

ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
CHƯƠNG TRÌNH KỸ SỰ CHẤT LƯỢNG CAO VIỆT PHÁP
KHOA ĐIỆN - ĐIỆN TỬ
BỘ MÔN VIỄN THÔNG



LUẬN VĂN TỐT NGHIỆP

**HỆ THỐNG GIÁM SÁT VIỆC SỬ
DỤNG THIẾT BỊ BẢO HỘ CÁ NHÂN
ỨNG DỤNG MẠNG HỌC SÂU**

NGUYỄN THÁI SƠN - 1512847

GIẢNG VIÊN HƯỚNG DẪN:
PGS. TS. HÀ HOÀNG KHA

Thành phố Hồ Chí Minh, tháng 6 năm 2020

Số: _____/BKDT
Khoa: Điện - Điện tử
Bộ môn: Viễn Thông

NHIỆM VỤ LUẬN VĂN TỐT NGHIỆP

- | | |
|--|-----------------|
| 1. HỌ VÀ TÊN: NGUYỄN THÁI SƠN | MSSV:1512847 |
| 2. NGÀNH: VIỄN THÔNG | LỚP: VP15VT |
| 3. Đề tài: HỆ THỐNG GIÁM SÁT VIỆC SỬ DỤNG THIẾT BỊ BẢO HỘ CÁ NHÂN ỨNG DỤNG MẠNG HỌC SÂU | |
| 4. Nhiệm vụ (Yêu cầu về nội dung và số liệu ban đầu): | |
| <ul style="list-style-type: none"> • Tìm hiểu các mô hình nhận diện đang được nghiên cứu và sử dụng. • Tìm hiểu về CNN và YOLOv3 • Xây dựng tập dữ liệu phù hợp cho bài toán đặt ra. • Xây dựng mô hình nhận diện. • Đánh giá, kết luận và viết luận văn. | |
| 5. Ngày giao nhiệm vụ luận văn: | |
| 6. Ngày hoàn thành nhiệm vụ: | |
| 7. Họ và tên người hướng dẫn: | Phần hướng dẫn |
| PGS. TS. Hà Hoàng Kha | Toàn phần |

Nội dung và yêu cầu LVTN đã được thông qua Bộ Môn.
Tp.HCM, ngày ... tháng ... năm 2020

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 luân văn:.....

LỜI CẢM ƠN

Khoảng thời gian học tập và rèn luyện tại **Trường Đại học Bách Khoa Thành phố Hồ Chí Minh** đã trang bị cho em rất nhiều kiến thức hữu ích và cần thiết về chuyên môn và xã hội để em có thể trở thành một công dân tốt và một kỹ sư có năng lực. Con đường học tập ở đại học trong suốt 5 năm vừa qua là không hề dễ dàng với muôn vàn thử thách và khó khăn. Để vượt qua những rào cản ấy, bên cạnh sự cố gắng của bản thân em còn là sự ủng hộ và giúp đỡ tận tình của quý **Thầy Cô, Gia đình và Bạn bè**. Em xin gửi lời cảm ơn chân thành và sâu sắc nhất đến quý **Thầy Cô Trường Đại học Bách Khoa Thành phố Hồ Chí Minh**, quý **Thầy Cô Khoa Điện – Điện Tử**, những người đã đi cùng với tri thức và tâm huyết truyền đạt vốn kiến thức quý báu của mình cho chúng em.

Em muốn giành riêng lời cảm ơn đặc biệt cho Thầy hướng dẫn của mình – PGS. TS. Hà Hoàng Kha – người đã luôn tận tình hướng dẫn và giúp đỡ em trong quá trình thực hiện luận văn này.

Ngoài ra, em cũng muốn giành cho gia đình và bạn bè lời cảm ơn chân thành và đặc biệt là ba mẹ và ông bà em, những người đã luôn là chỗ dựa tinh thần vững chắc cho em trong những thời điểm khó khăn.

Trong quá trình thực hiện luận văn chắc chắn không thể tránh khỏi những sai sót, em rất mong nhận được những ý kiến đóng góp quý báu của quý Thầy Cô để em có thể học hỏi thêm nhưng điều tốt đẹp và hoàn thiện luận văn của mình. Sau cùng, em xin kính chúc quý **Thầy Cô Trường Đại học Bách Khoa Thành phố Hồ Chí Minh** và **Thầy Hà Hoàng Kha** dồi dào sức khỏe, đạt được nhiều thành công trong cuộc sống và luôn giữ vững niềm đam mê nghiên cứu và giảng dạy để có thể tiếp tục truyền lửa tri thức cho những thế hệ sau.

Thành phố Hồ Chí Minh, tháng 6 năm 2020

Nguyễn Thái Sơn

TÓM TẮT LUẬN VĂN

Ý tưởng về việc kết hợp trí tuệ nhân tạo và thị giác máy tính để ứng dụng vào các bài toán giám sát đã xuất hiện từ lâu. Tuy nhiên chỉ đến những năm gần đây khi các thiết bị phần cứng có thể đáp ứng được yêu cầu về tính kĩ thuật và kinh tế thì các ứng dụng sử dụng các công nghệ này mới dần trở nên phổ biến. Một trong những ứng dụng đang rất được quan tâm là đảm bảo an toàn lao động của công nhân xây dựng thông qua một hệ thống giám sát sử dụng camera và trí tuệ nhân tạo để theo dõi việc sử dụng các thiết bị bảo hộ lao động của những người làm việc trong công trường.

Trong khuôn khổ của luận văn này, hệ thống giám sát việc sử dụng thiết bị bảo hộ cá nhân sẽ tập trung vào ba thiết bị thường gặp trong công trường xây dựng: mũ cứng, áo dạ quang bảo hộ và khẩu trang. Hệ thống sử dụng phương pháp phát hiện và phân loại vật thể YOLO, được xây dựng trên cơ sở mạng tích chập - CNN. Khi hoạt động, hệ thống sẽ có thể phát hiện và phân loại việc sử dụng các thiết bị bảo hộ lao động của công nhân ở công trường. Nếu phát hiện một trường hợp không sử dụng thiết bị bảo hộ lao động đang được theo dõi thì hệ thống sẽ gửi cảnh báo cho quan sát viên để kiểm tra và nhắc nhở. Việc này sẽ hỗ trợ rất nhiều trong công tác đảm bảo an toàn lao động trong xây dựng.

Hệ thống được viết bằng ngôn ngữ Python, mô hình máy học YOLOv3 và thư viện thị giác máy tính OpenCV.

Mục lục

1 Giới thiệu	1
1.1 Đặt vấn đề	1
1.2 Mục tiêu nghiên cứu	2
1.3 Phạm vi nghiên cứu	2
1.4 Phương pháp nghiên cứu	2
1.5 Cấu trúc luận văn	2
2 Cơ sở lý thuyết	3
2.1 Mạng neuron nhân tạo	4
2.1.1 Ý tưởng về một neuron nhân tạo	4
2.1.2 Mạng lưới các neuron	6
2.1.3 Hàm mất mát và bài toán tối ưu mạng neuron	8
2.2 Tối ưu hàm mất mát bằng Gradient Descent	9
2.2.1 Một ví dụ đơn giản về Gradient Descent	9
2.2.2 Gradient Descent cho hàm đa biến	12
2.2.3 Điều kiện dừng của giải thuật	13
2.3 Sử dụng backpropagation để giải quyết vấn đề cập nhật trọng số trong mạng neuron	14
2.4 Mạng neuron tích chập[1][2]	16
2.4.1 Lớp tích chập	16
2.4.2 Lớp pooling	19
2.4.3 Lớp đầy đủ kết nối	20
2.4.4 Mô hình mạng neuron tích chập	20
2.5 Mô hình YOLOv3	21
2.5.1 Unified Detection	21
2.5.2 Kiến trúc mạng YOLOv3	23
3 Phương pháp tiếp cận	25
3.1 Xây dựng tập dữ liệu	25
3.1.1 Xác định yêu cầu bài toán	25
3.1.2 Thu thập hình ảnh và dán nhãn	26
3.2 Huấn luyện mạng YOLOv3 sử dụng framework Darknet[3]	27
3.3 Sử dụng mô hình YOLOv3 để nhận diện trên camera hoặc video .	31
4 Phân tích kết quả	40
5 Kết luận và hướng phát triển	47
5.1 Kết luận	47
5.2 Hướng phát triển	47

Danh sách hình vẽ

2.1	Mô hình một neuron sinh học.	4
2.2	Mô hình một neuron nhân tạo.	5
2.3	Đồ thị và đạo hàm của hàm Sigmoid.	5
2.4	Đồ thị và đạo hàm của hàm Tanh.	6
2.5	Đồ thị và đạo hàm của hàm Rectified Linear Unit.	6
2.6	Mô hình mạng neuron nhân tạo gồm 3 lớp hidden layer, 1 đầu vào và 1 đầu ra.	7
2.7	Mô hình toán của các lớp ẩn trong mạng neuron.	8
2.8	Giá trị hàm mất mát theo với các trường hợp dự đoán khác nhau.	9
2.9	Cực tiểu địa phương (màu xanh) và cực tiểu toàn cục (màu đỏ)	10
2.10	Batch Gradient Descent với bài toán hồi quy tuyến tính. Toàn bộ số điểm đầu vào đều được dùng để cập nhật các vector trọng số (a, b) cho đường hồi quy tại mỗi bước, với a là độ dốc và b độ sai lệch.	11
2.11	Stochastic Gradient Descent với bài toán hồi quy tuyến tính. Một điểm đầu vào được chọn ngẫu nhiên để cập nhật các vector trọng số (a, b) cho đường hồi quy tại mỗi iteration, với a là độ dốc và b độ sai lệch.	11
2.12	Mini-batch Gradient Descent với bài toán hồi quy tuyến tính. Một batch sẽ gồm ba điểm đầu vào được chọn ngẫu nhiên để cập nhật các vector trọng số (a, b) cho đường hồi quy tại mỗi iteration, với a là độ dốc và b độ sai lệch. Một epoch sẽ gồm mười batch.	12
2.13	Đồ thị mặt phẳng hàm mất mát và Gradient Descent cho hàm nhiều biến.	13
2.14	Mô hình mạng neuron tích chập đơn giản. Lớp nhận hình ảnh vào màu đỏ là một lớp có cấu trúc ba chiều với chiều rộng và chiều cao là chiều rộng và chiều cao của hình ảnh đầu vào, chiều sâu bằng ba ứng với ba kênh màu đỏ, xanh lá và xanh dương. Các lớp của mạng neuron tích chập sẽ chuyển đổi một nhóm các ma trận thành một nhóm các ma trận khác. Lớp ngoài cùng là lớp phân loại, có kích thước các chiều tương ứng với một vector.	16
2.15	Hình ảnh đầu vào gồm ba kênh màu được mô hình hóa thành tensor với chiều cao và chiều rộng là chiều cao và chiều rộng của ảnh, chiều sâu là ba.	17
2.16	Hình ảnh sau khi được đưa qua đầu vào và chuyển đổi thành dữ liệu ba chiều sẽ được đưa vào lớp convolution đầu tiên. Một kernel có kích thước $3 \times 3 \times 3$ (góc trên bên trái của mô hình ngoài cùng bên phải) được trượt qua hình ảnh đầu vào.	17
2.17	Ví dụ về phép toán của số trượt với kích thước 3×3	17

2.18	Bên trái, ma trận 3×3 được zero padding với $padding = 1$. Bên phải, ma trận 3×3 được zero padding với $padding = 2$	18
2.19	Ví dụ về một kernel có kích thước $3 \times 3 \times 3$	18
2.20	Ví dụ về phép toán của một kernel lên một vị trí của ảnh trong lớp tích chập.	19
2.21	Một lớp tích chập có k kernel với kích thước $3 \times 3 \times 3$, $stride = 1$, $padding = 1$. Đầu vào là một tensor có kích thước $h \times w \times d$ đầu ra của phép tích chập lên tensor này khi khối tích chập có các thông số ở trên là một tensor có kích thước $h \times w \times k$	19
2.22	Bên trái, lớp pooling với cửa sổ trượt lấy giá trị lớn nhất với kích thước cửa sổ 2×2 , $stride = 1$, $padding = 0$. Bên phải, lớp pooling với cửa sổ trượt lấy giá trị trung bình với kích thước cửa sổ 2×2 , $stride = 2$, $padding = 0$	20
2.23	Mạng neuron tích chập gồm hai lớp tích chập và pooling, một lớp kết nối đầy đủ.	20
2.24	Hình ảnh được chia thành mạng lưới ô vuông $S \times S$	21
2.25	Miêu tả việc tính toán IOU.	22
2.26	Mô hình dự đoán bounding box của YOLO.	22
2.27	Kiến trúc mạng Darknet-53.	24
3.1	(1) Mũ bảo hộ, (2) Áo bảo hộ, (3) Khẩu trang	25
3.2	Kết quả nhận dạng mong muốn.	26
3.3	Dịnh dạng nhãn của YOLO.	26
3.4	Thông kê số lượng vật thể ứng với từng class. Wearing a hardhat: 27565, Not wearing a hardhat: 45888, Wearing a safety vest: 10264, Not wearing a safety vest: 52996, Wearing a mask: 12557, Not wearing a mask: 46501.	27
3.5	Sơ đồ khôi chương trình Python để nhận dạng trên video hoặc webcam.	32
4.1	Số lượng các object trong tập dữ liệu validation. Wearing a hardhat - 2113, Not wearing a hardhat - 3313, Wearing a safety vest - 814, Not wearing a safety vest - 3900, Wearing a mask - 887, Not wearing a mask - 3492.	41
4.2	Precision - màu đỏ, Recall - màu xanh dương, Mean Average Precision - màu xanh lá. Các thông số được tính toán mỗi 1000 iteration trên tập dữ liệu validation.	42
4.3	Kết quả dự đoán tốt - hình trên. Phóng to một phần của kết quả dự đoán - hình dưới.	43
4.4	Kết quả dự đoán tốt - hình trên. Phóng to một phần của kết quả dự đoán - hình dưới.	44
4.5	Kết quả dự đoán không tốt - hình trên. Phóng to một phần của kết quả dự đoán - hình dưới.	45
4.6	Kết quả dự đoán không tốt - hình trên. Phóng to một phần của kết quả dự đoán - hình dưới.	46

Danh sách bảng

- 2.1 So sánh hiệu năng của Darknet-53 với các mạng khác. Accuracy, Bn Ops - billions of operations, BFLOP/s - billion floating point operations per second, và FPS - frames per second. 24

Chương 1

Giới thiệu

1.1 Đặt vấn đề

Ngành xây dựng luôn được coi là một trong những ngành ẩn chứa nhiều rủi ro về tai nạn lao động và khả năng mắc các bệnh nghề nghiệp. Trên thực tế nhiều vụ tai nạn nghiêm trọng đã xảy ra, lấy đi sinh mạng hoặc để lại những thương tật nặng nề cho người lao động khiến họ mất khả năng làm việc, sinh hoạt như người bình thường. Kéo theo đó là nỗi đau về tinh thần và gánh nặng về kinh tế cho những thành viên trong gia đình người bị nạn. Do đó vấn đề đảm bảo an toàn vệ sinh lao động luôn là một trong những vấn đề được quan tâm hàng đầu trong ngành xây dựng.

Một trong những nguyên nhân chính gây ra những vụ tai nạn thương tâm là việc người lao động không sử dụng trang thiết bị bảo hộ cá nhân trong quá trình lao động. Vấn đề này không chỉ xuất phát từ sự chủ quan của cá nhân người lao động mà còn ở sự thiếu sót, lỏng lẻo trong quá trình giám sát công trình của nhà thầu và người sử dụng lao động. Đối với người lao động, những điều kiện khắc nghiệt của môi trường làm việc như nhiệt độ ngoài trời cao hay thường xuyên phải vận động mạnh khiến đổ mồ hôi liên tục đã khiến họ chấp nhận đánh đổi sự an toàn của bản thân để đổi lấy sự thoải mái. Còn đối với những người giám sát công trình, họ không thể bao quát được toàn bộ quá trình làm việc tại các nơi làm việc khác nhau, do đó không thể nhắc nhở người lao động kịp thời trước khi xảy ra những tai nạn mà hậu quả là có thể tránh khỏi hoặc được giảm nhẹ nếu người lao động có sử dụng trang thiết bị bảo hộ cá nhân.

Để tăng cường năng lực thực hiện đảm bảo an toàn vệ sinh lao động ở các công trình, nhiều chủ đầu tư và nhà thầu đã tiến hành lắp đặt các hệ thống camera giám sát quá trình làm việc. Các hệ thống này giúp giám sát viên có thể quan sát nhiều vị trí một lúc mà không cần phải di chuyển qua các địa điểm khác nhau trong công trình, giảm thiểu chi phí và thời gian thực hiện các công tác an toàn. Tuy nhiên, khi số lượng các khu vực cần quan sát tăng lên hoặc những người chịu trách nhiệm quan sát không tập trung vào nhiệm vụ thì việc giám sát thông qua màn hình dễ xảy ra sai sót. Việc tích hợp công nghệ AI vào các hệ thống giám sát sẽ là sẽ tăng thêm độ tin cậy cho công tác đảm bảo an toàn, giảm thiểu những sai sót không đáng có.

