

ĐẠI HỌC QUỐC GIA TP.HCM
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC VÀ KỸ THUẬT MÁY TÍNH



BÁO CÁO
ĐỒ ÁN CHUYÊN NGÀNH

MÔ HÌNH ROBOT PHÂN LOẠI HÀNG HOÁ

Ngành: KỸ THUẬT MÁY TÍNH

HỘI ĐỒNG: HỘI ĐỒNG 1 - ĐỒ ÁN MÔN HỌC
KỸ THUẬT MÁY TÍNH

GVHD: TS PHẠM HOÀNG ANH

THẦY PHẠM CÔNG THÁI

TKHD: Th.S HUỲNH HOÀNG KHA

SVTH: NGUYỄN TUẤN VINH - 1915944

TP. HỒ CHÍ MINH, 05/2023



Nhận xét của GVHD 1

.....
.....
.....
.....
.....

Chữ ký GVHD 1

....., ngày ... tháng 5 năm 2024

Nhận xét của GVHD 1

.....
.....
.....
.....
.....

Chữ ký GVHD 1

....., ngày ... tháng 5 năm 2024



LỜI CAM ĐOAN

Tôi xin cam đoan rằng đề tài *Mô hình Robot phân loại sản phẩm* được tiến hành một cách minh bạch, công khai. Mọi thứ được đưa trên sự cố gắng cũng như sự nỗ lực của bản thân cùng với sự giúp đỡ không nhỏ từ Th.S Phạm Hoàng Anh và Thầy Phạm Công Thái.

Các số liệu và kết quả nghiên cứu được đưa ra trong đồ án là trung thực và không sao chép hay sử dụng kết quả của bất kỳ đề tài nghiên cứu nào tương tự. Nếu như phát hiện rằng có sự sao chép kết quả nghiên cứu đề tài khác bản thân tôi xin chịu hoàn toàn trách nhiệm.

Sinh viên

Nguyễn Tuấn Vinh



LỜI CẢM ƠN

Trong quá trình thực hiện đề tài của mình tại Trường Đại học Bách Khoa TP. Hồ Chí Minh, em rất biết ơn sự hướng dẫn và hỗ trợ của Thầy Phạm Hoàng Anh và Thầy Phạm Công Thái cùng với sự ủng hộ của nhà trường.

Thầy Phạm Hoàng Anh và Thầy Phạm Công Thái đã là nguồn động viên và định hướng quan trọng cho em trong suốt quá trình nghiên cứu và thực hiện đề tài. Sự tận tâm và kiến thức chuyên sâu của hai Thầy đã giúp em vượt qua những thách thức và tiến triển mỗi ngày trong dự án của mình. Sự hướng dẫn chi tiết và sự khích lệ không ngừng từ Thầy Phạm Hoàng Anh và Thầy Phạm Công Thái đã giúp em không chỉ phát triển kiến thức mà còn rèn luyện được kỹ năng làm việc nhóm và giải quyết vấn đề.

Bên cạnh đó, em cũng xin gửi lời biết ơn chân thành đến nhà trường đã tạo điều kiện thuận lợi để em có thể tiếp cận và tham gia vào dự án này. Sự hỗ trợ về cơ sở vật chất và môi trường học tập năng động tại trường đã làm cho hành trình nghiên cứu của em trở nên trọn vẹn và hiệu quả hơn.

Những giá trị, kỹ năng và kinh nghiệm mà em thu được từ dự án này sẽ là nền tảng quý báu cho sự phát triển và hành trình học tập sắp tới của em. Em rất biết ơn về sự hỗ trợ và sự cống hiến của Thầy 1, Thầy 2 cũng như của nhà trường trong dự án này.

Và cuối cùng em xin trân trọng gửi lời cảm ơn đến những người thân yêu.

MỤC LỤC

LỜI CAM ĐOAN	2
LỜI CẢM ƠN	3
MỤC LỤC	4
DANH MỤC BẢNG	6
DANH MỤC HÌNH ẢNH	8
TỔNG QUAN	9
1 MỞ ĐẦU	11
1.1 Giới thiệu	11
1.2 Mục tiêu của đề tài	12
1.2.1 Mục tiêu chung	12
1.2.2 Mục tiêu cụ thể	14
1.3 Ý nghĩa thực tiễn của đề tài	15
1.4 Đối tượng nghiên cứu và phạm vi giới hạn của đề tài	16
2 CƠ SỞ LÝ THUYẾT	17
2.1 Deep Learning	17
2.1.1 Giới thiệu chung	17
2.1.2 Artificial Neural Networks (ANNs)	18
2.2 Convolutional Neural Networks (CNNs)	28
2.2.1 Giới thiệu chung	28
2.2.2 Các khái niệm cơ bản	29
2.2.3 Convolutional Layers[15]	35
2.2.4 Pooling Layers[15]	38
2.2.5 Fully Connected Layers	39
2.2.6 Các kiến trúc CNN cho bài toán phân loại đối tượng	40
2.3 Computer Vision	43
2.3.1 Bài toán object detection	43
2.3.2 R-CNN (Region with CNN feature)	44
2.3.3 Fast R-CNN	47
2.3.4 Faster R-CNN	49
2.3.5 YOLO Series	53



3 ĐỀ XUẤT GIẢI PHÁP	57
3.1 Nhắc lại nhiệm vụ của đề tài	57
3.2 Các giải pháp	57
3.2.1 Giải pháp phát hiện và phân loại sản phẩm	57
3.2.2 Giải pháp tìm vị trí các khu vực và đường đi	58
4 KẾ HOẠCH CHO GIA ĐOẠN 2	61
TÀI LIỆU THAM KHẢO	66

DANH MỤC BẢNG

1.1	Miêu các thành phần trong test case	13
2.1	Một số Kernel thông dụng[16]	35
4.1	Các nhiệm vụ cho phần Chuẩn bị khu vực test	61
4.2	Các nhiệm vụ cho phần Hiện thực phần cứng	63
4.3	Các nhiệm vụ cho phần Hiện thực phần mềm	64
4.4	Các nhiệm vụ cho phần Chuẩn bị khu vực test	65

DANH MỤC HÌNH ẢNH

1.1	Mô hình của test case	12
2.1	Cấu trúc mạng neural	17
2.2	AI, Machine Learning và Deep Learning	18
2.3	Cấu trúc của một Perceptron[14]	18
2.4	Dồ thị hàm Step	19
2.5	Dồ thị hàm ReLU	19
2.6	Dồ thị hàm leaky ReLU với $m = 0.01$	20
2.7	Dồ thị hàm Sigmoid	20
2.8	Dồ thị hàm Tanh	21
2.9	Cấu trúc mạng neural	22
2.10	Fully Connected	23
2.11	Pooling	23
2.12	Một node tại tầng hiện tại	25
2.13	Một node ở tầng trước đó	25
2.14	Mô phỏng cách tính lan truyền ngược. Tầng cuối có thể là tầng đầu ra.	27
2.15	Hệ màu RGB	29
2.16	Ảnh màu biểu diễn dưới dạng tensor 3 chiều[4]	31
2.17	Tính $X \otimes W = Y$	32
2.18	Padding	32
2.19	Thực hiện Flatten trên ma trận	33
2.20	Phép tính convolution trên ảnh màu với $k=3$.	36
2.21	Tensor X, W 3 chiều được viết dưới dạng 3 matrix	36
2.22	Thực hiện phép tính convolution trên ảnh màu	37
2.23	Convolution đầu tiên	37
2.24	Convolution tổng quát	38
2.25	Sau pooling layer (2×2)[3]	39
2.26	Kiến trúc Lenet-5	40
2.27	Kiến trúc AlexNet	41
2.28	Kiến trúc VGG 16	41
2.29	(a) Ánh xạ bên trong Khối Residual, (b) Ánh xạ trực tiếp	42
2.30	Mô hình dựa trên DenseNet với 3 khối dense.	43
2.31	Ví dụ cho object detection	43
2.32	Ảnh đầu vào	44
2.33	Ảnh đầu ra	44
2.34	Ảnh bị phân đoạn quá mức (oversegment)	45
2.35	Các bước của quá trình phân đoạn theo thứ bậc (hierarchical segmentation)[10]	45



2.36 Các bước trong R-CNN[13]	47
2.37 Các bước trong Fast R-CNN	48
2.38 Hình minh họa cách chia của ROI	48
2.39 Thực hiện ROI pooling	49
2.40 So sánh thời train train và test giữa R-CNN và Fast R-CNN	49
2.41 Kiến trúc mới Faster R-CNN	50
2.42 Mô tả các region proposal trong output của RPN	50
2.43 Ví dụ về anchor	51
2.44 Ví dụ về IoU	51
2.45 Chỉ số IoU	52
2.46 So sánh tốc độ test-time của các thuật toán object detection	52
2.47 Kiến trúc mạng YOLO	53
2.48 Minh họa cách YOLO hoạt động	53
2.49 Ví dụ về boundary box	55
3.1 Đặt tên các tuyến đường giữa các khu vực	58
3.2 Mô hình phân loại nhiều tầng	59



TỔNG QUAN

Phân loại hàng hoá và những lợi ích từ việc phân loại hàng hoá hiệu quả.

Trong quá trình Sản xuất hàng hoá và Phân phối hàng hoá, phân loại sản phẩm là một công việc quan trọng trong công đoạn xử lý. Đối với một doanh nghiệp, việc phân loại sản phẩm nhanh chóng và hiệu quả giúp:

- Nâng cao hiệu quả hoạt động:
 - Tăng tốc độ xử lý sản phẩm, xử lý nguyên liệu, giảm thời gian lưu kho, giúp doanh nghiệp tiết kiệm chi phí và nâng cao năng suất. Thông thường, một doanh nghiệp sẽ phải đầu tư 3 – 10% cho chi phí kho hàng, 3 – 5% cho chi phí về nhân lực cho các hoạt động giám sát và quản lý kho, 6 – 24% cho chi phí đầu tư vào hàng tồn kho và 2 – 5% cho chi phí về các thiết bị hàng hóa[12].
 - Giảm thiểu sai sót trong quá trình phân loại, đảm bảo chất lượng sản phẩm và hạn chế rủi ro cho doanh nghiệp.
 - Giảm thiểu tổn thất do hư hỏng, lỗi thời hoặc thất lạc.
 - Cải thiện khả năng truy xuất nguồn gốc sản phẩm, giúp doanh nghiệp quản lý hàng hóa hiệu quả hơn.

- Tăng cường khả năng cạnh tranh:
 - Giúp doanh nghiệp đáp ứng nhu cầu thị trường một cách nhanh chóng và hiệu quả.
 - Nâng cao uy tín và thương hiệu của doanh nghiệp, thu hút khách hàng tiềm năng.
 - Đáp ứng đơn hàng của khách hàng nhanh chóng và chính xác hơn. Từ đó, doanh nghiệp có thể cung cấp dịch vụ khách hàng tốt hơn bằng cách giải quyết các vấn đề về đơn hàng và giao hàng một cách nhanh chóng.
 - Tạo lợi thế cạnh tranh so với các đối thủ trong cùng ngành

- Thúc đẩy phát triển kinh tế:
 - Góp phần vào sự phát triển của ngành công nghiệp logistics và chuỗi cung ứng.
 - Tạo ra nhiều việc làm, thúc đẩy sự phát triển của kinh tế địa phương.
 - Nâng cao năng lực cạnh tranh của quốc gia trên thị trường quốc tế.

Bên cạnh đó, công việc phân loại sản phẩm là một công việc nhảm chán, lặp đi lặp lại và đòi hỏi tính chính xác, cẩn thận và tỉ mỉ cao. Để giảm thiểu chi phí đầu tư cho việc phân loại hàng hoá, thoả mãn các yêu cầu và tính chất công việc kể trên nhưng vẫn đáp ứng được tốc độ yêu cầu trong các quy trình sản xuất tự động thì việc ra đời của các robot phân loại sản phẩm là điều tất yếu trong thời đại công nghiệp hoá - hiện đại hoá hiện nay.

Tổng quan về thị trường Robot phân loại sản phẩm hiện nay

- Mức độ phát triển

Ngành công nghiệp robot phân loại sản phẩm ra đời vào đầu những năm 1980, với tốc độ phát triển nhanh chóng trong những thập kỷ qua. Tuy nhiên, ngành công nghiệp robot phân loại sản phẩm đã chứng kiến sự phát triển đáng kể trong thập kỷ gần đây, đặc biệt từ năm 2010 trở đi. Từ việc sử dụng công nghệ máy học đến các tiến bộ trong robot hình thái và cảm biến, ngành này đã phát triển với tốc độ nhanh chóng. Theo số liệu từ The International Federation of Robotics (IFR) trong Báo cáo thế giới về robot công nghiệp năm 2020 (IFR presents World Robotics Report 2020), Báo cáo cho thấy kỷ lục



2,7 triệu robot công nghiệp đang hoạt động trong các nhà máy trên khắp thế giới - tăng 12%[7]. Doanh số bán robot công nghiệp mới vẫn ở mức cao với 373.000 chiếc được xuất xưởng trên toàn cầu vào năm 2019. Con số này ít hơn 12% so với năm 2018, nhưng vẫn là doanh số bán hàng cao thứ 3 từng được ghi nhận[12]. Trong thời gian này, thị trường robot phân loại sản phẩm đã trở thành một phần không thể tách rời trong chuỗi cung ứng sản phẩm toàn cầu. Theo Mordor Intelligence, thị trường robot phân loại sản phẩm toàn cầu dự kiến đạt 114,67 tỷ USD vào năm 2023 và tăng trưởng với tốc độ CAGR 17,64% từ 2023 đến 2028[9].

- Các yếu tố ảnh hưởng

Các yếu tố thúc đẩy sự phát triển của ngành bao gồm sự tăng cao trong nhu cầu tự động hóa cao trong các ngành công nghiệp, đặc biệt là logistics, thực phẩm và đồ uống, dược phẩm, điện tử đi kèm với nhu cầu nâng cao hiệu quả, năng suất, giảm chi phí sản xuất đảm bảo an toàn lao động và giảm thiểu sai sót do con người. Bên cạnh đó còn có sự phát triển của các công nghệ như trí tuệ nhân tạo, thị giác máy tính, học máy. Tuy nhiên, ngành sản xuất và phát triển Robot phân loại sản phẩm còn gặp nhiều hạn chế như chi phí đầu tư ban đầu cao, nhu cầu về nhân lực có trình độ kỹ thuật cao để vận hành - bảo trì và khả năng ứng dụng hạn chế cho một số sản phẩm đặc biệt

- Đặc điểm robot phân loại sản phẩm hiện nay

- Cấu tạo: Các robot phân loại sản phẩm hiện nay thường được thiết kế để có khả năng xử lý sản phẩm ở nhiều kích thước, hình dạng và trọng lượng khác nhau. Chúng thường được trang bị cánh tay robot, hệ thống camera, các bộ cảm biến, hệ thống điều khiển, hệ thống băng tải.
- Nguyên lý hoạt động: Sử dụng camera để nhận diện sản phẩm, sau đó sử dụng cánh tay robot để gấp và phân loại sản phẩm theo các tiêu chí định trước.
- Số bậc tự do: Các robot phân loại sản phẩm hiện nay có từ 4 đến 6 bậc tự do, cho phép thực hiện các thao tác phức tạp như gấp, xoay, di chuyển sản phẩm.
- Hạn chế: Khả năng nhận diện sản phẩm phức tạp và khả năng thích ứng với môi trường thay đổi còn hạn chế.
- Công nghệ áp dụng: Trí tuệ nhân tạo, thị giác máy tính, học máy, robot cộng tác.

- Nhà cung cấp robot phân loại sản phẩm

Trong nước Việt Nam, một số công ty hàng đầu trong lĩnh vực này bao gồm Công ty TNHH Công Nghệ Robots Việt Nam và Công ty TNHH Sản Xuất và Thương Mại Robot Vina. Ngoài ra, các nhà sản xuất robot hàng đầu trên thế giới như Fanuc, ABB và Yaskawa cũng có mặt trên thị trường Việt Nam thông qua các đại lý và đối tác địa phương.

Chương 1

MỞ ĐẦU

1.1 Giới thiệu

Trong bối cảnh của ngành công nghiệp sản xuất hiện đại, việc tự động hóa quy trình phân loại sản phẩm trở nên ngày càng quan trọng. Dự án nhằm phát triển một mô hình robot có khả năng phân loại sản phẩm một cách tự động để cải thiện hiệu suất và giảm chi phí cho các nhà sản xuất.

Robot được thiết kế để tự động phát hiện và phân loại các loại sản phẩm khác nhau bằng cách sử dụng công nghệ thị giác máy tính (computer vision). Điều này cho phép robot nhận biết và phân loại sản phẩm dựa trên các đặc điểm hình dáng và màu sắc của chúng. Để thực hiện các nhiệm vụ này, robot sử dụng các vi điều khiển thuộc dòng ESP32 series, đảm bảo tính linh hoạt và hiệu suất trong việc xử lý dữ liệu và điều khiển các hành động của robot.

Không chỉ có khả năng phân loại sản phẩm, robot cũng được trang bị khả năng tự hành. Điều này cho phép robot tự điều hướng và thực hiện các nhiệm vụ phân loại sản phẩm một cách hiệu quả và đáng tin cậy. Bằng cách này, robot có thể hoạt động một cách độc lập và hiệu quả mà không cần sự can thiệp của con người.

Bên cạnh đó, với khả năng vận chuyển hàng hóa trên khoảng cách xa, robot trở thành một giải pháp lý tưởng cho các nhà kho hoặc nhà máy lớn. Sự kết hợp giữa khả năng phân loại sản phẩm và khả năng tự hành của robot giúp nâng cao hiệu suất và tăng cường tính linh hoạt trong quy trình vận chuyển hàng hóa, đặc biệt là trong môi trường công nghiệp đòi hỏi sự chính xác và hiệu quả.

Robot phân loại sản phẩm mà em phát triển có những điểm khác biệt so với các loại robot phân loại phổ biến hiện nay. Thay vì sử dụng cánh tay robot, em đã áp dụng một đầu kẹp để cầm nắm hàng hóa, tạo điểm nhấn độc đáo và giảm độ phức tạp trong quy trình thiết kế. Đồng thời, robot của em không được thiết lập cố định vị trí mà có thể di chuyển trên những tuyến đường được vẽ sẵn, giúp tối ưu hóa quy trình làm việc và tăng khả năng sử dụng trong các môi trường công nghiệp đa dạng.

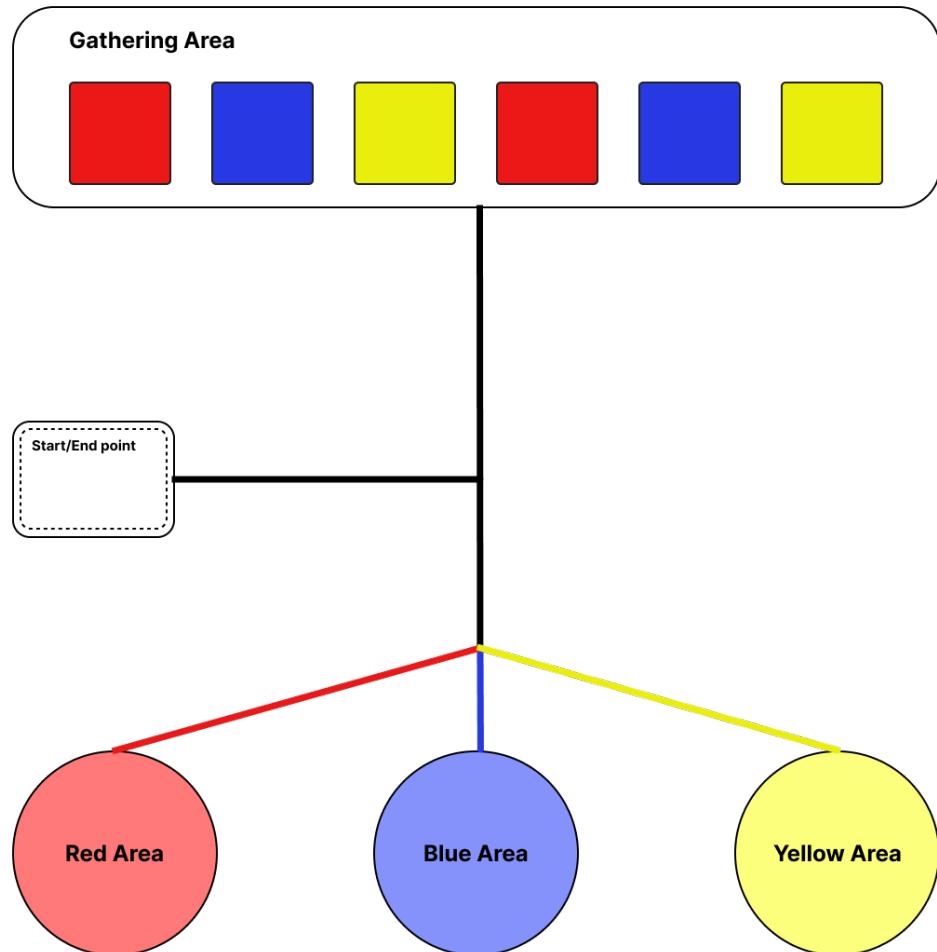
Robot có một số ưu điểm nổi bật như có khả năng phân loại được nhiều loại hàng hóa nếu được training đầy đủ, đồng thời việc thiết lập và cài đặt robot cũng rất dễ dàng. Chỉ cần training cho robot các loại hàng hóa cần phân loại, các vị trí của các khu vực phân loại và vẽ đường đi cho robot là có thể sử dụng được. Hơn nữa, khả năng vận chuyển ở khoảng cách xa của robot là một điểm mạnh lớn, làm cho nó trở thành lựa chọn phù hợp cho các nhà kho hoặc nhà máy lớn.

Tuy nhiên,劣势 điểm của robot cũng không thể bỏ qua. Độ chính xác không cao, dễ phân loại sai hoặc di chuyển nhầm đường là một trong những thách thức lớn đối với robot của em. Vấn đề về năng lượng cũng là một hạn chế đáng lưu ý, khi robot sử dụng pin để hoạt động và không thể hoạt động liên tục. Cuối cùng, cần có thời gian training lại khi có hàng hóa mới, tạo ra sự bất tiện cho quy trình sản xuất và vận hành của nhà máy hay nhà kho.

1.2 Mục tiêu của đề tài

1.2.1 Mục tiêu chung

Mục tiêu chung của đề tài là nghiên cứu, thiết kế và hiện thực một Robot phân loại sản phẩm và vận chuyển sản phẩm đến khu vực được chỉ định thỏa mãn yêu cầu của Test case sau:



Hình 1.1: Mô hình của test case

Trong đó:

Hình ảnh	Tên gọi	Mô tả
	Khối vuông (Cube)	Những khối vuông tượng trưng cho sản phẩm cần được phân loại. Số chỉ có 3 loại khối vuông chính là xanh, đỏ và vàng.
	Khu vực phân loại (Classified area)	Đây là các khu vực mà Robot cần phải đưa các khối vuông phía trên vào. Các khu vực chứa các sản phẩm đã được phân loại. Trong test case này, có 3 khu vực phân loại lần lượt là Blue area, Red area và Yellow area.
	Khu vực tập kết (Gathering area)	Đây là khu vực tập kết các khối vuông. Các khối vuông được đặt trong khu vực này để sẵn sàng cho việc phân loại.
	Điểm bắt đầu/ kết thúc (Start/End point)	Đây là nơi robot bắt đầu cũng như kết thúc chu trình làm việc của mình. Khu vực này có thể được tích hợp cổng sạc để sạc cho Robot khi Robot gần hết năng lượng.
	Đường kẻ (Line)	Những đường kẻ được vẽ sẵn để kết nối các khu vực với nhau. Robot dựa vào những đường vẽ này để xác định vị trí của các khu vực.

