mạng NN (Neural Network). Thay vì tăng kích thước mạng, YOLOv2 đơn giản hóa mạng NN và áp dụng các ý tưởng để cải thiện performance. Cải tiến chính của phiên bản này tốt hơn, nhanh hơn, tiên tiến hơn để bắt kịp faster R-CNN (phương pháp sử dụng Region Proposal Network), xử lý được những vấn đề gặp phải của YOLOv1. Dưới đây là những cải tiến chính của YOLOv2 so với YOLOv1

2.9.1. Anchor box

Đây là thay đổi đáng chú ý nhất của YOLOv2 so với YOLOv1. Trước tiên, anchor box là ý tưởng của FasterRCNN. Anchor box thực ra là các bounding box nhưng được tạo sẵn (chứ không phải kết quả của quá trình prediction). Điều này xuất phát từ việc các vật thể có một số bounding boxe tương đồng ví dụ như ô tô, xe đẹp có bounding box dạng hình chữ nhật nằm ngang, người có bounding box dạng hình chữ nhật đứng... Khái niệm anchor boxes này đã có trên Faster R-CNN, hoạt động khá hiệu quả, sẽ đi dự đoán bounding boxes dựa trên các anchor boxes này. Bằng việc dùng Kmean để phân cụm, ta sẽ tính ra được B anchor box đại diện cho các kích thước phổ biến. Như vậy, thay vì predict trực tiếp (x,y,w,h), ta predict ra bộ offset - tức độ lệch giữa ground-truth bounding box với các anchor box.

Trong YOLOv2, tác giả loại bỏ lớp fully connected ở giữa mạng và sử dụng kiến trúc anchor box để predict các bounding box. Việc dự đoán các offset so với anchor box sẽ dễ dàng hơn nhiều so với dự đoán tọa độ bounding box. Thay đổi này làm giảm mAP đi một chút nhưng làm recall tăng lên. Đồng thời Anchor box sẽ giúp

model phát hiện được nhiều vật thể trong cùng một grid cell, điều này đã khắc phục nhược điểm của YOLOv1 chỉ phát hiện được 1 object trong 1 grid cell.

2.9.2. Dự đoán vị trí trực tiếp

YOLOv1 không có các hạn chế trong việc dự đoán vị trí của bounding box. Khi các trọng số được khởi tạo ngẫu nhiên, bounding box có thể được dự đoán ở bất kỳ đâu trong ảnh. Điều này khiến mô hình không ổn định trong giai đoạn đầu của quá trình huấn luyện. Vị trí của bounding box có thể ở rất xa so với vị trí của grid cell.

YOLOv2 sử dụng hàm sigmoid (σ) để hạn chế giá trị trong khoảng 0 đến 1, từ đó có thể hạn chế các dự đoán bounding box ở xung quanh grid cell, từ đó giúp mô hình ổn định hơn trong quá trình huấn luyện.

Cho anchor box có kích thước (p_w, p_y) nằm tại grid cell với vị trí góc trên bên trái là (c_x, c_y) , mô hình sẽ dự đoán các offset và scale (t_x, t_y, t_w, t_h) và các thông số cho predicted bounding box (b_x, b_y, b_w, b_h) . Độ tự tin (confidence score) của dự đoán là $\sigma(t_0)$.

$$b_x = \sigma(t_x) + c_x$$

$$b_y = \sigma(t_y) + c_y$$

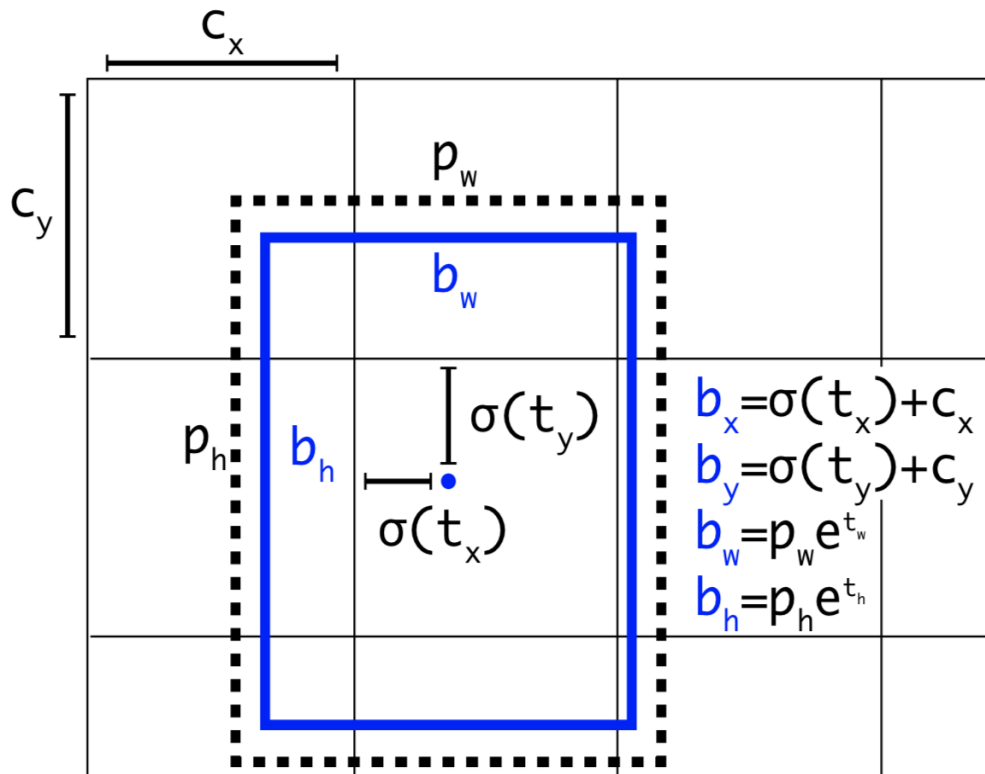
$$b_w = p_w e^{t_w}$$

$$b_h = p_h e^{t_h}$$

$$Pr(object).IoU(b, object) = \sigma(t_0)$$

Cách biểu diễn parameters của YOLOv2 khác với cách biểu diễn trong Faster R-CNN. Với cách biểu diễn trong YOLOv2, chúng ta đang ép tâm của predicted bounding box nằm trong grid cell ứng với anchor box.

Thực chất có mối liên hệ giữa (anchor box, grid cell) và ground-truth box, bây giờ như ở trên cũng có mối liên hệ giữa (anchor box, grid cell) và predicted bounding box. Từ đây nhận thấy thông qua anchor box và grid cell chúng ta đang đi khớp predicted bounding box với ground-truth box.



Hình 2.13: Dự đoán bounding box trong YOLOv2

Trong Hình 2.13, hình chữ nhật nét đứt bên ngoài là anchor box có kích thước là (p_w, p_h) . Tọa độ của một bounding box sẽ được xác định dựa trên đồng thời cả anchor box và grid cell mà nó thuộc về. Điều này giúp kiểm soát vị trí của bounding box dự đoán đâu đó quanh vị trí của grid cell và bounding box mà không vượt quá xa ra bên ngoài giới hạn này. Do đó quá trình huấn luyện sẽ ổn định hơn rất nhiều so với YOLOv1. YOLOv2 đã có thêm 5% mAP khi áp dụng phương pháp này.

2.9.3. Batch Normalization

Batch Normalization được thêm vào sau các lớp convolution trong YOLOv2. Nó giảm sự thay đổi giá trị của units trong các lớp ẩn, do đó sẽ cải thiện được tính ổn định của neural network. Việc thêm Batch Normalization giúp mAP tăng thêm 2%. Việc dùng Batch Normalization này đã giúp model thực hiện regularization, chúng ta có thể loại bỏ Dropout khỏi model mà không bị overfitting.

Batch Normalization là một phương pháp hiệu quả khi đào tạo một mô hình mạng nơ-ron. Mục tiêu của cách tiếp cận này là chuẩn hóa các tính năng (đầu ra của mỗi lớp sau khi vượt qua kích hoạt) về trạng thái trung bình bằng 0 với độ lệch chuẩn.

Batch Normalization có thể giúp ngăn giá trị x rơi vào trạng thái bão hòa sau khi vượt qua trình kích hoạt không chọn. Vì vậy, nó đảm bảo rằng không có kích hoạt nào quá cao hoặc quá thấp. Điều này giúp cho ta khi không sử dụng Batch Normalization, có thể sẽ không bao giờ học được, giờ có thể học bình thường. Điều này giúp YOLOv2 giảm sự phụ thuộc vào các giá trị khởi tạo của thông số.

2.9.4. Bộ phân loại độ phân giải cao.

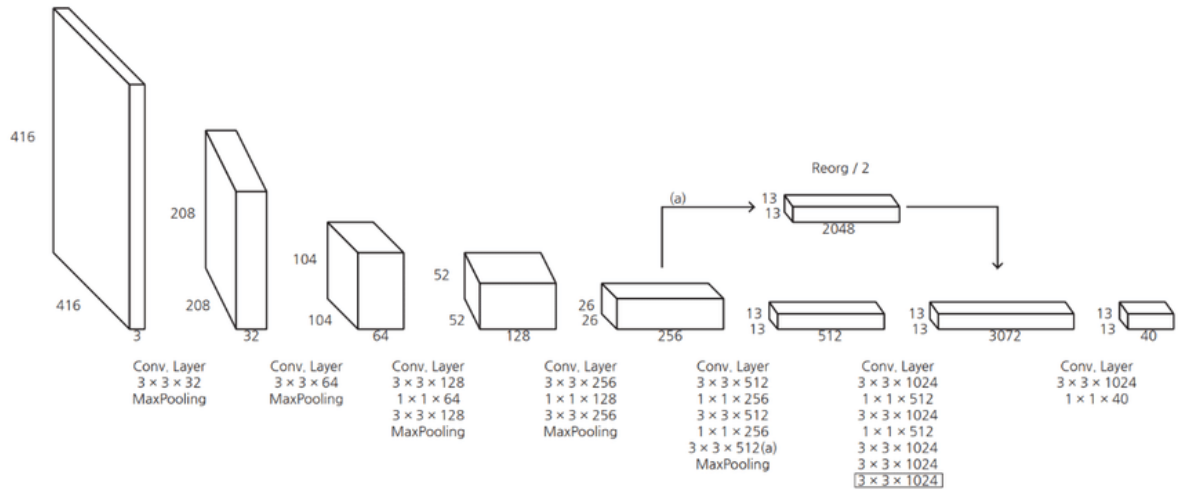
Trong YOLOv1, ban đầu chúng ta train base network cho classification task với kích thước ảnh đầu vào là 224×224 . Sau đó khi thực hiện training cho detection, kích thước ảnh đầu vào được thay đổi thành 448×448 .

Khác với YOLOv1, base network của YOLOv2 được train cho classification task với kích thước ảnh ban đầu là 448×448 trên ImageNet với 10 epochs. Sau đó thực hiện fine tune network cho detection task. Điều này giúp mAP tăng lên 4%.

2.9.5. Fine-Grained Features

YOLOv1 gặp vấn đề khó khăn khi phát hiện các vật thể nhỏ (chia ảnh thành 7×7 grid cells). YOLOv2 chia ảnh thành 13×13 grid cells, do đó có thể phát hiện được những vật thể nhỏ hơn, đồng thời cũng hiệu quả đối với các vật thể lớn.

Faster R-CNN và SSD đưa ra dự đoán ở nhiều tầng khác nhau trong mạng để tận dụng các feature map ở các kích thước khác nhau. Thay vì làm như vậy, YOLOv2 kết hợp các feature ở các tầng khác nhau lại để đưa ra dự đoán. Cụ thể ở Hình 2.14, YOLOv2 chuyển feature map $26 \times 26 \times 512$ thành $13 \times 13 \times 2048$ thông qua kỹ thuật Reorg, sau đó kết hợp với feature map $13 \times 13 \times 512$ feature map để được $13 \times 13 \times 3072$. Feature map này được dùng để đưa ra dự đoán.



Hình 2.14: Kiến trúc mô hình YOLOv2

Việc sử dụng fine-grained features giúp mAP của YOLOv2 được tăng thêm 1%.

2.9.6. Multi-Scale Training

YOLOv1 có điểm yếu khi phát hiện các đối tượng với các kích cỡ đầu vào khác nhau. Ví dụ YOLOv1 được huấn luyện với các ảnh có kích thước nhỏ của cùng loại vật thể, nó sẽ gặp vấn đề khi phát hiện vật thể tương tự trong ảnh có kích thước lớn hơn. Điều này được giải quyết với YOLOv2, nó được train với kích thước ảnh ngẫu nhiên trong khoảng 320x320, 352x352 đến 608x608. Điều này cho phép model học và dự đoán chính xác đối tượng với nhiều kích thước khác nhau. Kích thước mới của ảnh đầu vào được lấy ngẫu nhiên cứ sau 10 batches. Do Conv layers YOLOv2 giảm kích thước của ảnh đầu vào theo hệ số 32 (phụ thuộc vào các lớp convolution trong mạng NN) nên kích thước mới cần là số chia hết cho 32.