1.2 Mục tiêu nghiên cứu

Mục tiêu của luận văn là xây dựng, đánh giá một hệ thống nhận diện việc sử dụng thiết bị bảo hộ cá nhân của người lao động trong công trường. Khi phát hiện ra các trường hợp không sử dụng các trang thiết bị bảo hộ thì hệ thống sẽ đưa ra cảnh báo.

1.3 Phạm vi nghiên cứu

Phạm vi của luận văn là tiến hành nhận dạng trên các video trích xuất từ camera. Mô hình nhận diện được huấn luyện sử dụng framework được xây dựng sẵn. Tập dữ liệu sử dụng để huấn luyện và đánh giá mô hình nhận diện được thu thập và dán nhãn bởi người làm luận văn. Một phần hình ảnh trong tập dữ liệu này có nguồn gốc từ các tập dữ liệu khác nhưng không sử dụng lại các nhãn của các tập dữ liệu đó. Các thiết bị bảo hộ cá nhân được tích hợp trong hệ thống gồm: mũ cứng, áo bảo hộ và khẩu trang.

1.4 Phương pháp nghiên cứu

Các giai đoạn trong quá trình nghiên cứu và hoàn thiện luận văn:

1. Tìm hiểu các mô hình nhận diện đang được nghiên cứu và sử dụng.
2. Chọn mô hình phù hợp, tìm hiểu lý thuyết.
3. Xây dựng tập dữ liệu phù hợp cho bài toán đặt ra.
4. Huấn luyện mô hình nhận diện.
5. Đánh giá các tham số hiệu năng của mô hình nhận diện đã xây dựng.
6. Xây dựng hệ thống sử dụng mô hình nhận diện để đưa ra cảnh báo.
7. Đánh giá khả năng hoạt động của hệ thống.

Trong đó các giai đoạn 3, 4 và 5 được thực hiện luân phiên và nhiều lần để cải thiện hiệu năng của mô hình nhận diện. YOLOv3 được chọn để làm mô hình nhận diện vì đây là một trong số các bộ nhận diện có hiệu năng cao trong thời gian thực, đã được sử dụng và đánh giá trên nhiều tập dữ liệu khác nhau.

1.5 Cấu trúc luận văn

Luận văn này bao gồm 5 chương. Chương 1 là chương mở đầu, giới thiệu bao quát về vấn đề, mục tiêu, phạm vi và phương pháp nghiên cứu luận văn. Chương 2 sẽ cung cấp những lý thuyết về các khái niệm được sử dụng trong luận văn. Chương 3 cho người đọc biết về cách tập dữ liệu được xây dựng, cách mô hình YOLOv3 được huấn luyện sử dụng framework darknet và cách xây dựng hệ thống sử dụng mô hình nhận diện. Chương 4 sẽ chứa những thông số đánh giá hiệu năng của bộ nhận diện và hệ thống trong các trường hợp khác nhau. Cuối cùng, trong chương 5 sẽ là các nhận xét về các kết quả đạt được và kết luận.

Chương 2

Cơ sở lý thuyết

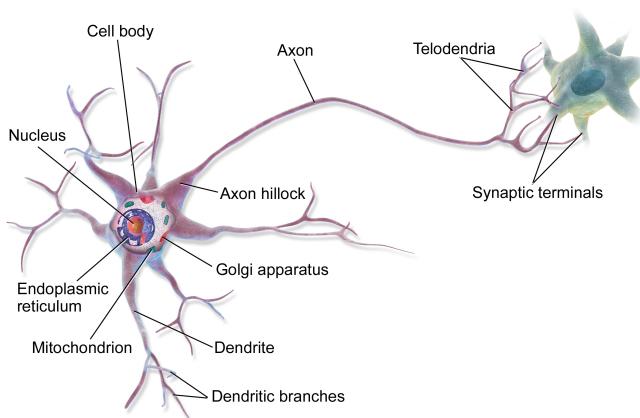
Việc nghiên cứu và mô phỏng cách mà não bộ con người hoạt động trên máy tính đã bắt đầu từ những năm 50 của thế kỷ XX. Những nghiên cứu mang tính đột phá như *The organization of behavior*[4] của Donald Olding Hebb, *Some Studies in Machine Learning Using the Game of Checkers*[5] của Arthur Lee Samuel đã đặt nền móng cho những mô hình mạng neuron đa lớp nhân tạo đầu tiên trong những năm 60 của thế kỷ XX, mở ra những hướng nghiên cứu mới về các kiến trúc mạng neuron feedforward. Sau đó vào những năm 70 lý thuyết về backpropagation bắt đầu được hình thành nhằm cho phép mạng neuron đa lớp có thể điều chỉnh các lớp ẩn để thích nghi với các tình huống khác nhau dựa vào hàm mất mát ở đầu ra. Tuy nhiên chỉ đến khi bài báo nổi tiếng của nhóm tác giả David Rumelhart, Geoffrey Hinton và Ronald Williams được công bố thì việc sử dụng backpropagation trong mạng neuron nhân tạo mới trở nên phổ biến. Bài báo *Learning representations by back-propagating errors*[6] công bố năm 1986 của nhóm tác giả đã đề xuất một số mô hình mạng neuron sử dụng backpropagation có thể học và xử lý nhanh hơn các mô hình truyền thống, điều này cho phép mạng neuron nhân tạo sử dụng backpropagation có thể giải quyết được những bài toán mà đối với mạng neuron truyền thống là không có lời giải. Năm 1989, Yann LeCun, Léon Bottou, Yoshua Bengio và Patrick Haffner trong bài báo *Gradient-Based Learning Applied to Document Recognition*[7] đã kết hợp kỹ thuật backpropagation vào mạng neuron tích chập để giải quyết bài toán nhận diện chữ viết tay và phát hiện khuôn mặt. Bài báo này đã tạo tiền đề cho những nghiên cứu khác về mạng neuron tích chập và mạng neuron dựa trên gradient trong thế kỷ XXI.

Chương này sẽ trình bày các lý thuyết về mạng neuron nhân tạo, vai trò của việc tối ưu hàm mất mát của một mạng neuron và cách mà một mạng neuron sử dụng giải thuật Gradient Descent kết hợp với kỹ thuật backpropagation để tối ưu hàm mất mát. Đồng thời lý thuyết về mạng neuron tích chập và ứng dụng của mạng này trong mạng YOLOv3 cũng sẽ được đề cập trong chương này để phục vụ cho các chương tiếp theo.

2.1 Mạng neuron nhân tạo

2.1.1 Ý tưởng về một neuron nhân tạo

Neuron[8] là một tế bào thần kinh cơ bản, cấu thành nên hệ thần kinh của con người. Trung bình một bộ não người sẽ có 86 tỷ neuron và $10^{14} - 10^{15}$ synapse. Hình 2.1 miêu tả các thành phần một neuron trong não người. Mỗi neuron sẽ nhận tín hiệu đầu vào từ các sợi nhánh (tiếng Anh: dendrite(s)) và gửi tín hiệu ra thông qua một sợi trực (tiếng Anh: axon). Sợi trực sẽ kết nối với các synapse và các synapse sẽ kết nối với các sợi nhánh của các neuron khác để gửi tín hiệu. Hình 2.2 miêu tả mô hình toán học của một neuron. Tín hiệu x_0 đi qua sợi trực

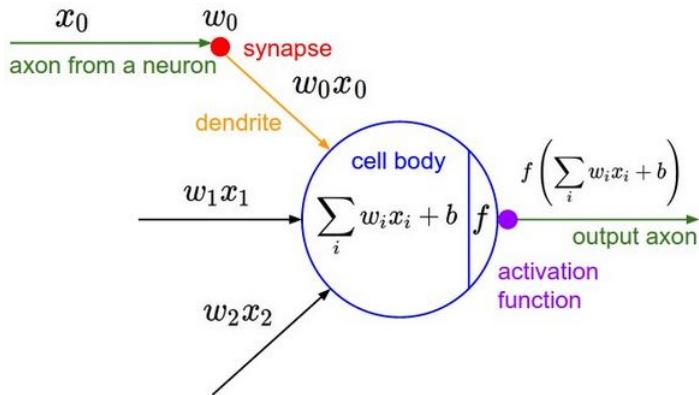


Hình 2.1: Mô hình một neuron sinh học.

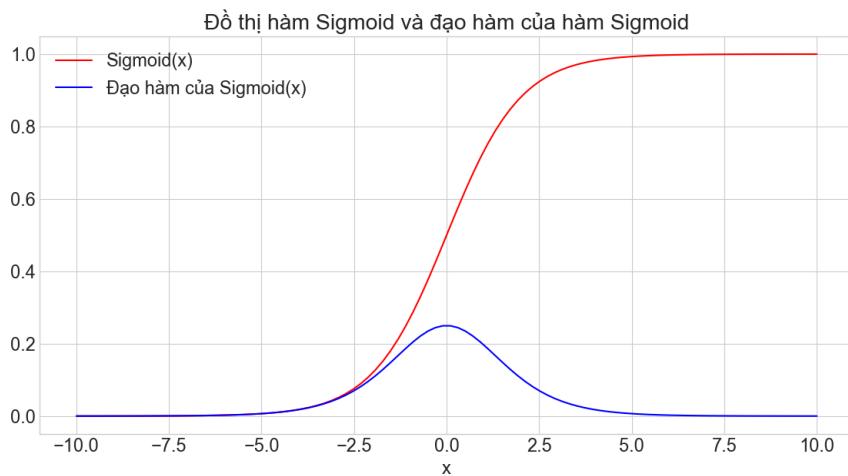
và tới synapse nơi kết nối với các sợi nhánh, mỗi sợi nhánh sẽ có một lực synapse (tiếng Anh: synaptic strength) được mô hình hóa bằng trọng số w biểu thị mức độ ảnh hưởng của sợi nhánh đó đến tín hiệu truyền đến neuron tiếp theo. Lực synapse có thể được học và điều chỉnh sự ảnh hưởng của neuron này đến neuron khác. Các tín hiệu sau khi đi qua sợi nhánh sẽ có dạng wx và sẽ được tổng hợp bằng hàm cộng trong tế bào neuron. Nếu như kết quả cao hơn một ngưỡng xác định thì tế bào neuron sẽ gửi tín hiệu đi. Hoạt động này được mô hình hóa bằng một hàm kích (tiếng Anh: activation function) f và cũng giống như mạng neuron sinh học, đầu ra sẽ chỉ là một kết quả duy nhất để gửi đến synapse tiếp theo.

Một số hàm số thường được sử dụng làm hàm kích cho một neuron nhân tạo gồm:

- Hàm sigmoid $\sigma(x) = \frac{1}{1+e^{-x}}$. Có thể thấy trên hình 2.3 đồ thị của hàm sigmoid, với mọi giá trị của x trên miền \mathbb{R} sau khi đi qua hàm sigmoid thì kết quả sẽ là một giá trị nằm trong khoảng $[0, 1]$. Đây là một trong những hàm số thường được sử dụng trong các mô hình mạng neuron cổ điển vì tính phi tuyến và khả năng mô hình hóa tương đối tốt quá trình một neuron gửi tín hiệu với cận 0 tương trưng cho tín hiệu không được truyền đi và cận 1 tương trưng cho tín hiệu được truyền đi. Tuy nhiên ta có thể thấy trên hình 2.3 hàm sigmoid có đạo hàm xấp xỉ bằng 0 tại các biên 0 và 1. Điều này ảnh hưởng tới việc sử dụng phương pháp backpropagation vì trong quá trình backpropagation thì đạo hàm của hàm kích sẽ được nhân với giá trị gradient của đầu ra tại neuron này, khi đó kết quả thu được có thể bằng 0



Hình 2.2: Mô hình một neuron nhân tạo.



Hình 2.3: Đồ thị và đạo hàm của hàm Sigmoid.

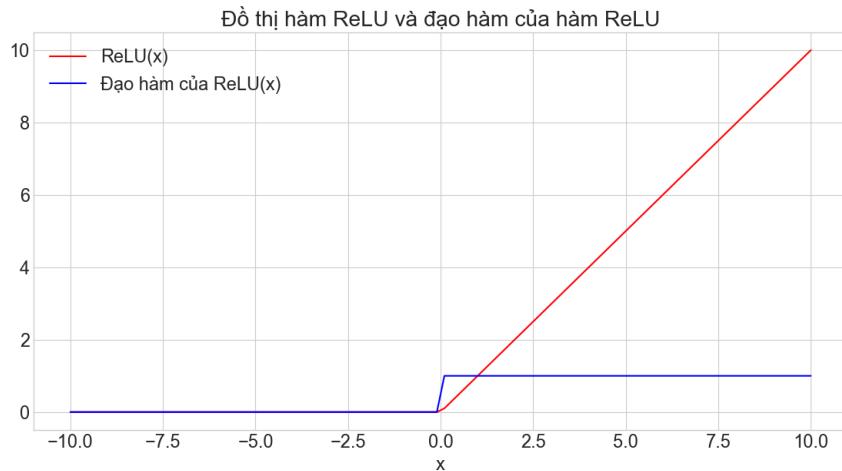
nếu kết quả đầu ra của neuron xấp xỉ 0 hoặc 1 và các trọng số kết nối với neuron này sẽ không được cập nhật. Ngoài ra kết quả của hàm sigmoid sẽ đổi xứng quanh 0.5 nên sẽ gây ra hiện tượng kết quả của gradient đổi dấu liên tục trong quá trình tính toán.

- Hàm tanh $\tanh(x) = 2\sigma(2x) - 1$. Hình 2.4 cho thấy hàm số này cũng có đạo hàm bằng 0 tại các biên 0 và 1 như hàm sigmoid nhưng kết quả của hàm tanh sẽ đổi xứng quanh điểm 0. Do vậy hàm tanh thường được ưu tiên dùng thay cho hàm sigmoid.
- Hàm Rectified Linear Unit với $ReLU(x) = \max(0, x), x \in \mathbb{R}$. Hình 2.5 miêu tả đồ thị hàm Rectified Linear Unit, khi sử dụng hàm số này, neuron sẽ cho kết quả đầu ra luôn dương tại ngưỡng 0. Hàm số này càng ngày càng được sử dụng nhiều trong các kiến trúc mạng neuron vì những ưu điểm như có khả năng tăng tốc độ hội tụ của một mô hình sử dụng gradient descent nhanh hơn gấp 6 lần[9] so với hàm sigmoid/tanh. Đồng thời việc tính toán đối với hàm số này cũng đơn giản hơn rất nhiều so với hàm sigmoid và tanh.

Về bản chất một neuron có thể đóng vai trò như một bộ phân loại nhị phân,



Hình 2.4: Đồ thị và đạo hàm của hàm Tanh.



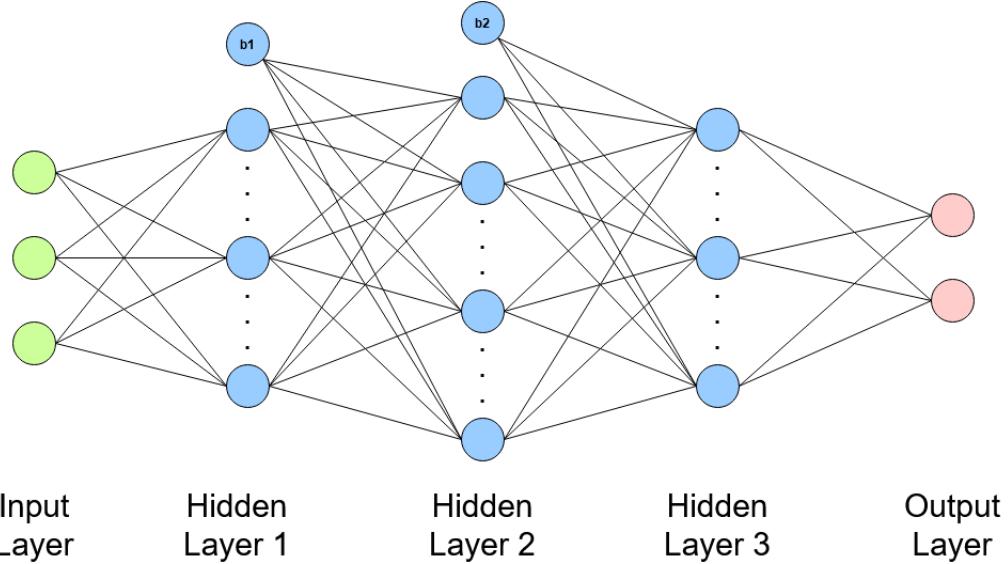
Hình 2.5: Đồ thị và đạo hàm của hàm Rectified Linear Unit.

tuy nhiên đối với các bài toán phức tạp, việc sử dụng một neuron đơn lẻ là bất khả thi. Để giải quyết các bài toán này, cần thiết phải có một mạng lưới gồm nhiều neuron liên kết với nhau. Đây cũng chính là nguồn cảm hứng cho nhiều nghiên cứu về các mô hình mạng neuron mở ra nhiều tiềm năng ứng dụng của trí tuệ nhân tạo trong cuộc sống.

2.1.2 Mạng lưới các neuron

Khác với mạng neuron sinh học, mạng neuron nhân tạo thường được tổ chức thành các lớp. Hình 2.6 thể hiện cấu trúc của một mạng neuron nhân tạo đơn giản. Lớp đầu tiên là lớp đầu vào (tiếng Anh: input layer) được dùng để lấy dữ liệu đầu vào cho mạng neuron. Lớp sau cùng là lớp đầu ra (tiếng Anh: output layer), lớp này sẽ cho ra kết quả tính toán sau cùng của các class cần được dự đoán với dữ liệu đầu vào tương ứng. Các lớp ở giữa được gọi là lớp ẩn (tiếng Anh: hidden layer) chứa các neuron có hàm kích và các liên kết có trọng số, đây chính là các lớp phụ trách việc trích xuất các đặc trưng và phân loại tín hiệu đầu vào.