Bảng 1.1: Miêu các thành phần trong test case

Yêu cầu:

Ban đầu, các khối vuông được đặt trong *Gathering area*. Từ **Start point**, Robot sẽ xuất phát và tiến đến *Gathering area*. Sau đó, Robot tiến hành nhận diện và phân loại các khối vuông rồi vận chuyển các khối vuông đến các khu vực phân loại (*Classified Area*) tương ứng với màu sắc của từng khối vuông. Khi đã phân loại hết tất cả các khối vuông trong *Gathering area*, Robot phải quay lại điểm *End point*.

Thông tin đầu vào:

- Các sản phẩm được đặt trong khu vực *Gathering*. Các sản phẩm được tượng trưng bằng các khối vuông



khác màu nhau.

2. Ba khu vực dành riêng cho các loại sản phẩm khác nhau. Các khu vực này là các khu vực để chứa các khối vuông theo màu sắc.
3. Các đường kẻ được vẽ sẵn để Robot có thể lựa chọn tuyến đường đến các khu vực.
4. Khu vực khởi đầu và kết thúc. Nơi đây sẽ là nơi bắt đầu cũng như kết thúc chu trình làm việc của Robot. Trong thực tế thì đây có thể là trạm sạc pin cho Robot.

Kết quả mong muôn:

1. Các khối vuông trong khu vực Gathering được vận chuyển và đặt tại các khu vực tương ứng với màu sắc của từng khối vuông.
2. Robot quay trở lại khu vực xuất phát.

Các giới hạn:

$$0 \leq \text{Số lượng khối vuông trong Gathering area} \leq 6 \quad (1.1)$$

$$\text{Chỉ có 3 loại khối vuông là Đỏ, Xanh và Vàng} \quad (1.2)$$

Nhiệm vụ của robot:

1. Tự động phát hiện được các sản phẩm,
2. Phân loại được loại sản phẩm,
3. Vận chuyển sản phẩm đó đến khu vực phân loại tương ứng,
4. Lặp lại các bước trên cho đến khi hết tất cả sản phẩm. Sau đó quay lại điểm bắt đầu.

1.2.2 Mục tiêu cụ thể

a) Về phần cứng

- Thiết kế và chế tạo khung cho robot chắc chắn, chịu lực tốt và di chuyển linh hoạt trên địa hình phẳng.
- Tính toán và xác định vị trí đặt camera trên robot sao cho có tầm nhìn bao quát có thể nhìn thấu các vật thể và cánh tay của Robot.
- Lựa chọn và tích hợp các cảm biến, động cơ và linh kiện phù hợp cho các chức năng đã được nêu.
- Thiết kế và chế tạo cánh tay robot có độ chính xác cao. Xác định chiều dài hoạt động của cánh tay, độ rộng khi mở và đóng của bàn tay sao cho phù hợp với hầu hết các kích thước của sản phẩm.

b) Về phần mềm

- Phát triển thuật toán thị giác máy tính:
 - Xác định được sản phẩm.
 - Phân loại sản phẩm theo các tiêu chí đã được định nghĩa.
 - Vẽ được khung bao quanh sản phẩm và xác định được vị trí tương đối của vật thể trên khung hình của camera.
- Lập trình phần mềm điều khiển robot:



- Điều khiển di chuyển robot trên địa hình, phát hiện đường kẻ và di chuyển theo nó.
- Điều khiển di chuyển camera.
- Điều khiển hoạt động của cánh tay robot.
- Giao tiếp giữa ESP32 WROOM 32 và ESP32 CAM.

c) Thử nghiệm và đánh giá

- Chuẩn bị bộ dataset dành cho việc training mô hình AI theo các tiêu chí.
- Xây dựng môi trường thử nghiệm.
- Dánh giá độ chính xác của mô hình AI với các sản phẩm khác nhau.
- Chạy thử và đánh giá robot trên môi trường thử nghiệm.
- Đề xuất các giải pháp khắc phục các trường hợp lỗi.

1.3 Ý nghĩa thực tiễn của đề tài

a) Nâng cao hiệu quả và năng suất tự động hóa

Robot phân loại hàng hóa có thể làm việc liên tục 24/7 mà không cần nghỉ ngơi, thay thế sức lao động thủ công, giảm thiểu sai sót và tai nạn lao động.

Robot có thể phân loại sản phẩm với tốc độ và độ chính xác cao hơn con người, giúp tăng năng suất và hiệu quả hoạt động trong các kho hàng, nhà máy, trung tâm thương mại,...

b) Giảm chi phí hoạt động

Sử dụng robot phân loại hàng hóa giúp tiết kiệm chi phí nhân công, chi phí bảo trì và chi phí sửa chữa so với hệ thống phân loại thủ công.

Robot có thể hoạt động trong môi trường khắc nghiệt, nguy hiểm, giảm thiểu chi phí bảo hộ lao động và chi phí y tế.

c) Tăng cường khả năng cạnh tranh

Doanh nghiệp áp dụng robot phân loại hàng hóa sẽ có lợi thế cạnh tranh về hiệu quả, năng suất và chi phí so với các doanh nghiệp sử dụng phương pháp phân loại thủ công.

Robot có thể giúp doanh nghiệp đáp ứng nhu cầu thị trường ngày càng cao về tốc độ và độ chính xác trong việc phân loại hàng hóa.

d) Mở ra tiềm năng ứng dụng rộng rãi

Robot phân loại hàng hóa có thể được ứng dụng trong nhiều lĩnh vực khác nhau như logistics, sản xuất, thương mại điện tử, nông nghiệp,...

Với sự phát triển của công nghệ trí tuệ nhân tạo và thị giác máy tính, robot phân loại hàng hóa sẽ ngày càng thông minh và linh hoạt hơn, mở ra nhiều tiềm năng ứng dụng mới trong tương lai.

Tóm lại, việc phát triển robot phân loại hàng hóa là một đề tài có ý nghĩa thực tiễn quan trọng, góp phần nâng cao hiệu quả, năng suất, giảm chi phí và tăng cường khả năng cạnh tranh cho doanh nghiệp.

Với tiềm năng ứng dụng rộng rãi, robot phân loại hàng hóa hứa hẹn sẽ đóng vai trò quan trọng trong việc tối ưu hóa hoạt động logistics và sản xuất trong tương lai.



1.4 Đối tượng nghiên cứu và phạm vi giới hạn của đề tài

Phần nội dung của đề tài tập trung vào nghiên cứu, thiết kế và hiện thực Robot có khả năng giải quyết được bài toán đã đề ra trong Test case của Mục tiêu đề tài. Cụ thể, phạm vi nghiên cứu bao gồm 5 đối tượng chính sau:

a) Khung Robot:

Nghiên cứu và thiết kế một khung cơ bản cho Robot, bao gồm cấu trúc cơ học và kết nối các thành phần điện tử.

b) Esp32 wroom 32 và esp32 cam:

Sử dụng vi điều khiển ESP32 WROOM 32 và ESP32 CAM để điều khiển và giao tiếp với các thiết bị và cảm biến khác nhau trên Robot.

c) Computer vision:

Nghiên cứu và triển khai các thuật toán phát hiện, nhận diện và phân loại vật thể sử dụng công nghệ computer vision, nhằm hỗ trợ quá trình phân loại sản phẩm của Robot.

d) Robot dò line:

Thiết kế và hiện thực chức năng dò line cho Robot, giúp nó có khả năng di chuyển trên các tuyến đường được vẽ sẵn và đến được đích một cách chính xác.

e) Thiết kế và nguyên lý hoạt động của cánh tay robot:

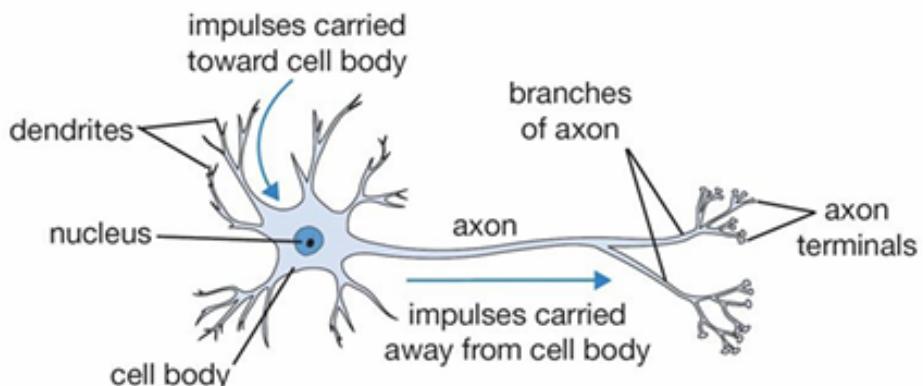
Nghiên cứu và thiết kế cánh tay robot, bao gồm cơ chế hoạt động, khả năng cầm nắm và di chuyển sản phẩm, đảm bảo tính linh hoạt và chính xác trong quá trình phân loại.

Chương 2

CƠ SỞ LÝ THUYẾT

2.1 Deep Learning

2.1.1 Giới thiệu chung



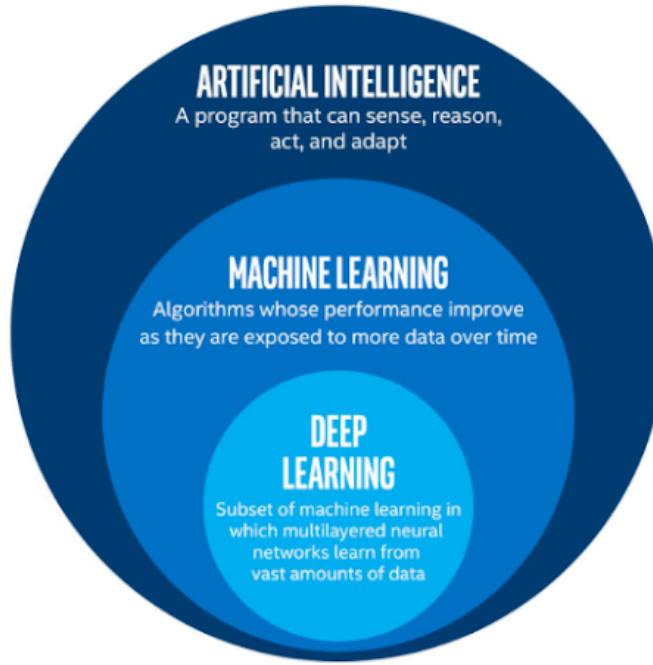
Hình 2.1: Cấu trúc mạng neural

Học sâu (Deep Learning) là một lĩnh vực con của Trí tuệ Nhân tạo (AI) lấy cảm hứng từ cấu trúc và hoạt động của não bộ con người.

Nó sử dụng các mạng nơ-ron nhân tạo (ANN) được xây dựng từ các "nơ-ron nhân tạo" đơn giản, mô phỏng hoạt động của các tế bào thần kinh.

Mỗi nơ-ron nhận nhiều tín hiệu đầu vào, xử lý chúng thông qua một hàm kích hoạt, và tạo ra một tín hiệu đầu ra. Các nơ-ron được sắp xếp thành nhiều lớp, với mỗi lớp thực hiện một chức năng cụ thể.

So với các phương pháp học máy truyền thống, Deep Learning có khả năng học hỏi từ dữ liệu phức tạp và phi tuyến tính tốt hơn, mang lại hiệu quả cao trong nhiều lĩnh vực như xử lý ảnh, nhận dạng giọng nói, dịch tự động và nhiều hơn nữa. Tuy nhiên, Deep Learning cũng đòi hỏi lượng dữ liệu lớn và tài nguyên tính toán mạnh mẽ để huấn luyện mô hình hiệu quả.



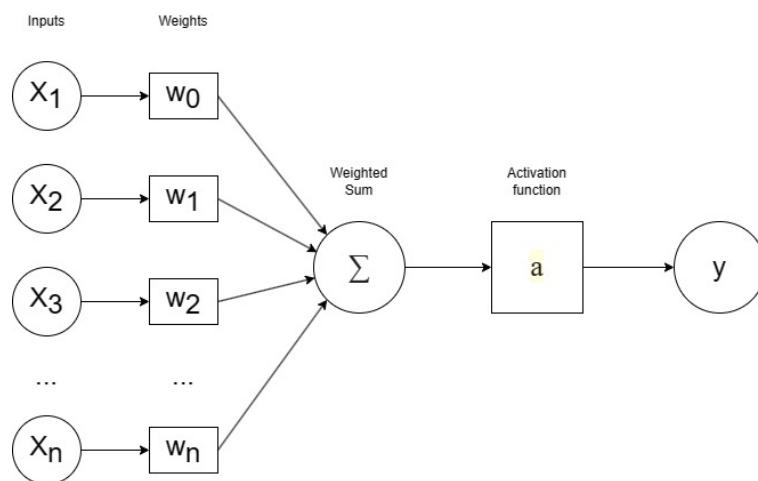
Hình 2.2: AI, Machine Learning và Deep Learning

2.1.2 Artificial Neural Networks (ANNs)

Perceptron

Perceptron là thành phần cơ bản nhất cấu thành nên một mạng neural. Chúng đơn giản là các hàm toán học nhận đầu vào từ một hoặc nhiều số, thực hiện các phép toán và trả về kết quả đầu ra.

Có thể hình dung hoạt động của perceptron trong hình vẽ sau:



Hình 2.3: Cấu trúc của một Perceptron[14]

Trong đó:

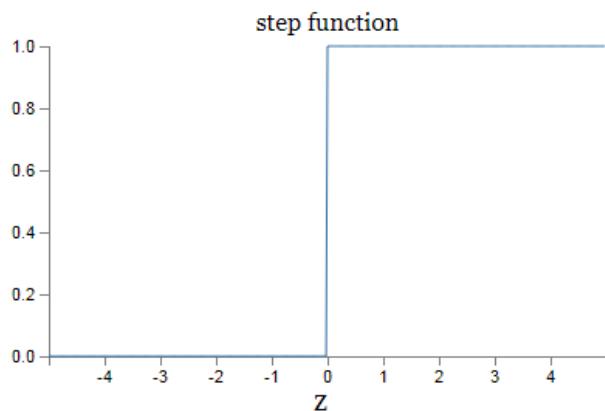
- Inputs $X = [x_0, x_1, x_2, \dots, x_n]$: Vector chứa các giá trị thực đầu vào.
- Weights $W = [w_0, w_1, w_2, \dots, w_n]$: Vector chứa các trọng số thực.
- Weight sum $z = \sum_{i=1}^n w_i * x_i$: Tổng các tích giữa trọng số và input.

- Activation function a: Hàm này áp dụng phép biến đổi phi tuyến tính cho tổng có trọng số và độ lệch (bias b), tạo ra kết quả đầu ra.
- Output $y = a(z + b)$: Kết quả của quá trình

Các hàm kích hoạt (Activation function) phổ biến:

a) **Hàm Step** [5]

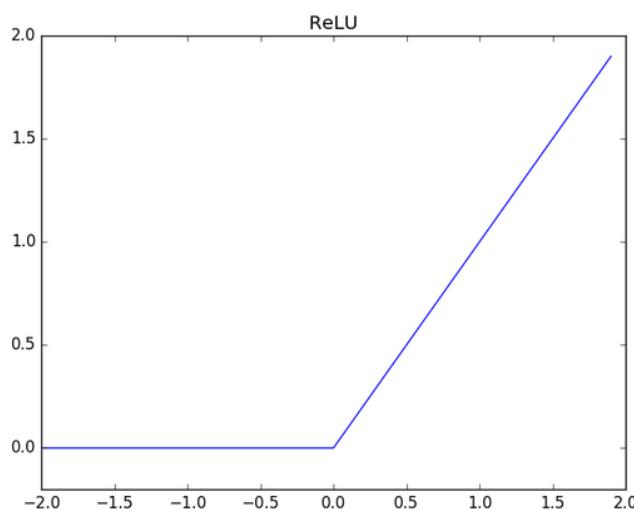
$$a(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases} \quad (2.1)$$



Hình 2.4: Đồ thị hàm Step

b) **Hàm ReLU (Rectified Linear Unit)** [1]

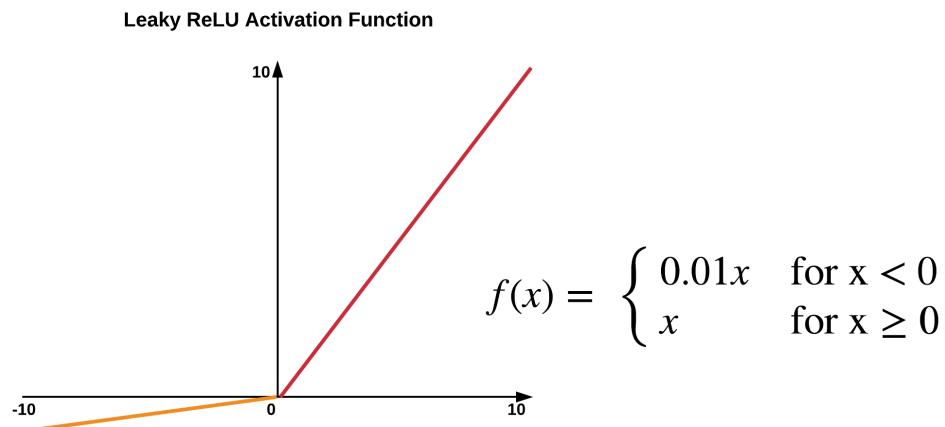
$$a(x) = \begin{cases} x & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases} \quad (2.2)$$



Hình 2.5: Đồ thị hàm ReLU

c) Hàm Leaky ReLU[1]

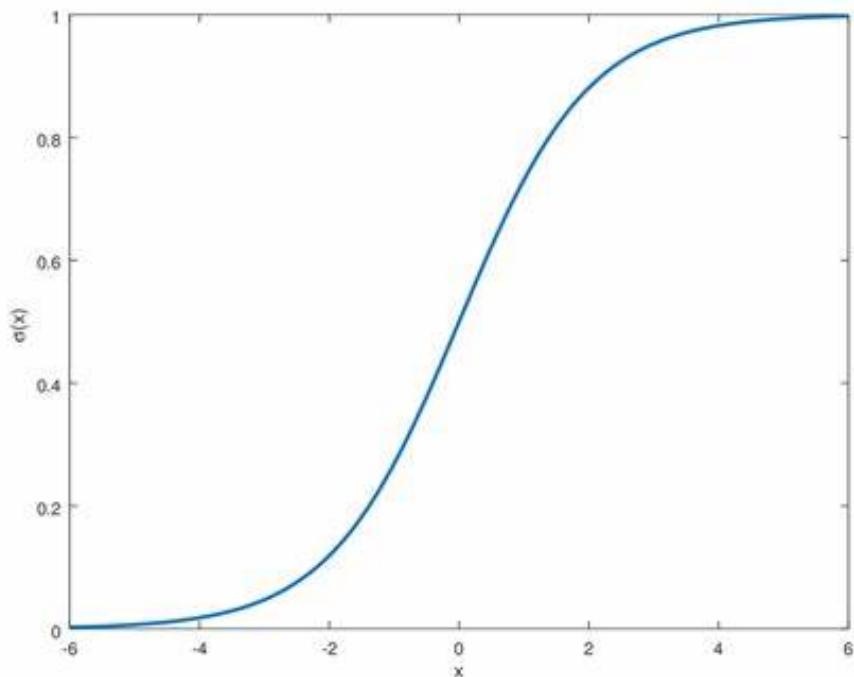
$$a(x) = \begin{cases} x & \text{if } x \geq 0 \\ mx & \text{if } x < 0 \end{cases} \quad (2.3)$$



Hình 2.6: Đồ thị hàm leaky ReLU với $m = 0.01$

d) Hàm Sigmoid[1]

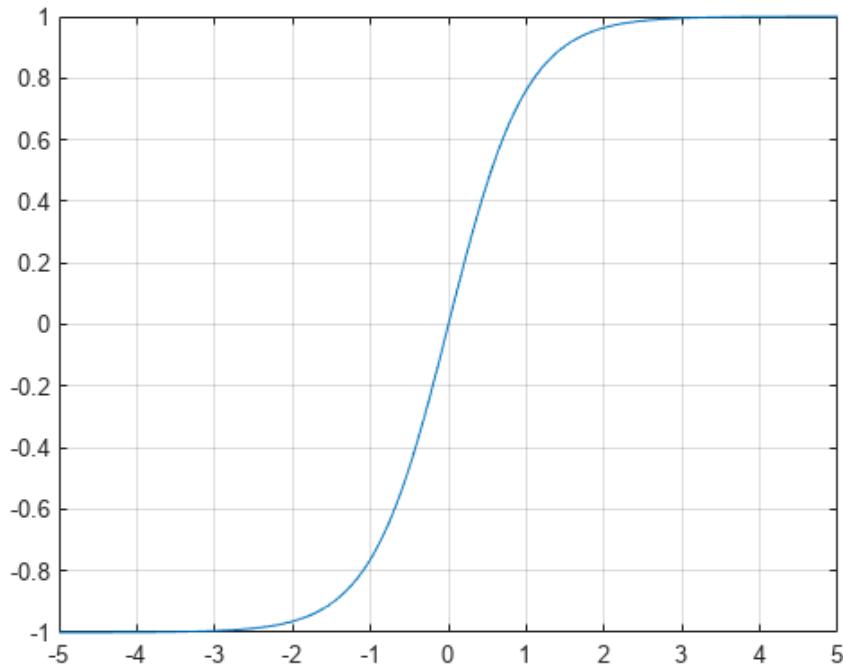
$$a(x) = \frac{1}{1 + e^{-x}} \quad (2.4)$$



Hình 2.7: Đồ thị hàm Sigmoid

e) **Hàm Tanh[1]**

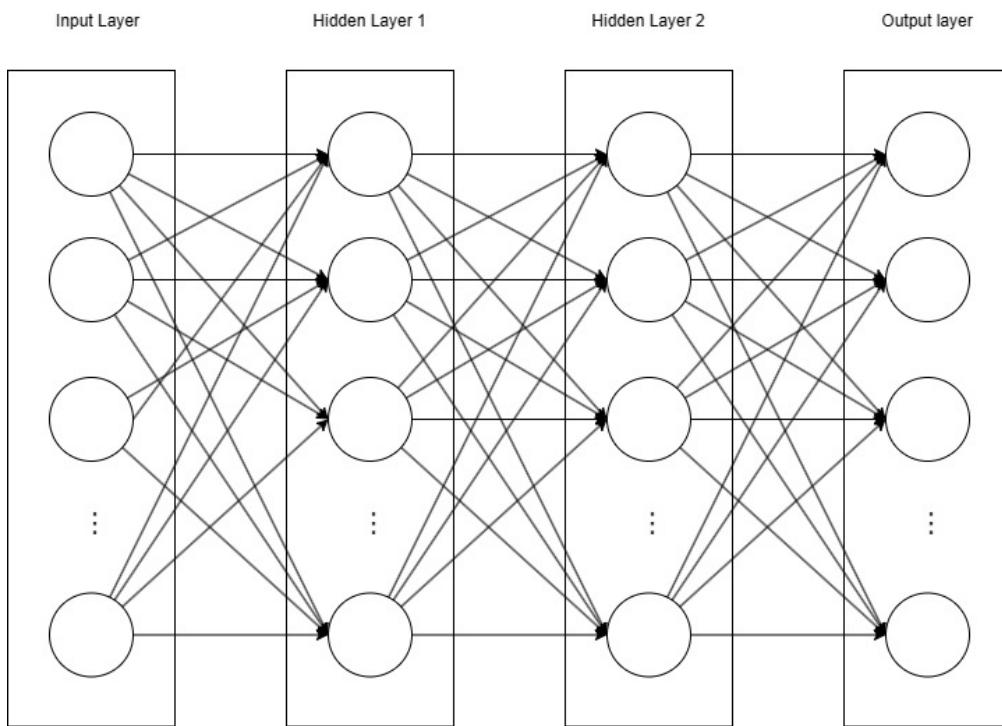
$$a(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \quad (2.5)$$



Hình 2.8: Đồ thị hàm Tanh

Cấu trúc mạng neural

Mạng neural là sự kết hợp của các tầng perceptron hay còn được gọi là perceptron đa tầng (multilayer perceptron) như hình vẽ bên dưới:



Hình 2.9: Cấu trúc mạng neural

Một mạng neural sẽ có 3 kiểu tầng:

- Tầng vào (Input layer): Là tầng bên trái cùng của mạng thể hiện cho các đầu vào của mạng.
- Tầng ra (Output layer): Là tầng bên phải cùng của mạng thể hiện cho các đầu ra của mạng.
- Tầng ẩn (Hidden layer): Là tầng nằm giữa tầng vào và tầng ra thể hiện cho việc suy luận logic của mạng. Lưu ý rằng, một NN chỉ có 1 tầng vào và 1 tầng ra nhưng có thể có nhiều tầng ẩn.