2.9.7. Light-weight backbone

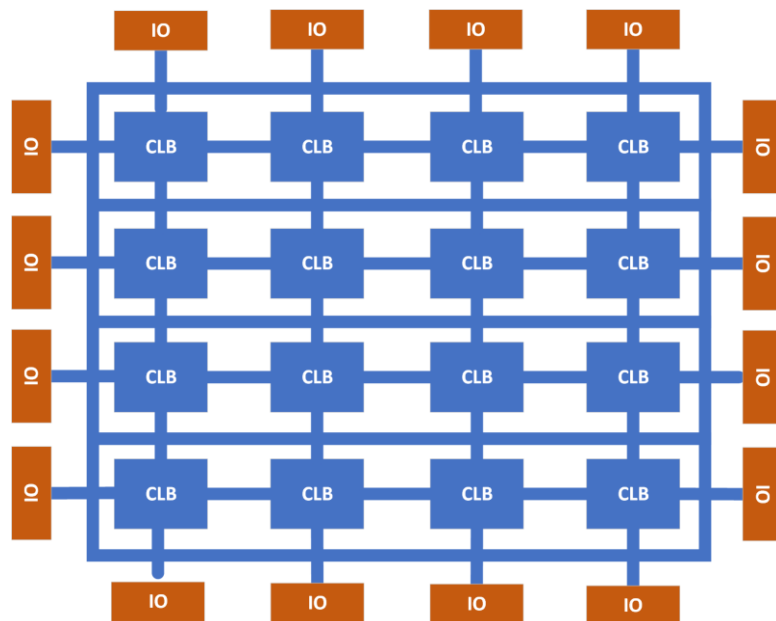
YOLOv1 tận dụng GoogleNet architecture, network này nhanh hơn VGG16, nó chỉ cần 8.52 tỉ operations để thực hiện forward pass. Tuy nhiên accuracy của nó thấp hơn so với VGG16.

Để tăng tốc độ YOLOv2 mà vẫn đảm bảo độ chính xác, YOLOv2 đã sử dụng kiến trúc Darknet-19 với 19 Conv layers, 5 MaxPooling layers, một Conv layer 1x1 để giảm số channels và 1 average pooling kết hợp với softmax layer cho phân loại vật

thể. Darknet được viết bằng ngôn ngữ C và CUDA. Darknet-19 cần 5.58 tỉ operations để xử lý image, nó đạt 72.9% top-1 accuracy và 91.2% top-5 accuracy trên ImageNet.

2.10. Field Programmable Gate Array (FPGA)

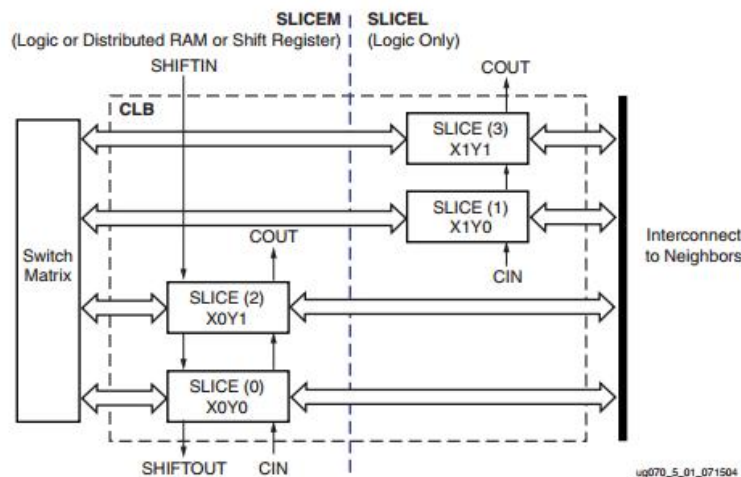
FPGA là một linh kiện bán dẫn có thể được lập trình bằng các ngôn ngữ lập trình như Verilog, VHDL, C, C ++... Nó được cấu tạo bởi một ma trận các khối Configurable Logic Block (CLB) được liên kết với nhau thông qua hệ thống Bus nội bộ. Ngược lại với mạch tích hợp dành riêng cho ứng dụng (ASIC) (xem Hình 2.15). FPGA có thể lập trình trên một ứng dụng mong muốn. Mặc dù kiến trúc FPGA là có thể lập trình một lần, nhưng ưu điểm của FPGA dựa trên SRAM là có thể lập trình lại theo ý định thiết kế. Do những ưu điểm có thể lập trình được, các sản phẩm FPGA là những sản phẩm lý tưởng như tạo mẫu ASIC, Ô tô, Truyền thông, xử lý video và hình ảnh...



Hình 2.15: Kiến trúc chung của FPGA

Cấu trúc của FPGA cơ bản:

- CLB: Đây là khối cơ bản nhất trong FPGA, cấu lạc bộ bao gồm một khối ma trận chuyển đổi gồm 4 đến 6 đầu vào có thể cấu hình, MUX và flipflop. Ma trận chuyển mạch có thể xử lý các mạch kết hợp và thanh ghi dịch chuyển hoặc RAM.

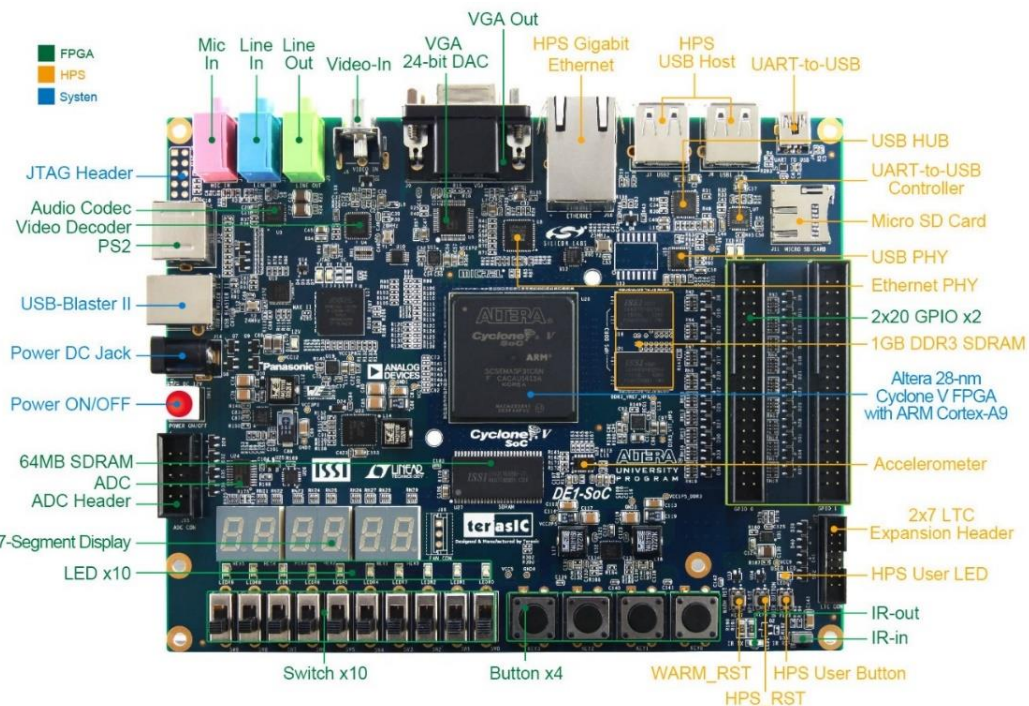


Hình 2.16: Cấu trúc của khối logic có thể định cấu hình (CLB)

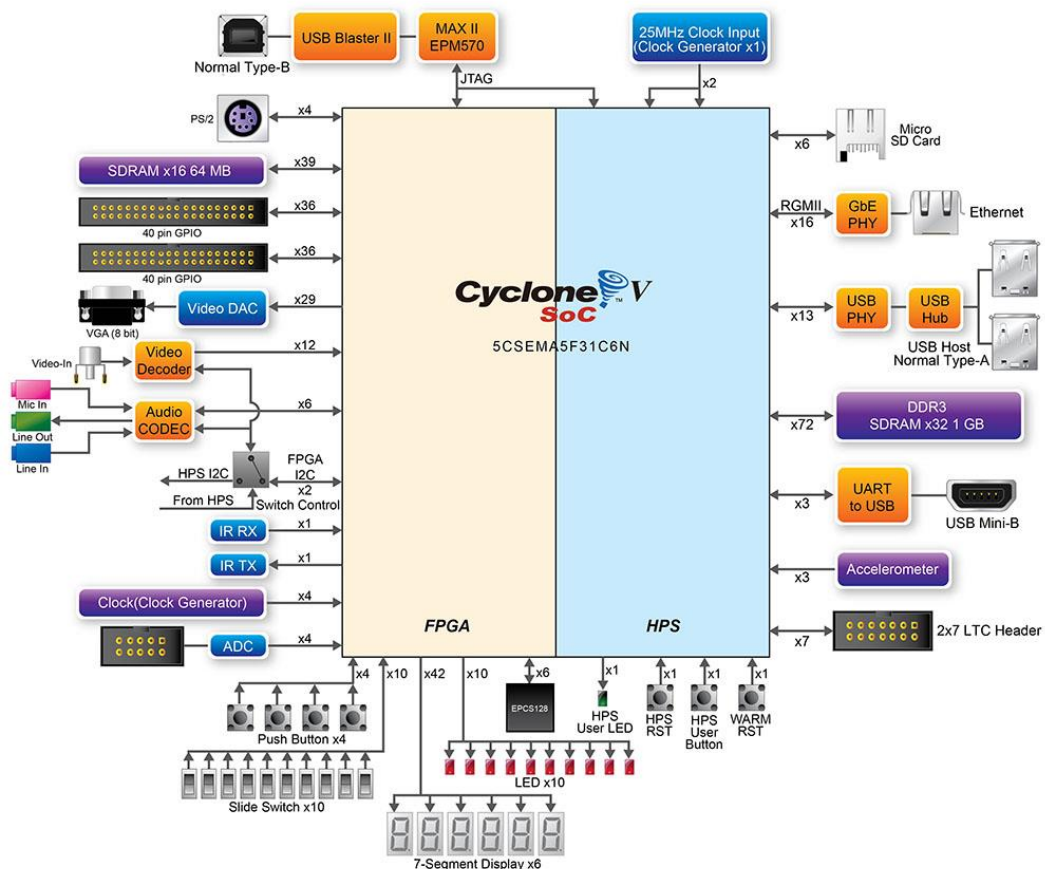
- Kết nối: các CLBs trong FPGA được kết nối với nhau thông qua I/Os Bus.
- Bộ nhớ (Memory): Hỗ trợ bộ nhớ trên chip, Bộ nhớ RAM khối được tìm thấy trong hầu hết các thiết bị FPGA.
- Quản lý xung nhịp: hỗ trợ xung nhịp kỹ thuật số cho người dùng quản lý tín hiệu xung nhịp gốc được tạo ra từ một bộ dao động trên FPGA và tạo xung clock mới theo ý muốn với tần số cao hơn hoặc thấp hơn.

2.11. Tổng quan về board DE1-SoC

Đề tài của nhóm được thực hiện trên board DE1-SoC do Terasic chế tạo và được dùng phổ biến trong các trường đại học trên thế giới với mục đích giáo dục. Kit phát triển DE1-SoC cung cấp một nền tảng phát triển thiết kế phần cứng được xây dựng dựa trên mô hình Altera (Intel) System-on-Chip (SoC) FPGA, mô hình bao gồm lõi nhúng dual-core Cortex-A9 và FPGA fabric công nghiệp hàng đầu của Altera. Ngoài ra hệ thống Altera SoC còn tích hợp hệ thống xử lý cứng dựa trên ARM (HPS - Hardware Processor System) bao gồm bộ xử lý, thiết bị ngoại vi và giao diện bộ nhớ được kết nối liền mạch với kết cấu FPGA bằng cách sử dụng đường trực kết nối băng thông cao. Do đó có board DE1-SoC nhiều tính năng cho phép người dùng thiết kế nhiều loại mạch, từ các mạch đơn giản đến các dự án đa phương tiện khác nhau. Hình 2.17 và Hình 2.18 mô tả layout thực tế và block diagram của board DE1-SoC.



Hình 2.17: Layout thực của board DE1-SoC



Hình 2.18: Block diagram của board DE1-SoC

Các thành phần chính được cung cấp trên board DE1-SoC bao gồm:

- FPGA Device
 - + Cyclone V SoC 5CSEMA5F31C6 Device
 - + Dual-core ARM Cortex-A9 (HPS)
 - + 85K Programmable Logic Elements
 - + 4,450 Kbits embedded memory
 - + 6 Fractional PLLs
 - + 2 Hard Memory Controllers
- Thành phần bộ nhớ
 - + 64MB (32Mx16) SDRAM on FPGA
 - + 1GB (2x256Mx16) DDR3 SDRAM on HPS
 - + Micro SD Card Socket on HPS
- Thành phần giao tiếp
 - + Two USB 2.0 Host Ports (ULPI interface with USB type A connector) on HPS
 - + UART to USB (USB Mini B connector)
 - + 10/100/1000 Ethernet
 - + PS/2 mouse/keyboard
 - + IR Emitter/Receiver

2.12. Intel® FPGA SDK for OpenCL™

2.12.1. Giới thiệu

Intel® FPGA SDK for OpenCL™ là môi trường lập trình song song không đồng nhất dựa trên OpenCL cho phép các nhà phát triển phần mềm tăng tốc ứng dụng của trên nền tảng không đồng nhất là CPU Intel và FPGA. Intel® FPGA SDK for OpenCL™ cho phép bạn tận dụng đầy đủ các khả năng độc đáo của FPGA để mang lại hiệu suất tăng tốc với hiệu suất năng lượng và độ trễ thấp.