Việc học của mạng neuron do vậy cũng diễn ra tại các lớp ẩn. Ta có thể thấy các neuron trong cùng một lớp thì không liên kết với nhau, ngoài ra mỗi neuron của lớp trước sẽ kết nối với mọi neuron của lớp sau ngoại trừ bias. Do đó các lớp này còn được gọi là các lớp kết nối hoàn toàn (tiếng Anh: fully connected layer).



Hình 2.6: Mô hình mạng neuron nhân tạo gồm 3 lớp hidden layer, 1 đầu vào và 1 đầu ra.

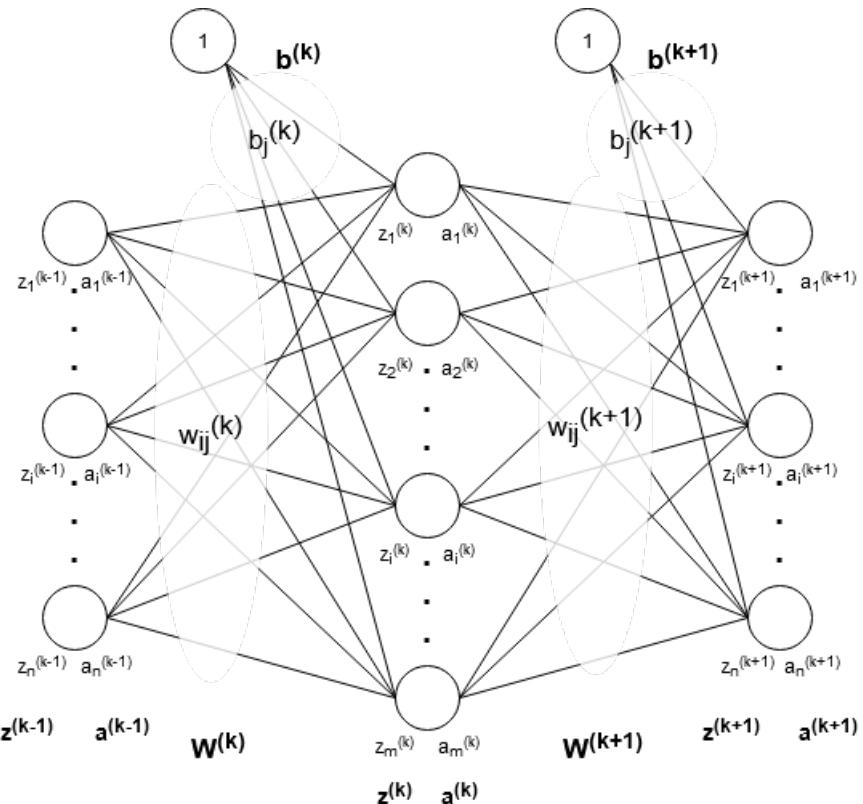
Quá trình mạng neuron nhân tạo lấy dữ liệu từ lớp đầu vào, đẩy dữ liệu qua các lớp ẩn và trả kết quả ở lớp đầu ra được gọi là quá trình *feed-forward*. Hình 2.7 và hệ phương trình 2.1 miêu tả mô hình toán học tổng quát của các lớp ẩn trong quá trình feed-forward của một mạng neuron.

$$\begin{aligned}
 a^{(0)} &= x \\
 z^{(k)} &= \mathbf{W}^{(k)T} a^{(k-1)} + \mathbf{b}^{(k)}, k = 1, 2, \dots, N \\
 a^{(k)} &= f^{(k)}(z^{(k)}), k = 1, 2, \dots, N \\
 \hat{y} &= a^{(N)}
 \end{aligned} \tag{2.1}$$

Với $\mathbf{W}^{(k)}$ là ma trận trọng số của các liên kết giữa lớp thứ $k - 1$ và lớp thứ k , $\mathbf{b}^{(k)}$ là vector bias của lớp thứ k , $z^{(k)}$ là vector đầu vào lớp ẩn thứ k , $a^{(k)}$ là vector đầu ra của lớp ẩn thứ k . Trong quá trình feed-forward, xét neuron thứ i của lớp thứ $k - 1$ với kết quả đầu ra $a_i^{(k-1)}$, đầu vào $z_j^{(k)}$ của neuron thứ j ở lớp thứ k là tổng tích các đầu ra ở lớp thứ $k - 1$ với các trọng số $w_{ij}^{(k)}$ cộng với bias $b_j^{(k)}$ như phương trình

$$z_j^{(k)} = \sum_i a_i^{(k-1)} \times w_{ij}^{(k)} + b_j^{(k)}. \tag{2.2}$$

Sau đó với mỗi giá trị đầu vào $z_i^{(k)}$ của một neuron thứ i của lớp thứ k ta sẽ có giá trị đầu ra của neuron là $a_i^{(k)} = f^{(k)}(z_i^{(k)})$ với $f^{(k)}$ là hàm kích của các neuron ở lớp thứ k . Quá trình này xảy ra liên tục giữa các lớp ẩn trong mạng neuron, do đó kết quả ở lớp đầu ra $\hat{y} = a^{(N)}$ với một dữ liệu đầu vào $a^{(0)} = x$ phụ thuộc vào các $\mathbf{W}^{(k)}$ và $\mathbf{b}^{(k)}$. Ban đầu các ma trận trọng số và vector bias được khởi tạo một



Hình 2.7: Mô hình toán của các lớp ẩn trong mạng neuron.

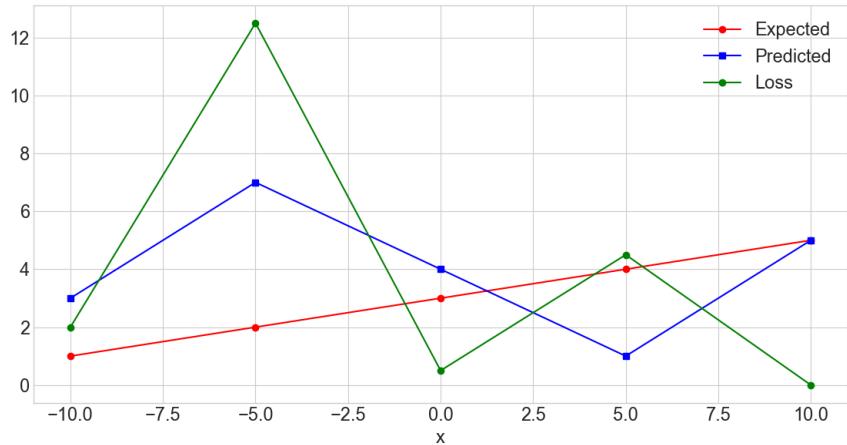
cách ngẫu nhiên hoặc theo một quy luật định trước, tuy nhiên kết quả dự đoán của mạng neuron với các giá trị khởi tạo này sẽ không chính xác cũng như con người không thể thuần thực một thao tác ngay lập tức mà phải trải qua một quá trình học, ý tưởng cho việc học của một mạng neuron là với mỗi giá trị đầu vào, sau quá trình feed-forward, mạng neuron nhân tạo có thể đánh giá được mức độ chính xác của việc tính toán ở các lớp ẩn và điều chỉnh các ma trận trọng số và vector bias để tăng độ chính xác hay nói cách khác là tối thiểu hóa độ sai lệch giữa kết quả dự đoán và kết quả thật sự.

2.1.3 Hàm mất mát và bài toán tối ưu mạng neuron

Để đánh giá được sự sai lệch giữa kết quả dự đoán được và kết quả thật sự, các mô hình mạng neuron sử dụng khái niệm hàm mất mát $J(\mathbf{W}, \mathbf{b}, \mathbf{X}, \mathbf{Y})$ (tiếng Anh: loss function). Một trong những hàm mất mát thường được sử dụng trong các mô hình mạng neuron nhân tạo cổ điển là hàm trung bình bình phương sai số (tiếng Anh: mean square error - MSE)

$$J(\mathbf{W}, \mathbf{b}, \mathbf{X}, \mathbf{Y}) = \frac{1}{M} \sum_{i=1}^M \|y_i - \hat{y}_i\|_2^2 = \frac{1}{M} \sum_{i=1}^M \|y_i - a_i^{(N)}\|_2^2. \quad (2.3)$$

Xét bài toán phân loại nhị phân với $M = 2$, trên hình 2.8 ta thấy rằng nếu mô hình càng đưa ra dự đoán chính xác thì J sẽ càng nhỏ do đó mục tiêu sau cùng của một mạng neuron nhân tạo trong quá trình học là tối thiểu hóa các giá trị của hàm mất mát với các điểm đầu vào khác nhau. Do J phụ thuộc vào các trọng



Hình 2.8: Giá trị hàm mất mát theo với các trường hợp dự đoán khác nhau.

số $w_i^{(k)} j$ và các bias $b_j^{(k)}$ nên sau khi đánh giá được mức độ chính xác của dự đoán, mạng neuron nhân tạo sẽ tiến hành điều chỉnh các giá trị này nhằm làm giảm giá trị J trong các lần tính sau. Phương pháp được dùng để thực hiện việc tối thiểu hàm mất mát là Gradient Descent.

2.2 Tối ưu hàm mất mát bằng Gradient Descent

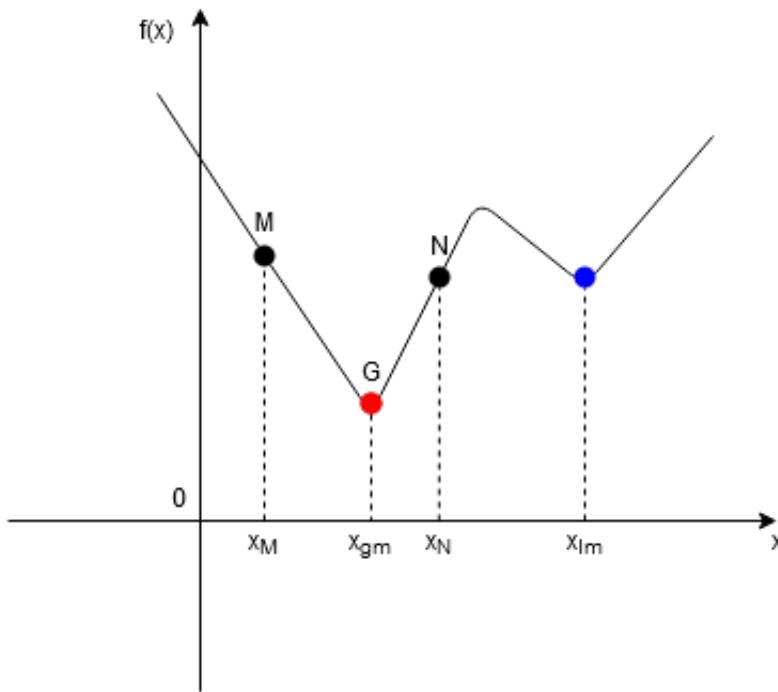
Để có thể hiểu cách mà Gradient Descent được dùng để tối thiểu hàm mất mát của một mô hình mạng neuron nhân tạo. Ta sẽ bắt đầu bằng một ví dụ đơn giản về Gradient Descent với một hàm số đơn biến, sau đó mở rộng ra cho hàm số đa biến.

2.2.1 Một ví dụ đơn giản về Gradient Descent

Một hàm số có thể có nhiều điểm cực tiểu địa phương (local minimum) và cực tiểu toàn cục (global minimum). Ta có thể thấy trên hình 2.9 là đồ thị của một hàm số đơn biến, điểm màu xanh là cực tiểu địa phương, điểm màu đỏ là cực tiểu toàn cục. Giả sử ta có hai điểm M tại x_M và N tại x_N trên đồ thị hình 2.9, ta gọi điểm cực tiểu toàn cục là G . Lúc này ta muốn đưa điểm M và N về xấp xỉ hoặc trùng với vị trí của G bằng giải thuật Gradient Descent. Ta nhận thấy M nằm bên trái G và $f'(x_M) < 0$, nếu M muốn di chuyển về G thì $x_{M_{k+1}} = x_{M_k} + \delta$ tại bước thứ $k + 1$. Ngược lại nếu ta muốn N có $f'(x_N) > 0$ tiến về phía G tại bước tính toán thứ $k + 1$ thì $x_{N_{k+1}} = x_{N_k} - \delta$. Như vậy để một điểm bất kì $(x, f(x))$ lân cận G trên đồ thị tiến về G thì vị trí của điểm đó phải được cập nhật sau mỗi bước tính toán bằng cách cộng với một lượng δ với $sign(\delta) = -sign(f'(x))$. Trong thực tế, công thức được sử dụng có dạng:

$$x_{k+1} = x_k - \mu f'(x_k) \quad (2.4)$$

Với μ là *tốc độ học*, $\mu \in \mathbb{R}$, $\mu > 0$. Nếu ta chọn μ lớn thì ta sẽ cần ít số bước tính toán hơn để đến gần vị trí cực tiểu mong muốn nhưng trong nhiều trường hợp độ sai lệch giữa vị trí của điểm tính toán được sau cùng và vị trí của điểm cực tiểu



Hình 2.9: Cực tiểu địa phương (màu xanh) và cực tiểu toàn cục (màu đỏ)

sẽ tương đối cao. Ngược lại, nếu ta chọn μ nhỏ thì ta sẽ cần nhiều hơn số bước tính toán, bù lại khoảng cách giữa vị trí điểm tính toán được sau cùng và vị trí điểm cực tiểu sẽ có thể rất nhỏ.

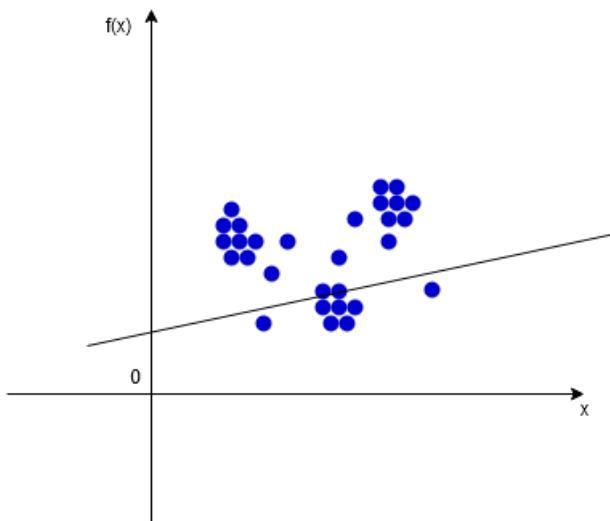
Hiện nay có ba phương pháp thuộc họ Gradient Descent đang được sử dụng:

1. Batch Gradient Descent

Giải thuật Batch Gradient Descent sử dụng tất cả các điểm đầu vào để cập nhật lại vector trọng số tại mỗi bước. Giả sử ta cần tối ưu hàm mất mát của một bài toán hồi quy tuyến tính gồm 30 điểm đầu vào với mỗi điểm gồm 2 tham số $(x, f(x))$ ở hình 2.10. Để tìm gradient cho mỗi điểm ta cần thực hiện 2 phép toán theo toán tử ∇ , đồng thời ta phải tìm gradient cho cả 30 điểm tại mỗi bước lặp và lấy trung bình của các kết quả này để cập nhật trọng số. Tổng số phép toán mà ta phải thực hiện cho tại mỗi bước là $30 \times 2 = 60$. Con số này sẽ tăng lên gấp nhiều lần đối với các bài toán thực tế khi số điểm và số tham số là vài triệu hoặc vài tỷ. Nói cách khác thuật toán này không hiệu quả về mặt tính toán với các bài toán máy học với dữ liệu lớn và phải cập nhật liên tục. Ngoài ra sau khi đã tìm được nghiệm tối ưu của bài toán. Nếu ta thêm một điểm đầu vào mới vào tập dữ liệu cũ thì việc tính toán phải thực hiện lại từ đầu với toàn bộ điểm đầu vào bao gồm tập điểm đầu vào cũ và điểm mới thêm vào.