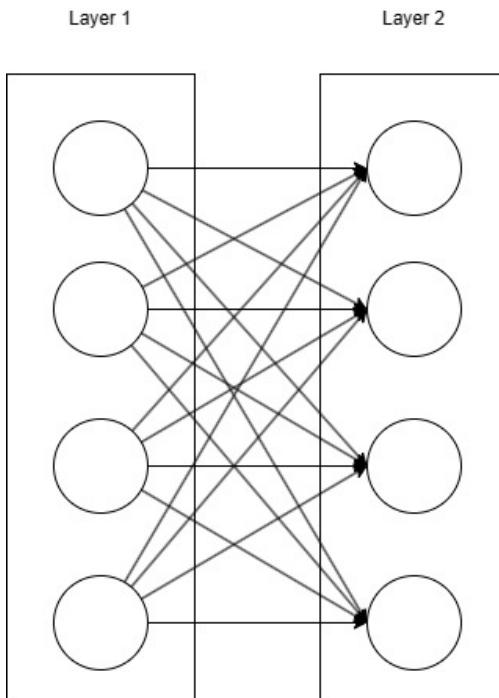
Số lượng của một mạng là không cố định. Nó được ký hiệu L với:

$$L = \text{số lượng Hidden Layer} + 1 [15] \quad (2.6)$$

Khi tính số tầng trong một mạng neural, ta sẽ không tính tầng Input.

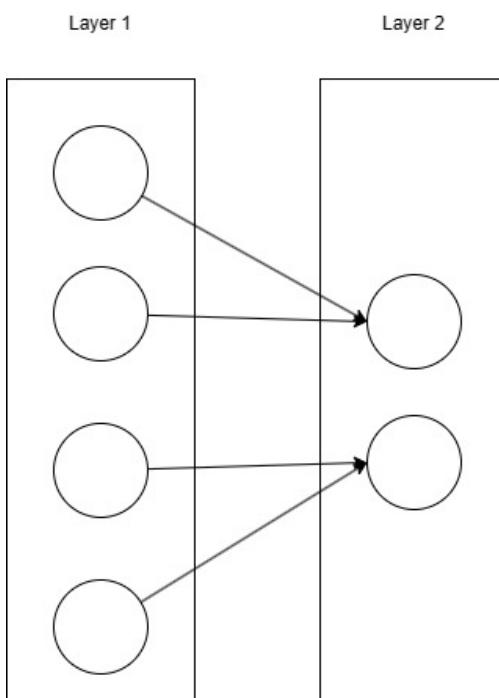
Các neural trong một lớp không được kết nối trực tiếp. Thông tin truyền một chiều từ lớp này sang lớp tiếp theo, với mỗi nơ-ron trong lớp nhận đầu vào từ lớp trước và gửi đầu ra của nó đến lớp tiếp theo. Có 2 kiểu kết nối chính:

- Fully-connected: mỗi neural trong cùng một tầng sẽ kết nối đôi một với các neural với tầng tiếp theo.



Hình 2.10: Fully Connected

- Pooling: Khác với fully-connected, pooling chỉ cho phép một vài neural trong cùng một tầng kết nối với một neural của tầng tiếp theo. Điều này sẽ giảm số lượng neural ở layer đó.



Hình 2.11: Pooling

Các kiến trúc mạng neural phổ biến

Có một số kiến trúc khác nhau cho các mạng neural, mỗi kiến trúc có điểm mạnh và điểm yếu riêng. Một số kiến trúc phổ biến nhất bao gồm:

- Mạng neural tiến nguồn (Feedforward Neural Networks): Đây là loại kiến trúc ANN đơn giản nhất, trong đó thông tin chảy theo một hướng từ đầu vào đến đầu ra. Các lớp được kết nối dày dặc, có nghĩa là mỗi neural trong một lớp được kết nối với tất cả các neural trong lớp tiếp theo.
- Mạng neural hồi quy (Recurrent Neural Networks - RNNs): Những mạng này có một thành phần "bộ nhớ", nơi thông tin có thể chảy trong chu kỳ qua mạng. Điều này cho phép mạng xử lý các chuỗi dữ liệu, như chuỗi thời gian hoặc âm thanh.
- Mạng neural tích chập (Convolutional Neural Networks - CNNs): Những mạng này được thiết kế để xử lý dữ liệu có một định dạng lưới, như hình ảnh. Các lớp bao gồm các lớp tích chập, có nhiệm vụ học để phát hiện các đặc điểm cụ thể trong dữ liệu, và các lớp gộp, giảm kích thước không gian của dữ liệu. Đây là kiến trúc chính cần tìm hiểu để giải quyết bài toán trong đề tài lần này.
- Bộ mã hóa tự động (Autoencoders): Đây là các mạng neural được sử dụng cho việc học không giám sát. Chúng bao gồm một bộ mã hóa chuyển dữ liệu đầu vào thành một biểu diễn chiều thấp hơn và một bộ giải mã chuyển biểu diễn trở lại dữ liệu ban đầu.
- Mạng neural sinh đối nghịch (Generative Adversarial Networks - GANs): Đây là các mạng neural được sử dụng cho việc mô hình hóa sinh. Chúng bao gồm hai phần: một bộ tạo ra học để tạo ra các mẫu dữ liệu mới, và một bộ phân biệt học để phân biệt giữa dữ liệu thực và dữ liệu được tạo ra.

Lan truyền tuyến (Forward Propagation)[11]

Như bạn thấy thì tất cả các nốt mạng (node) được kết hợp đôi một với nhau theo một chiều duy nhất từ tầng vào tới tầng ra. Tức là mỗi nốt ở một tầng nào đó sẽ nhận đầu vào là tất cả các nốt ở tầng trước đó mà không suy luận ngược lại. Hay nói cách khác, việc suy luận trong mạng NN là suy luận tiến (feedforward):

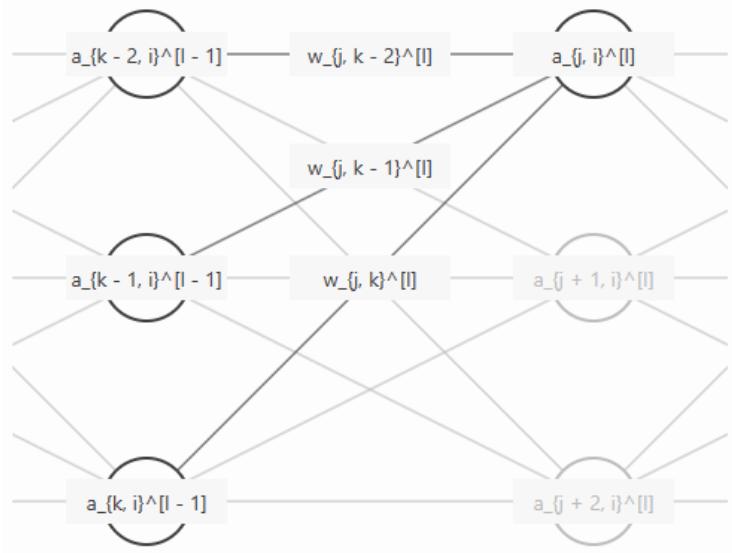
$$z_{j,i}^{[l]} = \sum_k w_{j,k}^{[l]} a_{k,i}^{[l-1]} + b_j^{[l]} \quad (2.7)$$

$$a_{j,i}^{[l]} = g_j^{[l]}(z_{1,i}^{[l]}, \dots, z_{j,i}^{[l]}, \dots, z_{n^{[l]},i}^{[l]}) \quad (2.8)$$

Trong đó:

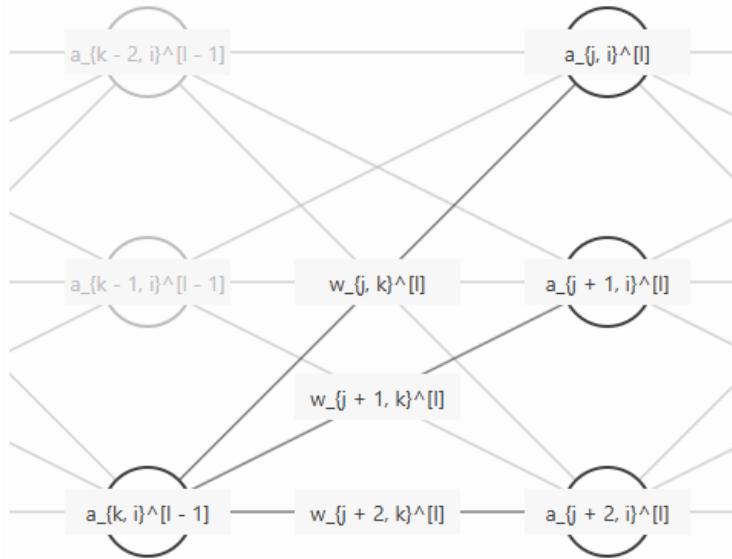
- l : Tầng hiện tại $l = 1, \dots, L$. Quy ước sử dụng $l = 0$ và $l = L$ để chỉ tầng inputs và output.
- $n^{[l]}$: Số lượng node tại tầng hiện tại
- $n^{[l-1]}$: Số lượng node tại tầng trước.
- j : Node thứ j của tầng hiện tại, $j = 1, \dots, n^l$
- k : Node thứ k của tầng trước đó, $k = 1, \dots, n^{[l-1]}$
- i : Ví dụ huấn luyện (training example) hiện tại $i = 1, \dots, m$ với m là số lượng ví dụ huấn luyện.
- $z_{j,i}^{[l]}$: Tổng trọng số của các lầm kích hoạt của lớp trước đó, được dịch chuyển theo độ lệch (bias).
- $w_{j,k}^{[l]}$: Trọng số cân bằng lầm kích hoạt thứ j của tầng trước.
- $b_j^{[l]}$: Giá trị bias của tầng hiện tại
- $a_{j,i}^{[l]}$: Lầm kích hoạt tại tầng hiện tại.
- $a_{k,i}^{[l-1]}$: Lầm kích hoạt tại tầng trước.
- $g_j^{[l]}$: Hàm kích hoạt $g_j^{[l]} : R^{n^{[l]}} \rightarrow R$ được sử dụng ở tầng hiện tại

Nói một cách ngắn gọn, một node trong lớp hiện tại phụ thuộc vào mọi node ở tầng trước và hình ảnh trực quan sau đây có thể giúp chúng ta thấy điều đó rõ ràng hơn:



Hình 2.12: Một node tại tầng hiện tại

Bên cạnh đó, một node ở lớp trước sẽ ảnh hưởng đến mọi node trong lớp hiện tại và với sự thay đổi về phần đánh dấu, chúng ta cũng có thể thấy điều đó rõ ràng hơn:



Hình 2.13: Một node ở tầng trước đó

Như vậy, ta có thể vector hóa các nút như sau:

$$\begin{bmatrix} z_{1,i}^{[l]} \\ \vdots \\ z_{j,i}^{[l]} \\ \vdots \\ z_{n^{[l]},i}^{[l]} \end{bmatrix} = \begin{bmatrix} w_{1,1}^{[l]} & \dots & w_{1,k}^{[l]} & \dots & w_{1,n^{[l-1]}}^{[l]} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ w_{j,1}^{[l]} & \dots & w_{j,k}^{[l]} & \dots & w_{j,n^{[l-1]}}^{[l]} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ w_{n^{[l]},1}^{[l]} & \dots & w_{n^{[l]},k}^{[l]} & \dots & w_{n^{[l]},n^{[l-1]}}^{[l]} \end{bmatrix} \begin{bmatrix} a_{1,i}^{[l-1]} \\ \vdots \\ a_{k,i}^{[l-1]} \\ \vdots \\ a_{n^{[l-1]},i}^{[l-1]} \end{bmatrix} + \begin{bmatrix} b_1^{[l]} \\ \vdots \\ b_j^{[l]} \\ \vdots \\ b_{n^{[l]}}^{[l]} \end{bmatrix} \quad (2.9)$$

$$\begin{bmatrix} a_{1,i}^{[l]} \\ \vdots \\ a_{k,i}^{[l]} \\ \vdots \\ a_{n^{[l]},i}^{[l]} \end{bmatrix} = \begin{bmatrix} g_1^{[l]}(z_{1,i}^{[l]}, \dots, z_{j,i}^{[l]}, \dots, z_{n^{[l]},i}^{[l]}) \\ \vdots \\ g_j^{[l]}(z_{1,i}^{[l]}, \dots, z_{j,i}^{[l]}, \dots, z_{n^{[l]},i}^{[l]}) \\ \vdots \\ g_{n^{[l]}}^{[l]}(z_{1,i}^{[l]}, \dots, z_{j,i}^{[l]}, \dots, z_{n^{[l]},i}^{[l]}) \end{bmatrix} \quad (2.10)$$

Hoặc ta có thể viết như sau:

$$z_{:,i}^{[l]} = W^{[l]} a_{:,i}^{[l-1]} + b^{[l]}, \quad (2.11)$$

$$a_{:,i}^{[l]} = g^{[l]}(z_{:,i}^{[l]}) \quad (2.12)$$

Trong đó $z_{:,i}^{[l]} \in R^{n^{[l]}}$, $W^{[l]} \in R^{n^{[l]} \times n^{[l-1]}}$, $b^{[l]} \in R^{n^{[l]}}$, $a_{:,i}^{[l]} \in R^{n^{[l]}}$, $a_{:,i}^{[l-1]} \in R^{n^{[l-1]}}$, và cuối cùng, $g^{[l]} : R^{n^{[l]}} \rightarrow R^{n^{[l]}}$. Đầu 2 chấm : ở đây dùng để làm rõ rằng $z_{:,i}^{[l]}$ là cột thứ i của $Z^{[l]}$. Tương tự với những cái khác.

Tiếp theo, chúng ta vector hoá các ví dụ huấn luyện:

$$\begin{aligned} Z^{[l]} &= [z_{:,i}^{[l]} \dots z_{:,i}^{[l]} \dots z_{:,m}^{[l]}] \\ &= W^{[l]} [a_{:,i}^{[l-1]} \dots a_{:,i}^{[l-1]} \dots a_{:,m}^{[l-1]}] + [b^{[l]} \dots b^{[l]} \dots b^{[l]}] \\ &= W^{[l]} A^{[l-1]} + \text{broadcast}(b^{[l]}) \end{aligned} \quad (2.13)$$

$$A^{[l]} = [a_{:,i}^{[l]} \dots a_{:,i}^{[l]} \dots a_{:,m}^{[l]}] \quad (2.14)$$

Trong đó, $Z^{[l]} \in R^{n^{[l]} \times m}$, $A^{[l]} \in R^{n^{[l]} \times m}$, và $A^{[l-1]} \in R^{n^{[l-1]} \times m}$.

Ta có thể lập hai ký hiệu bổ sung sau:

$$A^{[0]} = X \quad (2.15)$$

$$A^{[L]} = \hat{Y}, \quad (2.16)$$

Trong đó, $X \in R^{n^{[0]} \times m}$ là ma trận của tầng inputs và $\hat{Y} \in R^{n^{[L]} \times m}$ để chỉ ma trận tầng dự đoán/outputs. Cuối cùng, ta có thể định nghĩa hàm chi phí (cost function) như sau:

$$J = f(\hat{Y}, Y) = f(A^{[L]}, Y) \quad (2.17)$$

Trong đó, $Y \in R^{n^{[L]} \times m}$ để chỉ ma trận mục tiêu (the targets) và $f : R^{2n^L} \rightarrow R$ có thể được điều chỉnh theo nhu cầu của bài toán.

Lan truyền ngược (Backpropagation)[8]

Lan truyền ngược là quá trình giúp điều chỉnh các trọng số w và bias b của các node trong mạng neural để sai lệch giữa đầu ra dự đoán và đầu ra thực tế đạt tối thiểu.

Phương pháp phổ biến nhất để tối ưu mạng neural đa tầng chính là gradient descent (GD). Để áp dụng GD, chúng ta cần tính được gradient của hàm mất mát theo từng ma trận trọng số $W^{[l]}$ vector điều chỉnh $b^{[l]}$ (bias).

Giả sử $L(W, b, X, Y)$ là hàm mất mát của bài toán, trong đó W, b là tập hợp tất cả các ma trận trọng số và vector điều chỉnh. X, Y là cặp dữ liệu huấn luyện với mỗi cột tương ứng. Để có thể áp dụng các phương pháp gradient descent, chúng ta cần tính được các $\nabla_{W^{[l]}} J; \nabla_{b^{[l]}} J, \forall l = 1, 2, 3, \dots, L$.

Xét ví dụ của hàm mất mát là hàm sai số trung bình bình phương (MSE):

$$L(W, b, X, Y) = \frac{1}{N} n = 1N ||y_n - \hat{y}_n||_2^2 = \frac{1}{N} n = 1N ||y_n - a_n^{[L]}||_2^2 \quad (2.18)$$

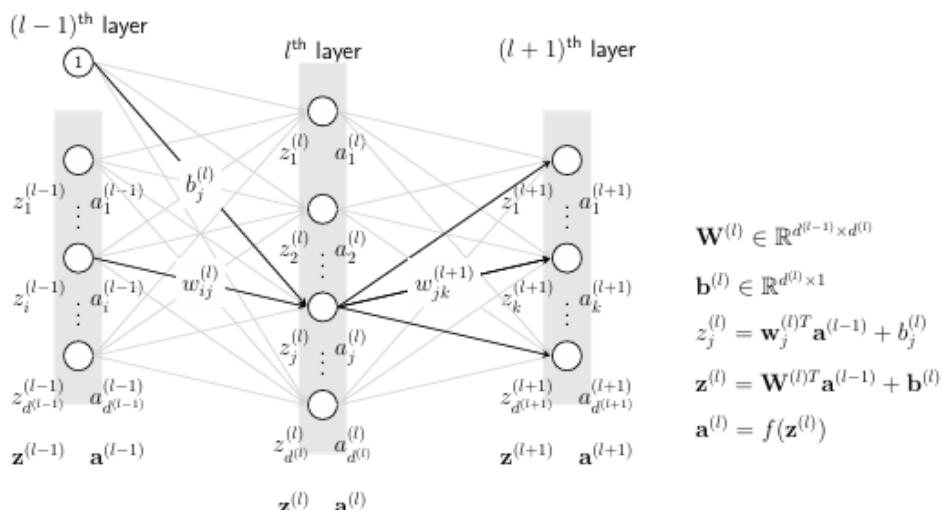
Với N là số cặp dữ liệu (x, y) trong tập huấn luyện. Theo các công thức này, việc tính toán trực tiếp các giá trị gradient tương đối phức tạp vì hàm mất mát không phụ thuộc trực tiếp vào các ma trận trọng số và vector điều chỉnh. Phương pháp phổ biến nhất được sử dụng là lan truyền ngược (backpropagation), giúp tính gradient ngược từ tầng cuối cùng đến tầng đầu tiên. Tầng cuối cùng được tính toán trước vì nó ảnh hưởng trực tiếp tới đầu ra dự đoán và hàm mất mát. Việc tính toán gradient của các ma trận trọng số trong các tầng trước được thực hiện dựa trên nguyên tắc chuỗi quen thuộc cho gradient của hàm hợp. Thuật toán Stochastic Gradient Descent thường được sử dụng để cập nhật các ma trận trọng số và vector điều chỉnh dựa trên một cặp điểm dữ liệu huấn luyện x, y . Đơn giản hơn, ta coi J là hàm mất mát nếu chỉ xét cặp điểm này. Ở đây J là hàm mất mát bất kỳ, không chỉ là hàm MSE như ở trên. Đạo hàm riêng của hàm mất mát theo chỉ một thành phần của ma trận trọng số của tầng đầu ra:

$$\frac{\partial L}{\partial w_{ij}^{[L]}} = \frac{\partial L}{\partial z_j^{[L]}} * \frac{\partial z_j^{[L]}}{\partial w_{ij}^{[l]}} = e_j^{[L]} * a_i^{[L-1]} \quad (2.19)$$

Trong đó $e_j^{[L]} = \frac{\partial L}{\partial z_j^{[L]}}$ thường là một đại lượng không quá khó để tính toán và $\frac{\partial z_j^{[L]}}{\partial w_{ij}^{[L]}} = a_i^{[L-1]}$ vì $z_j^{[L]} = w_j^{[L]T} a^{[L-1]} + b_j^{[L]}$. Tương tự, gradient của hàm mất mát theo hệ số tự do của tầng cuối cùng là

$$\frac{\partial L}{\partial b_j^{[L]}} = \frac{\partial L}{\partial z_j^{[L]}} * z_j^{[L]} b_j^{[L]} = e_j^{[L]} \quad (2.20)$$

Với đạo hàm riêng theo trọng số ở các tầng $l < L$, hãy quan sát hình minh họa bên dưới:



Hình 2.14: Mô phỏng cách tính lan truyền ngược. Tầng cuối có thể là tầng đầu ra.