Đặc biệt, Intel FPGA SDK for OpenCL giúp cho người dùng sử dụng OpenCL – một framework lập trình song song cho các hệ thống không đồng nhất dựa trên ngôn

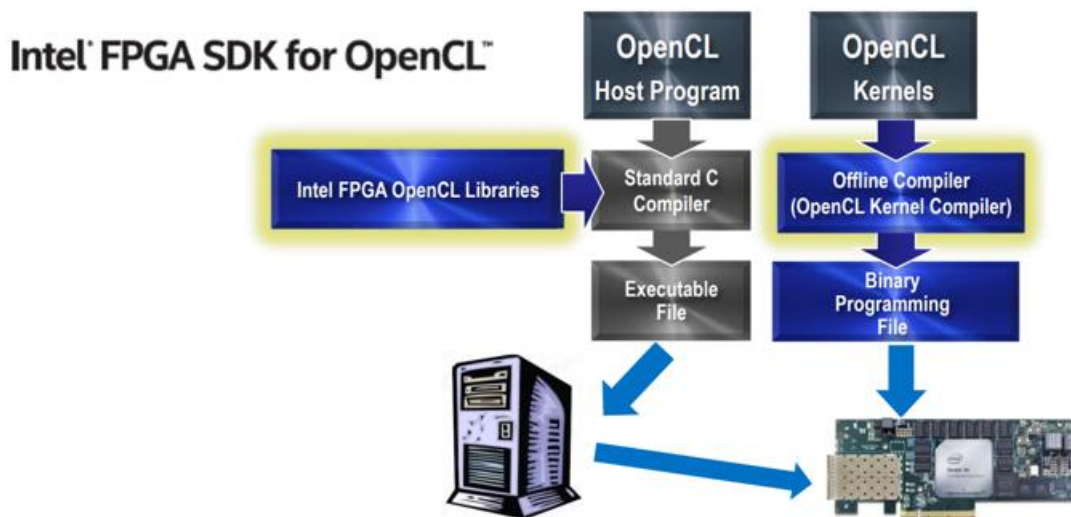
ngữ C) để thiết phần cứng thay vì sử dụng HDLs để lập trình lên các Intel FPGA. Công cụ chính của Intel FPGA SDK for OpenCL là Altera Offline Compiler (AOC), một trình biên dịch HLS (High Level Synthes) giúp biên dịch OpenCL kernel thành RTL (Register-transfer level) để từ đó tổng hợp chúng thông qua công cụ Quartus thành các FPGA bitstream và lập trình lên FPGA.

2.12.2. Quy trình programing FPGA sử dụng Intel® FPGA SDK for OpenCL™

Như đã đề cập ở phần đầu mục 2.10, cấu trúc của hệ thống trên board DE1-SoC bao gồm 2 thành phần:

- FPGA được sử dụng để tái cấu trúc phần cứng cho những ứng dụng cụ thể của thiết kế SoC.
- HPS dùng để thực thi phần mềm của thiết kế SoC.

Các ứng dụng sử dụng Intel® FPGA SDK cho OpenCL™ dùng để chạy trên FPGA và Intel CPU phải có hai file lập trình đó là OpenCL host program và OpenCL kernel. OpenCL host program sẽ giúp lập trình ứng dụng trên CPU để quản lý FPGA, trong khi OpenCL kernel dùng để thiết kế các phần tính trên được nạp lên FPGA.



Hình 2.19: Quy trình thiết kế và biên dịch của một ứng dụng sử dụng sử dụng Intel® FPGA SDK for OpenCL™

Quá thiết kế và biên dịch một ứng dụng sử dụng Intel® FPGA SDK for OpenCL™ được mô tả tại Hình 2.19. Tại file OpenCL host program trình biên dịch trên máy tính sẽ liên kết chương trình đó với các thư viện runtime của Intel® FPGA SDK for OpenCL™ để biên dịch file OpenCL host program ra các file thực thi (Exceutable File). Trong khi trình biên dịch AOC của Intel® FPGA SDK for OpenCL™ sẽ biên dịch OpenCL kernels thành các file bitstream (.aocx) dùng để programming FPGA.

Sau đó cả file thực thi của OpenCL host program và file bitstream (.aocx) của OpenCL kernels đều được được lên board Intel FPGA-SoC để chạy ứng dụng trên board.

Chương 3. XÂY DỰNG HỆ ĐIỀU HÀNH TÙY BIẾN LINUX CHO BOARD DE1-SOC

3.1. Tổng quan về hệ điều hành tùy biến linux cho board DE1-SoC

- Hệ điều hành armhf ubuntu 18.04 với nhân linux-socfpga phiên bản 3.18
- Tích hợp các chức năng
- LXDE (Lightweight X11 Desktop Environment)
- VNC (Virtual Network Computing)
- Intel® FPGA Runtime Environment (RTE) for OpenCL phiên bản 18.1
- Thư viện OpenCV phiên bản 2.4
- Hỗ trợ USB camera

3.2. Quy trình xây dựng linux SD Card image cho board DE1-SoC

3.2.1. Các công cụ cần thiết

Để tạo các file cần thiết cho hệ thống linux nhúng trên board DE1-SoC cần phải có các công cụ sau của Intel:

- Intel® Quartus® Prime Standard Edition hỗ trợ các thiết bị Cyclone V
- Intel® FPGA SDK for OpenCL™
- Intel® SoC FPGA Embedded Development Suite
- Intel® FPGA Runtime Environment for OpenCL™

Đồng thời quá trình tạo các file hệ thống được nhóm làm trên hệ điều hành Ubuntu và các bước thực hiện tương ứng với các mục dưới đây.

3.2.2. Tạo rbf (Raw Binary File)

File nhị phân với phần mở rộng là .rbf dùng để chứa các dữ liệu cấu hình để sử dụng bên ngoài phần mềm Quartus® Prime.

Ở đề tài này nhóm sử dụng các ví dụ về OpenCL (hello_world) được Terasic cung cấp trong các BSP (Board Support Package) for Altera SDK OpenCL để tạo ra các file rbf. Các BSP có thể được tải từ trang giới thiệu board DE1-SoC của Terasic.

BSP(Board Support Package) for Altera SDK OpenCL 18.1

Title	Version	Size	Date	Download
DE1-SoC OpenCL BSP(.zip)	1.0		2020-08-03	 
DE1-SoC OpenCL BSP(.tar.gz)	1.0		2020-08-03	 
DE1-SoC OpenCL User Manual	05		2020-08-03	

Hình 3.1: BSP(Board Support Package) for Altera SDK OpenCL tại trang chủ Terasic

Để tạo ra file rbf nhóm sử dụng AOC để biên dịch ví dụ hello_world bằng câu lệnh dưới đây:

```
aoc device/hello_world.cl -o bin/ hello_world.aocx -report -no-interleaving=default
```

Các việc thiết lập và hướng dẫn chi tiết về cách sử dụng AOC được hướng dẫn chi tiết trong hướng dẫn [1] .

Sau quá trình biên dịch sẽ đồng, AOC sẽ đồng thời tạo ra file rbf có tên là top.rbf bên cạnh quá trình biên dịch ví dụ hello_world.

3.2.3. Tạo file Preloader - preloader-mkpimage.bin

Chức năng của preloader sẽ được người dùng định nghĩa tuy nhiên các chức năng chính của preloader là:

- Khởi tạo giao tiếp SDRAM.
- Cấu hình các HPS I/O pin.
- Tải giai đoạn boot tiếp theo từ Flash sang SDRAM và đi tới giai đoạn đó.

Ở mục 3.2.2 sau quá trình biên dịch ví dụ hello_world thì thư mục hps_isw_handoff sẽ được tạo ra. Thư mục này được tạo ra mỗi khi một hệ thống SoC được tạo ra với Qsys (Platform designer) và được biên dịch bởi phần mềm Quartus. Thư mục hps_isw_handoff chứa các thông tin cấu hình cần thiết cho Preloader để cấu hình các clock, IO và bộ đồ khiển SDRAM.

Để tạo ra file preloader nhóm sử dụng sử dụng công cụ Preloader Generator (BSP Editor) có ở trong embedded command shell của Intel® SoC FPGA là Embedded Development Suite (EDS) với câu lệnh sau.

```
bsp-create-settings \  
  --type spl \  
  --bsp-dir software/spl_bsp \  
  --preloader-settings-dir "hps_isw_handoff/system_acl_iface_hps" \  
  --settings software/spl_bsp/settings.bsp
```

Sau khi chạy câu lệnh trên file preloader sẽ được tạo ra với tên preloader-mkpimage.bin.

3.2.4. Tạo file u-boot - u-boot.img.

Giai đoạn chính tiếp theo là bộ tải khởi động mã nguồn mở, U-boot - một bootloader cho board nhúng. Mục đích của U-Boot là để hệ thống có thể khởi động được nhân Linux (giống như cách mà preloader thiết lập hệ thống để chạy U-Boot).

Để tạo file u-boot, di chuyển đến thư mục spl_bsp ở mục 3.2.3, sau đó dùng lệnh make để tạo ra file u-boot như sau.

```
make uboot
```

Sau khi chạy xong file u-boot có tên là u-boot.img sẽ được tạo ra.

3.2.5. Tạo zImage

zImage là một phiên bản nén của Linux kernel image có thể tự giải nén được sử dụng bởi ARM Linux. Để tạo zImage, nhóm sử dụng linux-socfpga, một mã nguồn mở của altera và được cộng đồng hỗ trợ dùng để build linux kernel cho các board soc-fpga để tạo zImage cho đề tài.

Do image có tích hợp môi trường cho OpenCL được terasic cung cấp có linux kernel là 3.18 và chạy ổn định nên nhóm quyết định sẽ chọn linux-socfpga phiên bản 3.18 để build zImage.

Chi tiết các bước để tạo ra zImage được hướng dẫn chi tiết tại [2].

3.2.6. Tạo dtb (Device Tree Blob File)

Dtb (Device Tree Blob) file được sử dụng bởi linux kernel. Nó chứa dữ liệu nhị phân mô tả phần cứng của máy tính. File DTB cho phép hệ điều hành quản lý các thành phần của máy tính bằng cách cho hệ điều hành biết máy tính bao gồm phần cứng nào. Ở đây do mã nguồn mở linux-socfpga sử dụng ở mục 3.2.5 có hỗ trợ tạo file DTB nên tại thư mục linux-socfpga chỉ cần dùng lệnh make để tạo file DTB như sau.

```
make socfpga_cyclone5_de1soc.dtb
```

Sau khi chạy lệnh trên file socfpga_cyclone5_de1soc.dtb sẽ được tạo ra.

3.2.7. Xây dựng hệ thống file root (root filesystem)

Ở đề tài này nhóm build root filesystem bằng cách dựa trên các base root filesystem của armhf-Ubuntu-18.04.5. Base root filesystem của armhf-Ubuntu-18.04.5 có thể tải tại: <https://cdimage.ubuntu.com/>.

Để tránh ảnh hưởng tới hệ thống của máy tính khi tùy chỉnh root file system nhóm sử dụng chroot lên thư mục rootfs của armhf-Ubuntu-18.04.5. Và sử dụng trình giả lập QEMU với mode User mode emulation để cài đặt các gói cần thiết bên trong chroot. Mode này sẽ giúp biên dịch một CPU trên một CPU khác.

Sau khi vào môi trường giả lập QEMU và cài đặt các utils cần thiết thì tiếp theo sẽ là tiến hành cài đặt gói thư viện OpenCV. Các bước của quá trình được mô tả chi tiết tại [3]. Ở trong khóa luận này nhóm cài đặt thư viện OpenCV phiên bản 2.4

3.2.8. Xây dựng OpenCL driver

Để build driver hỗ trợ OpenCL trên board DE1-SoC, nhóm sử dụng gói package Intel cung cấp dùng để build các driver và chứa các thư viện cần thiết hỗ trợ OpenCL là Intel FPGA Runtime Environment for OpenCL Linux ARM SoC TGZ. Gói package trên có thể tải tại trang chủ Intel .

Sau khi tải gói package về, quá trình biên dịch và tạo ra OpenCL driver được hướng dẫn trong [4].

Sau khi OpenCL driver được tạo ra, copy thư chứa OpenCL driver vào thư trong root filesystem đã build ở mục 3.2.7 để sử dụng.

3.2.9. Xây dựng file U-Boot Script

Việc sử dụng u-boot cho phép người dùng cấu hình FPGA từ u-boot. Điều này có thể được thực hiện bằng các command củ u-boot hoặc từ một u-boot script. Ở đây nhóm sẽ sử dụng u-boot script để cấu hình FPGA từ file rbf, tải file dtb, tải root file system và tải linux kernel. Nội dung của script như sau:

```
fatload mmc 0:1 $fpgadata opencl.rbf;
fpga load 0 $fpgadata $filesize;
run bridge_enable_handoff;
run mmcload;
run mmcboot;
```

Để biên dịch và tạo thành file u-boot.src, trước tiên cần tạo 1 file văn bản có tên boot.script có nội dung như trên, sau đó mở embedded command shell (tham khảo mục 3.2.3) và sử dụng lệnh sau.

```
mkimage -A arm -O linux -T script -C none -a 0 -e 0 -n "U-boot
script" -d boot.script u-boot.scr
```

Sau lệnh trên file U-boot script được tạo ra có tên là u-boot.scr

3.2.10. Tạo SD Card image

Có nhiều cách để tạo SD card image nhưng ở đề tài nhóm sẽ sử dụng một python script được cung cấp trên trang web <https://rocketboards.org/> - một trang web hỗ trợ và cung cấp các tài liệu, công cụ cho việc phát triển Linux trên Altera SoC FPGAs.