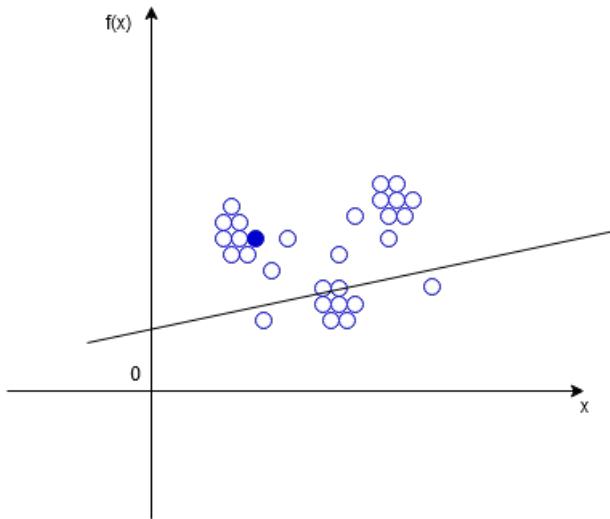
2. Stochastic Gradient Descent

Khác với Batch Gradient Descent giải thuật Stochastic Gradient Descent chỉ dùng gradient của một điểm ngẫu nhiên để cập nhật lại vector trọng số tại mỗi bước. Sau khi đi qua hết tất cả các điểm của tập đầu vào, thứ tự các điểm sẽ được xáo trộn và giải thuật lại tiếp tục với từng điểm. Mỗi một lần giải thuật Stochastic Gradient Descent tính toán xong với một điểm được



Hình 2.10: Batch Gradient Descent với bài toán hồi quy tuyến tính. Toàn bộ số điểm đầu vào đều được dùng để cập nhật các vector trọng số (a, b) cho đường hồi quy tại mỗi bước, với a là độ dốc và b độ sai lệch.

gọi là một *iteration* còn với toàn bộ tập điểm thì gọi là một *epoch*. Cũng bài toán hồi quy tuyến tính ở trên nhưng với giải thuật Stochastic Gradient Descent (hình 2.11), ta có thể thấy số iteration mà giải thuật Stochastic Gradient Descent phải thực hiện trong một epoch là 30. Số phép tính của một lần tính toán là 2.



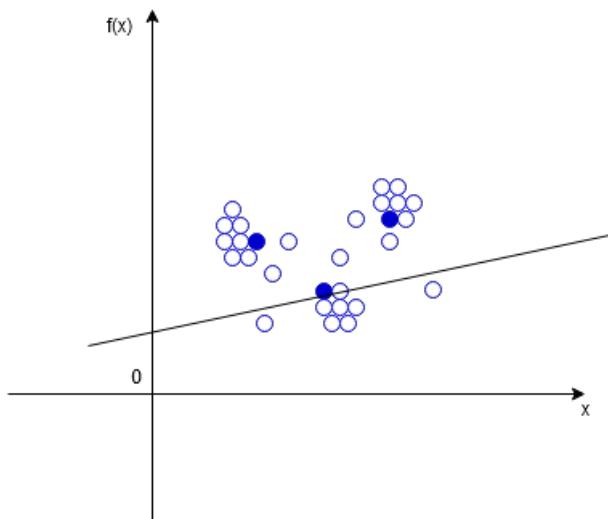
Hình 2.11: Stochastic Gradient Descent với bài toán hồi quy tuyến tính. Một điểm đầu vào được chọn ngẫu nhiên để cập nhật các vector trọng số (a, b) cho đường hồi quy tại mỗi iteration, với a là độ dốc và b độ sai lệch.

Do gradient của 1 điểm chỉ là xấp xỉ gần đúng của trung bình gradient của cả tập điểm nên việc cập nhật tại mỗi iteration sẽ có sai số nhặt định, đồng thời các giá trị gradient tính toán được có thể có sự dao động lớn do tập điểm đầu vào thường bị tác động bởi nhiều. Trên thực tế thì kết quả của giải thuật này có mức độ tối ưu khá tốt và hiệu quả tính toán cao. Sau khi

đã hoàn thành tính toán trên tập dữ liệu cũ, nếu như có những điểm mới được thêm vào thì ta chỉ cần chạy thuật toán với các điểm mới mà không cần phải chạy lại thuật toán bộ các điểm như Batch Gradient Descent.

3. Mini-batch Gradient Descent

Mini-batch Gradient Descent là sự kết hợp của Batch Gradient Descent và Stochastic Gradient Descent. Một mini-batch sẽ có n điểm với $1 < n \leq N$, N là tổng số điểm của tập dữ liệu đầu vào. Việc chia tập điểm ban đầu thành các batch sẽ được thực hiện một cách ngẫu nhiên. Mỗi một lần giải thuật xử lý xong một batch sẽ là một iteration và sau khi tất cả các batch được xử lý thì sẽ là một epoch. Như vậy $no_batch = \frac{N}{n}$. Phương pháp này cho kết quả gần với Batch Gradient Descent nhưng không dùng nhiều tài nguyên tính toán như Batch Gradient Descent và không cần phải lặp lại nhiều lần như Stochastic Gradient Descent.



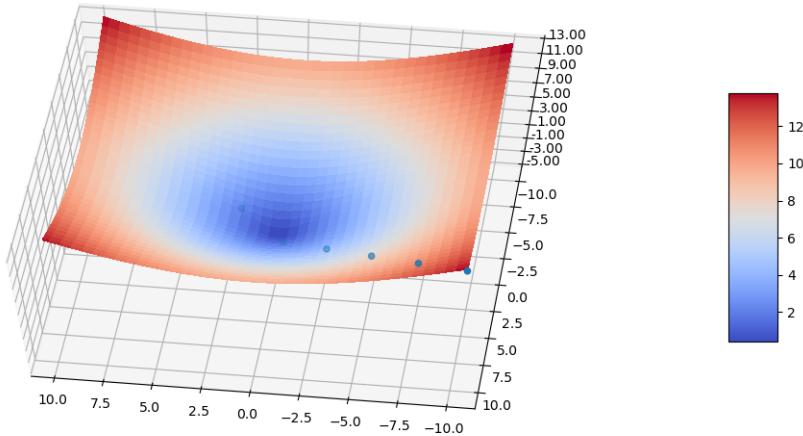
Hình 2.12: Mini-batch Gradient Descent với bài toán hồi quy tuyến tính. Một batch sẽ gồm ba điểm đầu vào được chọn ngẫu nhiên để cập nhật các vector trọng số (a, b) cho đường hồi quy tại mỗi iteration, với a là độ dốc và b độ sai lệch. Một epoch sẽ gồm mười batch.

2.2.2 Gradient Descent cho hàm đa biến

Việc áp dụng giải thuật Gradient Descent lên hàm đa biến là một sự mở rộng của ví dụ hàm đơn biến ở trên. Hình 2.13 miêu tả mặt phẳng của hàm măt măt trong trường hợp hàm đa biến và quá trình Gradient Descent trên mặt phẳng này. Cho hàm số $f : \mathbb{R}^n \rightarrow \mathbb{R}$, $n \in \mathbb{N}^*$, ta cần tìm cực tiểu cho $f(X)$ với $X = (x_0 \dots x_{n-1})$, $n \in \mathbb{N}^*$ từ một điểm khởi đầu X_0 bằng giải thuật Gradient Descent. Công thức để tính toán cho mỗi bước là:

$$X_{k+1} = X_k - \mu \nabla_X f (X_k). \quad (2.5)$$

Phương pháp Gradient Descent thường được dùng để cập nhật trọng số cho bài toán hàm măt măt đa biến là Mini-batch Gradient Descent.



Hình 2.13: Đồ thị mặt phẳng hàm măt măt và Gradient Descent cho hàm nhiều biến.

2.2.3 Điều kiện dừng của giải thuật

Ta đã biết các giải thuật Gradient Descent sẽ cần phải thực hiện rất nhiều vòng lặp tính toán để có thể hội tụ. Tuy nhiên rất khó để nói được khi nào có thể dừng được giải thuật. Trong thực tế có nhiều cách khác nhau được dùng để chọn số bước tính toán:

1. Chọn một số lượng vòng lặp nhất định dựa vào một số tiêu chí như số lượng dữ liệu đầu vào. Cách làm này có thể cho kết quả không đủ tốt, có thể nghiệm tối ưu nằm ở các bước trước hoặc sau điểm kết thúc.
2. Kiểm tra sự thay đổi của hàm măt măt giữa hai lần cập nhật liên tiếp, nếu sự sai lệch đạt tới ngưỡng đủ nhỏ thì ngưng giải thuật. Tuy nhiên nếu trên đồ thị của hàm măt măt có một vùng bằng phẳng nhưng không phải là cực tiểu thì giải thuật sẽ dừng tại điểm này mà không đạt được cực tiểu.
3. Kiểm tra sự thay đổi của gradient giữa hai lần cập nhật liên tiếp, nếu sự sai lệch đạt tới ngưỡng đủ nhỏ thì ngưng giải thuật. Nhược điểm của phương pháp này là việc tính gradient của các hàm phức tạp khó có thể thực hiện được.
4. Kiểm tra kết quả của giải thuật để ngừng việc lặp. Việc này cần người thực hiện việc huấn luyện mô hình phải thường xuyên kiểm tra các tham số hiệu năng của giải thuật lên một tập dữ liệu kiểm tra - *validation set* để xem tại thời điểm nào giải thuật có hiệu năng tốt nhất.

2.3 Sử dụng backpropagation để giải quyết vấn đề cập nhật trọng số trong mạng neuron

Hàm mất mát $J(\mathbf{W}, \mathbf{b}, \mathbf{X}, \mathbf{Y})$ sẽ phụ thuộc vào tập các ma trận trọng số \mathbf{W} và tập các vector bias của mỗi lớp \mathbf{b} . Việc tính gradient của hàm mất mát phụ thuộc vào việc tính các đạo hàm riêng $\frac{\partial J}{\partial \mathbf{W}^{(k)}}, \frac{\partial J}{\partial \mathbf{b}^{(k)}}, \forall k = 1, 2, \dots, N$.

Với N là số điểm trong tập điểm đầu vào. Ta nhận thấy để tìm các đạo hàm riêng của J với \mathbf{W} và \mathbf{b} trong trường hợp này là rất khó vì phương trình của J không phụ thuộc trực tiếp vào \mathbf{W} và \mathbf{b} . Để có thể hiện thực các giải thuật thuộc họ Gradient Descent thì phương pháp thường được sử dụng là Backpropagation. Phương pháp này sẽ cập nhật các trọng số theo chiều từ layer cuối cùng đến layer đầu tiên. Đầu tiên giải thuật sẽ tính đạo hàm của hàm mất mát theo ma trận trọng số của lớp cuối cùng.

$$\begin{aligned}\frac{\partial J}{\partial w_{ij}^{(N)}} &= \frac{\partial J}{\partial z_j^{(N)}} \cdot \frac{\partial z_j^{(N)}}{\partial w_{ij}^{(N)}} \\ &= e_j^{(N)} \frac{\partial (w_{ij}^{(N)T} a^{(N-1)} + b_j^{(N)})}{\partial w_{ij}^{(N)}} \\ &= e_j^{(N)} a_i^{(N-1)}\end{aligned}$$

Với $e_j^{(N)} = \frac{\partial J}{\partial z_j^{(N)}}$ có thể tính được tương đối dễ dàng. Tương tự ta có đạo hàm riêng của J với bias ở lớp cuối cùng.

$$\begin{aligned}\frac{\partial J}{\partial b_j^{(N)}} &= \frac{\partial J}{\partial z_j^{(N)}} \cdot \frac{\partial z_j^{(N)}}{\partial b_j^{(N)}} \\ &= e_j^{(N)}\end{aligned}$$

Các công thức trên cũng đúng với một lớp bất kỳ trong mạng neuron. Ta lấy mô hình hai lớp liên tiếp của một mạng neuron ở hình 2.7 để đưa ra công thức tổng quát như sau:

$$\begin{aligned}\frac{\partial J}{\partial w_{ij}^{(k)}} &= \frac{\partial J}{\partial z_j^{(k)}} \cdot \frac{\partial z_j^{(k)}}{\partial w_{ij}^{(k)}} \\ &= e_j^{(k)} \frac{\partial (w_{ij}^{(k)T} a^{(k-1)} + b_j^{(k)})}{\partial w_{ij}^{(k)}} \\ &= e_j^{(k)} a_i^{(k-1)}\end{aligned}$$

$$\begin{aligned}\frac{\partial J}{\partial b_j^{(k)}} &= \frac{\partial J}{\partial z_j^{(k)}} \cdot \frac{\partial z_j^{(k)}}{\partial b_j^{(k)}} \\ &= e_j^{(k)}\end{aligned}$$

Ta sẽ tính $e_j^{(k)}$ như sau:

$$\begin{aligned} e_j^{(k)} &= \frac{\partial J}{\partial z_j^{(k)}} = \frac{\partial J}{\partial a_j^{(k)}} \cdot \frac{\partial a_j^{(k)}}{\partial z_j^{(k)}} \\ &= \left(\sum_{l=1}^{d^{(k+1)}} \frac{\partial J}{\partial z_l^{k+1}} \cdot \frac{\partial z_l^{(k+1)}}{\partial a_j^{(k)}} \right) f^{(k)'}(z_j^{(k)}) \\ &= \left(\sum_{l=1}^{d^{(k+1)}} e_l^{(k+1)} \cdot w_{jl}^{(k+1)} \right) f^{(k)'}(z_j^{(k)}) \end{aligned}$$

Ta có $f : \mathbb{R} \rightarrow [0, 1]$ là hàm kích (activation function) hay còn gọi là hàm bao tại một node trong mạng neuron, $a_j^k = f(z_j^k)$, do đó ta có đạo hàm riêng của a_j^k theo z_j^k chính là đạo hàm của f . Ngoài ra do a_j^k trực tiếp tham gia vào việc tính các $z_l^{k+1}, l = 1, 2, \dots, d^{(k+1)}$ nên $\frac{\partial J}{\partial a_j^k}$ có thể tách ra thành tổng của tích các đạo hàm riêng như dòng thứ hai. Tương tự như vậy ta có thể tính

$$\frac{\partial J}{\partial b_j^{(k)}} = e_j^{(k)} \quad (2.6)$$

Việc tính e_j^k sẽ phụ thuộc vào kết quả của e_j^{k+1} do đó phương pháp này được gọi là Backpropagation. Các bước để thực hiện giải thuật Backpropagation cho một mạng neuron nhân tạo gồm:

1. Feedforward: Với mỗi giá trị đầu vào của x , tính giá trị đầu ra của mạng neuron, đồng thời lưu lại các kết quả $\mathbf{a}^{(k)}$ tại mỗi lớp.
2. Với mỗi node thứ j ở lớp ngoài cùng tính

$$e_j^{(N)} = \frac{\partial J}{\partial z_j^{(N)}} \quad (2.7)$$

3. Từ đó suy ra:

$$\begin{aligned} \frac{\partial J}{\partial w_{ij}^{(N)}} &= a_i^{(N-1)} e_j^{(N)} \\ \frac{\partial J}{\partial b_j^{(N)}} &= e_j^{(N)} \end{aligned}$$

4. Với $k = N - 1, N - 2, \dots, 1$ tìm $e_j^{(k)}$

$$e_j^{(k)} = \left(\sum_{l=1}^{d^{(k+1)}} e_l^{(k+1)} \cdot w_{jl}^{(k+1)} \right) f^{(k)'}(z_j^{(k)}) \quad (2.8)$$

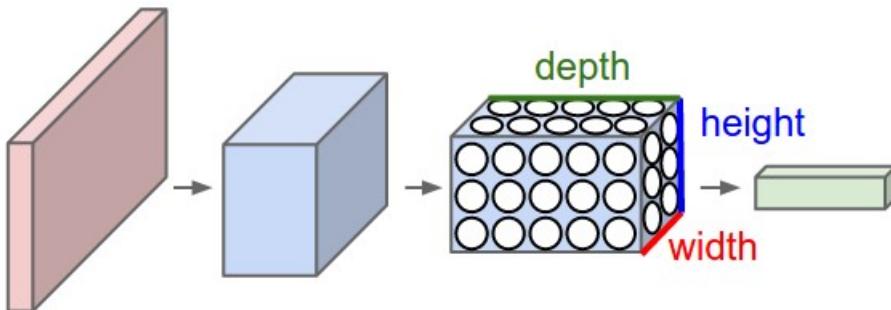
5. Cập nhật đạo hàm cho từng trọng số và bias:

$$\begin{aligned} \frac{\partial J}{\partial w_{ij}^{(k)}} &= a_i^{(k-1)} e_j^{(k)} \\ \frac{\partial J}{\partial b_j^{(k)}} &= e_j^{(k)} \end{aligned}$$

2.4 Mạng neuron tích chập[1][2]

Mạng neuron tích chập (tiếng Anh: Convolutional Neural Network - CNN) là một loại mạng neuron dùng riêng cho các bài toán về hình ảnh. Bên trong mạng neuron tích chập vẫn là các neuron có các trọng số và bias có thể cập nhật được để học các đặc trưng của hình ảnh.

Các lớp của mạng neuron tích chập được bố trí theo ba chiều: chiều rộng (tiếng Anh: width), chiều cao (tiếng Anh: height), chiều sâu (tiếng Anh: depth). Chiều sâu ở đây muốn nói tới chiều sâu của miền các neuron kích hoạt (tiếng Anh: activation volume) chứ không phải là chiều sâu của cả mạng neuron. Các neuron ở lớp sau sẽ chỉ được kết nối với một phần nhỏ các neuron ở lớp trước chứ không phải là toàn bộ như trong các mạng neuron thông thường. Ta lấy ví dụ mạng CIFAR-10 (hình 2.14), miền các neuron kích hoạt ở mạng neuron này có kích thước các chiều là $32 \times 32 \times 3$ (*rộng* \times *cao* \times *sâu*). Lớp cuối cùng của CIFAR-10 sẽ có kích thước các chiều là $1 \times 10 \times 10$ ứng với vector điểm cho các nhãn cần được phân loại (tiếng Anh: class scores). Một mạng neuron tích chập

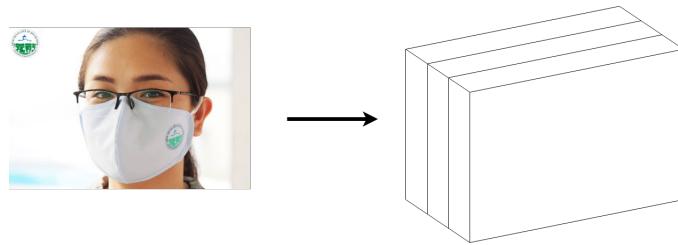


Hình 2.14: Mô hình mạng neuron tích chập đơn giản. Lớp nhận hình ảnh vào màu đỏ là một lớp có cấu trúc ba chiều với chiều rộng và chiều cao là chiều rộng và chiều cao của hình ảnh đầu vào, chiều sâu bằng ba ứng với ba kênh màu đỏ, xanh lá và xanh dương. Các lớp của mạng neuron tích chập sẽ chuyển đổi một nhóm các ma trận thành một nhóm các ma trận khác. Lớp ngoài cùng là lớp phân loại, có kích thước các chiều tương ứng với một vector.