Ở đây, tại các nút, đầu vào z và đầu ra a được viết riêng để tiện theo dõi. Dựa vào đó, bằng quy nạp ngược từ cuối, ta có thể tính được:

$$\frac{\partial L}{\partial w_{ij}^{[L]}} = \frac{\partial L}{\partial z_j^{[L]}} * \frac{\partial z_j^{[L]}}{\partial w_{ij}^{[l]}} = e_j^{[l]} a_i^{[l-1]} \quad (2.21)$$

Với:

$$\begin{aligned}
 e_j^{[l]} &= \frac{\partial L}{\partial z_j [l]} \\
 &= \frac{\partial L}{\partial a_j^{[l]}} * \frac{\partial a_j^{[l]}}{\partial z_j^{[l]}} \\
 &= \left(\sum_{k=1}^{d^{l+1}} \frac{\partial L}{\partial z_k^{[l+1]}} * \frac{\partial z_k^{[l+1]}}{\partial a_j^{[l]}} \right) f^{[l]'}(z_j^{[l]}) = \left(w_{j:}^{[l+1]} e^{l+1} f^{l'} \right) (z_j^l) \\
 &= \left(\sum_{k=1}^{d^{l+1}} e_k^{[l+1]} w_{jk}^{[l+1]} \right)
 \end{aligned}$$

(2.22)

Trong đó, $e^{[l+1]} = [e_1^{[l+1]}, e_2^{[l+1]}, \dots, e_{d^{[l+1]}}^{[l+1]}] \in R^{d^{[l+1]}}_1$ và $w_{j:}^{[l+1]}$ được hiểu là hàng thứ j của ma trận $W^{[l+1]}$ (Chú ý dấu hai chấm, khi không có dấu này, ta mặc định dùng nó để ký hiệu cho vector cột). Dấu \sum tính tổng ở dòng thứ hai trong phép tính trên xuất hiện vì $a_j^{[l]}$ đóng góp vào việc tính tất cả các $z_k^{[l+1]}, k = 1, 2, \dots, d^{l+1}$. Biểu thức đạo hàm ngoài dấu ngoặc lớn xuất hiện vì $a_j^{[l]} = f^{[l]}(z_j^{[l]})$. Tới đây, ta có thể thấy rằng việc hàm kích hoạt có đạo hàm đơn giản sẽ có ích rất nhiều trong việc tính toán. Với cách làm tương tự, ta có thể suy ra

$$\frac{\partial L}{\partial b_j^{[l]}} = e_j^{[l]} \quad (2.23)$$

Nhận thấy rằng trong những công thức trên, việc tính các $e_j^{[l]}$ đóng một vai trò quan trọng. Hơn nữa, để tính được giá trị này, ta cần tính được các $e_j^{[l+1]}$. Nói cách khác, ta cần tính ngược các giá trị này từ tầng cuối cùng.

Sau khi đã tính được $\frac{\partial L}{\partial w_{ij}^{[l]}}$ và $\frac{\partial L}{\partial b_j^{[l]}}$, ta có thể sử dụng thuật toán Gradient descent để điều chỉnh giá trị trọng số và bias tại các nút theo công thức sau:

$$w_{ij}^{[l]} = w_{ij}^{[l]} - lr * \frac{\partial L}{\partial w_{ij}^{[l]}} \quad (2.24)$$

$$b_i^{[l]} = b_i^{[l]} - lr * \frac{\partial L}{\partial b_i^{[l]}} \quad (2.25)$$

Trong đó lr là một số dương được gọi là tốc độ học (learning rate). Dấu trừ thể hiện việc $w_{ij}^{[l]}$ cần đi ngược với đạo hàm $\frac{\partial L}{\partial w_{ij}^{[l]}}$ và $b_i^{[l]}$ cần đi ngược với đạo hàm $\frac{\partial L}{\partial b_i^{[l]}}$.

2.2 Convolutional Neural Networks (CNNs)

2.2.1 Giới thiệu chung

Mạng neural nhân tạo (ANN) đã tạo nên cuộc cách mạng trong nhiều lĩnh vực nhờ khả năng học hỏi các mẫu phức tạp từ dữ liệu. Tuy nhiên, các kiến trúc ANN truyền thống lại gặp khó khăn trong việc xử lý hiệu quả dữ liệu dạng lưới, chẳng hạn như hình ảnh, nơi mối quan hệ giữa các pixel đóng vai trò quan trọng. Hạn chế này cản trở hiệu suất của chúng trong các nhiệm vụ như phân loại đối tượng, trong đó việc xác định và phân biệt các đối tượng trong ảnh là điều cần thiết.

Mạng neural tích chập (CNN) giải quyết thách thức này bằng cách giới thiệu một kiến trúc chuyên biệt được thiết kế cho xử lý ảnh và phân loại đối tượng. Lấy cảm hứng từ cấu trúc sinh học của vỏ não thị giác, CNN tận dụng các lớp tích chập để trích xuất các đặc trưng không gian từ hình ảnh. Các lớp này sử dụng các bộ lọc (kernels) di chuyển trên ảnh đầu vào, phát hiện các cạnh, đường thẳng và các đặc trưng cấp thấp

khác trong các vùng cụ thể. Khi mạng đi qua nhiều lớp tích chập hơn, nó học cách kết hợp các đặc trưng cấp thấp này thành các đặc trưng cấp cao phức tạp hơn, có liên quan đến việc xác định đối tượng. Khả năng trích xuất đặc trưng theo thứ bậc này cho phép CNN vượt trội trong các nhiệm vụ như phân loại đối tượng, trong đó việc nhận biết các đối tượng bên trong ảnh đòi hỏi phải hiểu các thành phần cấu thành và mối quan hệ không gian của chúng.

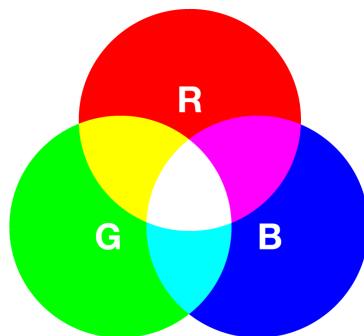
So với ANN truyền thống, CNN mang lại một số lợi thế cho việc phân loại đối tượng.

- Thứ nhất, chúng loại bỏ nhu cầu thiết kế thủ công các đặc trưng, một quá trình tốn thời gian và phụ thuộc vào lĩnh vực cụ thể. Bằng cách tự động học các đặc trưng thông qua phép tích chập, CNN có thể thích ứng với nhiều lớp đối tượng và độ phức tạp của hình ảnh khác nhau.
- Thứ hai, CNN tận dụng việc chia sẻ tham số, trong đó một bộ lọc duy nhất được áp dụng trên các phần khác nhau của ảnh. Điều này giúp giảm số lượng tham số cần học so với các mạng lưới dày dặc cùng kích thước, cải thiện hiệu quả đào tạo và giảm nguy cơ overfitting¹. Cuối cùng, kết nối cục bộ trong các lớp tích chập thúc đẩy tính bất biến tịnh tiến, một đặc tính quan trọng cho phân loại đối tượng. Điều này có nghĩa là mạng có thể nhận ra các đối tượng bất kể vị trí của chúng trong ảnh, giúp nó thích ứng với các biến thể trong bố cục hình ảnh.

2.2.2 Các khái niệm cơ bản

Ảnh trên máy tính

Thông thường, máy tính sử dụng hệ màu RGB². Khi trộn ba màu trên theo tỉ lệ nhất định có thể tạo thành các màu khác nhau.



Hình 2.15: Hệ màu RGB

Như vậy, khi ta chọn một màu thì sẽ ra một bộ ba số tương ứng (r, g, b) . Với mỗi bộ 3 số r, g, b nguyên trong khoảng $[0, 255]$ sẽ cho ra một màu khác nhau. Do có 256 cách chọn r , 256 cách chọn màu g , 256 cách chọn $b \Rightarrow$ tổng số màu có thể tạo ra bằng hệ màu RGB là: $256 * 256 * 256 = 16777216$ màu.

Ảnh trên máy tính được chia làm 2 loại là Ảnh màu và Ảnh xám

• Ảnh màu

Ảnh màu là một ma trận các pixel mà mỗi pixel biểu diễn một điểm màu. Mỗi điểm màu được biểu diễn bằng bộ 3 số (r, g, b) .

¹Overfitting (hay Quá khớp) là hiện tượng khi mô hình xây dựng thể hiện được chi tiết bộ dữ liệu huấn luyện. Điều này có nghĩa là cả dữ liệu nhiễu, hoặc dữ liệu bất thường trong tập huấn luyện đều được chọn và học để đưa ra quy luật mô hình. Những quy luật này sẽ không có ý nghĩa nhiều khi áp dụng với bộ dữ liệu mới có thể có dạng dữ liệu nhiễu khác. Khi đó, nó ảnh hưởng tiêu cực tới độ chính xác của mô hình nói chung.

²RGB viết tắt của Red (đỏ), Green (xanh lục), Blue (xanh lam), là ba màu chính của ánh sáng khi tách ra từ tia ánh sáng.



Ta có thể vector hóa một bức ảnh có kích thước $W \times H$ với W, H là chiều rộng và chiều dài của bức ảnh như sau:

$$\begin{bmatrix} (r_{1,1}, g_{1,1}, b_{1,1}) & (r_{1,2}, g_{1,2}, b_{1,2}) & \cdots & (r_{1,W}, g_{1,W}, b_{1,W}) \\ (r_{2,1}, g_{2,1}, b_{2,1}) & (r_{2,2}, g_{2,2}, b_{2,2}) & \cdots & (r_{2,W}, g_{2,W}, b_{2,W}) \\ \cdots & \cdots & \cdots & \cdots \\ (r_{H,1}, g_{H,1}, b_{H,1}) & (r_{H,2}, g_{H,2}, b_{H,2}) & \cdots & (r_{H,W}, g_{H,W}, b_{H,W}) \end{bmatrix} \quad (2.26)$$

Hoặc ta có thể tách ra thành 3 ma trận tương ứng cho mỗi màu như sau:

$$\begin{bmatrix} r_{1,1} & r_{1,2} & \cdots & r_{1,W} \\ r_{2,1} & r_{2,2} & \cdots & r_{2,W} \\ \cdots & \cdots & \cdots & \cdots \\ r_{H,1} & r_{H,2} & \cdots & r_{H,W} \end{bmatrix}, \begin{bmatrix} g_{1,1} & g_{1,2} & \cdots & g_{1,W} \\ g_{2,1} & g_{2,2} & \cdots & g_{2,W} \\ \cdots & \cdots & \cdots & \cdots \\ g_{H,1} & g_{H,2} & \cdots & g_{H,W} \end{bmatrix}, \begin{bmatrix} b_{1,1} & b_{1,2} & \cdots & b_{1,W} \\ b_{2,1} & b_{2,2} & \cdots & b_{2,W} \\ \cdots & \cdots & \cdots & \cdots \\ b_{H,1} & b_{H,2} & \cdots & b_{H,W} \end{bmatrix} \quad (2.27)$$

• Ảnh xám

Tương tự ảnh màu, ảnh xám cũng có kích thước $W \text{ pixel} \times H \text{ pixel}$, có thể biểu diễn dưới dạng một ma trận kích thước $W \times H$ (vì định nghĩa ma trận là số hàng nhân số cột).

Tuy nhiên mỗi pixel trong ảnh xám chỉ cần biểu diễn bằng một giá trị nguyên trong khoảng từ [0, 255] thay vì (r, g, b) như trong ảnh màu. Do đó khi biểu diễn ảnh xám trong máy tính chỉ cần một ma trận là đủ.

Tuy nhiên mỗi pixel trong ảnh xám chỉ cần biểu diễn bằng một giá trị nguyên trong khoảng từ [0, 255] thay vì (r, g, b) như trong ảnh màu. Do đó khi biểu diễn ảnh xám trong máy tính chỉ cần một ma trận là đủ.

Tensor

Khi dữ liệu biểu diễn dạng 1 chiều, người ta gọi là vector, mặc định khi viết vector sẽ viết dưới dạng cột. Khi dữ liệu dạng 2 chiều, người ta gọi là ma trận, kích thước là số hàng * số cột.

- $v = \begin{bmatrix} v_1 \\ v_2 \\ \cdots \\ v_n \end{bmatrix}$, vector v có kích thước là n .

- $W = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1n} \\ w_{21} & w_{22} & \cdots & w_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ w_{m1} & w_{m2} & \cdots & w_{mn} \end{bmatrix}$, ma trận W có kích thước là $m \times n$

Khi dữ liệu nhiều hơn 2 chiều thì sẽ được gọi là tensor, ví dụ như dữ liệu có 3 chiều.

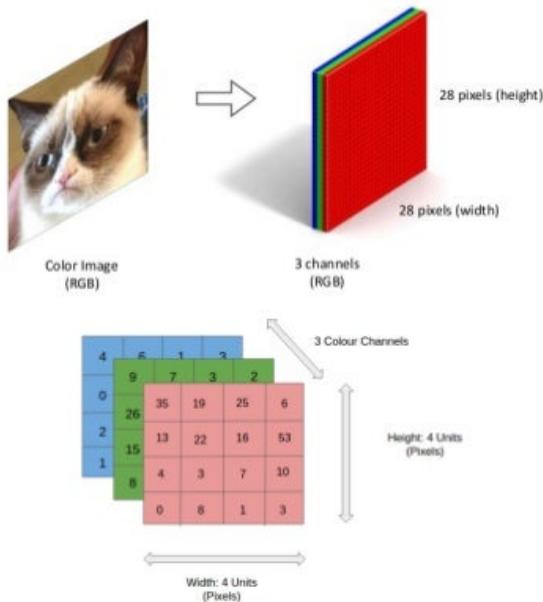
Để ý thì thấy là ma trận là sự kết hợp của các vector cùng kích thước. Xếp n vector kích thước m cạnh nhau thì sẽ được ma trận $m \times n$. Thì tensor 3 chiều cũng là sự kết hợp của các ma trận cùng kích thước, xếp k ma trận kích thước $m \times n$ lên nhau sẽ được tensor kích thước $m \times n \times k$.

Tưởng tượng mặt đáy là một ma trận kích thước $a \times b$, được tạo bởi b vector kích thước a . Cả hình hộp là tensor 3 chiều kích thước $a \times b \times h$, được tạo bởi h ma trận kích thước $a \times b$ lên nhau.

Do đó biểu diễn ảnh màu trên máy tính ở phần trên sẽ được biểu diễn dưới dạng tensor 3 chiều kích thước $600 \times 800 \times 3$ do có 3 ma trận (channel) màu red, green, blue kích thước 600×800 chồng lên nhau.

Ví dụ biểu diễn ảnh màu kích thước 28×28 , biểu diễn dưới dạng tensor $28 \times 28 \times 3$.

color image is 3rd-order tensor



Hình 2.16: Ảnh màu biểu diễn dưới dạng tensor 3 chiều[4]

Chuyển hệ màu của ảnh

Mỗi pixel trong ảnh màu được biểu diễn bằng 3 giá trị (r,g,b) còn trong ảnh xám chỉ cần 1 giá trị x để biểu diễn.

Khi chuyển từ ảnh màu sang ảnh xám ta có thể dùng công thức: $x = r \times 0.299 + g \times 0.587 + b \times 0.114$.

Tuy nhiên khi chuyển ngược lại, bạn chỉ biết giá trị x và cần đi tìm r,g,b nên sẽ không chính xác

Kernels và phép tính Convolution

[15] Kernels (hay Filter) là một ma trận vuông có kích thước $k \times k$ trong đó k là số lẻ. k có thể bằng 1, 3, 5, 7, 9, ... Ví dụ kernel kích thước 3×3

$$W = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} \quad (2.28)$$

Phép tính Convolution (tích chập) Trong toán học và đặc biệt là trong giải tích hàm, tích chập là 1 phép toán thực hiện đối với 2 hàm số f và g, kết quả cho ra 1 hàm số thứ 3. Phép tích chập khác với tương quan chéo ở chỗ nó cần lật kernel theo chiều ngang và đọc trước khi tính tổng của tích. Nó được ứng dụng trong xác suất, thống kê, thị giác máy tính (computer vision), xử lý ảnh, xử lý tín hiệu, kỹ thuật điện, học máy, và các phương trình vi phân.

Phép tính tích chập giữa 2 ma trận có ký hiệu là \otimes . Ví dụ: $Y = X \otimes W$.

Cách tính phép tính tích chập giữa ma trận X bất kỳ và kernel W, với X có kích thước $m \times n$, W có kích thước $k \times k$ và $k < m, n$, như sau: Kết quả phép tính tích chập này sẽ là 1 ma trận mới, tạm gọi là Y, có kích thước nhỏ hơn X là $(m - k + 1) \times (n - k + 1)$. Với mỗi phần tử x_{ij} trong ma trận X lấy ra một ma trận có kích thước bằng kích thước của kernel W có phần tử x_{ij} làm trung tâm (đây là vì sao kích thước của

kernel thường lẻ) gọi là ma trận A. Sau đó tính tổng các phần tử của phép tích element-wise³ của ma trận A và ma trận W, rồi viết vào ma trận kết quả Y.

1	0	1	2	1
1	2	3	3	1
2	1	1	0	1
0	2	4	2	1
1	0	0	0	2

 \otimes

1	0	1
0	1	0
1	0	1

=

1	0	1

X **W** **Y**

Hình 2.17: Tính $X \otimes W = Y$

Ví dụ khi tính tại x_{22} (ô khoanh đỏ trong hình), ma trận A cùng kích thước với W, có x_{22} làm trung tâm có màu nền da cam như trong hình. Sau đó tính $y_{11} = \text{sum}(A \otimes W) = x_{11} * w_{11} + x_{12} * w_{12} + x_{13} * w_{13} + x_{21} * w_{21} + x_{22} * w_{22} + x_{23} * w_{23} + x_{31} * w_{31} + x_{32} * w_{32} + x_{33} * w_{33} = 4$. Và làm tương tự với các phần tử còn lại trong ma trận.

Thế thì sẽ xử lý thế nào với phần tử ở viền ngoài như x_{11} ? Bình thường khi tính thì sẽ bỏ qua các phần tử ở viền ngoài, vì không tìm được ma trận A ở trong X

Padding[15]

Như vậy, sau mỗi lần tính convolution, ta sẽ thu được một ma trận có kích thước nhỏ hơn kích thước của ma trận đầu vào. Padding được sử dụng để giúp cho ma trận kết quả vẫn giữ được kích thước sau mỗi lần tính toán.

0	0	0	0	0	0
0	2	3	3	1	0
0	1	1	0	1	0
0	1	1	0	1	0
0	2	4	2	1	0
0	0	0	0	0	0

Hình 2.18: Padding

Rõ ràng là giờ đã giải quyết được vấn đề tìm A cho phần tử x_{11} , $v\text{matrn}Y\text{thu}\mathcal{O}csbngkchthcmatrnX\text{ban}\mathcal{O}u$.

Phép tính này gọi là convolution với padding=1. Padding=k nghĩa là thêm k vector 0 vào mỗi phía (trên, dưới, trái, phải) của ma trận.

³Phép tính element-wise (còn gọi là phép toán từng phần tử) là phép toán được thực hiện riêng biệt cho từng phần tử tương ứng giữa hai ma trận có cùng kích thước. Nói cách khác, phép toán này lấy giá trị tại mỗi vị trí (hàng, cột) tương ứng của hai ma trận và thực hiện phép toán nhị nguyên trên chúng, trả về một ma trận mới có cùng kích thước với hai ma trận ban đầu.

Tride[15]

Như ở trên ta thực hiện tuần tự các phần tử trong ma trận X, thu được ma trận Y cùng kích thước ma trận X, ta gọi là stride=1.

Tuy nhiên nếu stride=k ($k > 1$) thì ta chỉ thực hiện phép tính convolution trên các phần tử $x_{1+i*k, 1+j*k}$.

Hiểu đơn giản là bắt đầu từ vị trí x_{11} sau đó nhảy k bước theo chiều dọc và ngang cho đến hết ma trận X.

Kích thước của ma trận Y là 3×3 đã giảm đi đáng kể so với ma trận X.

Công thức tổng quát cho phép tính convolution của ma trận X kích thước $m \times n$ với kernel kích thước $k \times k$, stride = s, padding = p ra ma trận Y kích thước:

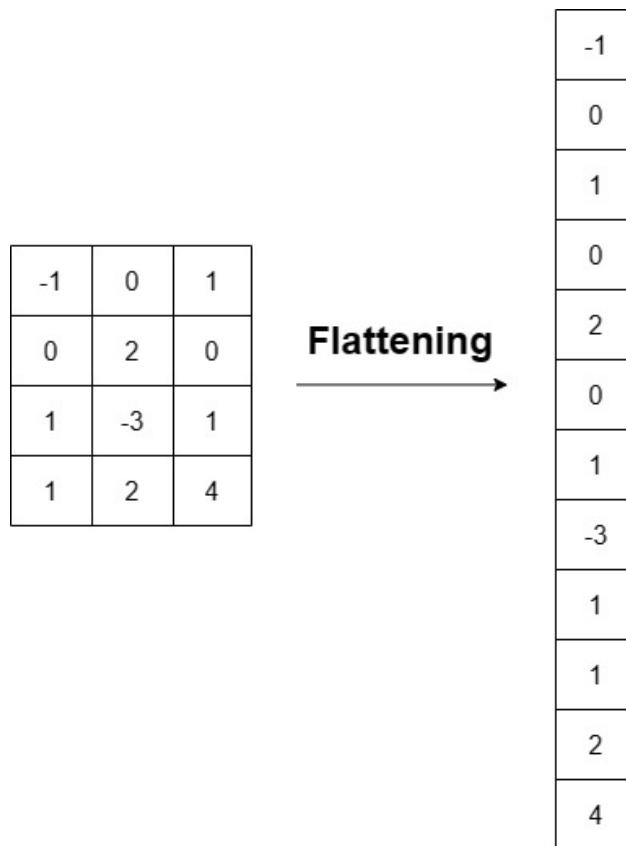
$$\left(\frac{m - k + 2p}{s} + 1 \right) \times \left(\frac{n - k + 2p}{s} + 1 \right) \quad (2.29)$$

Stride thường dùng để giảm kích thước của ma trận sau phép tính convolution.

Flatten

Trong quá trình xử lý ảnh bằng CNN, Flatten (làm phẳng) đóng vai trò quan trọng như một cầu nối giữa các lớp xử lý ảnh (chẳng hạn như lớp cuộn tích và lớp pooling) và các lớp fully-connected (hoàn toàn kết nối) thường được sử dụng ở giai đoạn sau của mạng.

Cụ thể, Flatten là việc thực hiện chuyển đổi một ma trận có kích thước (W, H) thành 1 vector có kích thước $(W * H, 1)$



Hình 2.19: Thực hiện Flatten trên ma trận

Mục đích của Flatten:

Các lớp cuộn tích và pooling trong CNN tạo ra các kết quả dạng tensor (mảng đa chiều) ba chiều, với kích thước (số lượng feature map, chiều cao, chiều rộng). Mỗi feature map là một ma trận hai chiều chứa đựng các kích hoạt của các neuron tại một vùng ảnh nhất định.