Trước tiên, cần tiến hành copy tất cả các file đã tạo ở trên vào cùng thư mục chứa đoạn python script và sửa tên file như mục 2.1. Sau đó dùng đoạn python script để tạo SD Card image bằng câu lệnh sau.

```
sudo python3 ./make_sdimage_p3.py -f \  
-P preloader-mkpimage.bin,num=3,format=raw,size=10M,type=A2 \  
-P zImage,u-boot.scr,opencl.rbf,socfpga.dtb,u-boot-  
img,num=1,format=fat32,size=300M \  
-P rootfs/*,num=2,format=ext4,size=6000M \  
-s 6500M \  
-n sdcard.img
```

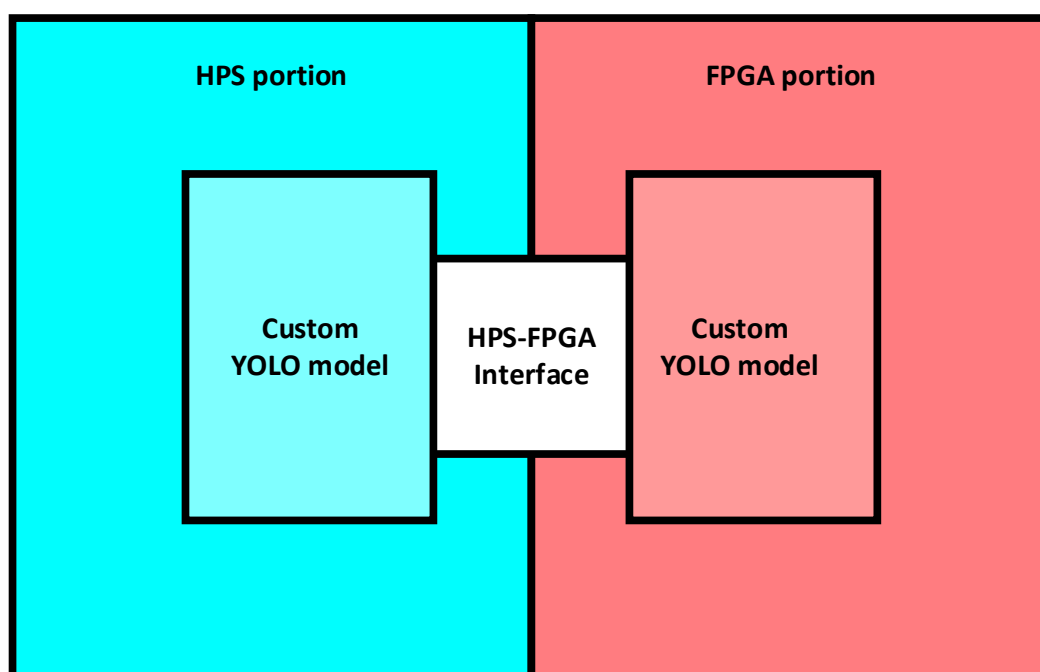
Cuối cùng ta sẽ thu được image để flash lên thẻ nhớ có tên là sdcard.img.

Chương 4. THIẾT KẾ VÀ HIỆN THỰC MÔ HÌNH CUSTOM YOLO TRÊN DE1-SOC SỬ DỤNG OPENCL

4.1. Hệ thống SoC-FPGA của đề tài

4.1.1. Tổng quan hệ thống

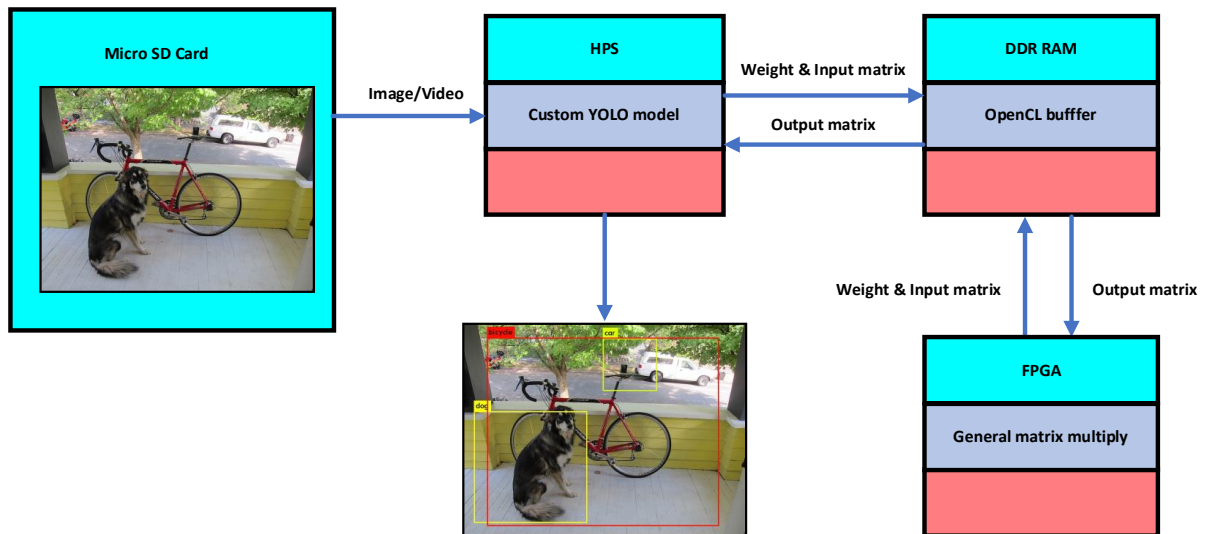
Như mục 3.1.2 đã đề cập các ứng dụng sử dụng Intel® FPGA SDK cho OpenCL™ có hai thành phần chính: FPGA programming bitstream(s) và host program. Nên kiến trúc hệ thống SoC-FPGA của nhóm đề xuất như hình dưới đây.



Hình 4.1: Kiến trúc hệ thống SoC-FPGA

- FPGA programming bitstream(s) sẽ có chức năng program FPGA để thực thi phép nhân ma trận (General matrix multiply).
- Host program sẽ đảm nhận việc thực thi mạng CNN trên HPS và truyền data để FPGA thực hiện phép nhân ma trận.

4.1.2. Luồng hiện thực hệ thống SoC trong nhận diện đối tượng



Hình 4.2: Luồng thiết kế của mô hình chạy trên board DE1-SoC

Hình 5.1 mô tả workflow của mô hình khi chạy trên board DE1-SoC. Với input là một ảnh hoặc video được lưu trên SD Card, sau đó được tải vào mô hình custom YOLO đang nằm ở HPS portion. Sau khi được convert từ ảnh sang dạng ma trận số, tại lớp convolution các ma trận input, weight được các OpenCL buffer được hỗ trợ lưu vào ram. Tiếp theo bộ nhân 2 ma trận được nạp trên FPGA sẽ lấy 2 ma trận là ma trận input và ma trận weight được lưu trên ram để thực hiện phép nhân sau đó chuyển kết quả phép toán là 1 ma trận output về lại ram và được OpenCL buffer chuyển về lại mô hình nằm ở HPS portion để thực hiện các công đoạn tiếp theo của mạng CNN như đi qua các hàm activation, qua các lớp maxpooling và sau đó lại làm input cho phép toán nhân ma trận ở lớp convolution cho đến khi tới lớp cuối cùng là lớp region để cho ra output là các hình ảnh, video được detect.

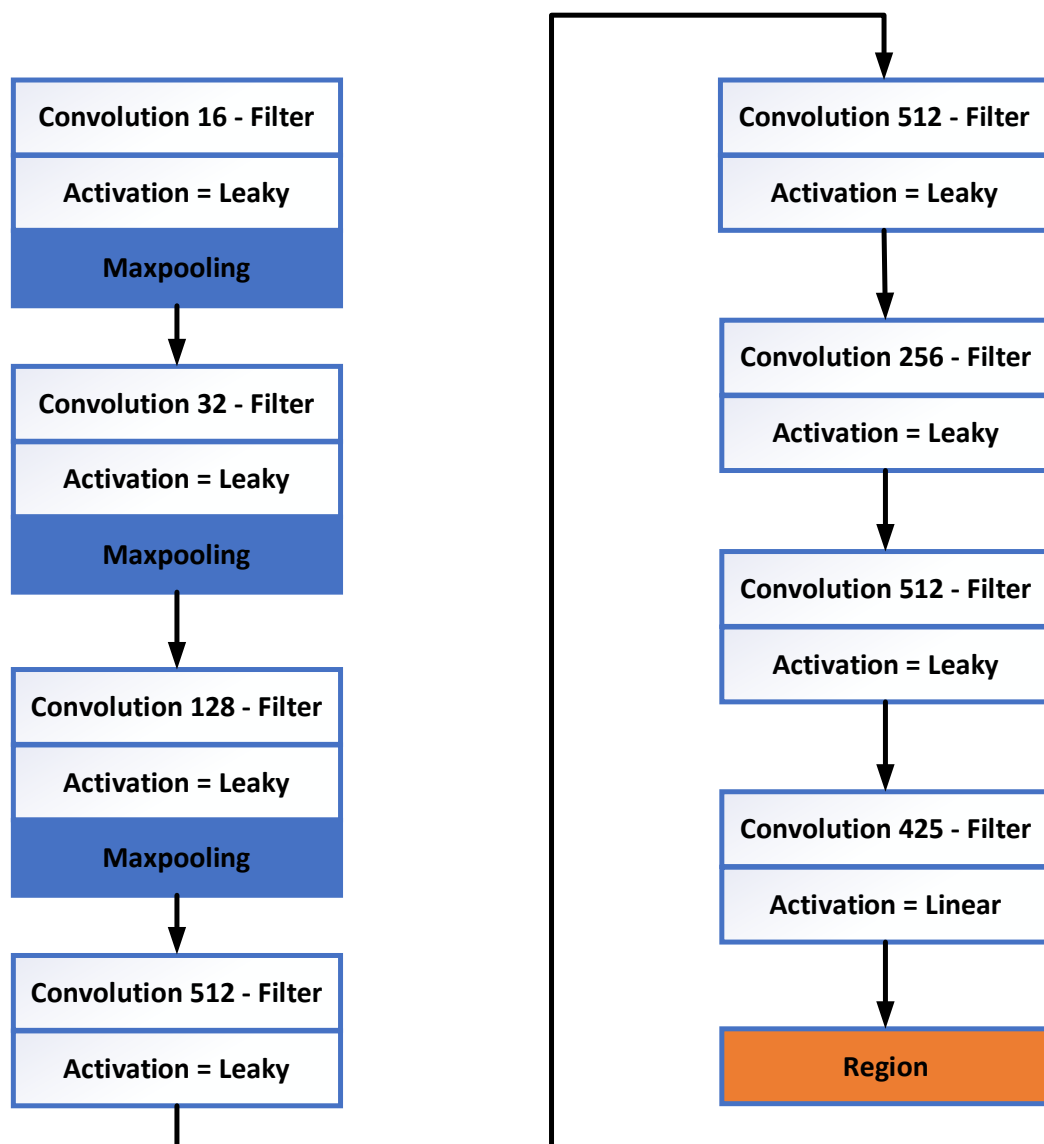
4.2. Thiết kế mô hình YOLO tùy biến trên HPS portion

Với mô hình Tiny-yolo-v2 mô hình này sử dụng 9 layer cho mô hình và đầu vào ảnh là 416x416. Vì vậy mô hình sẽ tốn nhiều tài nguyên để xử lý việc tính toán.

Nhóm sử dụng board DE1-SoC do đó tài nguyên của board khá là hạn chế vì vậy cần phải tùy chỉnh lại mô hình để hiện thực mô hình xuống board bên cạnh đó việc

custom mô hình giúp cho việc xử lý được tối ưu về mặt thời gian cũng như quá trình thực thi hơn so với mô hình ban đầu.

Nhóm đã quyết định giảm bớt một layer và giảm ảnh đầu vào thành 224x160 để mô hình tối ưu và phù hợp hơn với board DE1-SoC, Nhóm gọi mô hình này là mô hình custom tiny-yolo đây là mô hình nhóm đã chỉnh sửa lại kiến trúc để có thể chạy được trên tài nguyên của board DE1-SoC dựa trên framework Darknet dùng để giải quyết các bài toán về object detection. Mô hình của nhóm đề xuất được mô tả dưới hình sau đây.