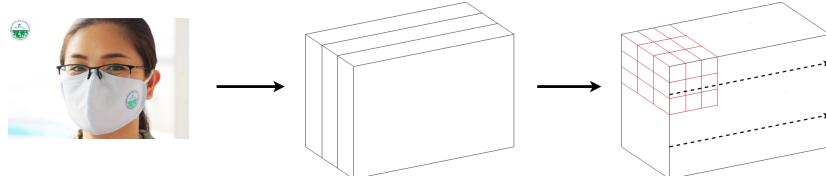
thông thường sẽ được cấu tạo từ ba loại lớp neuron: lớp tích chập (tiếng Anh: convolutional layer), lớp pooling (tiếng Anh: pooling layer) và lớp đầy đủ kết nối (tiếng Anh: fully-connected layer).

2.4.1 Lớp tích chập

Trong lớp này đầu vào lớp đầu tiên sẽ là một ảnh màu có ba kênh màu: đỏ, xanh lá, xanh dương (hình 2.15). Đầu ra của các lớp trước sẽ là đầu vào của các lớp sau. Các tensor trong mạng tích chập được gọi là các tensor. Sau đó một bộ lọc có kích thước $m \times n \times 3$ (tiếng Anh: kernel) sẽ được trượt qua tensor của ảnh đầu vào. Ở mỗi kênh màu, lớp tương ứng của kernel sẽ hoạt động như một cửa sổ trượt (tiếng Anh: sliding window). Nhắc lại một chút về phép toán của cửa sổ trượt trên ảnh trắng đen. Giả sử ta có một cửa sổ trượt có kích thước 3×3 đang quét qua một hình trắng đen (hình 2.17), tại vị trí như trên hình việc tính toán



Hình 2.15: Hình ảnh đầu vào gồm ba kênh màu được mô hình hóa thành tensor với chiều cao và chiều rộng là chiều cao và chiều rộng của ảnh, chiều sâu là ba.

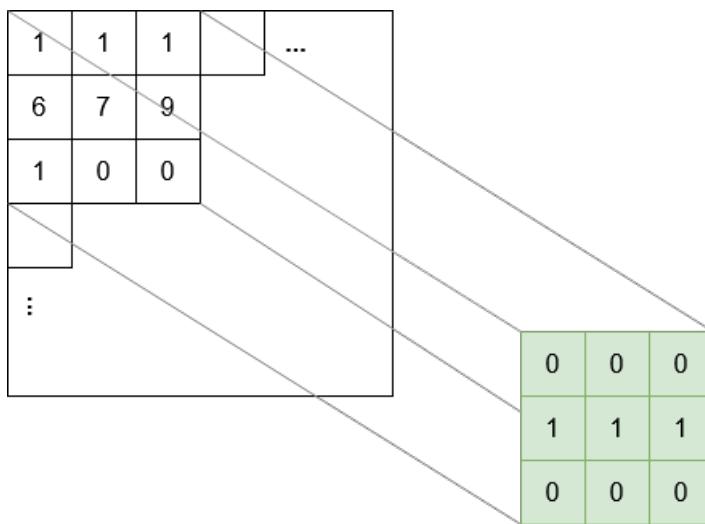


Hình 2.16: Hình ảnh sau khi được đưa vào và chuyển đổi thành dữ liệu ba chiều sẽ được đưa vào lớp convolution đầu tiên. Một kernel có kích thước $3 \times 3 \times 3$ (góc trên bên trái của mô hình ngoài cùng bên phải) được trượt qua hình đầu vào.

giá trị đầu ra được thực hiện như sau

$$1 \times 0 + 1 \times 0 + 1 \times 0 + 6 \times 1 + 7 \times 1 + 9 \times 1 + 1 \times 0 + 0 \times 0 + 0 \times 0 = 22 \quad (2.9)$$

Ngoài ra, ta còn hai khái niệm cần nhắc tới là stride và padding.



Hình 2.17: Ví dụ về phép toán cửa sổ trượt với kích thước 3×3 .

- Với stride bằng một thì cửa sổ trượt sẽ di chuyển tuần tự qua tất cả các ô của ma trận. Tổng quát với $stride = k$ thì các điểm ảnh được cửa sổ trượt đi qua của một ma trận có kích thước $m \times n$ sẽ là $x_{1+i \times k, 1+j \times k}$ với $i, j \in \mathbb{N}; 1 + i \times k \leq m; 1 + j \times k \leq n$.
- Đối với các điểm ảnh ở gần biên, nếu như trong vùng cửa sổ không có những chỗ không tồn tại giá trị điểm ảnh thì các phương pháp chèn giá trị (tiếng

Anh: padding) sẽ được sử dụng để thay làm các giá trị tính toán. Một trong các cách padding phổ biến là dùng các giá trị bằng không (tiếng Anh: zero padding).

0	0	0	0	0
0	2	3	-2	0
0	9	1	0	0
0	1	2	3	0
0	0	0	0	0

0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	2	3	-2	0	0
0	0	9	1	0	0	0
0	0	1	2	3	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

Hình 2.18: Bên trái, ma trận 3×3 được zero padding với $padding = 1$. Bên phải, ma trận 3×3 được zero padding với $padding = 2$

Như vậy, nếu đầu vào của phép tính tích chập là ma trận X có kích thước $m \times n$ với cửa sổ trượt có kích thước $k \times k$, $stride = s$, $padding = p$ thì đầu ra sẽ là một ma trận Y có kích thước $(\frac{m-k+2p}{s} + 1) \times (\frac{n-k+2p}{s} + 1)$

Việc tính toán tại mỗi kênh màu của hình khi kernel đi qua cũng gần tương tự với cửa sổ trượt. Kết quả phép toán của ba kênh màu và một bias sẽ được cộng lại và đưa vào ma trận kết quả. Giả sử ta có một kernel có kích thước $3 \times 3 \times 3$ như hình 2.19. Dữ liệu một ảnh đầu vào gồm ba kênh màu, khi đi qua lớp tích

-1	0	1
0	1	-1
1	-1	-1

0	0	0
1	1	1
0	0	0

0	1	0
0	1	0
0	1	0

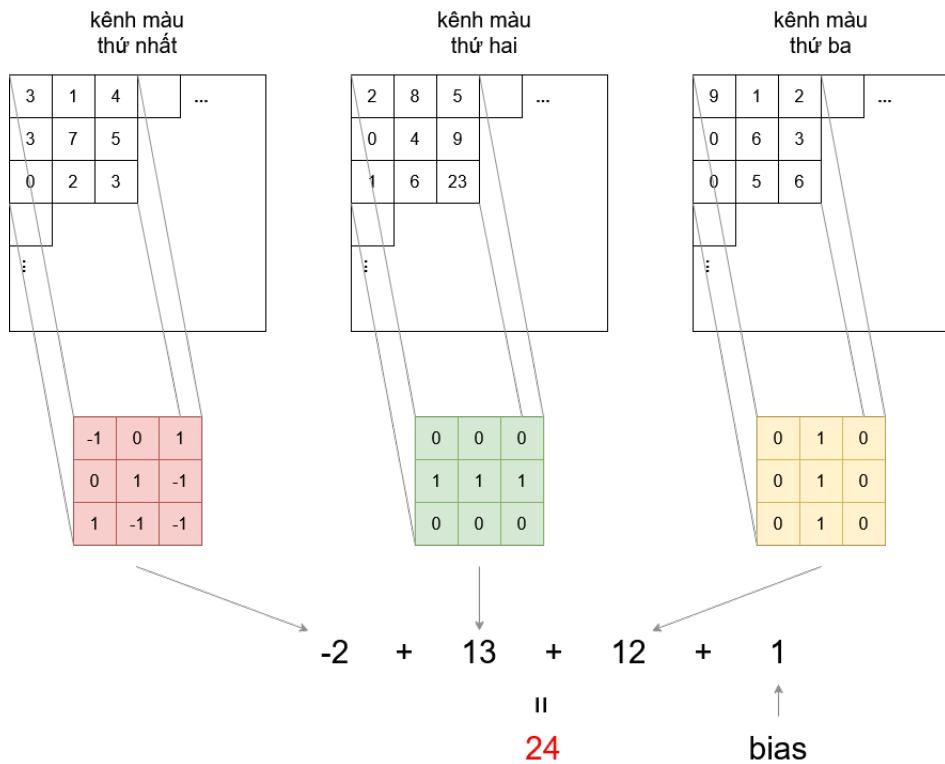
kênh kernel
thứ nhất

kênh kernel
thứ hai

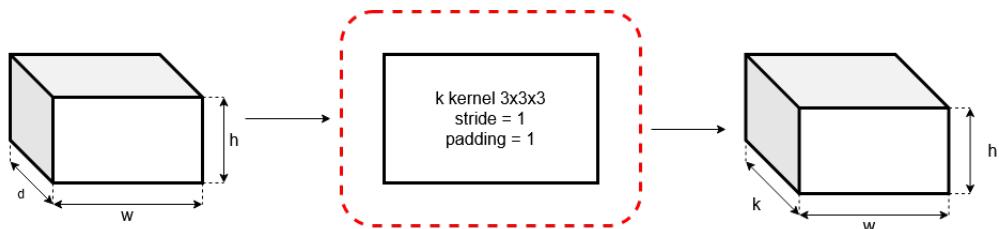
kênh kernel
thứ ba

Hình 2.19: Ví dụ về một kernel có kích thước $3 \times 3 \times 3$.

chập đầu tiên sẽ được tính toán như hình 2.20 Sau khi kernel đã quét qua hết các điểm ảnh mong muốn thì kết quả nhận được sẽ là một ma trận. Mỗi một kernel khác nhau sẽ trích xuất ra được một đặc trưng khác nhau của ảnh. Do đó một lớp tích chập sẽ có nhiều kernel để lấy các đặc trưng khác nhau. Lúc này đầu ra sẽ là một tensor gồm nhiều ma trận. Nếu như có K kernel được dùng tại một lớp tích chập thì đầu ra sẽ có chiều sâu bằng K , chiều rộng và chiều cao sẽ bằng chiều rộng và chiều cao của ảnh đầu vào. Đầu ra của lớp tích chập trước sẽ là đầu vào của lớp tích chập sau. Tổng quát hóa, với một lớp tích chập với K kernel có kích thước $N \times N \times D$ (với D là chiều sâu của đầu vào và là số lẻ), $stride = S$,



Hình 2.20: Ví dụ về phép toán của một kernel lên một vị trí của ảnh trong lớp tích chập.



Hình 2.21: Một lớp tích chập có k kernel với kích thước $3 \times 3 \times 3$, $stride = 1$, $padding = 1$. Đầu vào là một tensor có kích thước $h \times w \times d$ đầu ra của phép tích chập lên tensor này khi khôi tích chập có các thông số ở trên là một tensor có kích thước $h \times w \times k$

$padding = P$. Đầu vào là một tensor có kích thước $H \times W \times D$ thì kích thước của tensor đầu ra sẽ là $(\frac{H-F+2P}{S} + 1) \times (\frac{W-F+2P}{S} + 1) \times K$

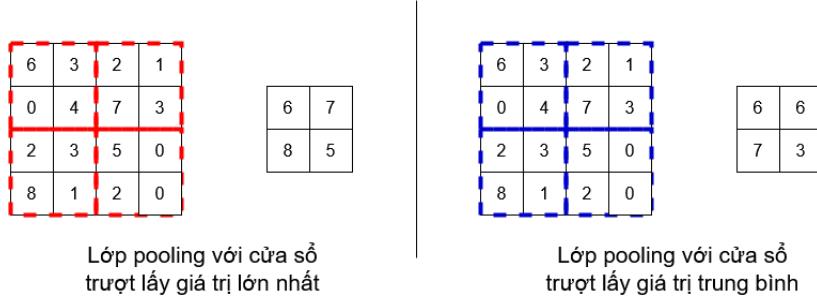
Dầu ra của lớp tích chập sẽ đi qua hàm kích hoạt trước khi được đưa vào lớp tích chập tiếp theo. Mỗi kernel với kích thước $N \times N \times D$ sẽ có một hệ số bias tương ứng với tổng số các tham số của một kernel là $N \times N \times D + 1$, với K kernel thì số tham số sẽ là $K \times (N \times N \times D + 1)$.

2.4.2 Lớp pooling

Việc tính toán với toàn bộ dữ liệu đầu vào của ảnh có độ phân giải lớn và kích thước lớn trên mạng neuron tích chập thường không hiệu quả về mặt tính toán do sẽ có nhiều điểm ảnh miêu tả cùng một đặc trưng. Do đó lớp pooling được dùng ở giữa các lớp tích chập để giảm kích thước của các tensor nhưng vẫn

không làm mất đi các đặc trưng của dữ liệu.

Cho một lớp pooling có kích thước cửa sổ trượt là $N \times N$, đầu vào là một tensor có kích thước $H \times W \times D$. Ta chia tensor này thành D ma trận $H \times W$. Với mỗi ma trận ta lần lượt trượt cửa sổ trượt của lớp pooling lên từng điểm ảnh. Trong vùng dữ liệu của cửa sổ trượt ta sẽ tìm giá trị lớn nhất hoặc trung bình của các giá trị để đưa vào ma trận mới. Một số mô hình mạng neuron tích chập



Hình 2.22: Bên trái, lớp pooling với cửa sổ trượt lấy giá trị lớn nhất với kích thước cửa sổ 2×2 , $stride = 1$, $padding = 0$. Bên phải, lớp pooling với cửa sổ trượt lấy giá trị trung bình với kích thước cửa sổ 2×2 , $stride = 2$, $padding = 0$.

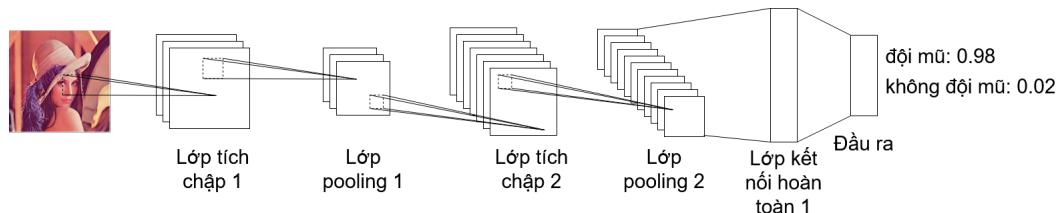
sẽ dùng $stride > 1$ trong lớp tích chập để làm giảm kích thước dữ liệu thay vì dùng lớp pooling. Ngoài ra, trong thực tế lớp pooling thường được sử dụng với kích thước cửa sổ trượt 2×2 , $stride = 2$, $padding = 0$. Chiều cao và chiều rộng của tensor đầu ra sẽ giảm đi một nửa còn chiều sâu vẫn giữ nguyên.

2.4.3 Lớp đầy đủ kết nối

Hình ảnh sau khi qua các lớp tích chập và pooling thì đầu ra sẽ là một tensor chứa các đặc trưng mà mô hình trích xuất được. Tensor có kích thước $H \times W \times D$ tại lớp tích chập cuối cùng sẽ được chuyển thành một vector có chiều dài $H \times W \times D$. Sau đó vector này sẽ được đưa vào các lớp đầy đủ kết nối để đưa ra kết quả dự đoán cho ảnh.

2.4.4 Mô hình mạng neuron tích chập

Ảnh đầu vào \rightarrow [Lớp tích chập \rightarrow Lớp pooling] $\times n \rightarrow$ [Lớp liên kết hoàn toàn] $\times m \rightarrow$ Đầu ra, với $m, n \in \mathbb{N}^*$.



Hình 2.23: Mạng neuron tích chập gồm hai lớp tích chập và pooling, một lớp kết nối đầy đủ.

2.5 Mô hình YOLOv3

YOLO - You Only Look Once là một trong những mô hình nhận diện thời gian thực hiện đại nhất và đang được sử dụng cho nhiều bài toán nhận diện, theo dõi khác nhau. Không như những mạng CNN hay R-CNN trước đây, thay vì sử dụng phương pháp dự đoán trên từng miền (tiếng Anh: region proposal method) và cửa sổ trượt (tiếng Anh: sliding window) để phát hiện vật thể trong từng vùng nhỏ trong khung hình và tiến hành phân loại vật thể đó. YOLO sử dụng cả khung hình để nhận diện vật thể, các đặc trưng của vùng nền phía sau (tiếng Anh: background) cũng được dùng trong quá trình huấn luyện. Do vậy YOLO có thể nhận diện vật thể một cách nhanh chóng trong một khung hình với độ chính xác cao chỉ với một lần xử lý, đó cũng là lý do vì sao mô hình này được gọi là Bạn Chỉ Cần Nhìn Một Lần (tiếng Anh: You Only Look Once).