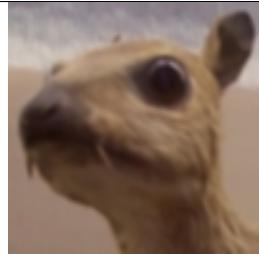
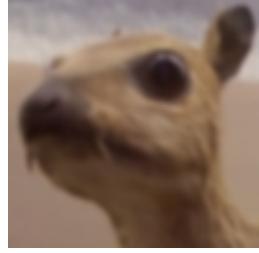
Tuy nhiên, các lớp fully-connected đòi hỏi đầu vào là một vector một chiều (mảng một chiều). Do đó, lớp Flatten thực hiện việc chuyển đổi các feature map trong tensor ba chiều thành một vector một chiều dài bằng tích của tổng số phần tử trong tất cả các feature map.

Ý nghĩa của phép tính convolutional[15]

Mục đích của phép tính convolution trên ảnh là làm mờ, làm nét ảnh; xác định các đường;... Mỗi kernel khác nhau thì sẽ phép tính convolution sẽ có ý nghĩa khác nhau.

Một số kernel thường được sử dụng:

Operation	Kernel ω	Image result $g(x, y)$
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Ridge hay edge detection	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	

Gaussian blur 3×3 (approximation)	$\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	
Gaussian blur 5×5 (approximation)	$\frac{1}{256} \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$	
Unsharp masking 5×5 Based on Gaussian blur with amount as 1 and threshold as 0 (with no image mask)	$\frac{-1}{256} \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & -476 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$	

Bảng 2.1: Một số Kernel thông dụng[16]

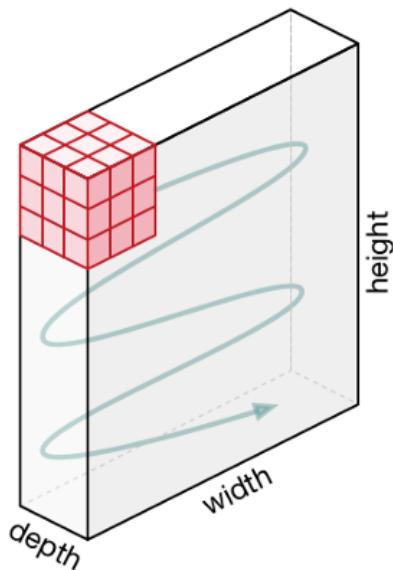
Feature map

Trong CNN, bản đồ đặc trưng (feature map) là một ma trận hai chiều chứa đựng các đặc trưng được trích xuất từ ảnh đầu vào qua quá trình thực hiện tích chập (convolution). Nó đóng vai trò then chốt trong việc giúp CNN phân biệt và nhận dạng các đối tượng trong ảnh. Mỗi lớp cuộn tích trong CNN sẽ tạo ra một tập hợp các bản đồ đặc trưng, phản ánh các mức độ kích hoạt khác nhau ở những vùng ảnh nhất định.

Nói một cách đơn giản thì feature map là kết quả của hình ảnh sau khi được các kernel lọc các đặc trưng.

2.2.3 Convolutional Layers[15]

Như chúng ta đã biết, ảnh màu có tối 3 channels là red, green và blue nên khi biểu diễn ảnh thì ta sẽ biểu diễn ảnh dưới dạng tensor 3 chiều. Vì vậy để thực hiện phép tính convolution, ta cần một kernel là 1 tensor 3 chiều có kích thước $k \times k \times 3$.



Hình 2.20: Phép tính convolution trên ảnh màu với $k=3$.

Ta định nghĩa kernel có cùng độ sâu (depth) với biểu diễn ảnh, rồi sau đó thực hiện di chuyển khối kernel tương tự như khi thực hiện trên ảnh xám.

1	0	1	2	1
1	2	3	3	1
2	1	1	0	1
0	2	4	2	1
1	0	0	0	2

1	0	1
0	1	0
1	0	1

1	0	1	2	1
1	2	3	3	1
2	1	1	0	1
0	2	4	2	1
1	0	0	0	2

 \otimes

1	0	1
0	1	0
1	0	1

 $=$

1	0	1	2	1
1	2	3	3	1
2	1	1	0	1
0	2	4	2	1
1	0	0	0	2

1	0	1
0	1	0
1	0	1

X W Y

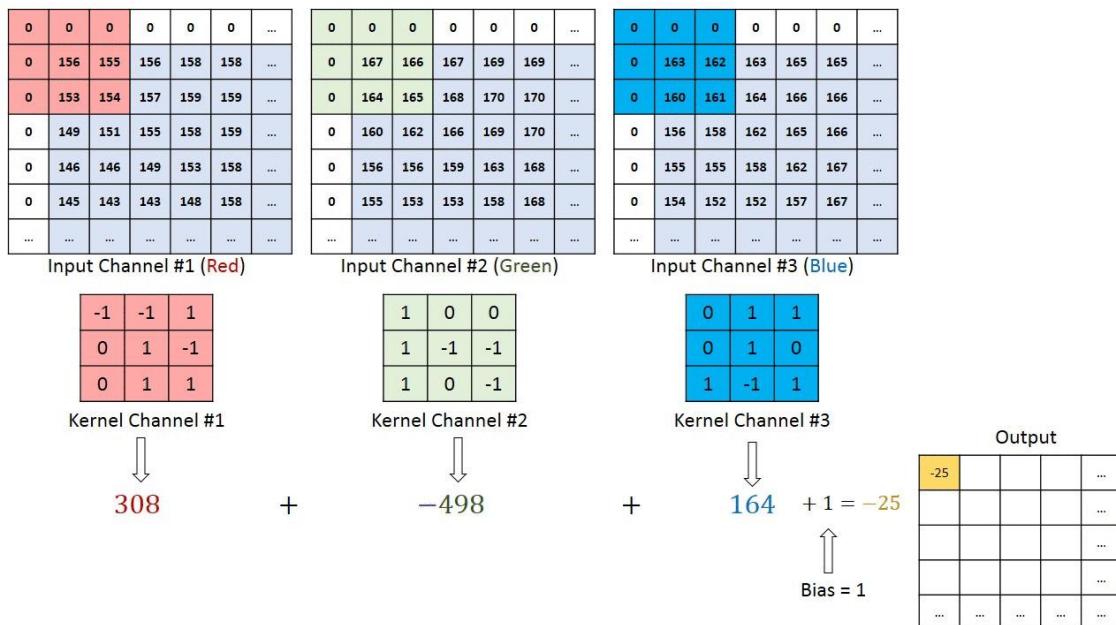
Hình 2.21: Tensor X, W 3 chiều được viết dưới dạng 3 matrix

Khi biểu diễn ma trận ta cần 2 chỉ số hàng và cột: i và j, thì khi biểu diễn ở dạng tensor 3 chiều cần thêm chỉ số độ sâu k. Nên chỉ số mỗi phần tử trong tensor là x_{ijk} .

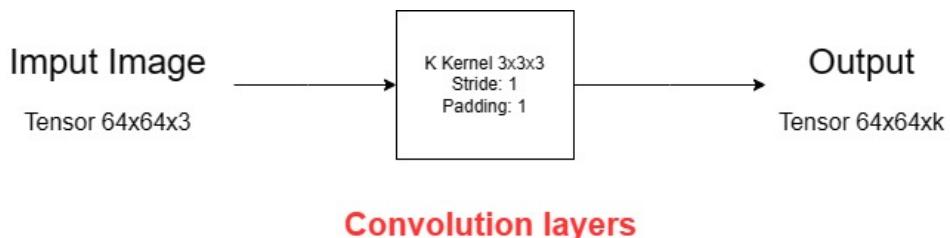
Nhận xét:

- Output Y của phép tính convolution trên ảnh màu là 1 matrix.
- Có 1 hệ số bias được cộng vào sau bước tính tổng các phần tử của phép tính element-wise

Với mỗi kernel khác nhau ta sẽ học được những đặc trưng khác nhau của ảnh, nên trong mỗi convolutional layer ta sẽ dùng nhiều kernel để học được nhiều thuộc tính của ảnh. Vì mỗi kernel cho ra output là 1 matrix nên k kernel sẽ cho ra k output matrix. Ta kết hợp k output matrix này lại thành 1 tensor 3 chiều có chiều sâu k.



Hình 2.22: Thực hiện phép tính convolution trên ảnh màu



Hình 2.23: Convolution đầu tiên

Output của convolutional layer đầu tiên sẽ thành input của convolutional layer tiếp theo.

Convolutional layer tổng quát:

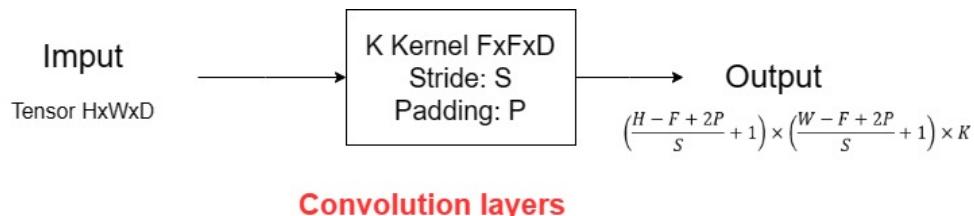
Giả sử input của 1 convolutional layer tổng quát là tensor kích thước $H \times W \times D$.

Kernel có kích thước $F \times F \times D$ (kernel luôn có depth bằng depth của input và F là số lẻ), stride: S, padding: P.

Convolutional layer áp dụng K kernel.

⇒ Output của layer là tensor 3 chiều có kích thước:

$$\left(\frac{H - F + 2P}{S} + 1 \right) \times \left(\frac{W - F + 2P}{S} + 1 \right) \times K \quad (2.30)$$



Hình 2.24: Convolution tổng quát

Lưu ý:

- Output của convolutional layer sẽ qua hàm non-linear activation function trước khi trở thành input của convolutional layer tiếp theo.
- Tổng số parameter của layer: Mỗi kernel có kích thước $F \times F \times D$ và có 1 hệ số bias, nên tổng parameter của 1 kernel là $F \times F \times D + 1$. Mà convolutional layer áp dụng K kernel \Rightarrow Tổng số parameter trong layer này là $K \times (F \times F \times D + 1)$.

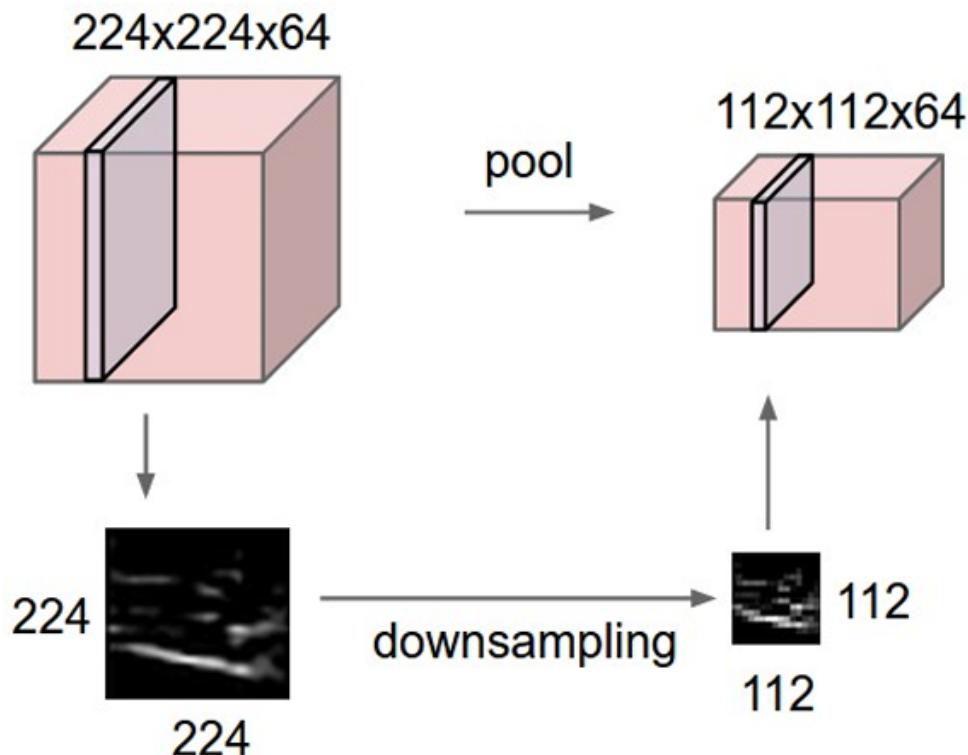
2.2.4 Pooling Layers[15]

Pooling layer thường được dùng giữa các convolutional layer, để giảm kích thước dữ liệu nhưng vẫn giữ được các thuộc tính quan trọng. Việc giảm kích thước dữ liệu giúp giảm các phép tính toán trong model.

Bên cạnh đó, với phép pooling kích thước ảnh giảm, do đó lớp convolution học được các vùng có kích thước lớn hơn. Ví dụ như ảnh kích thước 224×224 qua pooling về 112×112 thì vùng 3×3 ở ảnh 112×112 tương ứng với vùng 6×6 ở ảnh ban đầu. Vì vậy qua các pooling thì kích thước ảnh nhỏ đi và convolutional layer sẽ học được các thuộc tính lớn hơn.

Gọi pooling size kích thước $K \times K$. Input của pooling layer có kích thước $H \times W \times D$, ta tách ra làm D ma trận kích thước $H \times W$. Với mỗi ma trận, trên vùng kích thước $K \times K$ trên ma trận ta tìm maximum hoặc average của dữ liệu rồi viết vào ma trận kết quả. Quy tắc về stride và padding áp dụng như phép tính convolution trên ảnh.

Hầu hết khi dùng pooling layer thì sẽ dùng size=(2,2), stride=2, padding=0. Khi đó output width và height của dữ liệu giảm đi một nửa, depth thì được giữ nguyên.



Hình 2.25: Sau pooling layer (2×2)[3]

Có 2 loại pooling layer phổ biến là: max pooling và average pooling.

- Max pooling: chọn phần tử có giá trị lớn nhất trong pooling layer.
- Average pooling: tính giá trị trung bình của các phần tử trong pooling layer.

Công thức tìm kích thước feature map đầu ra sau thực hiện pooling như sau:

$$h' = \left\lceil \frac{h - f}{s} \right\rceil \quad (2.31)$$

$$w' = \left\lceil \frac{w - f}{s} \right\rceil \quad (2.32)$$

Trong đó:

- h' : Chiều cao của feature map đầu ra,
- w' : Chiều rộng của feature map đầu ra,
- h : Chiều cao của feature map đầu vào,
- w : Chiều rộng của feature map đầu vào,
- f : Kích thước của vùng pooling.
- s : Giá trị của stride trong việc thực hiện pooling.

2.2.5 Fully Connected Layers

Sau khi ảnh được truyền qua nhiều convolutional layer và pooling layer thì model đã học được tương đối các đặc điểm của ảnh (ví dụ mắt, mũi, khung mặt,...) thì tensor của output của layer cuối cùng, kích thước $H \times W \times D$, sẽ được flatten về 1 vector kích thước $(H \times W \times D, 1)$.

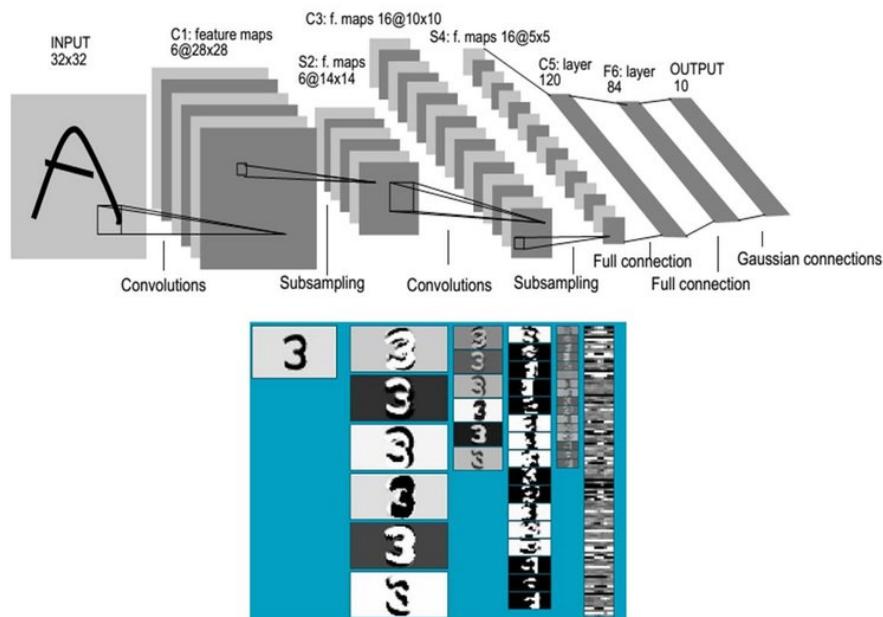
Sau đó ta dùng các fully connected layer để kết hợp các đặc điểm của ảnh để ra được output của model.

2.2.6 Các kiến trúc CNN cho bài toán phân loại đối tượng

LeNet-5[6]

Kiến trúc LeNet-5 là một trong những kiến trúc CNN sớm nhất, được thiết kế để phân loại các chữ số viết tay. Nó được giới thiệu bởi LeCun và đồng nghiệp vào năm 1998. LeNet-5 có 5 lớp có trọng số (có thể huấn luyện), gồm ba lớp tích chập và hai lớp FC. Trong số đó, mỗi lớp tích chập đầu tiên được sau bởi một lớp max-pooling (để lấy mẫu con của các bản đồ đặc trưng) và sau đó, lớp tích chập cuối cùng được sau bởi hai lớp fully connected. Lớp cuối cùng của các lớp fully connected đó được sử dụng làm bộ phân loại, có thể phân loại 10 chữ số. **Chú ý quan trọng:**

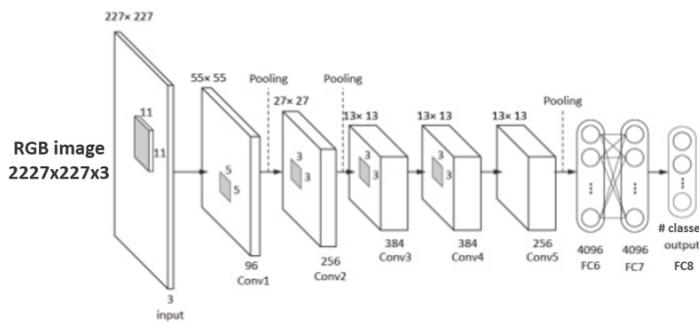
- LeNet-5 được huấn luyện trên bộ dữ liệu chữ số MNIST.
- Nó sử dụng hàm kích hoạt sigmoid làm hàm không tuyến tính.
- Nó sử dụng thuật toán học SGD (stochastic gradient descent) với 20 training epoch.
- Nó sử dụng hệ số động lượng (momentum factor) = 0.02.
- Nó giảm tỷ lệ lỗi thử nghiệm xuống 0.95% trên tập dữ liệu MNIST.



Hình 2.26: Kiến trúc Lenet-5

AlexNet[6]

Lấy cảm hứng từ LeNet, Krizhevsky và đồng nghiệp thiết kế mô hình CNN quy mô lớn đầu tiên, được gọi là AlexNet vào năm 2012, được thiết kế để phân loại dữ liệu ImageNet. Nó bao gồm tám lớp có trọng số (có thể học được), trong đó năm lớp đầu tiên là các lớp tích chập, và sau đó, ba lớp cuối cùng là các lớp fully connected. Vì nó được thiết kế cho dữ liệu ImageNet, nên lớp đầu ra cuối cùng phân loại các hình ảnh đầu vào thành một trong các lớp thuộc 1.000 lớp của tập dữ liệu ImageNet với sự trợ giúp của 1.000 đơn vị.



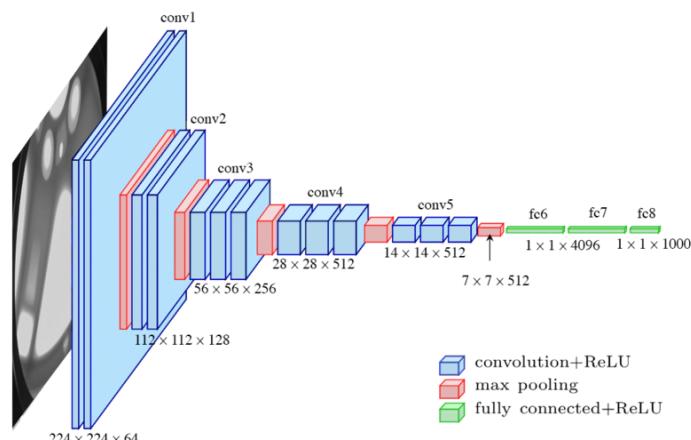
Hình 2.27: Kiến trúc AlexNet

Chú ý quan trọng:

- AlexNet sử dụng hàm kích hoạt phi tuyến tính đơn vị tuyến tính được sửa đổi (ReLU) sau mỗi lớp tích chập và fully connected.
- Nó sử dụng lớp max-pooling sau mỗi lớp LRN và lớp tích chập cuối cùng.
- Do có một số lượng lớn trọng số (có thể học được), AlexNet sử dụng một số kỹ thuật điều chỉnh như dropout và augmentation dữ liệu để tránh việc quá mức.
- AlexNet được huấn luyện bằng thuật toán học gradient giả ngẫu nhiên (SGD) với kích thước mini-batch là 128, giảm trọng lượng (weight decay) là 0.00005 và hệ số động lượng (momentum factor) là 0.9.
- AlexNet được huấn luyện trên hai NVIDIA GTX 580 (với bộ nhớ 3 GB) bằng cách song song hóa trên nhiều GPU và mất khoảng sáu ngày để hoàn thành (trên tập dữ liệu ImageNet).
- AlexNet là người chiến thắng của ILSVRC-2012.

VGG 16 (Visual Geometry Group 16)[15]

VGG16 là mạng convolutional neural network được đề xuất bởi K. Simonyan and A. Zisserman, University of Oxford. Model sau khi train bởi mạng VGG16 đạt độ chính xác 92.7% top-5 test trong dữ liệu ImageNet gồm 14 triệu hình ảnh thuộc 1000 lớp khác nhau. Giờ áp dụng kiến thức ở trên để phân tích mạng VGG 16.