Hình 4.3: Hình ảnh kiến trúc mô hình custom YOLO

Mô hình bao gồm tổng cộng 12-layer trong đó bao gồm 8 layer convolution, 3 layer max pooling và 1 lớp region. Chi tiết các lớp được mô tả tại Bảng 4.1

Bảng 4.1: Các thông số của mạng custom YOLO

Layer	Name	Filter	Size / Stride	Input	Output
0	Conv	16	3 x 3 / 1	224 x 160 x 3	224 x 160 x 16
1	Max		2 x 2 / 2	224 x 160 x 16	112 x 180 x 16
2	Conv	32	3 x 3 / 1	112 x 180 x 16	112 x 180 x 32
3	Max		4 x 4 / 4	112 x 180 x 32	28 x 20 x 32
4	Conv	128	3 x 3 / 1	28 x 20 x 32	28 x 20 x 128
5	Max		4 x 4 / 4	28 x 20 x 128	7 x 5 x 128
6	Conv	512	3 x 3 / 1	7 x 5 x 128	7 x 5 x 512
7	Conv	512	3 x 3 / 1	7 x 5 x 512	7 x 5 x 512
8	Conv	256	1 x 1 / 1	7 x 5 x 512	7 x 5 x 256
9	Conv	512	3 x 3 / 1	7 x 5 x 256	7 x 5 x 512
10	Conv	425	1 x 1 / 1	7 x 5 x 512	7 x 5 x 425
11	Region				

4.3. Thiết kế bộ nhân ma trận (General matrix multiply) trên FPGA portion.

Mạng thần kinh sâu (DNN) yêu cầu rất lớn số lượng tính toán cho cả đào tạo và suy luận khi triển khai tại hiện trường. Một cách tiếp cận chung để thực hiện DNN là để viết lại các hoạt động tính toán tốn kém nhất như phép nhân ma trận tổng quát (GEMM). Các thuật toán này làm giảm đáng kể chi phí không gian của DNN chập, làm cho nó phù hợp hơn nhiều cho bộ nhớ hạn chế những hệ thống.

Thuật toán GEMM khá cần thiết trong phép tích chập convolution. Mục đích khóa luận của chúng tôi sử dụng bộ tăng tốc FPGA dựa trên OpenCL cho thuật toán

GEMM. Framework Darknet có các chức năng GEMM được viết bởi ngôn ngữ C nhưng nó là mã mẫu vì vậy nó quá chậm trong việc xử lý các phép toán.

Những vấn đề chính của phương pháp này là

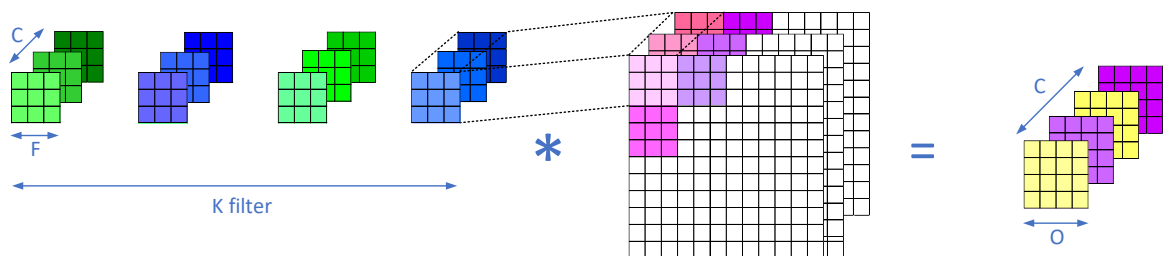
- Không gian làm việc bộ nhớ lớn cần thiết để lưu trữ các ma trận trung gian được tạo ra bởi biến đổi im2col/im2row
- Thời gian để thực hiện biến đổi im2col/im2row, không đáng kể đối với các mạng nơ-ron phức tạp.

Do đó dựa trên hàm GEMM mẫu của Darknet được viết bằng ngôn ngữ C chúng tôi triển khai OpenCL.

Nhóm thực hiện mô tả OpenCL kernel và biên dịch với AOC (Altera OpenCL Compiler)

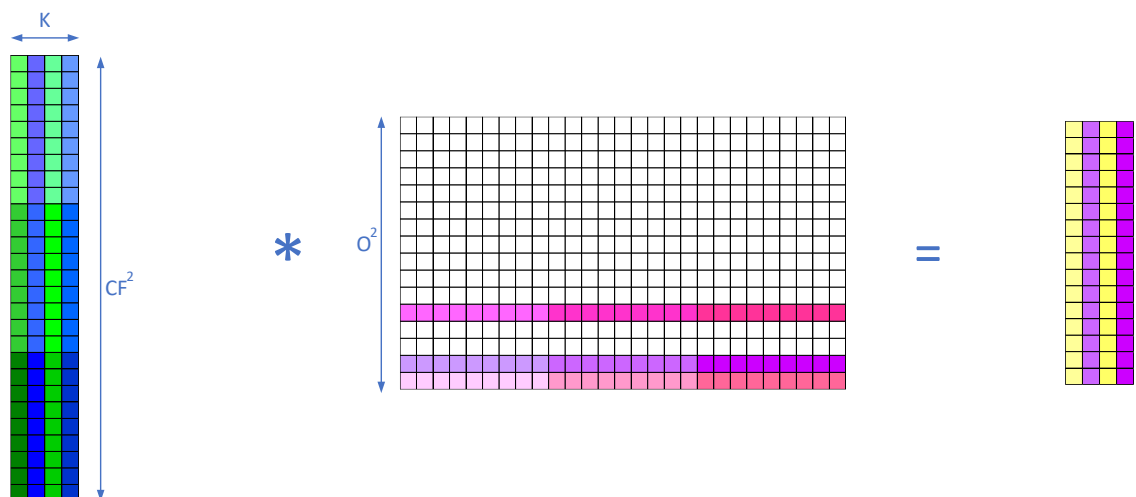
Ngoài ra chúng tôi sử dụng một dạng chuẩn FP16 ieee-754-2008 binary16 để tối ưu quá trình tính toán trong phép nhân so với floating point 32 bit. Trình biên dịch ARM gcc compiler hỗ trợ định dạng half precision floating point.

Phép toán sử dụng tensor convolution được thể hiện dưới Hình 4.4.



Hình 4.4: Tensor convolution

Biểu diễn ma trận và phép nhân xếp lớp bằng cách sử dụng thuật toán GEMM. Với thuật toán GEMM hiện thực trên FPGA thì chúng tôi sử dụng hàm im2row thay vì hàm im2col chung hay sử dụng. Hàm chung im2col sao chép và mở rộng dữ liệu hình ảnh thành ma trận cột. Hình 4.5 dưới đây minh họa cho thuật toán GEMM.



Hình 4.5: Thuật toán GEMM

4.4. Huấn luyện mô hình custom yolo

4.4.1. Chuẩn bị dữ liệu

Nhóm sử dụng hai bộ dataset cho hai mục đích khác nhau trong đó tập dữ liệu ImageNet dùng để phân loại đối tượng (image classification) và hai tập dữ liệu VOC dataset, COCO dataset dùng để định vị vật thể (object localization). Hai giai đoạn này kết hợp lại gọi chung là phát hiện đối tượng (object detection). Quá trình này được thể hiện bằng cách vẽ một hộp giới hạn (bounding box) xung quanh từng đối tượng quan tâm trong ảnh và gán cho chúng một nhãn.

Với tập dữ liệu VOC dataset thì chỉ gồm hai mươi đối tượng vì vậy nhóm muốn mở rộng việc phát hiện nhiều đối tượng hơn nên COCO dataset là một sự lựa chọn phù hợp.

ImageNet là một tập dữ liệu rất nổi tiếng trong lĩnh vực computer vision. Nó thực sự là một tập dữ liệu khổng lồ với 14,197,122 ảnh và 1000 class. Nó là tập dữ liệu được sử dụng để tổ chức các competition hàng năm và là một tập dữ liệu tiêu chuẩn để đo benchmark cho các thuật toán phân loại hình ảnh mới được ra đời.

VOC dataset là một dataset nổi tiếng cho bài toán object detection, classification và segmentation. Đã có 8 cuộc thi sử dụng PASCAL VOC trong khoảng từ 2005 tới 2012. PASCAL VOC có khoảng 10000 ảnh cho training và validation bao gồm

bounding box và object detect. Mặc dù chỉ có khoảng 20 categories, nó vẫn được dùng như một dataset tham chiếu cho các mô hình Deep Learning.

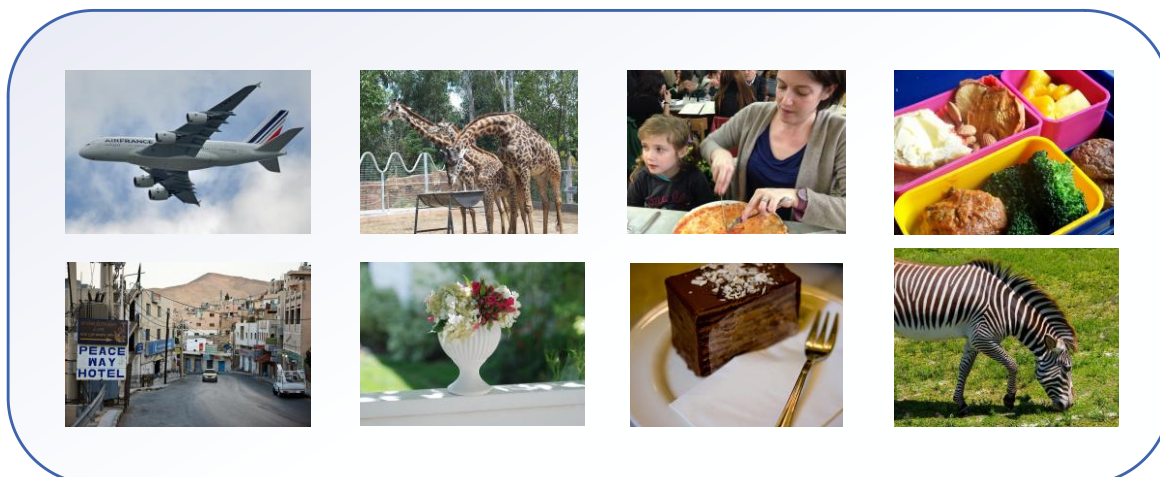
COCO dataset là một tập datasets phục vụ cho các bài toán Object Detection, Segmentation, Image Captioning. Tập dữ liệu tổng cộng có khoảng 1.5 triệu objects thuộc về 80 class khác nhau

Nhóm sử dụng tập dữ liệu COCO dataset với tập train gồm 117000 ảnh, tập valid gồm 5000 ảnh để nhận diện 80 labels. Bảng 2.1 thể hiện số lượng nhãn của 2 tập dữ liệu VOC dataset và COCO dataset.

Bảng 4.2: Thể hiện số lượng nhãn trong tập dữ liệu VOC và COCO

Labels VOC data	Labels COCO data
20 labels	80 labels
aeroplane, bicycle, bird, boat, bottle, bus, car, cat, chair, cow, diningtable, dog, horse, motorbike, person, pottedplant, sheep, sofa, train, tvmonitor	person, bicycle, car, motorbike, aeroplane, bus, train, truck, boat, traffic light, fire hydrant, stop sign, parking meter, bench, bird, cat, dog, horse, sheep, cow, elephant, bear, zebra, giraffe, backpack, umbrella, handbag, tie, suitcase, frisbee, skis, snowboard, sports ball, kite, baseball bat, baseball glove, skateboard, surfboard, tennis racket, bottle, wine glass, cup, fork, knife, spoon, bowl, banana, apple, sandwich, orange, broccoli, carrot, hot dog, pizza, donut, cake, chair, sofa, pottedplant, bed, diningtable, toilet, tvmonitor, laptop, mouse, remote, keyboard, cell phone, microwave, oven, toaster, sink, refrigerator, book, clock, vase, scissors, teddy bear, hair drier, toothbrush

Hình 4.6 dưới đây thể hiện một số labels trong tập train COCO dataset.



Hình 4.6: Một số labels trong tập train COCO dataset

4.4.2. Huấn luyện mô hình

Mô hình được huấn luyện dựa trên framework Darknet. Darknet là mạng thần kinh mã nguồn mở được viết bằng C và CUDA. Nó nhanh chóng, dễ cài đặt và hỗ trợ tính toán CPU và GPU. Vì vậy nhóm sử dụng framework darknet để huấn luyện mô hình.

Nhóm sử dụng GPU Geforce GTX 1050, OpenCV để huấn luyện mô hình custom YOLO.

- Cấu hình darknet
 - + Tạo file config data:
 - + File config data sẽ khai báo một số thông tin như sau:
 - Số lượng classes
 - Đường dẫn tới các file train.txt, test.txt
 - Đường dẫn tới file obj.names
 - Thư mục backup mô hình huấn luyện.
 - + Tạo file config model

Đây là bước quan trọng nhất khi huấn luyện model YOLO. Chúng ta sẽ sử dụng file “.cfg” để cấu hình mô hình huấn luyện.

File “.cfg” cần được chỉnh sửa các thông số phù hợp cho quá trình huấn luyện mô hình như : batch, subdivisions, max_batches, width, height, classes và số filter dựa

trên số classes. Ở framework darknet này công thức tính được thể hiện sau đây:
 $\text{filters} = (\text{classes} + 5) \times 3$. Sau đó để huấn luyện mô hình sử dụng lệnh “detector train”.

Mô hình huấn luyện hơn 1 tuần với số lượng parameter của mô hình là 4521737 (4521737x4) Mb cho file weights.