2.5.1 Unified Detection

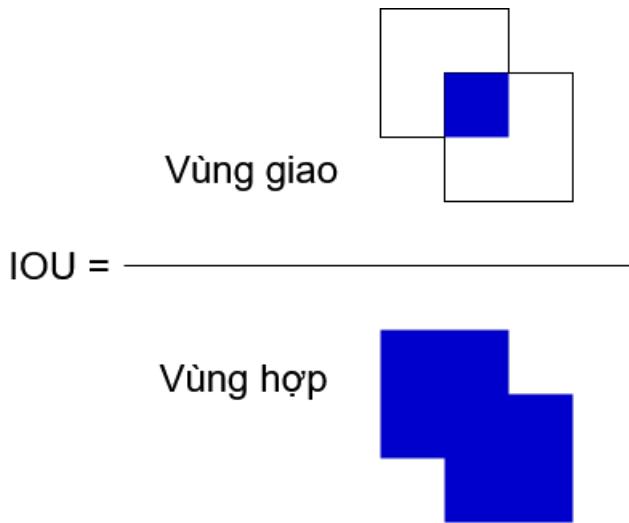
Hình ảnh đầu vào được chia thành một mạng lưới ô vuông (tiếng Anh: grid cell) có kích thước $S \times S$. Nếu như tâm của một vật thể nằm ở tâm của một ô thì ô đó sẽ chịu trách nhiệm trong việc nhận diện vật thể đó. Mỗi ô sẽ



Hình 2.24: Hình ảnh được chia thành mạng lưới ô vuông $S \times S$.

dự đoán B bounding box và độ tin cậy (tiếng Anh: confidence score) của từng bounding box. Gọi $Pr(\text{Object})$ là xác suất của vật thể nằm trong một ô với $Pr(\text{Object}) \in \mathbb{R}, 0 \leq Pr(\text{Object}) \leq 1$. IOU - intersection over union là tỷ lệ giữa diện tích miền giao và diện tích miền hợp của bounding box dự đoán được và bounding box được tạo sẵn để huấn luyện (tiếng Anh: ground truth) hình 2.25, $IOU \in \mathbb{R}, 0 \leq IOU \leq 1$. Độ tin cậy sẽ được định nghĩa bằng $Pr(\text{Object}) \times IOU$, nếu như một ô không chứa vật thể thì $Pr(\text{Object}) = 0$ suy ra độ tin cậy sẽ bằng không, ngược lại nếu một ô chứa vật thể thì $Pr(\text{Object}) = 1$, lúc này độ tin cậy bằng IOU .

Mỗi một bounding box sẽ có năm tham số cần dự đoán: t_x, t_y, t_w, t_h và độ tin cậy. (t_x, t_y) là tọa độ tương đối của tâm bounding box với một ô, nếu gọi offset của một ô trong hình là (c_x, c_y) thì tọa độ của một bounding box so với hình sẽ là $(\sigma(t_x) + c_x, \sigma(t_y) + c_y)$. Nếu chiều dài và chiều rộng của bounding box cho trước là (p_w, p_h) chiều dài và chiều rộng của bounding box được sự đoán sẽ là

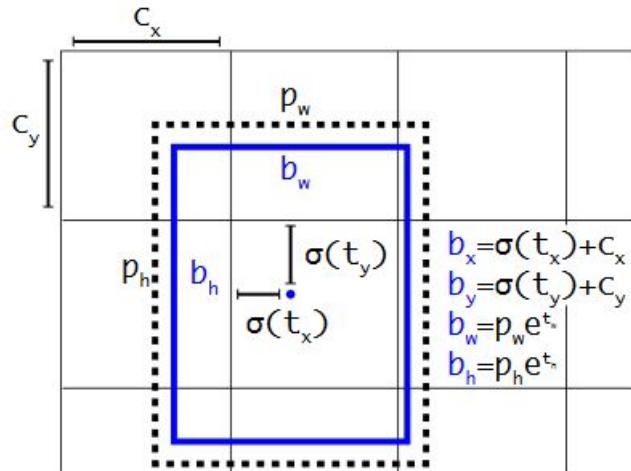


Hình 2.25: Miêu tả việc tính toán IOU.

$$(p_w \times e^{t_w}, p_h \times e^{t_h}).$$

$$\begin{aligned} b_x &= \sigma(t_x) + c_x \\ b_y &= \sigma(t_y) + c_y \\ b_w &= p_w \times e^{t_w} \\ b_h &= p_h \times e^{t_h} \end{aligned}$$

Trong quá trình huấn luyện, hàm mất mát tổng bình phương sai số được sử dụng



Hình 2.26: Mô hình dự đoán bounding box của YOLO.

cho các tọa độ của bounding box. Với một dự đoán t_* có ground truth là \hat{t}_* thì gradient sẽ là hiệu giữa ground truth và kết quả dự đoán được $\hat{t}_* - t_*$.

YOLOv3 sử dụng hàm hồi quy logistic để dự đoán xem một bounding box có chứa vật thể hay không. Nếu như bounding box dự đoán được giao với ground truth nhiều hơn các bounding box trước đó thì kết quả là 1. Nếu như một bounding box dự đoán được không phải là trường hợp có diện tích giao lớn nhất với ground truth nhưng vẫn có độ tin cậy lớn hơn một ngưỡng xác quyết thì dự đoán với bounding box này sẽ bị bỏ qua. Ngưỡng xác quyết được dùng là 0.5. Nếu như

một bounding box không được đặt vào một ground truth thì sẽ không có các dự đoán về tọa độ và class mà chỉ có dự đoán về sự tồn tại của vật thể.

Dộ tin cậy chính là IOU giữa bounding box dự đoán được và bounding box được tạo sẵn. Đối với những ô được dự đoán có vật thể, mô hình sẽ dự đoán thêm C xác suất của các class mà vật thể đó thuộc về $Pr(Class_i|Object)$ với $Pr(Class_i|Object) \in \mathbb{R}, 0 \leq Pr(Class_i|Object) \leq 1, i = 1, \dots, C$. Mỗi ô sẽ chỉ có một tập các giá trị $Pr(Class_i|Object)$ mà không liên quan tới số lượng bounding box B .

Khi tiến hành dự đoán, YOLO sẽ nhân các xác suất và độ tin cậy lại với nhau để được xác suất của một class trên một bounding box.

$$Pr(Class_i|Object) \times Pr(Object) \times IOU = Pr(Class_i) \times IOU \quad (2.10)$$

Hàm mất mát được sử dụng khi huấn luyện để dự đoán các class là hàm binary cross-entropy nhằm giúp mô hình có thể dự đoán đa lớp trong cùng một ô.

Sau cùng ta có hàm mất mát tổng của YOLOv3

$$\begin{aligned} Loss &= \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbf{1}_{ij}^{obj} \left[(\hat{t}_{x_{ij}} - t_{x_{ij}})^2 + (\hat{t}_{y_{ij}} - t_{y_{ij}})^2 + (\hat{t}_{w_{ij}} - t_{w_{ij}})^2 + (\hat{t}_{h_{ij}} - t_{h_{ij}})^2 \right] \\ &\quad + \sum_{i=0}^{S^2} \sum_{j=0}^B - [\hat{p}_{o_{ij}} \times \log(p_{o_{ij}}) + (1 - \hat{p}_{o_{ij}}) \times \log(1 - p_{o_{ij}})] \\ &\quad + \sum_{i=0}^{S^2} \mathbf{1}_{ij}^{obj} \sum_{k=1}^C - [\hat{p}_i(c_k) \times \log(p_i(c_k)) + (1 - \hat{p}_i(c_k)) \times \log(1 - p_i(c_k))] \end{aligned}$$

2.5.2 Kiến trúc mạng YOLOv3

Các đặc trưng từ ảnh được trích xuất theo ba tỷ lệ khác nhau bằng phương pháp giống như phương pháp được sử dụng trong mạng trích xuất đặc trưng dạng kim tự tháp (tiếng Anh: feature pyramid networks). Đầu ra của miền trích xuất đặc trưng là một tensor ba chiều chứa vị trí bounding box, xác suất tồn tại vật thể, xác suất các class, kích thước của tensor là $S \times S \times [3 \times (4 + 1 + C)]$, với $S \times S$ là số ô mà ảnh được chia thành, 4 là các giá trị dự đoán của bounding box (t_x, t_y, t_w, t_h), 1 là giá trị dự đoán sự tồn tại của vật thể trong ô, C là vector các giá trị dự đoán của các class.

Sau đó các ma trận đặc trưng từ hai lớp trước và upsample lên hai lần. Ngoài ra các ma trận đặc trưng từ các lớp đầu cũng được ghép lại với các lớp sau. Việc này giúp mô hình lấy được nhiều thông tin có ý nghĩa từ các ma trận đặc trưng được upsample và vẫn giữ được được những đặc trưng nhỏ hơn từ các lớp đầu. Sau đó một vài lớp tích chập được sử dụng để kết hợp các ma trận đặc trưng và đưa ra tensor sau cùng chứa các dự đoán có kích thước gấp đôi.

Với tỷ lệ cuối, mô hình như ở trên được lặp đi lặp lại nhiều lần, điều này cho phép các bounding box ở tỷ lệ cuối có thể dùng các giá trị đã được tính toán từ các tỷ lệ trước cũng nhưng các đặc trưng có ý nghĩa.

Ban đầu sẽ có chín bounding box: (10×13) , (16×30) , (33×23) , (30×61) , (62×45) , (59×119) , (116×90) , (156×198) , (373×326) . Các bounding box này sẽ được chia vào ba tỷ lệ bằng giải thuật k-means. Các bounding box này sẽ được dùng để dự đoán ở các tỷ lệ tương ứng. Ảnh đầu vào sẽ được downsample với các stride

bằng 32, 16 và 8 cho từng tỷ lệ. Nhờ vậy YOLOv3 có thể dự đoán được các vật thể nhỏ rất tốt.

YOLOv3 sử dụng kiến trúc mạng kết hợp giữa YOLOv2, Darknet-19 và mạng residual. Các lớp tích chập có kích thước 3×3 , 1×1 và có thêm các liên kết tắt (tiếng Anh: short cut). Có tất cả 53 lớp tích chập được dùng nên kiến trúc này được gọi là Darknet-53. Kiến trúc này cho khả năng nhận diện tốt hơn Darknet-19

Type	Filters	Size	Output
Convolutional	32	3×3	256×256
Convolutional	64	$3 \times 3 / 2$	128×128
1x	Convolutional	32	1×1
1x	Convolutional	64	3×3
1x	Residual		128×128
2x	Convolutional	128	$3 \times 3 / 2$
2x	Convolutional	64	1×1
2x	Convolutional	128	3×3
2x	Residual		64×64
8x	Convolutional	256	$3 \times 3 / 2$
8x	Convolutional	128	1×1
8x	Convolutional	256	3×3
8x	Residual		32×32
8x	Convolutional	512	$3 \times 3 / 2$
8x	Convolutional	256	1×1
8x	Convolutional	512	3×3
8x	Residual		16×16
4x	Convolutional	1024	$3 \times 3 / 2$
4x	Convolutional	512	1×1
4x	Convolutional	1024	3×3
4x	Residual		8×8
	Avgpool		Global
	Connected		1000
	Softmax		

Hình 2.27: Kiến trúc mạng Darknet-53.

và hiệu năng cao hơn ResNet-101 hoặc ResNet-152 bảng 2.1.

Backbone	Top-1	Top-5	Bn Ops	BFLOP/s	FPS
Darknet-19	74.1	91.8	7.29	1246	171
ResNet-101	77.1	93.7	19.7	1039	53
ResNet-152	77.6	93.8	29.4	1090	37
Darknet-53	77.2	93.8	18.7	1457	78

Bảng 2.1: So sánh hiệu năng của Darknet-53 với các mạng khác. Accuracy, Bn Ops - billions of operations, BFLOP/s - billion floating point operations per second, và FPS - frames per second.

Chương 3

Phương pháp tiếp cận

3.1 Xây dựng tập dữ liệu

3.1.1 Xác định yêu cầu bài toán

Bài toán đặt ra là nhận diện người trong khung hình có đang đeo các thiết bị bảo hộ cá nhân (tiếng Anh: personal protective equipment) hay không. Các thiết bị bảo hộ cá nhân được chọn để nhận diện là: mũ bảo hộ, áo bảo hộ và khẩu trang.



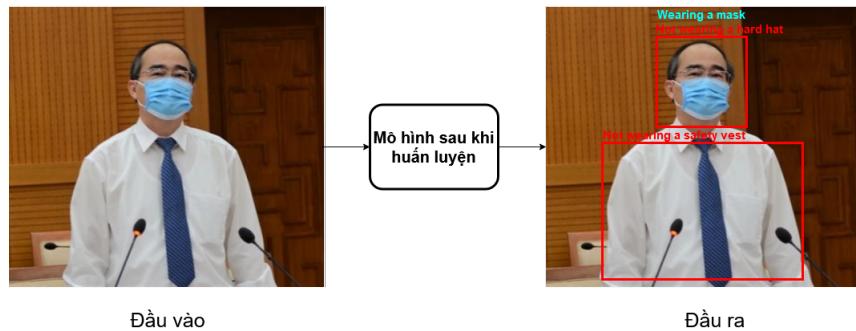
Hình 3.1: (1) Mũ bảo hộ, (2) Áo bảo hộ, (3) Khẩu trang

Mục tiêu đầu ra của hệ thống là có thể xác định được vị trí đầu người và thân người và phân loại việc sử dụng các thiết bị bảo hộ cá nhân đối với các vật thể đã được phát hiện. Ứng với mỗi thiết bị, vật thể sẽ được phân loại thành hai trạng thái, một là *Wearing - Mặc*, hai là *Not wearing - Không mặc*.

- Wearing a hardhat
- Not wearing a hardhat
- Wearing a safety vest
- Not wearing a safety vest
- Wearing a mask
- Not wearing a mask

Hình 3.2 minh họa đầu vào đầu ra mong muốn của hệ thống.

Do vậy tập dữ liệu sẽ được xây dựng với các hình ảnh chứa con người và các nhãn được đánh đúng với mong muốn của đầu ra.



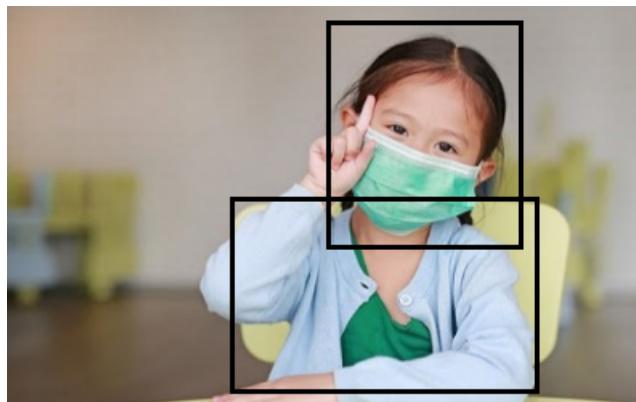
Hình 3.2: Kết quả nhận dạng mong muốn.

3.1.2 Thu thập hình ảnh và dán nhãn

Tập dữ liệu gồm 11586 hình trong đó

- 3541 hình được lấy từ tập dữ liệu *Hardhat and Safety Vest Image for Object Detection*[10]
- 3174 hình được lấy từ tập dữ liệu *GDUT-HWD*[11]
- 4871 hình được lấy từ công cụ tìm kiếm hình ảnh *Google image*

Các hình được dán nhãn bằng phần mềm *LabelImg*[12]. Mỗi hình sẽ có tương ứng một tệp tin văn bản với đuôi *.txt*. Bên trong tệp tin này là các nhãn được đánh dấu bằng định dạng của YOLO với các thông tin gồm *object class id* - đây là id tương ứng với thứ tự của một nhãn trong danh sách các nhãn, *x* và *y* là tọa độ tương đối của bounding box được đánh dấu với hình, *w* và *h* là chiều rộng và chiều cao tương đối của bounding box được đánh dấu với hình, hình ??.

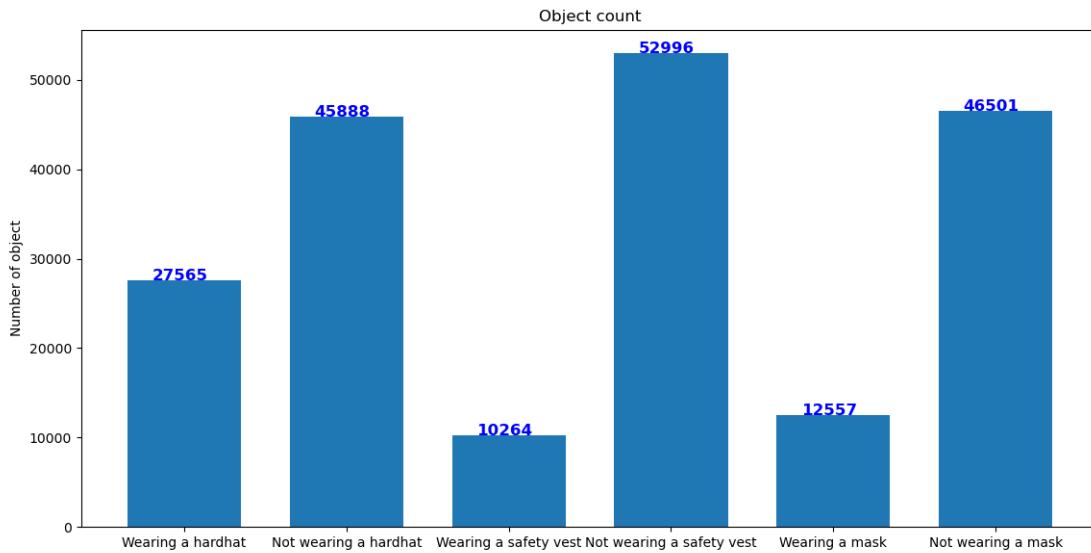


```
3 0.590820 0.732087 0.404297 0.448598
1 0.649414 0.323988 0.306641 0.560748
4 0.649414 0.323988 0.306641 0.560748
```

Hình 3.3: Định dạng nhãn của YOLO.