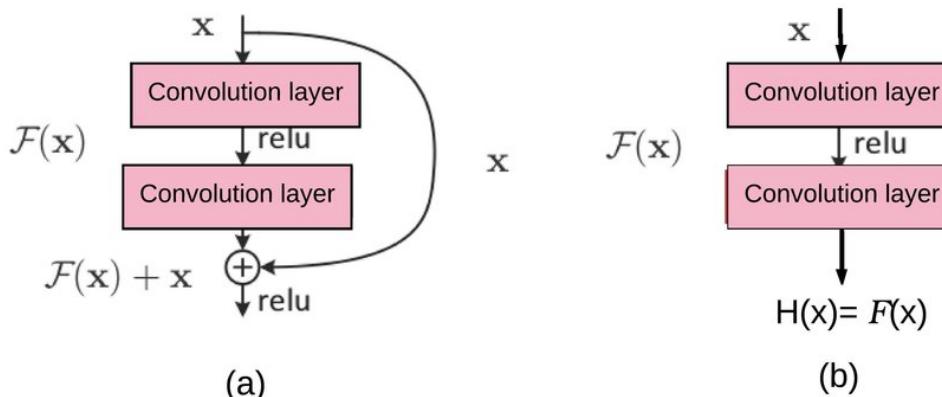
Hình 2.28: Kiến trúc VGG 16

Chú ý quan trọng:

- Convolutional layer: kích thước filter, padding=1, stride=1. Tại sao không ghi stride, padding mà vẫn biết? Vì mặc định stride=1 và padding được sử dụng để giữ cho output có cùng width và height với input.
- Pool/2: max pooling layer với size 2x2.
- 3×3 conv, 64: thì 64 là số kernel được áp dụng trong layer đấy, hay depth của output của layer này.
- Càng các convolutional layer sau thì kích thước width, height càng giảm nhưng depth càng tăng.
- Sau khá nhiều convolutional layer và pooling layer thì dữ liệu được flatten và cho vào fully connected layer.

ResNet (Residual Network)[6]

Do hạn chế của các mạng CNN sâu về vấn đề mất gradient (vanishing gradient) như chúng ta đã thảo luận trước đó, He et al. từ Microsoft đã giới thiệu khái niệm kết nối身份 (identity skip connection) để giải quyết vấn đề này bằng cách đề xuất mô hình ResNet. Kiến trúc ResNet sử dụng ánh xạ residual ($H(x) = F(x) + x$) thay vì học ánh xạ trực tiếp ($H(x) = F(x)$). Các khối này được gọi là khối residual. Toàn bộ kiến trúc ResNet được cấu thành từ nhiều khối residual với các lớp tích chập (convolution) kích thước 3×3 .



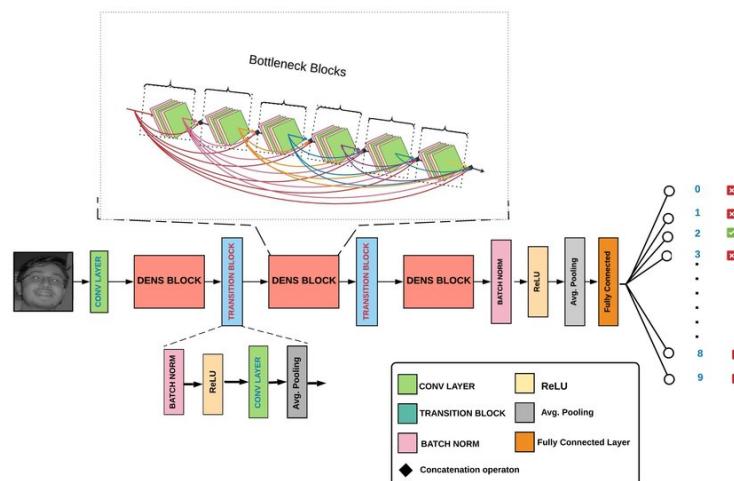
Hình 2.29: (a) Ánh xạ bên trong Khối Residual, (b) Ánh xạ trực tiếp

Chú ý quan trọng:

- Các tác giả đề xuất nhiều phiên bản ResNet với độ sâu khác nhau.
- Họ sử dụng lớp "bottleneck" (thắt cổ chai) để giảm kích thước (dimensionality reduction) trong mỗi kiến trúc ResNet có độ sâu hơn 50 lớp.
- Mặc dù ResNet (với 152 lớp) có độ sâu gấp 8 lần VGGNets (22 lớp), nhưng độ phức tạp của nó lại thấp hơn VGGNets (16/19).
- ResNet sử dụng thuật toán học tập SGD (Stochastic Gradient Descent) với kích thước mini-batch là 128, weight decay (suy giảm trọng số) là 0.0001. momentum (động lượng) là 0.9.
- ResNet đã giành chiến thắng trong cuộc thi ILSVRC-2015 với bước tiến vượt bậc về hiệu suất, giảm tỷ lệ lỗi top-5 xuống 3,6%. Nhà vô địch năm trước là GoogleNet có tỷ lệ lỗi top-5 là 6,7%.

DenseNet (Densely Connected Convolutional Networks)[6]

DenseNet mở rộng khái niệm ánh xạ residual bằng cách truyền lan đầu ra của mỗi khối tới tất cả các khối tiếp theo bên trong mỗi khối dense trong mạng. Bằng cách truyền lan thông tin theo cả hướng thuận và ngược dòng trong quá trình huấn luyện mô hình, DenseNet tăng cường khả năng lan truyền đặc trưng và giải quyết vấn đề mất gradient. DenseNet được Huang et al. giới thiệu vào năm 2016 và đã giành chiến thắng trong cuộc thi ILSVRC-2016.



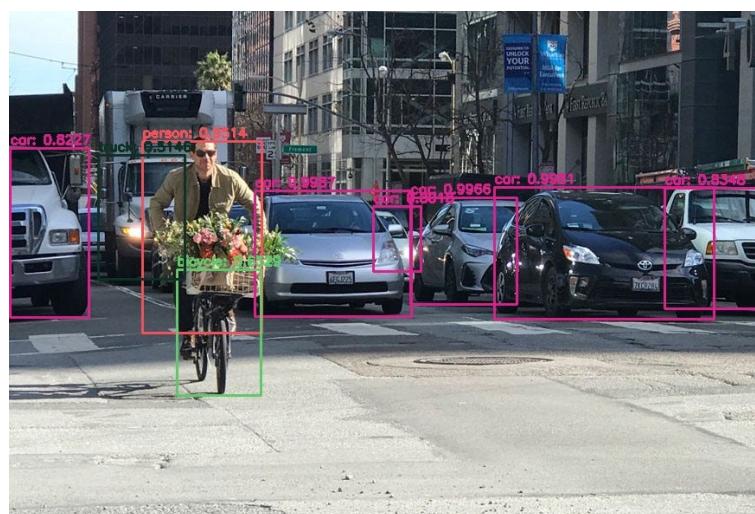
Hình 2.30: Mô hình dựa trên DenseNet với 3 khối dense.

2.3 Computer Vision

2.3.1 Bài toán object detection

Đối với những mô hình CNN bình thường, các ảnh input chỉ bao gồm 01 đối tượng cụ thể như chữ số hay 01 loài hoa.

Tuy nhiên là ảnh trong cuộc sống bình thường thì không chỉ có 01 đối tượng mà thường chứa nhiều các đối tượng khác. Từ đó nảy sinh vấn đề cần tìm vị trí của từng đối tượng trong ảnh. Đó là bài toán object detection.[15]



Hình 2.31: Ví dụ cho object detection

Bài toán object detection có input là ảnh màu và output là vị trí của các đối tượng trong ảnh. Ta thấy nó bao gồm 2 bài toán nhỏ:

- Xác định các bounding box (hình chữ nhật) quanh đối tượng.
- Với mỗi bounding box thì cần phân loại xem đây là đối tượng gì (chó, ngựa, ô tô,...) với bao nhiêu phần trăm chắc chắn.

Việc lựa chọn có bao nhiêu loại đối tượng thì phụ thuộc vào bài toán mà ta đang giải quyết.

Như vậy, ta có thể thấy được các mô hình CNN truyền thống không thể giải quyết được bài toán object detection do không biết trước có bao nhiêu đối tượng trong ảnh, nên không thiết kế được output layer hiệu quả. Để khắc phục được các vấn đề này, các mô hình R-CNN (regional convolutional neural network) ra đời.

2.3.2 R-CNN (Region with CNN feature)

Ý tưởng của R-CNN

Ý tưởng thuật toán R-CNN khá đơn giản

- Bước 1: Dùng Selective Search algorithm để lấy ra khoảng 2000 bounding box trong input mà có khả năng chứa đối tượng.
- Bước 2: Với mỗi bounding box ta xác định xem nó là đối tượng nào (người, ô tô, xe đạp,...)

Thuật toán Selective search

[2] Selective Search là một thuật toán để xuất vùng sử dụng trong phát hiện đối tượng. Nó được thiết kế để nhanh chóng với tỷ lệ gọi lại rất cao. Nó dựa trên việc tính toán việc nhóm tương tự theo cấp bậc dựa trên sự tương thích về màu sắc, kết cấu, kích thước và hình dạng.

Selective Search bắt đầu bằng cách chia nhỏ hình ảnh dựa trên cường độ pixel bằng phương pháp phân đoạn dựa trên đồ thị (graph-based segmentation method)[2] được phát triển bởi Felzenszwalb và Huttenlocher. Kết quả của thuật toán được hiển thị bên dưới. Hình ảnh bên phải chứa các vùng phân đoạn được biểu diễn bằng các màu đồng nhất (solid colors).



Hình 2.32: Ảnh đầu vào



Hình 2.33: Ảnh đầu ra

Chúng ta không thể sử dụng các vùng phân đoạn trong ảnh này làm gợi ý vùng vì hai lý do sau:

- Hầu hết các đối tượng thực tế trong ảnh gốc chứa 2 hoặc nhiều vùng phân đoạn.
- Phương pháp này không thể tạo ra các gợi ý vùng cho các đối tượng bị che khuất, chẳng hạn như cái dĩa bị che bởi cái ly hoặc cái ly đựng đầy cà phê.

Nếu chúng ta cố gắng giải quyết vấn đề đầu tiên bằng cách hợp nhất thêm các vùng liền kề có đặc điểm tương đồng, chúng ta sẽ kết thúc với một vùng phân đoạn duy nhất bao gồm hai đối tượng.

Mục tiêu ở đây không phải là phân vùng hoàn hảo. Chúng ta chỉ muốn dự đoán nhiều gợi ý vùng sao cho một số vùng trong số đó có độ trùng lặp cao với các đối tượng thực tế.

Selective Search sử dụng các vùng phân đoạn quá mức từ phương pháp của Felzenszwalb và Huttenlocher làm nền tảng ban đầu. Một hình ảnh được phân đoạn quá mức trông như thế này

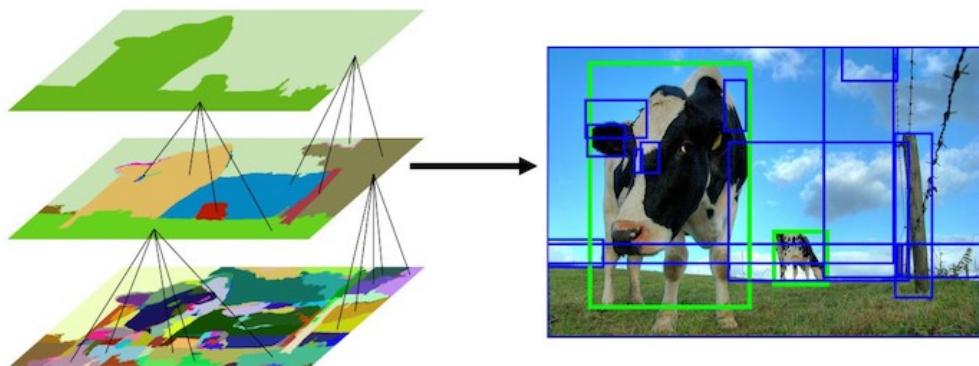


Hình 2.34: Ảnh bị phân đoạn quá mức (oversegment)

Thuật toán Selective Search sử dụng các phân đoạn này như là đầu vào ban đầu và thực hiện các bước sau:

1. Thêm tất cả các khung hình bao quanh tương ứng với các vùng phân đoạn vào danh sách gợi ý vùng,
2. Nhóm các phân đoạn kề nhau dựa trên sự tương đồng,
3. Quay lại bước 1.

Tại mỗi vòng lặp, các phân đoạn lớn hơn được hình thành và được thêm vào danh sách các đề xuất vùng. Do đó, chúng ta tạo ra các đề xuất vùng từ các phân đoạn nhỏ đến các phân đoạn lớn theo cách tiếp cận từ dưới lên (bottom-up approach). Đây chính là ý nghĩa của việc tính toán "phân đoạn theo cấp bậc" ("hierarchical" segmentations)[2] sử dụng các phân đoạn từ phương pháp oversegments của Felzenszwalb và Huttenlocher.



Hình 2.35: Các bước của quá trình phân đoạn theo thứ bậc (hierarchical segmentation)[10]

Selective Search sử dụng 4 thước đo tương tự dựa trên khả năng tương thích về màu sắc, kết cấu, kích thước và hình dạng.

a Độ tương đồng màu sắc (Color Similarity)[2]

Một biểu đồ màu (color histogram) gồm 25 bin được tính cho mỗi channel của hình ảnh và các biểu đồ cho tất cả các kênh được ghép nối để thu được bộ mô tả màu dẫn đến bộ mô tả màu (color descriptor⁴) $25 \times 3 = 75$ chiều.

Dộ tương đồng về màu sắc của hai vùng được dựa trên giao cắt histogram (histogram intersection) và có thể được tính toán theo công thức:

$$s_{color}(r_i, r_j) = \sum_{k=1}^n \min(c_i^k, c_j^k) \quad (2.33)$$

Với c_i^k là giá trị histogram với bin thứ k trong bộ mô tả màu.

b Độ tương đồng hoạ tiết (Texture Similarity)[2]

Các đặc điểm hoạ tiết (Texture features) được tính bằng cách trích xuất đạo hàm Gaussian tại 8 hướng cho mỗi kênh. Đối với mỗi hướng và mỗi kênh màu, một histogram 10-bin được tính toán, dẫn đến một mô tả đặc trưng (feature descriptor) có kích thước $10 \times 8 \times 3 = 240$ chiều.

Sự tương đồng về hoạ tiết của hai khu vực cũng được tính bằng cách sử dụng giao nhau của các histogram.

$$s_{texture}(r_i, r_j) = \sum_{k=1}^n \min(t_i^k, t_j^k) \quad (2.34)$$

Với t_i^k là giá trị histogram với bin thứ k trong bộ mô tả hoạ tiết.

c Độ tương đồng kích thước (Size Similarity)[2]

Dộ tương đồng kích thước khuyến khích các vùng nhỏ hơn hợp nhất sớm hơn. Điều này đảm bảo rằng các gợi ý vùng ở mọi kích thước được tạo thành trên tất cả các phần của ảnh. Nếu phép đo độ tương đồng này không được tính đến, một vùng đơn lẻ sẽ tiếp tục gộp tất cả các vùng nhỏ hơn liền kề, từng vùng một, và do đó các gợi ý vùng ở nhiều kích thước chỉ được tạo ra ở vị trí này. Độ tương đồng kích thước được định nghĩa như:

$$s_{size}(r_i, r_j) = 1 - \frac{\text{size}(r_i) + \text{size}(r_j)}{\text{size}(im)} \quad (2.35)$$

Với $\text{size}(im)$ là kích thước của ảnh được đo bằng đơn vị pixel.

d Độ tương đồng hình dạng (Shape Compatibility)[2]

Dộ tương đồng hình dạng đo lường mức độ hai vùng ($r_i v r_j$) phù hợp với nhau. Nếu r_i vừa khít vào r_j , chúng ta muốn gộp chúng lại với nhau để lắp đầy khoảng trống và nếu chúng không tiếp xúc với nhau thì chúng không nên được gộp lại. Độ tương thích hình dạng được xác định như sau:

$$s_{fill}(r_i, r_j) = 1 - \frac{\text{size}(BB_{ij}) - \text{size}(r_i) - \text{size}(r_j)}{\text{size}(im)} \quad (2.36)$$

Với $\text{size}(BB_{ij})$ là khung bao quan r_i và r_j .

⁴Bộ mô tả màu (color descriptor) là một biểu diễn số hóa của màu sắc trong một hình ảnh. Nó có thể được sử dụng để mô tả các đặc điểm màu sắc của các vùng trong hình ảnh và thường được sử dụng trong các ứng dụng nhận dạng và phân loại hình ảnh. Một mô tả màu thường bao gồm các thông tin về phân phôi màu sắc, các thành phần màu chính, hoặc các tính toán thống kê về màu sắc như histogram màu.

e Độ tương đồng cuối cùng (Final Similarity)[2]

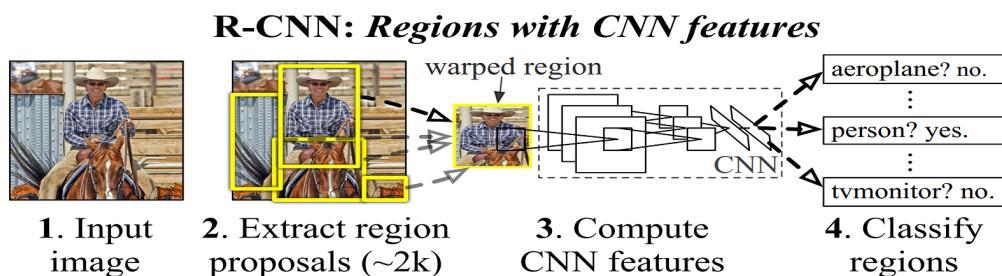
Độ tương đồng cuối cùng giữa hai vùng được định nghĩa là một tổ hợp tuyến tính (linear combination) của 4 phép đo độ tương đồng được đề cập trước đó (màu sắc, họa tiết, kích thước và hình dạng).

$$s(r_i, r_j) = a_1 s_{color}(r_i, r_j) + a_2 s_{texture}(r_i, r_j) + a_3 s_{size}(r_i, r_j) + a_4 s_{file}(r_i, r_j) \quad (2.37)$$

Với r_i và r_j là 2 vùng hoặc cạnh trong hình và $a_i \in 0, 1$ để chỉ ra rằng liệu 2 vùng có tương đồng hay không.

Phân loại region proposal

Bài toán trở thành phân loại ảnh cho các region proposal. Do thuật toán selective search cho tới 2000 region proposal nên có rất nhiều region proposal không chứa đối tượng nào. Vậy nên ta cần thêm 1 lớp background (không chứa đối tượng nào). Ví dụ như hình dưới ta có 4 region proposal, ta sẽ phân loại mỗi bounding box là người, ngựa hay background.



Hình 2.36: Các bước trong R-CNN[13]

Sau đó các region proposal được resize lại về cùng kích thước và thực hiện transfer learning với feature extractor, sau đó các extracted feature được cho vào thuật toán SVM để phân loại ảnh.

Bên cạnh đó thì extracted feature cũng được dùng để dự đoán 4 offset values cho mỗi cạnh. Ví dụ như khi region proposal chứa người nhưng chỉ có phần thân và nửa mặt, nửa mặt còn lại không có trong region proposal đó thì offset value có thể giúp mở rộng region proposal để lấy được toàn bộ người.

Vấn đề với R-CNN

Hồi mới xuất hiện thì thuật toán hoạt động khá tốt cho với các thuật toán về computer vision trước đó nhờ vào CNN, tuy nhiên nó vẫn có khá nhiều hạn chế:

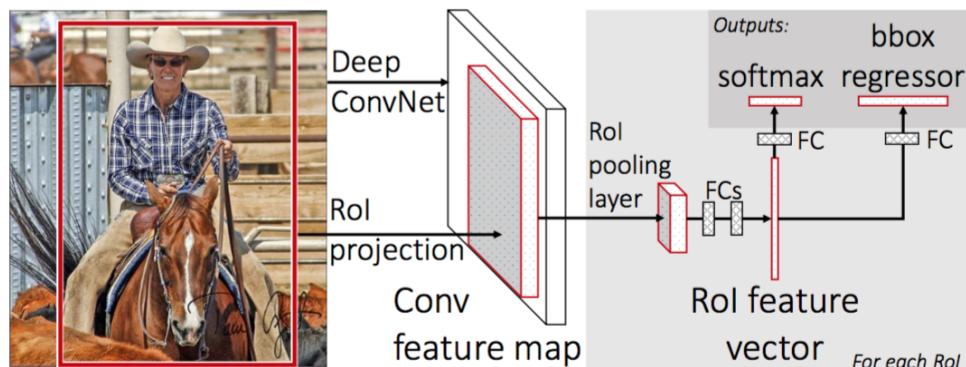
- Vì với mỗi ảnh ta cần phân loại các class cho 2000 region proposal nên thời gian train rất lâu.
- Không thể áp dụng cho real-time vì mỗi ảnh trong test set mất tới 47s để xử lý.

2.3.3 Fast R-CNN

Giới thiệu

Khoảng 1.5 năm sau khi R-CNN ra đời, Fast R-CNN được giới thiệu bởi cùng tác giả của R-CNN, nó giải quyết được một số hạn chế của R-CNN để cải thiện tốc độ. Tương tự như R-CNN thì Fast R-CNN vẫn dùng selective search để lấy ra các region proposal. Tuy nhiên là nó không tách 2000 region proposal ra khỏi ảnh và thực hiện bài toán image classification cho mỗi ảnh. Fast R-CNN cho cả bức ảnh vào ConvNet (một vài convolutional layer + max pooling layer) để tạo ra convolutional feature map. Sau đó các vùng

region proposal được lấy ra tương ứng từ convolutional feature map. Tiếp đó được Flatten và thêm 2 Fully connected layer (FCs) để dự đoán lớp của region proposal và giá trị offset values của bounding box.



Hình 2.37: Các bước trong Fast R-CNN

Tuy nhiên là kích thước của các region proposal khác nhau nên khi Flatten sẽ ra các vector có kích thước khác nhau nên không thể áp dụng neural network được. Thử nhìn lại xem ở trên R-CNN đã xử lý như thế nào? Nó đã resize các region proposal về cùng kích thước trước khi dùng transfer learning. Tuy nhiên ở feature map ta không thể resize được, nên ta phải có cách gì đấy để chuyển các region proposal trong feature map về cùng kích thước \Rightarrow Region of Interest (ROI) pooling ra đời.

Region of Internet (ROI) pooling

ROI pooling là một dạng của pooling layer. Điểm khác so với max pooling hay average pooling là bất kể kích thước của tensor input, ROI pooling luôn cho ra output có kích thước cố định được định nghĩa trước.

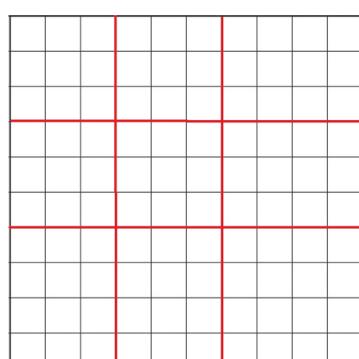
Ta ký hiệu a/b là phần nguyên của a khi chia cho b và $a\%b$ là phần dư của a khi chia cho b .

Ví dụ: $10/3 = 3$ và $10\%3 = 1$.

Gọi input của ROI pooling kích thước $m \times n$ và output có kích thước $h \times k$ (thông thường h, k khá nhỏ; ví dụ 7×7).