Chương 5. KẾT QUẢ VÀ ĐÁNH GIÁ

5.1. Kết quả training mô hình Custom YOLO

Sau quá trình huấn luyện dữ liệu để nhận diện 80 lớp đối tượng sử dụng framework Darknet thì nhóm thu được file weight kết quả thể hiện như sau

Mean average precision (mAp) của mô hình đạt được khoảng 13.54 %. Bảng 5.1 thể hiện độ chính xác trung bình của một số labels.

Bảng 5.1: Thể hiện độ chính xác AP của từng labels

class_id = 0	name = person	ap = 16.82%
class_id = 1	name = bicycle	ap = 1.14%
class_id = 2	name = car	ap = 1.56%
class_id = 3	name = motorbike	ap = 22.08%
class_id = 4	name = aeroplane	ap = 37.70%
class_id = 5	name = bus	ap = 37.13%
class_id = 6	name = train	ap = 38.28%
class_id = 7	name = truck	ap = 4.66%
class_id = 8	name = boat	ap = 2.99%
class_id = 9	name = traffic light	ap = 0.06%
class_id = 10	name = fire hydrant	ap = 51.06%
class_id = 11	name = stop sign	ap = 36.67%
class_id = 12	name = parking meter	ap = 14.29%
class_id = 13	name = bench	ap = 4.45%
class_id = 14	name = bird	ap = 6.96%
class_id = 15	name = cat	ap = 36.75%
class_id = 16	name = dog	ap = 20.07%
class_id = 17	name = horse	ap = 22.92%
class_id = 18	name = sheep	ap = 6.27%
class_id = 19	name = cow	ap = 30.58%

class_id = 20	name = elephant	ap = 45.88%
class_id = 21	name = bear	ap = 63.69%
class_id = 22	name = zebra	ap = 48.29%
class_id = 23	name = giraffe	ap = 51.24%
class_id = 24	name = backpack	ap = 0.07%
class_id = 25	name = umbrella	ap = 5.57%
class_id = 26	name = handbag	ap = 0.01%
class_id = 27	name = tie	ap = 1.59%
class_id = 28	name = suitcase	ap = 7.59%
class_id = 29	name = frisbee	ap = 10.13%
class_id = 30	name = skis	ap = 3.81%
class_id = 31	name = snowboard	ap = 8.75%
class_id = 32	name = sports ball	ap = 1.71%
class_id = 33	name = kite	ap = 5.02%
class_id = 34	name = baseball bat	ap = 0.08%
class_id = 35	name = baseball glove	ap = 0.00%
class_id = 36	name = skateboard	ap = 3.28%
class_id = 37	name = surfboard	ap = 9.85%
class_id = 38	name = tennis racket	ap = 4.09%
class_id = 39	name = bottle	ap = 1.28%
class_id = 40	name = wine glass	ap = 5.00%
class_id = 41	name = cup	ap = 2.63%
class_id = 42	name = fork	ap = 2.53%
class_id = 43	name = knife	ap = 0.71%
class_id = 44	name = spoon	ap = 0.77%
class_id = 45	name = bowl	ap = 15.80%
class_id = 46	name = banana	ap = 6.78%

class_id = 47	name = apple	ap = 0.88%
class_id = 48	name = sandwich	ap = 16.11%
class_id = 49	name = orange	ap = 1.31%
class_id = 50	name = broccoli	ap = 11.58%
class_id = 51	name = carrot	ap = 2.67%
class_id = 52	name = hot dog	ap = 8.32%
class_id = 53	name = pizza	ap = 48.79%
class_id = 54	name = donut	ap = 5.55%
class_id = 55	name = cake	ap = 17.80%
class_id = 56	name = chair	ap = 3.71%
class_id = 57	name = sofa	ap = 16.91%
class_id = 58	name = pottedplant	ap = 2.74%
class_id = 59	name = bed	ap = 50.26%
class_id = 60	name = diningtable	ap = 29.84%
class_id = 61	name = toilet	ap = 29.87%
class_id = 62	name = tvmonitor	ap = 22.81%
class_id = 63	name = laptop	ap = 18.23%
class_id = 64	name = mouse	ap = 0.03%
class_id = 65	name = remote	ap = 2.22%
class_id = 66	name = keyboard	ap = 3.48%
class_id = 67	name = cell phone	ap = 0.85%
class_id = 68	name = microwave	ap = 2.50%
class_id = 69	name = oven	ap = 3.73%
class_id = 70	name = toaster	ap = 50.00%
class_id = 71	name = sink	ap = 8.42%
class_id = 72	name = refrigerator	ap = 19.82%
class_id = 73	name = book	ap = 0.51%

class_id = 74	name = clock	ap = 14.93%
class_id = 75	name = vase	ap = 11.77%
class_id = 76	name = scissors	ap = 4.77%
class_id = 77	name = teddy bear	ap = 14.88%
class_id = 78	name = hair drier	ap = 0.00%
class_id = 79	name = toothbrush	ap = 0.89%

Bảng 5.2 dưới đây thể hiện mAP của mô hình Tiny YOLO được công bố trên trang chủ Darknet sử dụng tập dữ liệu COCO dataset.

Bảng 5.2: Hiệu suất mô hình Tiny YOLO tập dữ liệu COCO

Model	Train	Test	mAP	FLOPS	FPS
Tiny YOLO	COCO trainval	Test-dev	23.7	5.41 Bn	244

Từ Bảng 5.3 độ chính xác của mô hình tuy giảm nhưng việc số lượng phép toán của mô hình giảm hơn 2.5 lần số phép toán của mô hình gốc vì vậy không tránh khỏi việc độ chính xác của mô hình bị giảm.

Bảng 5.3: mAP và FLOPS của Tiny Yolov2 và Custom Yolo

	Tiny YOLO	Custom Yolo
mAP	23.7 %	13.54 %
FLOPS	5.41 Bn	2.148 Bn

5.2. Kết quả thực nghiệm mô hình

5.2.1. Kết quả chạy mô hình với ảnh

Để kiểm tra độ chính xác của mô hình có đáp ứng mục tiêu đưa ra hay không nhóm sử dụng tập dữ liệu test gồm 1000 tấm ảnh ngẫu nhiên để cho 2 mô hình Tiny Yolov2 và mô hình Custom Yolo của nhóm dự đoán. Bảng 5.2 thể hiện kết quả đánh giá giữa 2 mô hình.

Bảng 5.4: IoU và Recall của Tiny-Yolov2 vs Custom-Yolo

	Tiny Yolo v2	Custom Yolo
Recall	51.62473 %	23.40193 %
IoU	47.33565 %	29.59615 %

Tiếp theo để kiểm tra tốc độ thực thi của mô hình, nhóm sử dụng tập dữ liệu test gồm 1000 tấm ảnh để dự đoán của mô hình trên Intel Core i5 Gen 8th và DE1-SoC. Hình 5.1 là kết quả khi dự đoán Intel CPU và Hình 5.2 là kết quả chạy mô hình trên DE1-SoC.

```

classes = 80
train = /home/tnt/darknet_sd1_v2/data/coco/trainvalno5k.txt
valid = /home/tnt/darknet_sd1_v2/data/coco/1k.txt
names = data/coco.names
backup = COCO_WEIGHTS

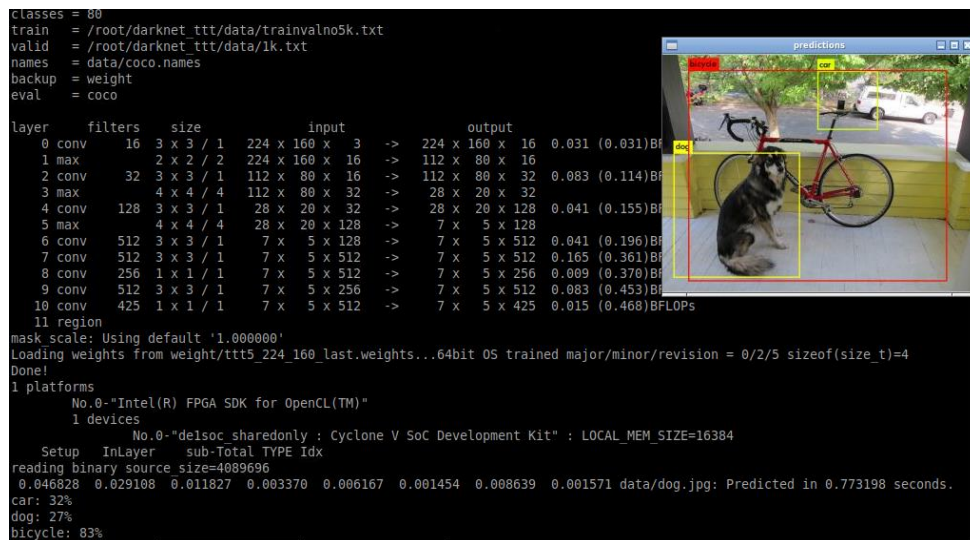
layer      filters  size      input      output      0.031 (0.031)BFLOPs
0 conv     16  3 x 3 / 1  224 x 160 x 3  -> 224 x 160 x 16
1 max       2  2 x 2 / 2  224 x 160 x 16  -> 112 x 80 x 16
2 conv     32  3 x 3 / 1  112 x 80 x 16  -> 112 x 80 x 32  0.083 (0.114)BFLOPs
3 max       4  4 x 4 / 4  112 x 80 x 32  -> 28 x 20 x 32
4 conv    128  3 x 3 / 1  28 x 20 x 32  -> 28 x 20 x 128  0.041 (0.155)BFLOPs
5 max       4  4 x 4 / 4  28 x 20 x 128  -> 7 x 5 x 128
6 conv    512  3 x 3 / 1  7 x 5 x 128  -> 7 x 5 x 512  0.041 (0.196)BFLOPs
7 conv    512  3 x 3 / 1  7 x 5 x 512  -> 7 x 5 x 512  0.165 (0.361)BFLOPs
8 conv    256  1 x 1 / 1  7 x 5 x 512  -> 7 x 5 x 256  0.009 (0.370)BFLOPs
9 conv    512  3 x 3 / 1  7 x 5 x 256  -> 7 x 5 x 512  0.083 (0.453)BFLOPs
10 conv   425  1 x 1 / 1  7 x 5 x 512  -> 7 x 5 x 425  0.015 (0.468)BFLOPs
11 region

mask_scale: Using default '1.000000'
Loading weights from weights/ttt5_224_160_last.weights...04bit 05 trained major/minor/revision = 0/2/5 sizeof(size_t)=8
Done!
0.000000 emulator_model:gemm_emu.aocx
1 platforms
No.0-"Intel(R) FPGA SDK for OpenCL(TM)"
1 devices
No.0-"EmulatorDevice : Emulated Device" : LOCAL_MEM_SIZE=16384
Support SVM_COARSE_GRAIN
Support SVM_FINE_GRAIN
Setup InLayer sub-Total TYPE Idx
reading binary source_size=1370028
CL_DEVICE_MAX_CLOCK_FREQUENCY: 1000
0.029297 0.029297 0000 000
0.000000 0.002155 0.031451 0003 001
0.000000 0.030866 0.068318 0000 002
0.000000 0.000623 0.068940 0003 003
0.000000 0.009426 0.078366 0000 004
0.000000 0.000149 0.078515 0003 005
0.000000 0.018039 0.097354 0000 006
0.000000 0.150031 0.253385 0000 007
0.000000 0.001953 0.255338 0000 008
0.000000 0.045890 0.301229 0000 009
0.000000 0.003564 0.304792 0000 010
0.000000 0.000154 0.304946 0022 011
data/dog.jpg: Predicted in 0.320573 seconds.
car: 32%
dog: 27%
bicycle: 83%