Tập dữ liệu này có tổng cộng 195771 vật thể được dán nhãn với thống kê số vật thể của từng class được thể hiện trong hình 3.4.

Việc chênh lệch lớn về số lượng các bounding box giữa các class cụ thể là các class *Not wearing* có số lượng lớn hơn rất nhiều so với các class *Wearing* tương ứng sẽ giúp bộ nhận dạng nhạy hơn với các trường hợp vi phạm trang phục bảo



Hình 3.4: Thống kê số lượng vật thể ứng với từng class. Wearing a hardhat: 27565, Not wearing a hardhat: 45888, Wearing a safety vest: 10264, Not wearing a safety vest: 52996, Wearing a mask: 12557, Not wearing a mask: 46501.

về lao động. Tuy có sự chênh lệch lớn nhưng số lượng các bounding box của mỗi class là đủ lớn để mô hình có thể học được các đặc trưng cần thiết để có thể phân loại class tốt cho một bounding box.

3.2 Huấn luyện mạng YOLOv3 sử dụng framework Darknet[3]

Darknet là một framework được xây dựng bởi Joseph Redmon cũng là cha đẻ của YOLO, framework này được viết bằng C/C++ và được dùng để huấn luyện mô hình YOLOv3 với tập dữ liệu riêng cho từng vấn đề. Kiến trúc của Darknet đã được đề cập ở phần lý thuyết và sẽ không được nhắc lại ở chương này.

Mô hình YOLOv3 trong luận văn này được huấn luyện trên *GoogleColab*, về bản chất môi trường trên *GoogleColab* là môi trường máy ảo chạy Linux với các thông số tại thời điểm thực hiện luận văn:

- Hệ điều hành: Ubuntu 18.04.3 LTS
- Chip xử lý: Intel 2-core Xeon 2.2GHz
- RAM: 13Gb
- HDD: 33Gb
- GPU: Tesla K80 with 12GB memory

Để có thể huấn luyện được mô hình YOLOv3 cho bộ dữ liệu riêng, ta cần thực hiện một số bước.

1. Tải framework Darknet từ github repository của AlexeyAB.

```
https://github.com/AlexeyAB/darknet
```

Sau đó ta chọn Clone → Download ZIP và tiến hành tải thư mục về, sau khi tải xong ta tiến hành giải nén vào một thư mục mà ta tạo sẵn.

2. Ta chép tệp tin *yolov3.cfg* trong thư mục *cfg* ra thư mục làm việc *darknet* và chỉnh sửa như sau. Đầu tiên ta sẽ sửa các giá trị *batch* là số hình trong một mini-batch mà ta muốn dùng để huấn luyện, *subdivision* là thông số để chia nhỏ một mini-batch để đảm bảo mô hình có thể chạy trên các tài nguyên GPU khác nhau, *width* và *height* là chiều rộng và chiều cao của ảnh đầu vào, các ảnh có kích thước khác nhau sẽ được resize lại kích thước này trước khi được đưa vào để huấn luyện. *max_batches* là tổng số batch mà mô hình sẽ chạy qua, đây còn được gọi là số *iteration*. *steps* sẽ có giá trị lần lượt bằng 80% và 90% của *max_batches*.

```
# [net]
# Testing
# batch=1
# subdivisions=1
# Training
batch=64
subdivisions=64
width=608
height=608
channels=3
momentum=0.9
decay=0.0005
angle=0
saturation = 1.5
exposure = 1.5
hue=.1

learning_rate=0.001
burn_in=1000
max_batches = 22000
policy=steps
steps=17600,19800
scales=.1,.1
```

Sau đó tại các dòng 610, 696 và 783 ta sẽ thay số *classes* bằng sáu, chính là số class của bài toán. Đồng thời ta sẽ sửa số *filters* của lớp convolution ngay phía trên theo công thức $3 \times (5 + C)$ với C là số class, khi $C = 6$ ta có *filters* = 33.

```
[convolutional]
size=1
stride=1
pad=1
filters=33
activation=linear

[yolo]
mask = 0, 1, 2
anchors = 10, 13, 16, 30, 33, 23, 30, 61, 62, 45, 59, 119, 116, 90, 156
, 198, 373, 326
classes=6
num=9
jitter=.3
ignore_thresh = .7
truth_thresh = 1
random=1
```

3. Sửa các thông số ở đầu của tệp *Makefile* như sau.

```
GPU=1
CUDNN=0
CUDNN_HALF=0
OPENCV=1
AVX=0
OPENMP=0
LIBSO=0
ZED_CAMERA=0 # ZED SDK 3.0 and above
ZED_CAMERA_v2_8=0 # ZED SDK 2.X
```

4. Tạo tệp có tên *obj.names* chứa tên các class.

```
Wearing a hardhat
Not wearing a hardhat
Wearing a safety vest
Not wearing a safety vest
Wearing a mask
Not wearing a mask
```

5. Sao chép tập dữ liệu gồm hình ảnh và tệp tin dán nhãn vào đường dẫn *./data/objects*
6. Tạo hai tệp text, *train.txt* và *val.txt* chứa đường dẫn đến các hình ảnh. *train.txt* sẽ được dùng để huấn luyện còn *val.txt* sẽ được dùng để validate mô hình trong quá trình huấn luyện. Việc chia tập dữ liệu được thực hiện ngẫu nhiên. Có 900 hình được dùng để validate và 10686 hình được dùng để huấn luyện. Việc chia này được thực hiện bằng một chương trình Python.

```

import os
import glob
import cv2
import random

basenames = [os.path.basename(x) for x in glob.glob("./data/objects/*.jpg")]
basenamesNotEmpty = []

for name in basenames:
    if "empty" not in name:
        basenamesNotEmpty.append(name)

train = random.sample(basenames, len(basenames))
valid = random.sample(basenamesNotEmpty, 900)

with open("./val.txt", "w") as f:
    for name in valid:
        f.write("data/objects/" + name + "\n")

with open("./train.txt", "w") as f:
    for name in train:
        if name not in valid:
            f.write("data/objects/" + name + "\n")

```

7. Tạo tệp có tên *obj.data* chứa tên các class.

```

classes = 6
train = train.txt
valid = val.txt
names = obj.names
backup = backup/

```

8. Vào đường dẫn này để tải tệp tin trọng số cho các lớp tích chập được huấn luyện từ mạng Imagenet. Sau đó sao chép tệp tin vừa tải về vào thư mục *darknet*

<https://pjreddie.com/media/files/darknet53.conv.74>

9. Nén thư mục làm việc lại thành một tệp tin *zip*, tạo một thư mục tên *Darknet* trên *Google Drive* và upload tệp tin đã nén vào thư mục này. Đồng thời tạo một thư mục con có tên *backup* trong *Darknet*.
10. Trong *Google Colab*, chọn *Runtime* → *Change runtime type* → *Hardware accelerator* → *GPU*, chạy đoạn code sau để *build Darknet*.

```

from google.colab import drive
drive.mount('/content/drive')
%cd /content
!unzip /content/drive/'My Drive'/Darknet/darknet.zip
%cd /content/darknet
!sudo apt-get install dos2unix
!make
!chmod +x ./darknet
!rm /content/darknet/backup -r
!ln -s /content/drive/'My Drive'/Darknet/backup /content/darknet
%cd /content/darknet
!find . -type f -name "*txt" -print0 | xargs -0 dos2unix

```

11. Đối với lần đầu tiên ta sẽ chạy dòng lệnh sau.

```
!./darknet detector train obj.data yolov3.cfg darknet53.conv.74
```

Dối với các lần huấn luyện sau ta chỉ cần dùng tệp tin *yolov3_last.weights* để huấn luyện tiếp mà không cần huấn luyện lại từ đầu.

```
!./darknet      detector      train      obj.data      yolov3.cfg
./backup/yolov3_last.weights
```

12. *Darknet* sẽ tự động lưu các tệp tin trọng số mỗi 1000 *iteration* ví dụ: *yolov3_1000.weights*, *yolov3_2000.weights*. Ta có thể dừng việc huấn luyện để kiểm tra các tham số hiệu năng của mô hình như *precision* và *recall*. Việc kiểm tra này được thực hiện trên tập dữ liệu validate trong tệp *val.txt*. Để thực hiện việc kiểm tra này ta sẽ chạy dòng lệnh sau.

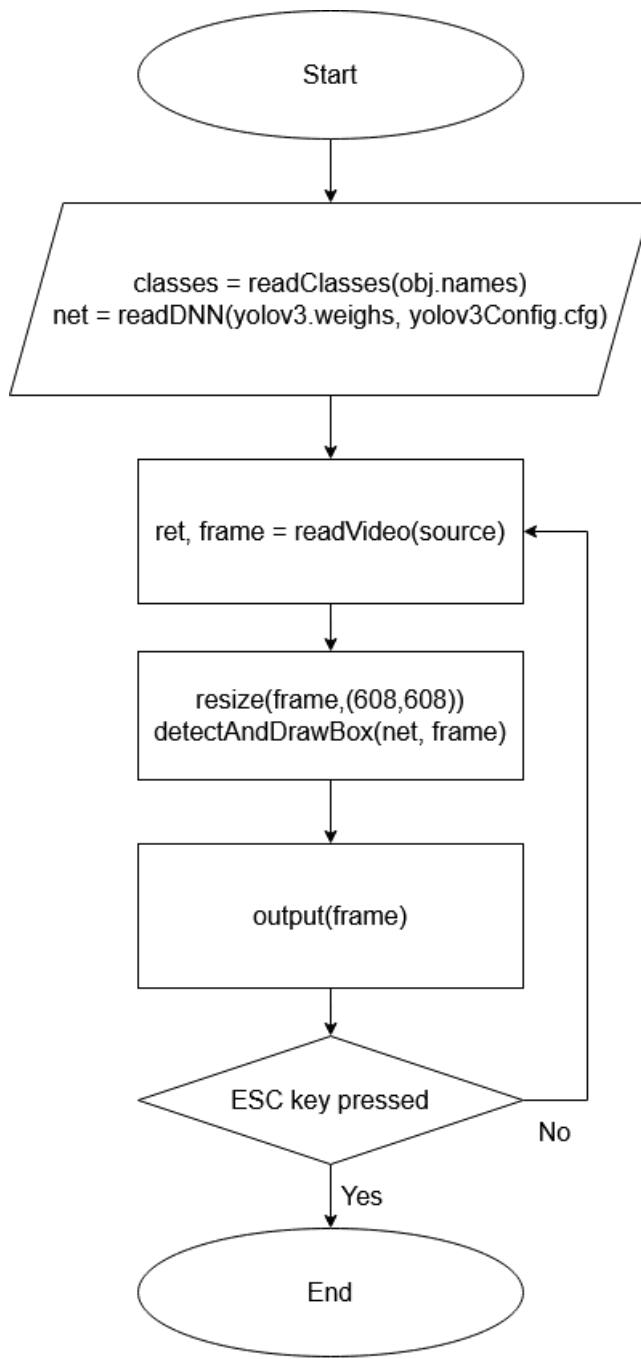
```
!./darknet      detector      map      obj.data      yolov3.cfg
./backup/yolov3_1000.weights
```

Ta có thể thay *yolov3_1000.weights* bằng tệp tin trọng số mà ta muốn kiểm tra.

3.3 Sử dụng mô hình YOLOv3 để nhận diện trên camera hoặc video

Sau khi việc huấn luyện đã hoàn thành, ta sẽ dùng tệp tin trọng số để tiến hành nhận diện trên video hoặc camera. Để làm được điều này ta sẽ viết một chương trình Python có sơ đồ khống như hình 3.5.

- Chương trình sẽ lấy tên các class được định nghĩa sẵn trong tệp tin *obj.names*, đồng thời sẽ đọc các giá trị trọng số của mô hình trong tệp tin *yolov3.weights* dựa trên các thông số về kiến trúc đã cài đặt trong tệp tin *yolov3Config.cfg*. Hàm để đọc mô hình DNN trong OpenCV là *cv2.dnn.readNet(weightsFile, configFile)* với *weightsFile* là tệp tin trọng số và *configFile* là tệp tin kiến trúc của mạng.



Hình 3.5: Sơ đồ khái quát chương trình Python để nhận dạng trên video hoặc webcam.

2. Sau đó ta sẽ dùng hàm đọc video của OpenCV để đọc dữ liệu từ webcam. Đầu tiên ta sẽ khởi tạo một đối tượng đại diện cho webcam `cap = VideoCapture(0)`. Sau đó ta sẽ đọc từng frame từ đối tượng vừa được khởi tạo `ret, image = cap.read()` với `image` là biến chứa một frame, `ret` sẽ có giá trị `True` nếu có frame được đọc từ đầu vào và `False` nếu không có frame được đọc từ đầu vào.
3. Để tiến hành nhận dạng trên hình, ta sẽ dùng các câu lệnh sau.

```

blob = cv2.dnn.blobFromImage(image, scale, (604, 604), (0, 0, 0),
    True, crop=False)
net.setInput(blob)
outs = net.forward(get_output_layers(net))

```

Dòng lệnh đầu tiên sẽ resize hình về kích thước (604, 604), sau đó từng pixel trong hình sẽ được trừ cho (0, 0, 0) và chia cho *scale*. *swapRB=True* sẽ hoán đổi vị trí hai kênh đỏ và xanh dương trong tensor của hình. *crop=False* sẽ không crop hình sau khi resize. Giả sử kênh màu đỏ của hình sau khi được resize có giá trị là *R*, *scale=σ*, giá trị để trừ là *μ_{red}*, giá trị của kênh màu đỏ sau khi đi qua hàm *blob* sẽ là

$$\frac{R - \mu_{red}}{\sigma} \quad (3.1)$$

Kết quả của hàm *blob* sẽ được đưa vào đối tượng *net* và đầu ra sẽ là kết quả của quá trình feed forward.

4. Tuy nhiên tại đầu ra *outs* sẽ có rất nhiều bounding box bị trùng lặp do với cùng một vật thể và class. Do đó kết quả đầu ra cần phải đi qua giải thuật non-maximum suppression để có thể lấy một bounding box riêng biệt và duy nhất cho một vật thể.

Về cơ bản giải thuật non-maximum suppression hoạt động như sau.