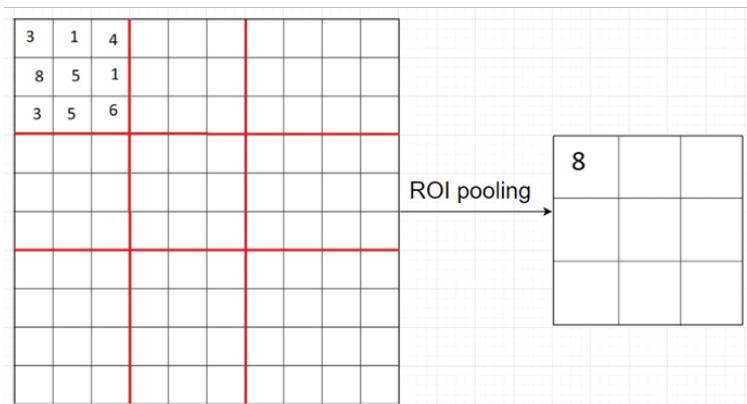
- Ta chia chiều rộng thành h phần, $(h-1)$ phần có kích thước m/h , phần cuối có kích thước $m/h + m\%h$.
- Tương tự ta chia chiều dài thành k phần, $(k-1)$ phần có kích thước n/k , phần cuối có kích thước $n/k + n\%k$.

Ví dụ $m = n = 10$, $h = k = 3$, do $m/h = 3$ và $m\%h = 1$, nên ta sẽ chia chiều rộng thành 3 phần, 2 phần có kích thước 3, và 1 phần có kích thước 4.



Hình 2.38: Hình minh họa cách chia của ROI

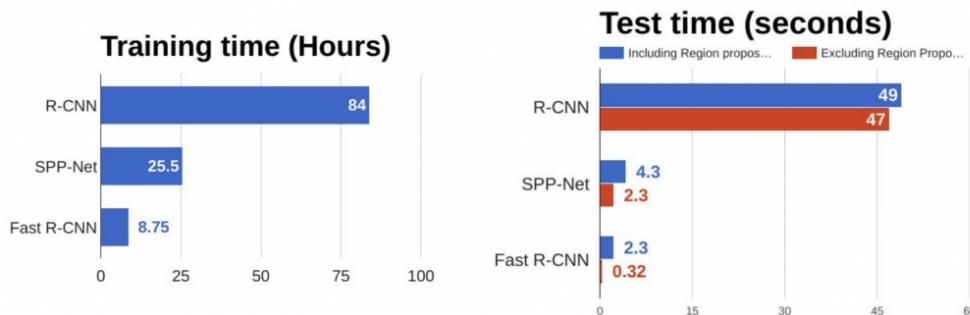
Sau đó với mỗi khối được tạo ra bằng các đường dọc và cạnh, ta thực hiện max pooling lấy ra 1 giá trị.



Hình 2.39: Thực hiện ROI pooling

Ta có thể thấy là kích thước sau khi thực hiện ROI pooling về đúng $h*k$ như ta mong muốn.

Dánh giá Fast R-CNN



Hình 2.40: So sánh thời train và test giữa R-CNN và Fast R-CNN

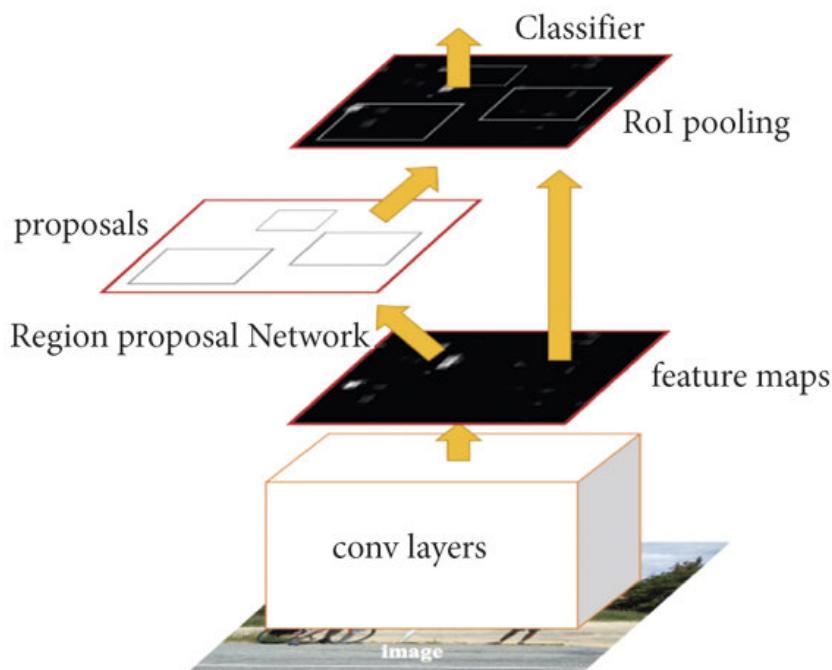
Fast R-CNN khác với R-CNN là nó thực hiện feature map với cả ảnh sau đó với lấy các region proposal ra từ feature map, còn R-CNN thực hiện tách các region proposal ra rồi mới thực hiện CNN trên từng region proposal. Do đó Fast R-CNN nhanh hơn đáng kể nhờ tối ưu việc tính toán bằng Vectorization.

Tuy nhiên nhìn hình trên ở phần test time với mục Fast R-CNN thì thời gian tính region proposal rất lâu và làm chậm thuật toán \Rightarrow Cần thay thế thuật toán selective search. Giờ người ta nghĩ đến việc dùng deep learning để tạo ra region proposal \Rightarrow Faster R-CNN ra đời.

2.3.4 Faster R-CNN

Giới thiệu

Faster R-CNN không dùng thuật toán selective search để lấy ra các region proposal, mà nó thêm một mạng CNN mới gọi là Region Proposal Network (RPN) để tìm các region proposal.

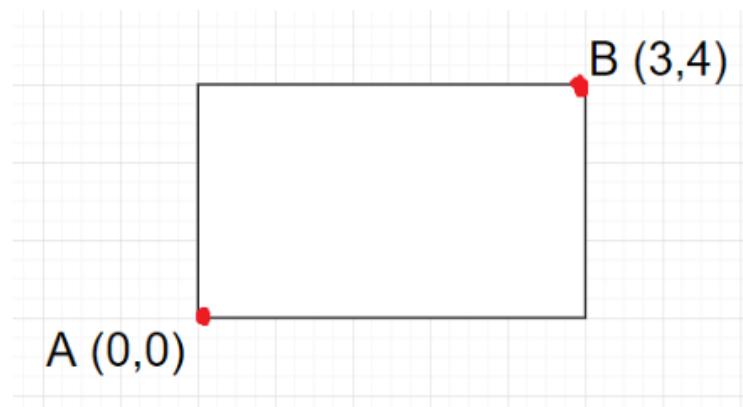


Hình 2.41: Kiến trúc mới Faster R-CNN

Dầu tiên cả bức ảnh được cho qua pre-trained model để lấy feature map. Sau đó feature map được dùng cho Region Proposal Network để lấy được các region proposal. Sau khi lấy được vị trí các region proposal thì thực hiện tương tự Fast R-CNN.

Region Proposal Network (RPN)

Input của RPN là feature map và output là các region proposal. Ta thấy các region proposal là hình chữ nhật.



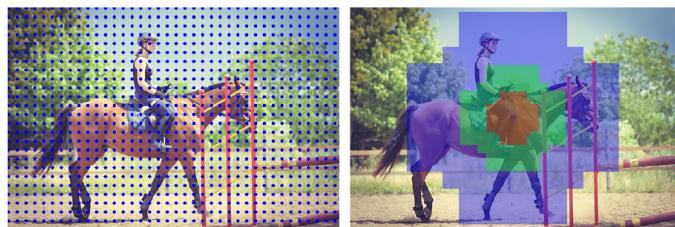
Hình 2.42: Mô tả các region proposal trong output của RPN

Mà một hình chữ nhật được xác định bằng 2 điểm ở 2 góc, ví dụ $A(x_{min}, y_{min})$ và $B(x_{max}, y_{max})$. Nhận xét:

- Khi RPN dự đoán ta phải rồng buộc $x_{min} < x_{max}$ và $y_{min} < y_{max}$.
- Hơn nữa các giá trị x,y khi dự đoán có thể ra ngoài khói bức ảnh.
⇒ Cần một kĩ thuật mới để biểu diễn region proposal ⇒ Anchor ra đời.

Ý tưởng là thay vì dự đoán 2 góc ta sẽ dự đoán điểm trung tâm (x_{center}, y_{center}) và width, height của hình chữ nhật. Như vậy mỗi anchor được xác định bằng 4 tham số ($x_{center}, y_{center}, width, height$).

Vì không sử dụng Selective search nên RPN ban đầu cần xác định các anchor box có thể là region proposal, sau đó qua RPN thì chỉ output những anchor box chắc chắn chứa đối tượng.



Hình 2.43: Ví dụ về anchor

Ảnh bên trái kích thước 400×600 pixel, tác tâm của anchor box màu xanh, cách nhau 16 pixel \Rightarrow có khoảng $\frac{400 \times 600}{16 \times 16} = 938$ tâm. Do các object trong ảnh có thể có kích thước và tỉ lệ khác nhau nên với mỗi tâm ta định nghĩa 9 anchors với kích thước $64 \times 64, 128 \times 128, 256 \times 256$, mỗi kích thước có 3 tỉ lệ tương ứng: 11, 12 và 21.

Giống như hình bên phải với tâm ở giữa 3 kích thước ứng với màu da cam, xanh lam, xanh lục và với mỗi kích thước có 3 tỉ lệ.

\Rightarrow Số lượng anchor box giờ là $938 * 9 = 8442$ anchors. Tuy nhiên sau RPN ta chỉ giữ lại khoảng 1000 anchors box để thực hiện như trong Fast R-CNN. \Rightarrow Việc của RPN là lấy ra các region proposal giống như selective search thôi chứ không phải là phân loại ảnh.

Mô hình RPN khá đơn giản, feature map được cho qua Conv layer 3×3 , 512 kernels. Sau đó, với mỗi anchor lấy được ở trên, RPN thực hiện 2 bước:

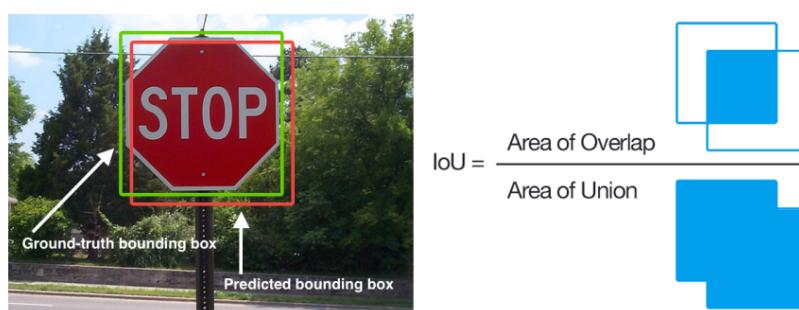
1. Dự đoán xem anchor đây là foreground (chứa object) hay background (không chứa object)
2. Dự đoán 4 offset value cho $x_{center}, y_{center}, width, height$ cho các anchor.

Nhận xét: có rất nhiều anchor bị chồng lên nhau nên non-maxima suppression được dùng để loại bỏ các anchor chồng lên nhau.

Sau cùng dựa vào phần trăm dự đoán background RPN sẽ lấy N anchor (N có thể 2000, 1000, thậm chí 100 vẫn chạy tốt) để làm region proposal.

Intersection over Union (IoU)

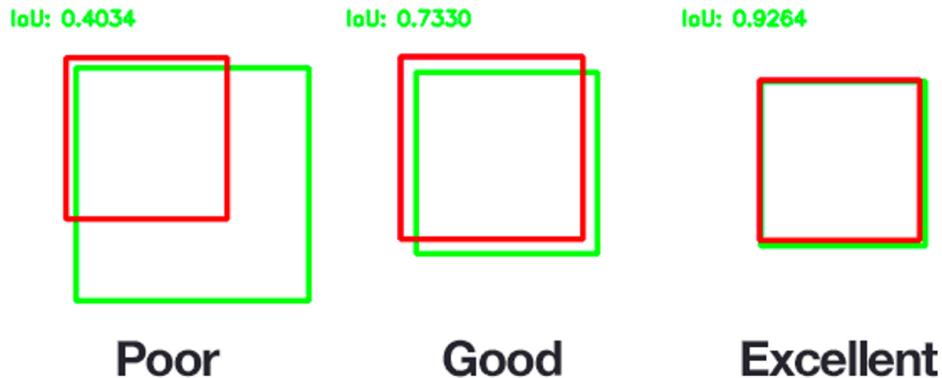
IoU được sử dụng trong bài toán object detection, để đánh giá xem bounding box dự đoán đối tượng khớp với ground truth thật của đối tượng.



Hình 2.44: Ví dụ về IoU

Ví dụ về hệ số IoU, nhận xét:

- Chỉ số IoU trong khoảng [0,1]
- IoU càng gần 1 thì bounding box dự đoán càng gần ground truth



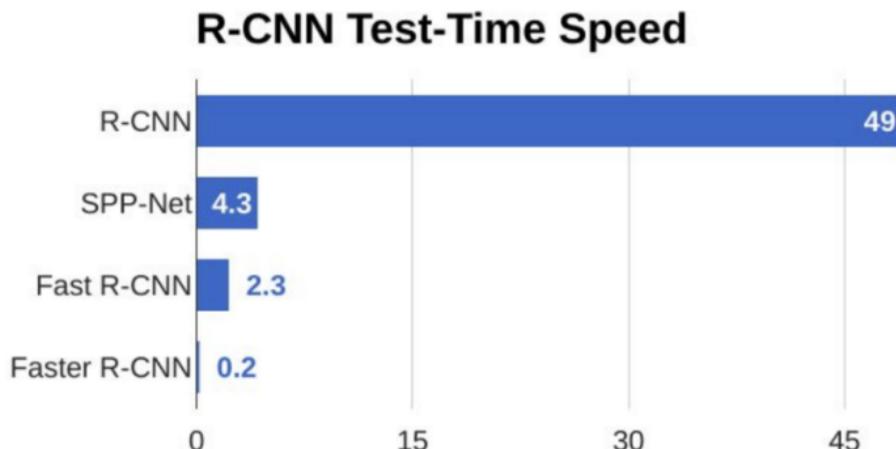
Hình 2.45: Chỉ số IoU

Non-maxima suppression

Ở trong Region Proposal Network đầu tiên ta có khoảng 9000 anchor box (tập Input) tuy nhiên ta chỉ muốn giữ lại 100 anchor (tập Ouput) làm region proposal. Ta sẽ làm như sau:

- Bước 1: Chọn ra anchor box (A) có xác xuất là foreground lớn nhất trong tập Input
- Bước 2: Thêm A vào tập Ouput.
- Bước 3: Loại bỏ A và các anchor box trong tập Input mà có hệ số IoU với A lớn hơn 0.5 ra khỏi tập Input.
- Bước 4: Kiểm tra nếu tập Input rỗng hoặc tập Output đủ 100 anchor thì dừng lại, nếu không quay lại bước 1.

Kết quả của Faster R-CNN



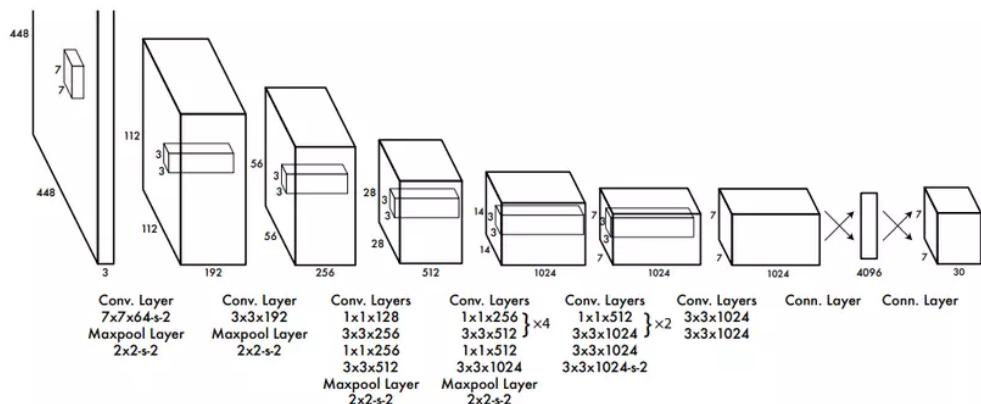
Hình 2.46: So sánh tốc độ test-time của các thuật toán object detection

Nhìn ở hình trên ta thấy Faster R-CNN nhanh hơn hẳn các dòng R-CNN trước đó, vì vậy có thể dùng cho real-time objec detection.

2.3.5 YOLO Series

Giới thiệu

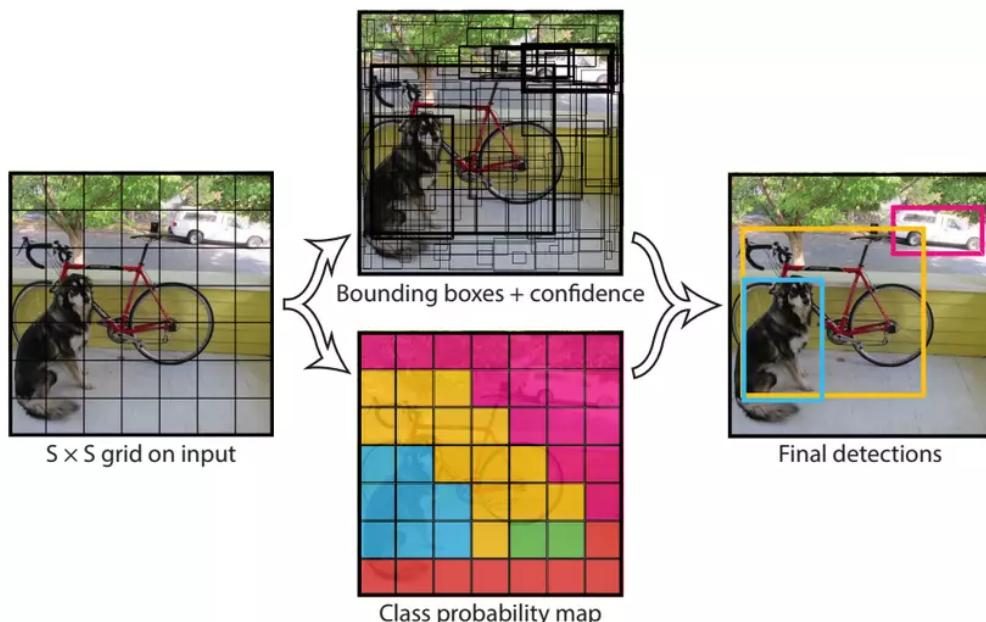
Yolo là một mô hình mạng CNN cho việc phát hiện, nhận dạng, phân loại đối tượng real-time. Yolo được tạo ra từ việc kết hợp giữa các convolutional layers và connected layers. Trong đó, các convolutional layers sẽ trích xuất ra các feature của ảnh, còn full-connected layers sẽ dự đoán ra xác suất đó và tọa độ của đối tượng.



Hình 2.47: Kiến trúc mạng YOLO

Cách hoạt động

Đầu vào của mô hình 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. Ảnh đầu vào được chia thành $S \times S$ ô thường thì sẽ là 3×3 , 7×7 , 9×9 ,... việc chia ô này có ảnh hưởng tới việc mô hình phát hiện đối tượng, điều này sẽ được trình bày ở phần sau.



Hình 2.48: Minh họa cách YOLO hoạt động

Với Input là 1 ảnh, đầu ra mô hình là một ma trận 3 chiều có kích thước

$$S \times S \times (5 \times N + M) \quad (2.38)$$

với số lượng tham số mỗi ô là $(5 \times N + M)$ với N và M lần lượt là số lượng Box và Class mà mỗi ô cần dự đoán. Ví dụ với hình ảnh trên chia thành 7×7 ô, mỗi ô cần dự đoán 2 bounding box và 3 object : con chó, ô tô, xe đạp thì output là $7 \times 7 \times 13$, mỗi ô sẽ có 13 tham số, kết quả trả về ($7 \times 7 \times 2 = 98$) bounding box. Chúng ta sẽ cùng giải thích con số $(5 \times N + M)$ được tính như thế nào.

Dự đoán mỗi bounding box gồm 5 thành phần : $(x, y, w, h, prediction)$ với (x, y) là tọa độ tâm của *boundingbox*, (w, h) lần lượt là chiều rộng và chiều cao của bounding box, prediction được định nghĩa $Pr(Object) * IOU(pred, truth)$. Với hình ảnh trên như ta tính mỗi ô sẽ có 13 tham số, ta có thể hiểu đơn giản như sau tham số thứ 1 sẽ chỉ ra ô đó có chứa đối tượng nào hay không $P(Object)$, tham số 2, 3, 4, 5 sẽ trả về x, y, w, h của Box1. Tham số 6, 7, 8, 9, 10 tương tự sẽ Box2, tham số 11, 12, 13 lần lượt là xác suất ô đó có chứa $object1(P(\text{chó}|object), object2(P(\text{ô tô}|object)), object3(P(\text{xe đạp}|object))$.

Lưu ý rằng tâm của bounding box nằm ở ô nào thì ô đó sẽ chứa đối tượng, cho dù đối tượng có thể ở các ô khác thì cũng sẽ trả về là 0. Vì vậy việc mà 1 ô chứa 2 hay nhiều tâm của bounding box hay đối tượng thì sẽ không thể detect được, đó là một hạn chế của mô hình YOLO1, vậy ta cần phải tăng số lượng ô chia trong 1 ảnh lên đó là lí do vì sao mình nói việc chia ô có thể làm ảnh hưởng tới việc mô hình phát hiện đối tượng.