```

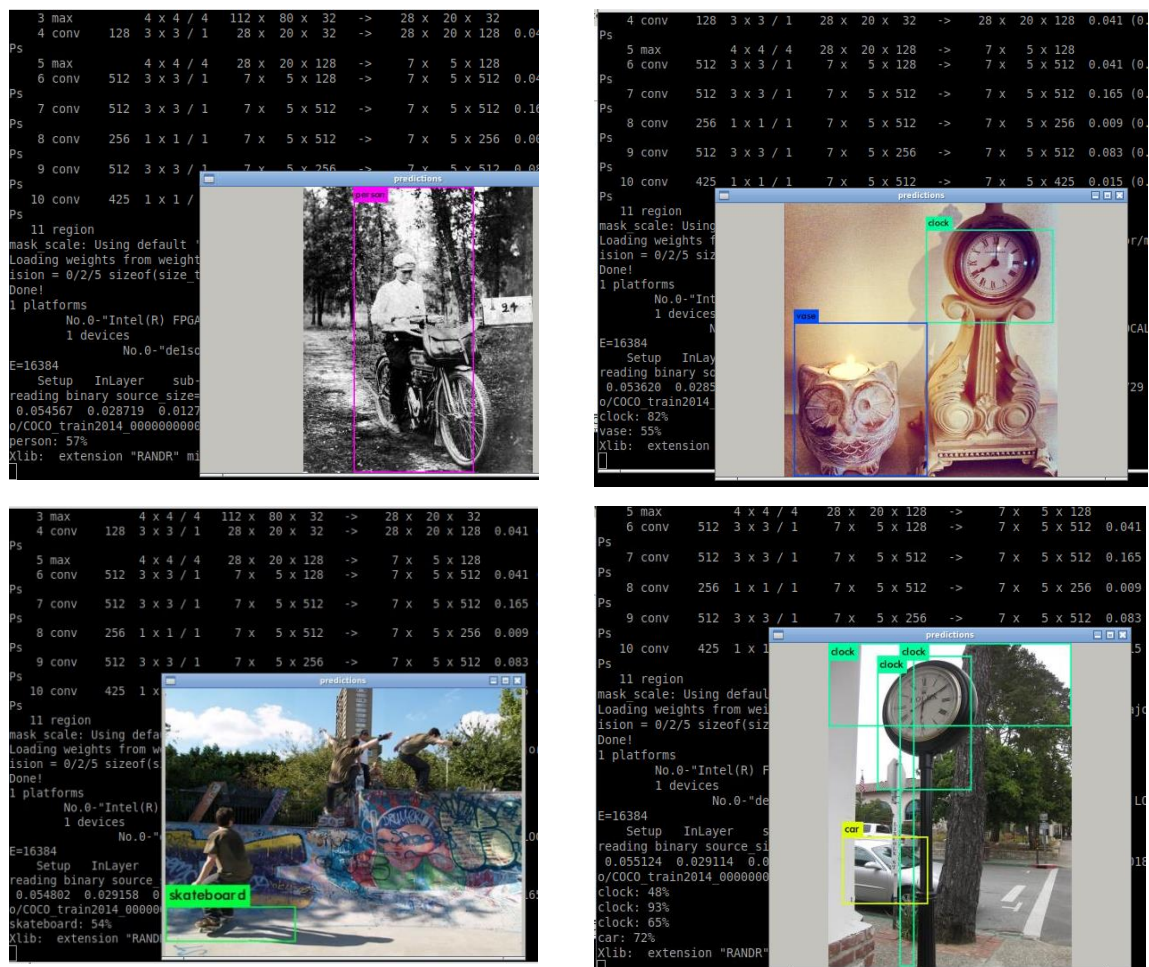


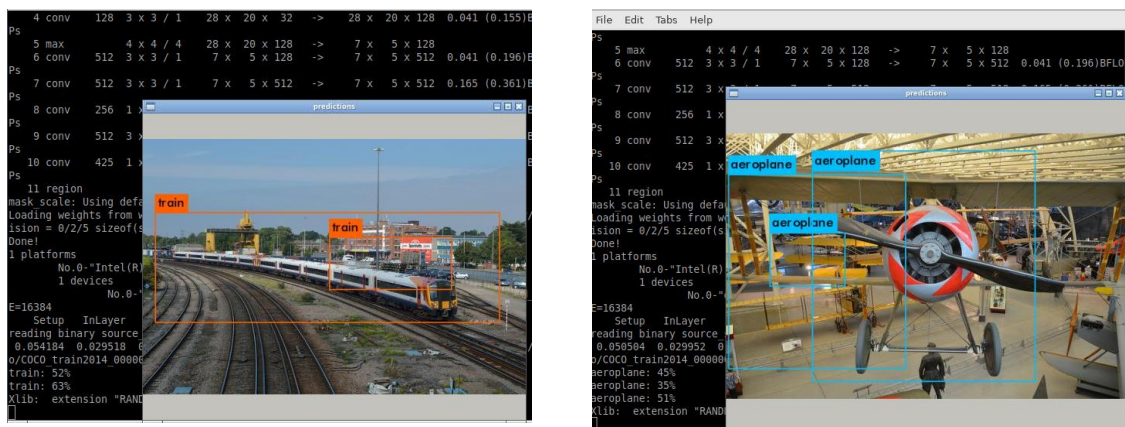
Hình 5.1: Kết quả chạy mô hình trên Intel CPU



Hình 5.2: Kết quả chạy mô hình trên Board DE1-SoC

Hình 5.3 thể hiện kết quả dự đoán khác của mô hình chạy trên board DE1-SoC.





Hình 5.3: Một vài kết quả dự đoán khác của mô hình.

Bảng 5.5 thể hiện thời gian dự đoán trung bình của 1000 tấm trên Intel CPU và board DE1-SoC.

Bảng 5.5: Tốc độ trung bình giữa CPU Intel và DE1-SoC


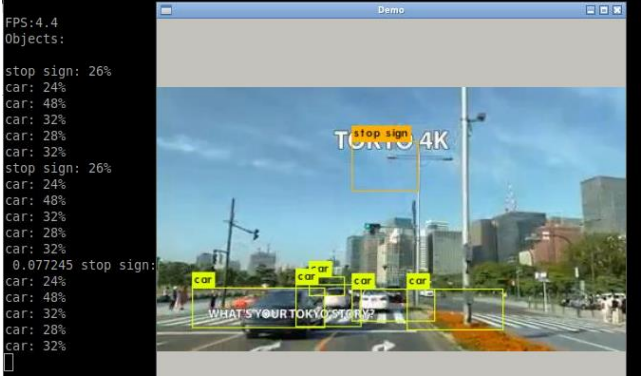
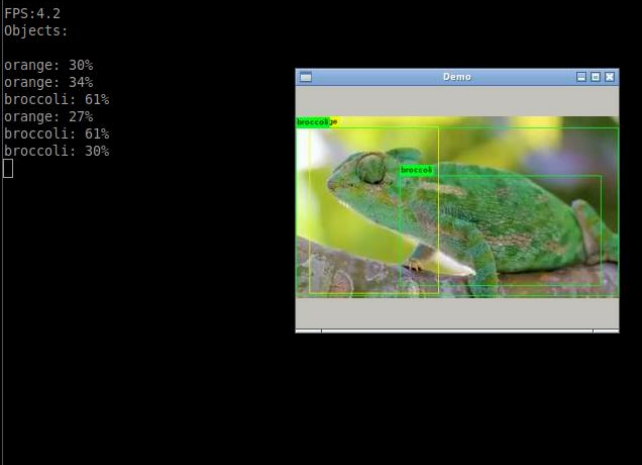
CPU Intel Core i5 Gen 8th	DE1-SoC
0.327837s	1.120932s

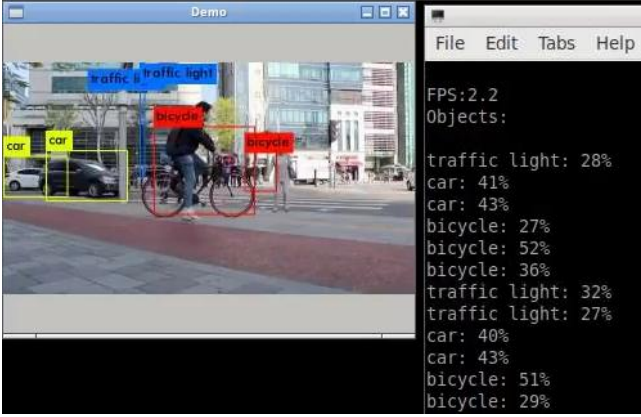
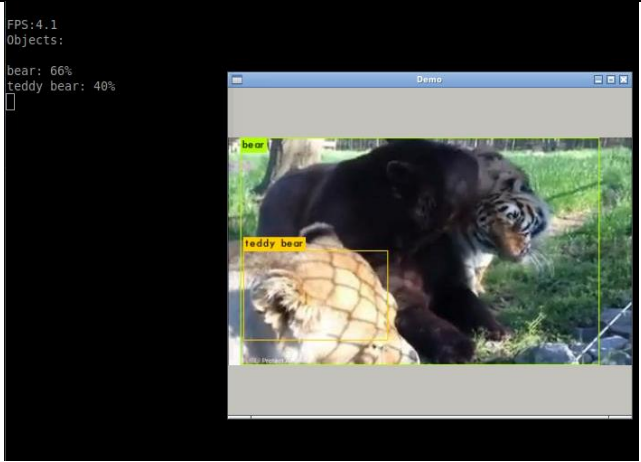
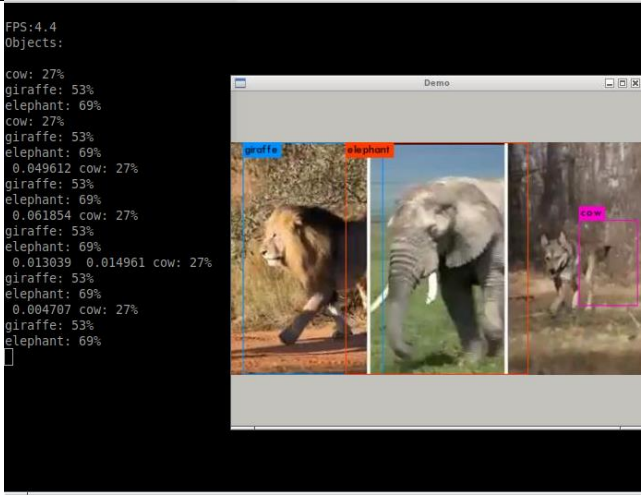
Nhóm so sánh tốc độ thực thi của bài báo tham khảo [5] và kết quả của nhóm.

[5]	DE1-SoC
1.4s	1.120932s


5.2.2. Kết quả chạy mô hình với video

Sau khi chạy thử mô hình với ảnh nhóm tiếp tục chạy thử nghiệm mô hình với video. Kết quả độ chính xác khi dự đoán đối tượng lên tới 90% và tốc độ xử lý đạt 3~4 FPS. Hình 5.4 thể hiện kết dự đoán trên 10 video trên board DE1-SoC.

STT	Độ chính xác cao nhất đạt được	FPS	Hình ảnh từ video
1	92%	1~6	
2	80%	1~6	
3	75%	1~6	

4	60%	1~6	 <pre> FPS:2.2 Objects: traffic light: 28% car: 41% car: 43% bicycle: 27% bicycle: 52% bicycle: 36% traffic light: 32% traffic light: 27% car: 40% car: 43% bicycle: 51% bicycle: 29% </pre>
5	70%	1~6	<pre> FPS:4.1 Objects: bear: 66% teddy bear: 48% </pre> 
6	75%	1~6	<pre> FPS:4.4 Objects: cow: 27% giraffe: 53% elephant: 69% cow: 27% giraffe: 53% elephant: 69% 0.049612 cow: 27% giraffe: 53% elephant: 69% 0.061854 cow: 27% elephant: 69% 0.013039 0.014961 cow: 27% giraffe: 53% elephant: 69% 0.004707 cow: 27% giraffe: 53% elephant: 69% </pre> 

7	60%	1~6	<div> <div> FPS:4.1 Objects: sports ball: 27% bear: 48% sports ball: 27% bear: 51% 0.045934 sports ball: 27% bear: 48% 0.076493 sports ball: 27% bear: 48% □ </div> <div> </div> </div>
8	75%	1~6	<div> <div> </div> <div> File Edit Tabs car: 32% keyboard: 50% car: 26% car: 67% car: 30% car: 35% car: 33% keyboard: 50% car: 36% car: 65% car: 25% car: 34% car: 32% </div> </div>
9	60%	1~6	<div> <div> FPS:3.6 Objects: vase: 45% vase: 31% person: 36% diningtable: 45% vase: 45% vase: 31% person: 36% diningtable: 45% 0.044483 0.028991 0.02216 vase: 31% person: 36% diningtable: 45% 0.004184 0.004208 vase: 45% vase: 31% person: 36% diningtable: 45% □ </div> <div> </div> </div>

10		1~6	<div> <div> FPS:2.5 Objects: fire hydrant: 73% fire hydrant: 47% fire hydrant: 74% person: 38% 0.040246 fire hydrant fire hydrant: 47% </div> <div>  </div> </div>
----	--	-----	---

Hình 5.4: Kết quả dự đoán video trên board DE1-SoC

5.3. Kết quả và đánh giá

5.3.1. Kết quả mong đợi

- Xây dựng thành công hệ điều hành linux và hiện thực thành công mô hình nhận diện 80 đối tượng phổ biến, tốc độ xử lý video đạt 5 FPS.
- Tài nguyên trên board: Nhóm muốn tạo ra một hệ thống mà nó có thể sử dụng ít tài nguyên nhất có thể và đáp ứng được tốc độ xử lý tốt nhất cho mô hình custom YOLO để phát hiện đối tượng.
- Kết nối với camera: Như mong đợi nhóm muốn triển khai mô hình custom YOLO trên board DE1-SoC để phát hiện đối tượng theo thời gian thực.

5.3.2. Kết quả thực tế

Với những kết quả mà nhóm đã đề ra sau quá trình thực hiện kết quả thực tế nhóm đạt được như sau:

- Với số lượng labels lớn cùng với việc tùy biến mô hình để phù hợp với tài nguyên của board DE1-SoC thì file weight của mô hình sau quá trình huấn luyện đạt độ chính xác không cao so với dự kiến nhóm đưa ra nhưng vẫn phát hiện được đối tượng tùy theo các labels.

- Xây dựng thành công hệ điều hành linux trên board DE1-SoC và hiện thực thành công mô hình nhận diện 80 đối tượng phổ biến, tốc độ xử lý video trung bình đạt 3~4 FPS sử dụng OpenCL.
- Chưa thể sử dụng camera để dự đoán đối tượng theo thời gian thực
- Tài nguyên hệ thống được nói chi tiết **Error! Reference source not found.** t hệ hiện kernel sử dụng bao nhiêu tài nguyên trên board DE1-SoC và so với tài nguyên sử dụng trong bài báo [5]

Bảng 5.6: Tài nguyên sử dụng trên FPGA của kernel.