- (a) Bắt đầu giải thuật có sẽ có hai mảng một chiều A và B với mảng A chứa các bounding box cần xử lý, mảng B rỗng.
- (b) Lấy bounding box có giá trị *confidence* lớn nhất trong A và đưa vào B và loại bounding box này ra khỏi A.
- (c) Với mọi bounding box còn lại trong A, tìm IOU với bounding box vừa đưa vào B. Nếu IOU lớn hơn ngưỡng *nmsThreshold* thì loại bounding đang xét ra khỏi A.
- (d) Lặp lại bước (b) và (c) cho đến khi không còn bounding box nào trong A.

```

indices = cv2.dnn.NMSBoxes(boxes, confidences, confThreshold, nm-
sThreshold)

```

5. Cuối cùng, ta sẽ vẽ các bounding box vào frame và xuất ra màn hình.

```

for i in indices:
    i = i[0]
    box = boxes[i]
    x = box[0]
    y = box[1]
    w = box[2]
    h = box[3]
    image = draw_prediction(image, class_ids[i], confidences[i],
    round(x), round(y), round(x + w), round(y + h))

cv2.imshow("test", image)

```

Chương trình Python để sau cùng để nhận diện trên webcam.

```
from PIL import Image
import time
import cv2
import argparse
import numpy as np

def get_output_layers(net):
    layer_names = net.getLayerNames()

    output_layers = [layer_names[i[0] - 1] for i in
net.getUnconnectedOutLayers()]

    return output_layers

def draw_prediction(img, class_id, confidence, x, y, x_plus_w,
y_plus_h):
    label = str(classes[class_id])

    if class_id == 5 or class_id == 3 or class_id == 1:
        color = [0,0,255]
    else:
        color = COLORS[class_id]

    cv2.rectangle(img, (x, y), (x_plus_w, y_plus_h), color, 2)

    if class_id == 5 or class_id == 4:
        cv2.putText(img, label, (x - 10, y - 35),
cv2.FONT_HERSHEY_SIMPLEX, 1, color, 1)
    else:
        cv2.putText(img, label, (x - 10, y - 10),
cv2.FONT_HERSHEY_SIMPLEX, 1, color, 1)

    return img
```

```
def eudistance(v1, v2):
    dist = [(a - b)**2 for a, b in zip(v1, v2)]
    dist = math.sqrt(sum(dist))
    return dist

# Create a VideoCapture object
cap = cv2.VideoCapture(0)

# Check if camera opened successfully
if (cap.isOpened() == False):
    print("Unable to read camera feed")

# Default resolutions of the frame are obtained. The default
resolutions are system dependent.
# We convert the resolutions from float to integer.
frame_width = int(cap.get(3))
frame_height = int(cap.get(4))

classes = []
with open("obj.names", 'r') as f:
    classes = [line.strip() for line in f.readlines()]

COLORS=[[0,128,0],[255,0,0],[255,165,0],[0,0,255],[255,255,0],[255,69,0]]

net = cv2.dnn.readNet("yolov3-tiny_12000.weights", "yolov3-
tiny.cfg")

conf_threshold = 0.2
nms_threshold = 0.4

Width = frame_width
Height = frame_height
scale = 0.00392
```

```

start = time.time()
count = 0
while(True):
    ret, image = cap.read()

    if ret == True:
        if count
            blob = cv2.dnn.blobFromImage(image, scale, (604, 604), (0,
0, 0), True, crop=False)

            net.setInput(blob)

            outs = net.forward(get_output_layers(net))

            class_ids = []
            confidences = []
            boxes = []
            class_ids_mask = []
            confidences_mask = []
            boxes_mask = []
            class_ids_hat = []
            confidences_hat = []
            boxes_hat = []

        for out in outs:
            for detection in out:
                scores = detection[5:]
                class_id = np.argmax(scores)
                confidence = scores[class_id]
                if class_id == 0 or class_id == 1 or class_id == 4
or class_id == 5:
                    mask_id = 0
                    hat_id = 0

                    if float(scores[0]) > float(scores[1]):
                        hat_id = 0
                    else:
                        hat_id = 1

```

```

if float(scores[4]) > float(scores[5]):
    mask_id = 4
else:
    mask_id = 5

if confidence > 0.25:
    center_x = int(detection[0] * Width)
    center_y = int(detection[1] * Height)
    w = int(detection[2] * Width)
    h = int(detection[3] * Height)
    x = center_x - w / 2
    y = center_y - h / 2

    class_ids_mask.append(mask_id)
    confidences_mask.append(float(scores[hat_id]))
    boxes_mask.append([x, y, w, h])

    class_ids_hat.append(hat_id)
    confidences_hat.append(float(scores[hat_id]))
    boxes_hat.append([x, y, w, h])

else:
    if confidence > 0.25:
        center_x = int(detection[0] * Width)
        center_y = int(detection[1] * Height)
        w = int(detection[2] * Width)
        h = int(detection[3] * Height)
        x = center_x - w / 2
        y = center_y - h / 2
        class_ids.append(class_id)
        confidences.append(float(scores[class_id]))
        boxes.append([x, y, w, h])

    indices      = cv2.dnn.NMSBoxes(boxes,      confidences,
conf_threshold, nms_threshold)
    indices_mask = cv2.dnn.NMSBoxes(boxes_mask, confidences_mask,
conf_threshold, nms_threshold)

```

```
    indices_hat = cv2.dnn.NMSBoxes(boxes_hat, confidences_hat, conf_threshold, nms_threshold)

    for i in indices:
        i = i[0]
        box = boxes[i]
        x = box[0]
        y = box[1]
        w = box[2]
        h = box[3]
        image = draw_prediction(image, class_ids[i], confidences[i],
                                round(x), round(y), round(x + w), round(y + h))

    for i in indices_mask:
        i = i[0]
        if class_ids_mask[i] == 5:
            noMask = True
        else:
            noMask = False
        box = boxes_mask[i]
        x = box[0]
        y = box[1]
        w = box[2]
        h = box[3]
        image = draw_prediction(image, class_ids_mask[i], confidences_mask[i],
                                round(x), round(y), round(x + w), round(y + h))
```

```
for i in indices_hat:  
    i = i[0]  
    if class_ids_hat[i] == 1:  
        noHat = True  
    else:  
        noHat = False  
    box = boxes_hat[i]  
    x = box[0]  
    y = box[1]  
    w = box[2]  
    h = box[3]  
    image = draw_prediction(image, class_ids_hat[i], confidences_hat[i], round(x), round(y), round(x + w), round(y + h))  
  
    cv2.putText(image, "FPS: " + str(round(count/(time.time()-start))), (30, 30), cv2.FONT_HERSHEY_SIMPLEX, 1, [0,255,0], 1)  
    # Show webcam  
    cv2.imshow("test", image)  
  
    k = cv2.waitKey(1)  
    if k  
        # ESC pressed  
        print("Escape hit, closing...")  
        break  
    count+=1  
  
cap.release()  
cv2.destroyAllWindows()  
  
end = time.time()  
print("YOLO Execution time: " + str(end-start))
```

Chương 4

Phân tích kết quả

Mô hình được huấn luyện trong ba ngày với 12000 iteration. Trong quá trình huấn luyện, việc tính toán các thông số hiệu năng của mô hình được thực hiện với mỗi 1000 iteration trên tập dữ liệu validation. Nhắc lại về cách tính các thông số hiệu năng:

- Precision là thông số thể hiện độ chính xác của các dự đoán. *TruePositive* (viết tắt: TP) là những bounding box được dán nhãn đúng và thực sự đúng. *FalsePositive* (viết tắt: FP) là những bounding box được dán nhãn đúng và không thực sự đúng. Precision được tính bằng công thức

$$precision = \frac{TP}{TP + FP} \quad (4.1)$$

- Recall là thông số thể hiện độ nhạy của mô hình với các đối tượng cần nhận dạng. *TruePositive* (viết tắt: TP) là những bounding box được dán nhãn đúng và thực sự đúng. *FalseNegative* (viết tắt: FN) là những bounding box được dán nhãn không đúng hoặc không được dán nhãn và thực sự đúng. Recall được tính bằng công thức

$$precision = \frac{TP}{TP + FN} \quad (4.2)$$

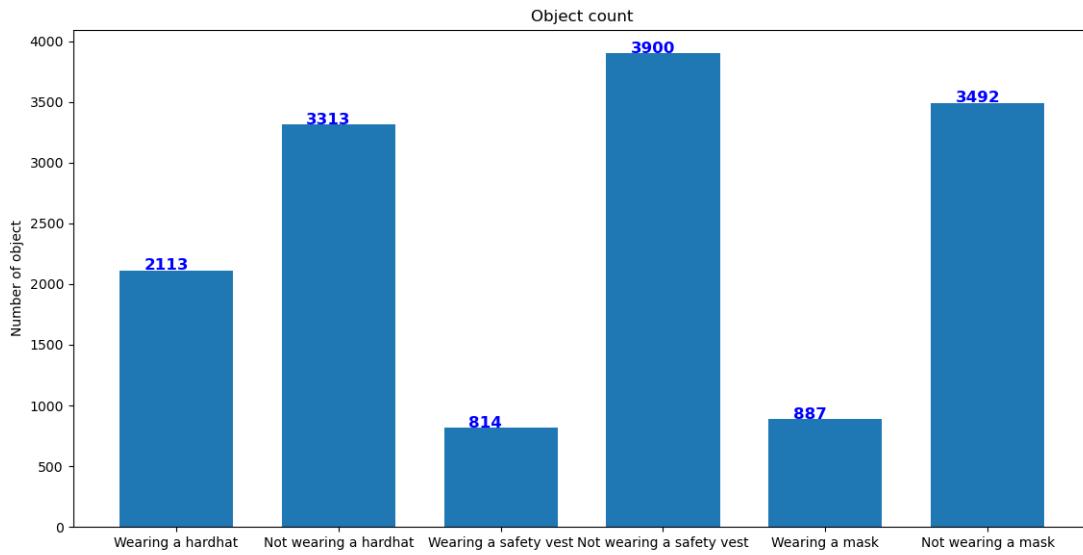
- Average precision là thông số được tính cho một class. Với mỗi class, trong quá trình đánh giá trên tập dữ liệu validation, các giá trị precision và recall sẽ được lưu lại. Sau đó ta sẽ vẽ đồ thị của precision theo recall, average precision của một class sẽ là phần diện tích dưới đồ thị này. Gọi $p(r) : [0, 1] \rightarrow [0, 1]$ là hàm số biểu diễn quan hệ của precision và recall. Average precision của một class sẽ được tính theo công thức

$$\text{average precision} = \int_0^1 p(r)dr \quad (4.3)$$

- Mean average precision là trung bình của các average precision của các class

$$\text{mean average precision} = \frac{\sum_{i=0}^N ap_i}{N} \quad (4.4)$$

Với $N \in \mathbb{N}^*$ là số class của mô hình.



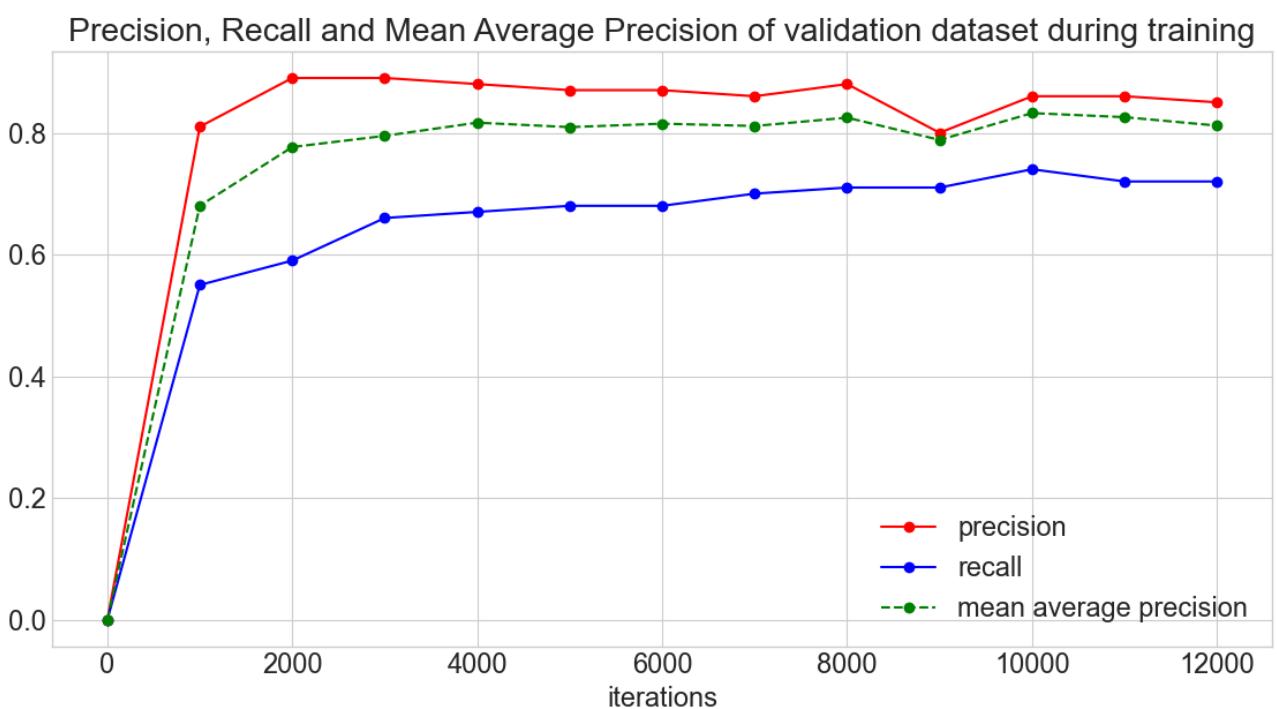
Hình 4.1: Số lượng các object trong tập dữ liệu validation. Wearing a hardhat - 2113, Not wearing a hardhat - 3313, Wearing a safety vest - 814, Not wearing a safety vest - 3900, Wearing a mask - 887, Not wearing a mask - 3492.

Tập dữ liệu validation gồm 900 hình được chia một cách ngẫu nhiên từ tập dữ liệu gốc và không được dùng để huấn luyện, số lượng các object trong tập dữ liệu này được thể hiện trong hình 4.1

Các thông số được tính toán gồm: precision, recall và mean average precision 4.6.

Một số kết quả của mô hình khi dự đoán trên hình:

Nhận xét: Mô hình có thể phát hiện rất tốt các vị trí cần nhận diện gồm phần đầu và cơ thể. Đồng thời có khả năng phân loại tương đối tốt với các vật thể được phát hiện. Tuy nhiên đối với một số class mang đặc tính *wearing* thì đôi khi vẫn bị nhầm lẫn với các class *not wearing*. Điều này có thể xuất phát từ vị trí góc nghiêng của vật thể được phát hiện không cho phép mô hình đưa ra dự đoán chính xác.



Hình 4.2: Precision - màu đỏ, Recall - màu xanh dương, Mean Average Precision - màu xanh lá. Các thông số được tính toán mỗi 1000 iteration trên tập dữ liệu validation.



Hình 4.3: Kết quả dự đoán tốt - hình trên. Phóng to một phần của kết quả dự đoán - hình dưới.



Hình 4.4: Kết quả dự đoán tốt - hình trên. Phóng to một phần của kết quả dự đoán - hình dưới.



Hình 4.5: Kết quả dự đoán không tốt - hình trên. Phóng to một phần của kết quả dự đoán - hình dưới.



Hình 4.6: Kết quả dự đoán không tốt - hình trên. Phóng to một phần của kết quả dự đoán - hình dưới.

Chương 5

Kết luận và hướng phát triển

5.1 Kết luận

Luận văn này đã xây dựng thành công hệ thống giám sát việc sử dụng thiết bị bảo hộ cá nhân trên cơ sở lý thuyết mạng neuron tích chập. Hệ thống sử dụng mô hình YOLOv3 được huấn luyện từ đầu trên tập dữ liệu tự xây dựng với tổng cộng 11586 hình với nhiều khung cảnh và thời gian khác nhau để đảm bảo mô hình không quá quen thuộc với một ngữ cảnh nhất định. Mô hình có khả năng dự đoán tương đối tốt với độ chính xác 86% và có thể được ứng dụng trong thực tế với vai trò hỗ trợ việc giám sát sử dụng thiết bị bảo hộ lao động.

Tuy nhiên mô hình vẫn còn một số khuyết điểm cần phải giải quyết:

- Cần khả năng xử lý lớn để có thể chạy trong thời gian thực trong thực tế đặc biệt là với các hệ thống có nhiều camera.
- Độ chính xác 86% là chưa đủ cao so với ngưỡng chấp nhận 95% để nhưng sản phẩm ứng trí tuệ nhân tạo có thể dùng được trong thực tế.

5.2 Hướng phát triển

Vấn đề đảm bảo an toàn lao động luôn nhận được sự chú ý của xã hội, việc cải thiện khả năng giám sát trong các khu vực cần đảm bảo việc trang bị các thiết bị bảo hộ lao động sẽ giúp nâng cao chất lượng môi trường làm việc không chỉ trong ngành xây dựng mà còn các ngành có rủi ro tai nạn cao khác như hầm mỏ, chế tạo sắt,... Để có thể thực sự đáng tin cậy để có thể dùng trong thực tế hệ thống cần phải đảm bảo độ chính xác trên 95%. Nhằm đạt được mục tiêu này một số hướng tiếp cận có thể được thực hiện:

- Tăng số lượng ảnh trong tập dữ liệu với nhiều background khác nhau để mô hình có thể học được nhiều trường hợp đặc biệt hơn.
- Dùng các kiến trúc mạng tích chập mới hơn như YOLOv4 để tăng độ chính xác.

Tài liệu tham khảo

- [1] Standford University. Convolutional neural networks (cnns / convnets), 2020.
- [2] Nguyễn Thanh Tuấn. Bài 6: Convolutional neural network, 2019.
- [3] AlexeyAB. Darknet, 2020.
- [4] D. O. Hebb. *The organization of behavior: A neuropsychological theory*. Wiley, 1949.
- [5] Arthur Lee Samuel. Some Studies in Machine Learning Using the Game of Checkers. *IBM Journal of Research and Development*, page 44:206–44:226, 1959.
- [6] David E. Rumelhart, Geoffrey E. Hinton and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, page 533–536, 1986.
- [7] Y. LeCun, L. Bottou, Y. Bengio and P. Haffner. Gradient-Based Learning Applied to Document Recognition. *Proceedings of the IEEE*, pages 86(11):2278–2324, 1998.
- [8] Suzana Herculano-Houzel. The Human Brain in Numbers: A Linearly Scaled-up Primate Brain. *Frontiers in Human Neuroscience*, pages 3:31:1–3:31:11, 2009.
- [9] Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. *LSVRC*, 2012.
- [10] John Syin. Hardhat and safety vest image for object detection, 2020.
- [11] Jixiu Wu, Nian Cai, Wenjie Chen, Huiheng Wang, Guotian Wang. Automatic detection of hardhats worn by construction personnel: A deep learning approach and benchmark dataset. *Automation in Construction*, 2019.
- [12] Tzutalin. Labelimg, 2018.
- [13] Vũ Hữu Tiệp. Bài 7: Gradient descent (phần 1/2), 2017.
- [14] Vũ Hữu Tiệp. Bài 14: Multi-layer perceptron và backpropagation, 2017.
- [15] Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi. You Only Look Once: Unified, Real-Time Object Detection. *CVPR 2016*, 2016.
- [16] Joseph Redmon, Ali Farhadi. YOLOv3: An Incremental Improvement. *Tech report*, 2018.