Loss function

Hàm lỗi trong YOLO được tính trên việc dự đoán và nhãn mô hình để tính. Cụ thể hơn nó là tổng độ lỗi của 3 thành phần sau :

- Độ lỗi của việc dự đoán loại nhãn của object - Classification loss.
- Độ lỗi của dự đoán tọa độ tâm, chiều dài, rộng của boundary box (x, y, w, h) - Localization loss.
- Độ lỗi của việc dự đoán bounding box đó chứa object so với nhãn thực tế tại ô vuông đó - Confidence loss.

a) Classification loss

Classification loss - độ lỗi của việc dự đoán loại nhãn của object, hàm lỗi này chỉ tính trên những ô vuông có xuất hiện object, còn những ô vuông khác ta không quan tâm. Classification loss được tính bằng công thức sau:

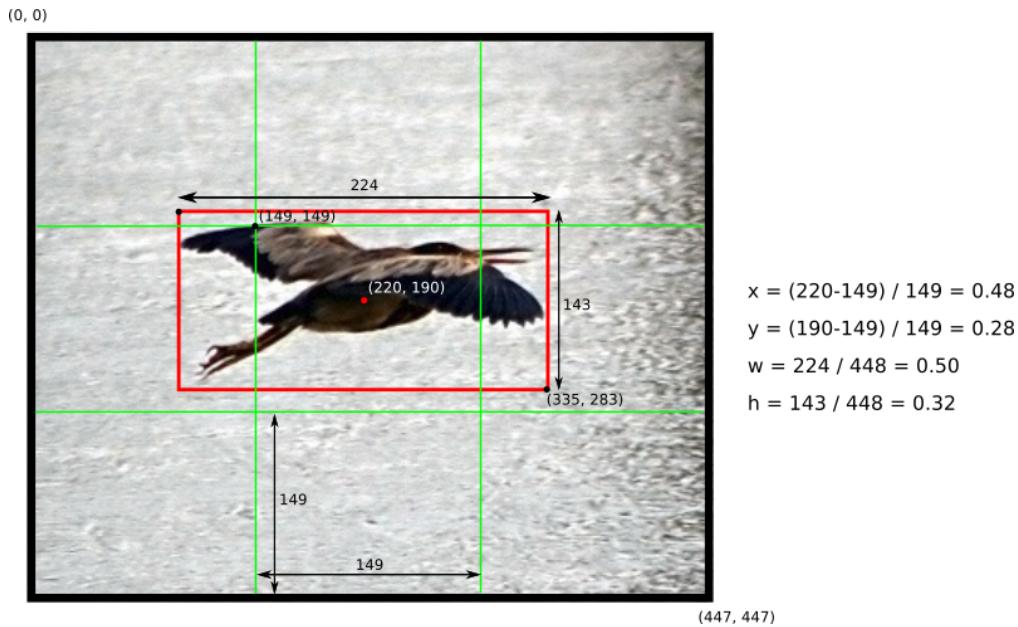
$$L_{classification} = \sum_{i=0}^{S^2} 1_i^{obj} \sum_{c \in class} (p_i(c) - \hat{p}_i(c))^2 \quad (2.39)$$

Trong đó:

- 1_i^{obj} : bằng 1 nếu ô vuông đang xét có object, ngược lại thì bằng 0,
- $\hat{p}_i(c)$: là xác suất có điều kiện của lớp c tại ô vuông tương ứng mô hình dự đoán.

b) Localization loss

Localization loss là hàm lỗi dùng để tính giá trị lỗi cho boundary box được dự đoán bao gồm tọa độ tâm, chiều rộng, chiều cao của so với vị trí thực tế từ dữ liệu huấn luyện của mô hình. Lưu ý rằng chúng ta không nên tính giá trị hàm lỗi này trực tiếp từ kích thước ảnh thực tế mà cần phải chuẩn hóa về $[0, 1]$ so với tâm của bounding box. Việc chuẩn hóa này kích thước này giúp cho mô hình dự đoán nhanh hơn và chính xác hơn so với để giá trị mặc định của ảnh. Hãy cùng xem một ví dụ:



Hình 2.49: Ví dụ về boundary box

Giá trị hàm Localization loss ($L_{localization}$) được tính trên tổng giá trị lỗi dự đoán toạ độ tâm (x, y) và (w, h) của predicted bounding box với ground-truth bounding box. Tại mỗi ô có chứa object, ta chọn 1 boundary box có IOU (Intersect over union) tốt nhất, rồi sau đó tính độ lỗi theo các boundary box này.

Giá trị hàm lỗi dự đoán tọa độ tâm (x, y) của predicted bounding box và (\hat{x}, \hat{y}) là tọa độ tâm của truth bounding box được tính như sau :

$$\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^B 1_{ij}^{obj} (x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \quad (2.40)$$

Giá trị hàm lỗi dự đoán (w, h) của predicted bounding box so với truth bounding box được tính như sau:

$$\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^B 1_{ij}^{obj} \left(\sqrt{w_i} - \sqrt{\widehat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\widehat{h}_i} \right)^2 \quad (2.41)$$

Trong đó:

- 1_{ij}^{obj} là hàm indicator có giá trị 0, 1, $1_{ij}^{obj} = 1$ nếu cell i chứa object và $1_{ij}^{obj} = 0$ nếu không chứa object.

Với ví dụ trên thì $S = 7$, $B = 2$, còn λ_{coord} là trọng số thành phần trong paper gốc tác giả lấy giá trị là 5

c) Confidence loss

Confidence loss là độ lỗi giữa dự đoán boundary box đó chứa object so với nhãn thực tế tại ô vuông đó. Độ lỗi này tính trên cả những ô vuông chứa object và không chứa object.

$$L_{confidence} = \sum_{i=0}^{S^2} \sum_{j=0}^B I_{ij}^{obj} \left(C_i - \widehat{C}_i \right)^2 + \lambda_{noobject} \sum_{i=0}^{S^2} \sum_{j=0}^B I_{ij}^{noobj} \left(C_i - \widehat{C}_i \right)^2 \quad (2.42)$$

Với ví dụ trên thì $S = 7$, $B = 2$, còn noobject là trọng số thành phần trong paper gốc tác giả lấy giá trị là 0.5. Đối với các hộp j của ô thứ i nếu xuất hiện object thì $C_j = 1$ và ngược lại.



d) Total loss Tổng lại chúng ta có hàm lỗi là tổng của 3 hàm lỗi trên:

$$L_{total} = L_{classification} + L_{localization} + L_{confidence} \quad (2.43)$$

Chương 3

ĐỀ XUẤT GIẢI PHÁP

3.1 Nhắc lại nhiệm vụ của đề tài

Trong đề tài này, Robot có 2 nhiệm vụ chính là:

1. Phát hiện và phân loại sản phẩm.
2. Tìm đường và di chuyển đến các khu vực.

Vậy để có thể hiện thực được 1 Robot thực hiện được các nhiệm vụ đó thì em cần giải quyết được các vấn đề sau:

1. Làm sao để Robot có thể nhận dạng được sản phẩm?
2. Làm sao để Robot có thể biết được vị trí các khu vực và đường đi đến các khu vực này?

3.2 Các giải pháp

3.2.1 Giải pháp phát hiện và phân loại sản phẩm

Để có thể phát hiện và phân loại sản phẩm, em sẽ áp dụng công nghệ Computer Vision cho mục đích Object Detection.

Cụ thể, em sẽ lựa chọn và xây dựng một mạng neural nhân tạo theo các kiến trúc thuộc YOLO series đã được nêu ở trên. Đây là một kiến trúc rất phù hợp cho nhiệm vụ này vì ưu điểm của nó là tốc độ phát hiện và phân loại vật thể rất nhanh, phát hiện nhiều vật thể cùng lúc và nó còn có thể cung cấp vị trí của các vật thể trên khung hình camera.

Với tốc độ xử lý nhanh và phân loại được nhiều vật thể cùng lúc thì đây là mô hình phù hợp cho việc phân loại thời gian thực theo video được ghi bằng camera. Ngoài ra, thông tin vị trí và tên của vật thể là những thông tin cực kỳ quan trọng để giúp Robot giải quyết được vấn đề *Tìm đường và di chuyển đến các khu vực*.

Tuy nhiên, giải pháp này cũng có một số hạn chế nhất định như:

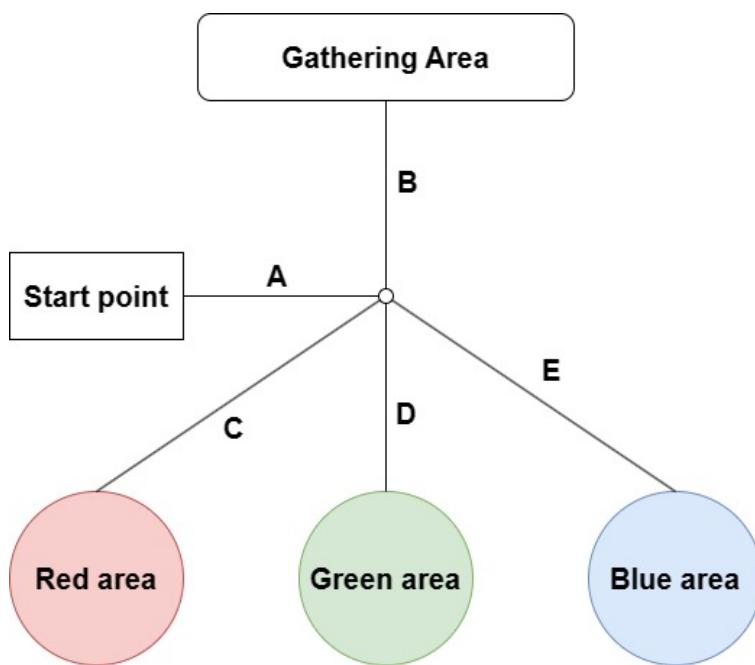
1. Xây dựng mô hình rất phức tạp và cần một chip xử lý đủ mạnh để thực hiện rất nhiều phép tính trong khoảng thời gian ngắn theo tỉ lệ FPS của camera.
2. Cần một bộ dữ liệu huấn luyện lớn bao gồm các hình ảnh được gắn nhãn. Đây là vấn đề cần rất nhiều thời gian để chuẩn bị.

3.2.2 Giải pháp tìm vị trí các khu vực và đường đi

Dựa vào miêu tả test case tại phần 1.2, ta có thể thấy Robot sẽ di chuyển trên các nhóm tuyến đường chính sau:

- **Nhóm 1:** Tuyến đường từ *Start point* đến *Gathering Area* và ngược lại.
- **Nhóm 2:** Tuyến đường từ *Gathering Area* đến các khu vực phân loại và từ các khu vực phân loại đến *Gathering Area*.
- **Nhóm 3:** Tuyến đường từ các *Khu vực phân loại* đến *Start point*. Nhóm này không có tuyến đường từ *Start point* đến các khu vực phân loại vì tuyến đường này không cần thiết và không xảy ra.

Nếu đặt tên cho các tuyến đường đi của Robot như hình bên dưới.



Hình 3.1: Đặt tên các tuyến đường giữa các khu vực

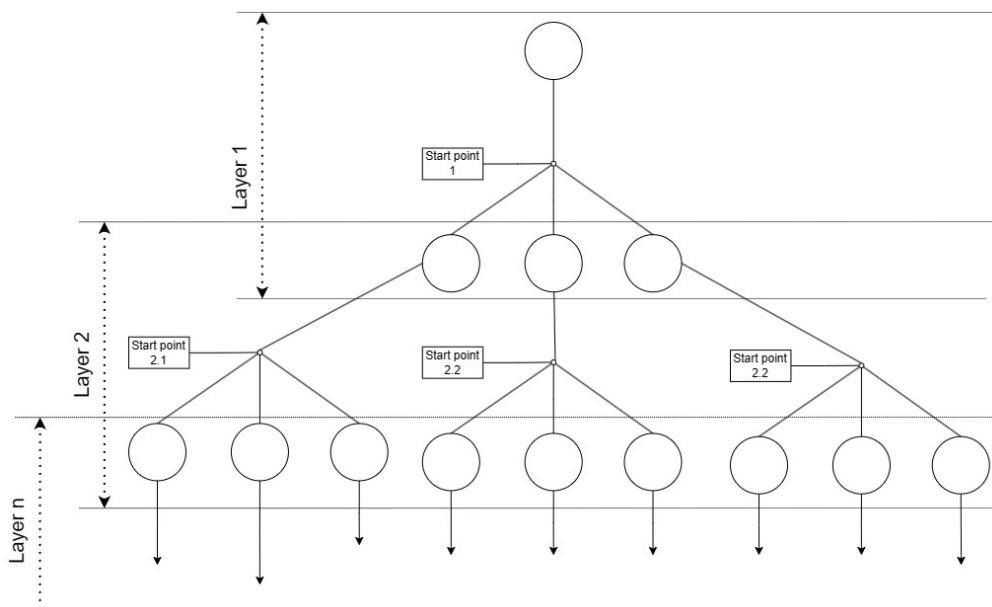
- Nhóm 1 bao gồm:
 - $A \rightarrow B$ và $B \rightarrow A$
- Nhóm 2 bao gồm:
 - $B \rightarrow C$ và $C \rightarrow B$
 - $B \rightarrow D$ và $D \rightarrow B$
 - $B \rightarrow E$ và $E \rightarrow B$
- Nhóm 3 bao gồm:
 - $C \rightarrow A$
 - $D \rightarrow A$
 - $E \rightarrow A$

Như vậy, để Robot có thể nhận biết được rằng mình đang trên đoạn đường nào thì Robot cần phát hiện được đặt trưng của đoạn đường đó.

Giải pháp của em là đối với mỗi đoạn đường, em sẽ sử dụng một màu sắc khác nhau hoàn toàn với các đoạn đường còn lại để vẽ nối. Về Robot, em sẽ sử dụng một cảm biến màu sắc được đặt dây của Robot để phát hiện sự thay đổi màu sắc trên các đoạn đường.

Nhưng giải pháp này có 1 hạn chế là số lượng màu sắc có thể sử dụng là giới hạn. Lý do là vì mỗi màu sắc khi được cảm biến nhận vào sẽ được chuyển thành hệ màu RGB với tổ hợp 3 giá trị màu là (r, g, b) . Tuy nhiên giá trị của mỗi thành phần lại chỉ có 256 giá trị ($r \in [0, 255]$, $g \in [0, 255]$ và $b \in [0, 255]$). Ngoài ra, việc nhận biết các màu có tổ hợp quá gần nhau như $(212, 10, 10)$ và $(213, 10, 10)$ là vô cùng khó khăn và dễ gây ra sai sót trong quá trình phân loại.

Để giải quyết được vấn đề này, giải pháp của em là một hệ thống phân loại nhiều tầng. Cụ thể sẽ như hình vẽ sau:



Hình 3.2: Mô hình phân loại nhiều tầng

Mô hình này sẽ phân 1 mặt hàng thành nhiều tầng khác nhau để phân loại. Ví dụ như đối với mặt hàng là điện thoại thì sẽ có:

- Samsung:
 - Galaxy Note
 - Galaxy S
 - Galaxy Z
- Iphone:
 - Iphone 5
 - * Pro Max
 - * Plus
 - Iphone 6
 - * Pro Max
 - * Plus
 - Iphone 7



* Pro Max

* Plus

Üng với mỗi *Start point* thì sẽ có ít nhất 1 Robot chịu trách nhiệm phân loại tầng sản phẩm đó.

Tuy sẽ không giải quyết triệt để hạn chế đã nêu trên nhưng với mô hình phân loại đa lớp này thì gần như có thể phân loại 1 lượng lớn loại sản phẩm trên thị trường.

Chương 4

KẾ HOẠCH CHO GIA ĐOẠN 2

Với lý do Giai đoạn 2 không thể thực hiện ở kỳ hè 233 mà chỉ thực hiện được ở kỳ 241 nên thời gian cho Giai đoạn 2 của em sẽ kéo dài hơn 1 kỳ. Do đó, với khoảng thời gian hoàn thành khá lớn, kế hoạch thực hiện đề tài của em được chia thành 4 phần bao gồm: Chuẩn bị khu vực test, Hiện thực phần cứng, Hiện thực phần mềm và Viết báo cáo.

Cụ thể các nhiệm vụ trong mỗi phần như sau:

1. Chuẩn bị khu vực test

Tên nhiệm vụ	Yêu cầu đầu ra	Ước tính thời gian thực hiện
Vẽ các khu vực được miêu tả trong đề tài	<ul style="list-style-type: none">Khu vực test có khả năng tháo lắp dễ dàng vận chuyểnCác khu vực được vẽ rõ ràng	5
Vẽ các đường nối các khu vực	<ul style="list-style-type: none">Các đường nối được vẽ bởi các màu khác nhau rõ rệtCác đường vẽ phải rõ ràng và nổi bật so với sàn	3
Chuẩn bị các khối vuông nhiều màu	<ul style="list-style-type: none">Tối thiểu 5 khối vuông.Mỗi khối vuông chỉ có 1 màu cho cả 6 mặt và màu đó nằm trong tập hợp 3 màu Đỏ, Vàng, XanhKích thước của khối vuông không được quá 4cm	3

Bảng 4.1: Các nhiệm vụ cho phần Chuẩn bị khu vực test

2. Hiện thực phần cứng



Tên nhiệm vụ	Yêu cầu đầu ra	Ước tính thời gian thực hiện
Xác định phạm vi hoạt động của các bộ phận của Robot.	Xác định được: <ul style="list-style-type: none">Tốc độ của các motor.Góc quay cần thiết cho các servo.Chiều dài min và max của cánh tay robot.	5
Lựa chọn và liệt kê các tất cả linh kiện phù hợp với Robot	Lựa chọn và liệt kê được tối thiểu 80% số linh kiện cần thiết để hiện thực Robot	3
Mua các linh kiện	Mua được đầy đủ các linh kiện đã được liệt kê	2
Kiểm tra hoạt động của các linh kiện	Đảm bảo các linh kiện hoạt động ổn định và khớp với thông tin được ghi trên datasheet của linh kiện. Nếu không phải điều chỉnh hoặc mua lại linh kiện bị hỏng	2
Thiết kế sơ đồ mạch điện và sơ đồ liên kết các linh kiện	<ul style="list-style-type: none">Sơ đồ hoạt động ổn định trên môi trường mô phỏng.Nguồn điện cung cấp đủ điện năng cho toàn bộ hệ thống.	7
Thiết kế mô hình 3D cho Robot	Mô hình phải đầy đủ các bộ phận của Robot và sát 70% so với thực tế	7
Thực hiện video mô phỏng hoạt động của Robot	<ul style="list-style-type: none">Video rõ ràng, độ dài không quá 10 phút.Video mô tả đầy đủ quy trình phân loại của Robot từ lúc bắt đầu đến khi phân loại hết hàng hóa.	8
Hiện thực khung Robot	<ul style="list-style-type: none">Khung Robot chắc chắn, chịu tải tốt.Không rung lắc khi di chuyển.Robot vẫn giữ được thăng bằng và ổn định khi di chuyển có kèm sản phẩm cần phân loại.	14



Lắp ráp và hoàn chỉnh phần cứng của Robot	<ul style="list-style-type: none">Robot được lắp ráp hoàn chỉnh, đầy đủ các bộ phận đã được thiết kế.Các linh kiện được kết nối chắc chắn.Các mạch điện được hàn chắc chắn, đảm bảo các đường dây gọn gàng, không chồng chéo, có lớp cách điện hợp lý.	14
Kiểm thử và chỉnh sửa	<ul style="list-style-type: none">Đảm bảo các linh kiện vẫn hoạt động bình thường.Tín hiệu từ board điều khiển ổn định.Chỉnh sửa nếu cần thiết.	7

Bảng 4.2: Các nhiệm vụ cho phần Hiện thực phần cứng

3. Hiện thực phần mềm

Tên nhiệm vụ	Yêu cầu đầu ra	Ước tính thời gian thực hiện
Chuẩn bị data set cho việc training model AI	<ul style="list-style-type: none">Bộ data bao gồm ít nhất 300 ảnh cho mỗi loại sản phẩm, tổng cộng là 1000 ảnh.Các ảnh phải có ít nhất 9 góc nhìn khác nhau của vật thể.Kích thước mỗi ảnh phải khớp với khung hình của Camera.	10
Xây dựng model AI	Model hoàn thiện và chạy ổn định	16
Training model và kiểm thử model bằng webcam laptop hoặc điện thoại	Model cần nhận diện được vật thể nhanh chóng và có thể hoạt động ổn định với tốc độ khung hình là 20fps	4



Tạo dự án trên Visual Studio và setup dự án	<ul style="list-style-type: none">Dự án được tạo thành công.Hoàn thành các setup ban đầu.Tạo các cây thư mục cần thiết.Chạy test môi trường ổn định.Thêm README.md hướng dẫn chạy dự án.Upload dự án lên Github ở chế độ public.	3
Khai báo các chân cần sử dụng trên board	Các chân được khai báo đầy đủ trong 1 file riêng để tiện sửa chữa và cập nhật	1
Thiết kế các sơ đồ giao tiếp giữa các component	<ul style="list-style-type: none">Các component được thiết kế rõ ràng về mục đích, đầu ra, đầu vào, cách hoạt động.Các component liên kết hợp lý và logic.	5
Hiện thực các component	<ul style="list-style-type: none">Các component được hiện thực chính xác theo phần thiết kế.Nếu có lỗi xảy ra, phải sửa lỗi đó trên sơ đồ thiết trước. Sau đó mới sửa ở phần code	21
Tích hợp model AI	<ul style="list-style-type: none">Tích hợp thành công đoạn mã AI vào hệ thống.Model chạy ổn định, phân loại chính xác và không xảy ra lỗi.	7
Nạp code vào board điều khiển và kiểm thử	Nạp code vào board thành công	5
Chạy thử trên môi trường chính và sửa lỗi	<ul style="list-style-type: none">Chạy thử trên môi trường test đã được thiết kế.Sửa lỗi nếu có.	14

Bảng 4.3: Các nhiệm vụ cho phần Hiện thực phần mềm



4. Viết báo cáo

Tên nhiệm vụ	Yêu cầu đầu ra	Ước tính thời gian thực hiện
Viết báo cáo	<ul style="list-style-type: none">• Hoàn thành nội dung báo cáo và gửi cho GVHD để đánh giá.• Nhận đánh giá, sửa lỗi và tiếp tục hoàn thiện báo cáo.	7

Bảng 4.4: Các nhiệm vụ cho phần Chuẩn bị khu vực test

Tổng thời gian thực hiện đồ án là 210 ngày, tương đương với 31 tuần.

TÀI LIỆU THAM KHẢO

- [1] P. Baheti. Activation functions in neural networks [12 types & use cases], 5 2021. Accessed: 2024-05-10.
- [2] V. S. Chadel. Selective search for object detection (c++ / python), 9 2017. Accessed: 2024-05-11.
- [3] cs231. Convolutional neural networks (cnns / convnets), 2017. Accessed: 2024-05-11.
- [4] B. Earnshaw. A brief survey of tensors, 2017. Accessed: 2024-05-11.
- [5] G. for Geeks. Types of activation function in ann, 1 2021. Accessed: 2024-05-10.
- [6] A. Ghosh, F. Sultana, A. Sufia, and A. Chakrabarti. *Fundamental Concepts of Convolutional*. India, 2020.
- [7] C. Heer. Ifr presents world robotics report 2020, 9 2020. Accessed: 2024-02-27.
- [8] V. H. Hiệp. *Machine Learning cơ bản*. 2020.
- [9] M. Intelligence. Robotics industry size & share analysis - growth trends & forecasts (2024 - 2029), 2024. Accessed: 2024-02-27.
- [10] Koen. Segmentation as selective search for object recognition, 2007. Accessed: 2024-05-11.
- [11] J. Lalin. Feedforward neural networks in depth, part 1: Forward and backward propagations, 12 2021. Accessed: 2024-05-11.
- [12] L. V. Nam. Logistics việt nam, 7 2022. Accessed: 2024-02-27.
- [13] T. D. J. M. U. B. I. Ross Girshick, Jeff Donahue. Rich feature hierarchies for accurate object detection and semantic segmentation. *IEEE Explore*, 2013.
- [14] P. H. Toàn. [handbook cv with dl - phần 1] các khái niệm cơ bản trong computer vision và deep learning, 2 2019. Accessed: 2024-05-10.
- [15] N. T. Tuấn. *Deep Learning cơ bản*. 2020.
- [16] Wikipedia. Kernel (image processing), 10 2023. Accessed: 2024-05-05.