	[5]	Mô hình của nhóm
ALUTs	33%	91%
FFs	27%	--
RAMs	35%	98%
DSPs	9%	100%

5.3.3. Đánh giá kết quả

Iou và Recall của mô hình custom thấp hơn mô hình Tiny YoloV2 tuy nhiên phép toán cũng như đầu vào ảnh được giảm xuống rất nhiều để đáp ứng việc phát hiện đối tượng theo thời gian thực.

Việc vẽ bounding box dự đoán đối tượng của mô hình vẫn còn sai và lệch khỏi đối tượng dự đoán khá nhiều. Tuy nhiên đối với những đối tượng chưa đưa training thì mô hình vẫn vẽ bounding box đúng đối tượng cho dù nhãn còn sai.

Kết quả thực tế đạt được tuy độ chính xác không được cao nhưng vẫn đủ đáp ứng một vài ứng dụng thực tế như: phát hiện xe, động vật... Đồng thời tốc độ xử lý video cũng đáp ứng được kết quả mong đợi của nhóm.

Chương 6. KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

6.1. Kết luận

6.1.1. Những gì nhóm đã đạt được

Về lý thuyết đã tìm hiểu về thuật toán GEMM để áp dụng vào mô hình DNN, và áp dụng những phương pháp phổ biến trong xử lý ảnh hiện nay là sử dụng mô hình mạng CNN, sử dụng những nghiên cứu đã có sẵn từ trước là mô hình Tiny YOLO sau đó custom lại mô hình phù hợp với tài nguyên của board DE1-SoC. Đồng thời nhóm đã biết tìm hiểu và áp dụng được phương pháp hiện thực mạng CNN mới trên FPGA là sử dụng OpenCL để tạo ra mạch có thể tổng hợp được thay vì sử dụng các ngôn ngữ mô tả phần cứng.

Bên cạnh đó hiểu được bộ phát triển DE1-SoC - một nền tảng thiết kế phần cứng mạnh mẽ được xây dựng bởi Altera Hệ thống trên chip (SoC) FPGA. Tận dụng được sức mạnh của khả năng tái cấu hình to lớn kết hợp với bộ xử lý hiệu suất cao, công suất thấp của FPGA để phát triển và phục vụ cho nhu cầu của nhóm.

Về mặt thực hành nhóm biết được cách thức để xây dựng một hệ điều hành linux cho board DE1-SoC và biết cách cài đặt và tích hợp các tiện ích, thư viện, các driver vào hệ điều hành Linux SoC-FPGA. Đồng thời nhóm đã áp dụng thành công phương pháp tổng hợp mạch phần cứng lên FPGA sử dụng OpenCL bằng cách sử dụng công nghệ Intel® FPGA SDK for OpenCL™. Ngoài ra còn áp dụng phương pháp Object Detection vào trong đề tài để nhận diện 80 đối tượng phổ biến sử dụng COCO dataset.

6.1.2. Khó khăn

Việc xây dựng hệ điều hành cho linux có hỗ trợ OpenCL gặp khá nhiều khó khăn do nhóm sử dụng linux kernel phiên bản 3.18 dẫn tới việc source đã bị ngưng hỗ trợ từ lâu khiến việc tìm kiếm tốn nhiều thời gian. Đồng thời do các thư viện được sử dụng trong source dùng để build linux đã khá cũ dẫn tới việc khi biên dịch gặp nhiều lỗi và gặp nhiều khó khăn trong việc sửa lỗi.

Framework nhóm sử dụng hiện tại khá cũ và không được phát triển. Trong quá trình nghiên cứu và tìm hiểu, các kiến thức nhóm phải tiếp nhận khá nhiều và phân bổ khá rộng trên nhiều lĩnh vực liên quan đến FPGA.

Tài nguyên của FPGA khá giới hạn nên việc sử dụng mô hình cần phải tính toán để cho phù hợp cũng như việc training mô hình cũng ảnh hưởng. Vì vậy việc huấn luyện dữ liệu khá mất thời gian và độ chính xác sẽ tương ứng với mô hình bị giới hạn.

Cuối cùng kiến thức về FPGA và OpenCL khá là rộng lớn mà kinh nghiệm của nhóm còn hạn chế vì vậy gặp không ít sai sót cũng như quá trình sửa lỗi, làm việc nhóm khá mất nhiều thời gian.

6.2. Hướng phát triển

Tuy đề tài đã hiện thực thành công mô hình YOLO trên board DE1-SoC nhưng vẫn cần cải thiện về các mặt sau:

- Về tốc độ:
 - + Do cách hiện thực thuật toán nhân ma trận GEMM trên FPGA còn dùng vòng lặp khá nhiều và chưa có pipeline nên tốc độ xử lý chưa được tối ưu. Do đó có thể tối ưu kernel bằng cách sử dụng pipeline hoặc các cách nhân 2 ma trận với nhau mà không cần dùng vòng lặp.
 - + Đề tài hiện đang sử dụng DDR3 SDRAM trên HPS dẫn tốc độ truy cập của FPGA vào SDRAM khá dẫn đến việc đọc ghi dữ liệu tốn nhiều thời gian. Do đó có thể tăng tốc độ thực thi của mô hình bằng cách cải thiện tốc độ truy cập của FPGA vào SDRAM
- Về độ chính xác:
 - + Nâng cao độ chính xác bằng các cải thiện tập dữ liệu.
 - + Tinh chỉnh lại kiến trúc mô hình để đạt được độ chính xác cao hơn.
- Về tính năng: Tích hợp camera để đáp ứng nhu cầu realtime. Ở đây nhóm đã thử kết nối với camera, tuy đã kết nối thành công với camera nhưng việc dự đoán qua camera còn bị delay so với thời gian thực dẫn tới không thể sử dụng trong thực tế. Đây là một trong những hướng có thể phát triển đề tài này.

TÀI LIỆU THAM KHẢO

- [1] Intel, DE1-SoC OpenCL, 2020.
- [2] Rocketboards, “Guide, Embedded Linux Beginners,” [Trực tuyến]. Available: <https://rocketboards.org/foswiki/Documentation/EmbeddedLinuxBeginnerSGuide>.
- [3] “Incorporate OpenCV into Cyclone VSoC to display USB camera video on Linux Desktop,” [Trực tuyến]. Available: <https://qiita.com/element/items/9f22ad95f227b6421b9c#opencv%E3%81%AE%E3%82%A4%E3%83%B3%E3%82%B9%E3%83%88%E3%83%BC%E3%83%AB-1>.
- [4] Yap June Wai, Zulkalnain bin Mohd Yussof, Sani Irwan bin Salim, Lim Kim Chuan, “Fixed Point Implementation of Tiny-Yolo-v2 using OpenCL on FPGA,” trong *(IJACSA) International Journal of Advanced Computer Science and Applications, Vol. 9, No. 10, 2018*, 2018.
- [5] Yap June Wai, Zulkanain Mohd Yussof, Sani Irwan Md Salim, “A scalable FPGA based accelerator for Tiny-YOLO-v2,” trong *International Journal of Reconfigurable and Embedded Systems (IJRES) Vol. 8, No. 3, November 2019, pp. 206~214 ISSN: 2089-4864, DOI: 10.11591/ijres.v8.i3.pp206-214*, 2019.
- [6] J. Solawetz, “What is Mean Average Precision (mAP) in Object Detection?,” roboflow, 6 May 2020. [Trực tuyến]. Available: <https://blog.roboflow.com/mean-average-precision/>. [Đã truy cập 2021].
- [7] RocketBoards.org. [Trực tuyến]. Available: <https://rocketboards.org/foswiki/Documentation/GSRDSdCard>.

- [8] Max Ferguson, Ronay Ak, Yung-Tsun Tina Lee, Kincho H. Law, “Automatic localization of casting defects with convolutional neural networks,” trong *2017 IEEE International Conference on Big Data (Big Data)*, Boston, MA, USA, 2017.
- [9] Alex Kost, Wael A. Altabey, Mohammad Noori, Taher Awad, “Applying Neural Networks for Tire Pressure Monitoring Systems,” trong *SDHM Structural Durability and Health Monitoring*, 2019.
- [10] J. Redmon, S. Divvala, R. Girshick and A. Farhadi, “You Only Look Once: Unified, Real-Time Object Detection,” trong *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 779-788, doi: 10.1109/CVPR.2016.91., 2016.
- [11] J. Ma, L. Chen, Gao, “Hardware Implementation and Optimization of Tiny-YOLO Network,” trong *International Forum on Digital TV and Wireless Multimedia Communications, Springer, Singapore, 2017*, pp. 224-234.
- [12] J. Redmon, and Farhadi, “A. YOLO9000: better, faster, stronger,” trong *arXiv*, 2017..
- [13] J. Zhang, and J. Li, “Improving the performance of OpenCL-based fpga accelerator for convolutional neural network,” trong *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2017, pp. 25-34.
- [14] Bochkovskiy, Alexey & Wang, Chien-Yao & Liao, Hong-yuan, “YOLOv4: Optimal Speed and Accuracy of Object Detection,” 2020.
- [15] R. Padilla, S. L. Netto and E. A. B. da Silva, “A Survey on Performance Metrics for Object-Detection Algorithms,” trong *2020 International Conference on Systems, Signals and Image Processing (IWSSIP)*, 2020, pp. 237-242, doi: 10.1109/IWSSIP48289.2020.9145130..

- [16] T. Technologies, “Terasic - DE Main Boards - Cyclone - DE1-SoC Board,” [Trực tuyến]. Available: <http://de1-soc.terasic.com..>
- [17] J. J. Redmon, “Darknet: Open source neural networks in c,” 2013–2016. [Trực tuyến]. Available: <http://pjreddie.com/darknet/>.
- [18] Zhang, Ning, Xin Wei, He Chen, and Wenchao Liu, “FPGA Implementation for CNN-Based Optical Remote Sensing Object Detection,” trong *Electronics* 10, no. 3: 282, 2021.
- [19] Rodriguez-Conde, I., Campos, C. & Fdez-Riverola, “Optimized convolutional neural network architectures for efficient on-device vision-based object detection,” trong *Neural Comput & Applic*, 2021.
- [20] T. S. C. e. al, “From opencl to high-performance hardware on FPGAS,” trong *22nd International Conference on Field Programmable Logic and Applications (FPL)*, 2012, pp. 531-534, doi: 10.1109/FPL.2012.6339272..
- [21] J. Z. e. al, “A Low-Latency FPGA Implementation for Real-Time Object Detection,” trong *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2021, pp. 1-5, doi: 10.1109/ISCAS51556.2021.9401577..
- [22] A. Anderson, A. Vasudevan, C. Keane and D. Gregg, “High-Performance Low-Memory Lowering: GEMM-based Algorithms for DNN Convolution,” trong *2020 IEEE 32nd International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, 2020, pp. 99-106, doi: 10.1109/SBAC-PAD49847.2020.00024..
- [23] Pilipović, Ratko, Vladimir Risojević, Janko Božić, Patricio Bulić, and Uroš Lotrič, “An Approximate GEMM Unit for Energy-Efficient Object Detection,” trong *Sensors* 21, no. 12: 4195.

- [24] Cai, Liangwei & Wang, Ceng & Xu, Yuan, “A Real-Time FPGA Accelerator Based on Winograd Algorithm for Underwater Object Detection,” trong *Electronics*. 10. 2889. 10.3390/electronics10232889., 2021.
- [25] S. Kala, J. Mathew, B. R. Jose, and S. Nalesh, “UniWiG: Unified Winograd-GEMM Architecture for Accelerating CNN on FPGAs,” trong *in 2019 32nd International Conference on VLSI Design and 2019 18th International Conference on Embedded Systems (VLSID), Delhi, NCR, India, 2019*, pp. 209–214. DOI: 10.1109/VLSID.2019.00055.
- [26] T. Adiono, A. Putra, N. Sutisna, I. Syafalni and R. Mulyawan, “Low Latency YOLOv3-Tiny Accelerator for Low-Cost FPGA Using General Matrix Multiplication Principle,” trong *IEEE Access*, vol. 9, pp. 141890-141913, 2021, doi: 10.1109/ACCESS.2021.3120629.
- [27] Q. Xiao, L. Lu, J. Xie and Y. Liang, “FCNNLib: An Efficient and Flexible Convolution Algorithm Library on FPGAs,” trong *57th ACM/IEEE Design Automation Conference (DAC), 2020*, pp. 1-6, doi: 10.1109/DAC18072.2020.9218748., 2020.
- [28] L. Lu, Y. Liang, Q. Xiao and S. Yan, “Evaluating Fast Algorithms for Convolutional Neural Networks on FPGAs,” trong *IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), 2017*, pp. 101-108, doi: 10.1109/FCCM.2017.64., 2017.
- [29] Intel, Intel® FPGA SDK for OpenCL™ Intel® Cyclone® V SoC Development Kit Reference Platform Porting Guide.
- [30] D. T. Nguyen, T. N. Nguyen, H. Kim and H. Lee, “A High-Throughput and Power-Efficient FPGA Implementation of YOLO CNN for Object Detection,” trong *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 8, pp. 1861-1873, Aug. 2019, doi: 10.1109/TVLSI.2019.2905242., 2019.

