TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI
PHÂN HIỆU TẠI TP. HỒ CHÍ MINH
BỘ MÔN CÔNG NGHỆ THÔNG TIN

BÁO CÁO ĐỒ ÁN TỐT NGHIỆP

ĐỀ TÀI: NGHIÊN CỨU TRIỂN KHAI THUẬT TOÁN YOLOV3 VÀ XÂY DỰNG HỆ THỐNG HƯỚNG DẪN ĐẬU XE Ô TÔ PARKING VISION

Giảng viên hướng dẫn: ThS. PHẠM THỊ MIÊN

Sinh viên thực hiện : NGUYỄN VĂN HIỆP

Mã sinh viên : 5851071024

Lớp : CÔNG NGHỆ THÔNG TIN

Khoá : 58

TP. Hồ Chí Minh, tháng 06 năm 2021

TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI
PHÂN HIỆU TẠI TP. HỒ CHÍ MINH
BỘ MÔN CÔNG NGHỆ THÔNG TIN

BÁO CÁO ĐỒ ÁN TỐT NGHIỆP

ĐỀ TÀI: NGHIÊN CỨU TRIỂN KHAI THUẬT TOÁN YOLOV3 VÀ XÂY DỰNG HỆ THỐNG HƯỚNG DẪN ĐẬU XE Ô TÔ PARKING VISION

Giảng viên hướng dẫn: ThS. PHẠM THỊ MIÊN

Sinh viên thực hiện : NGUYỄN VĂN HIỆP

Mã sinh viên : 5851071024

Lớp : CÔNG NGHỆ THÔNG TIN

Khoá : 58

TP. Hồ Chí Minh, tháng 06 năm 2021

TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI
PHÂN HIỆU TẠI THÀNH PHỐ HỒ CHÍ MINH

CỘNG HÒA XÃ HỘI CHỦ NGHIÃ VIỆT NAM
Độc lập – Tự do – Hạnh phúc

NHIỆM VỤ THIẾT KẾ TỐT NGHIỆP

BỘ MÔN: **CÔNG NGHỆ THÔNG TIN**

-------***-------

Mã sinh viên: 5851071024 **Họ tên SV:** Nguyễn Văn Hiệp

Khóa: 58 **Lớp:** Công Nghệ Thông Tin

1. Tên đề tài đồ án tốt nghiệp:

NGHIÊN CỨU TRIỂN KHAI THUẬT TOÁN YOLOV3 VÀ XÂY DỰNG HỆ THỐNG HƯỚNG DẪN ĐẬU XE Ô TÔ PARKING VISION.

2. Mục đích, yêu cầu:

a. Mục đích:

- Xây dựng website theo dõi các vị trí còn trống và đã đỗ trong bãi đỗ xe và hiển thị lên màn hình số vị trí còn trống, đã đỗ và tổng số lượng chỗ đỗ trong bãi đỗ xe ô tô theo thời gian thực.
- Xây dựng mô hình phát hiện các chỗ còn trống và đã đỗ trong bãi đỗ xe bằng hình ảnh, video, camera.
- Xây dựng website giới thiệu sản phẩm.
- Xây dựng giải pháp cho Hệ Thống Hướng Dẫn Bãi Đỗ Xe (Parking Guidance System) thông minh tiết kiệm, nhanh hơn, tiện dụng, chính xác và có thể mở rộng quy mô sau này.

b. Yêu cầu:

- Tìm hiểu về Thị Giác Máy Tính (Computer Vision) và Học Sâu (Deep Learning).
- Nghiên cứu về xử lý ảnh.
- Nghiên cứu những quy trình trong xử lý ảnh.

- Nghiên cứu về Mạng Neural Tích Chập (Convolution Neural Network) và những ứng dụng của nó trong Deep Learning.
- Nghiên cứu thuật toán YOLOv3 (You Only Look Once Version 3).
- Tìm hiểu một số khái niệm liên quan đến lĩnh lực thống kê học áp dụng vào mô hình Học Sâu.
- Tìm hiểu về thuật toán hồi quy logistic.
- Tìm hiểu về Gradient Descent.
- Tìm hiểu về Batch normalization.
- Tìm hiểu nguồn cơ sở dữ liệu hình ảnh để huấn luyện mô hình Học Sâu.
- Thu thập dữ liệu hình ảnh về những chỗ trống và đã đỗ trong bãi đỗ xe. Gắn nhãn, tiền xử lý.
- Tìm hiểu về Transfer Learning, và ứng dụng vào huấn luyện mô hình.
- Ứng dụng kiến trúc Darknet huấn luyện mô hình trên tập dữ liệu lớn và tập dữ liệu mô phỏng bằng công cụ Google Colab.
- Ứng dụng thuật toán YOLOv3 (You Only Look Once version 3) để phát hiện những vị trí còn trống, đã đỗ trên hình ảnh, video, camera bằng Pytorch.
- Ứng dụng Nicepage thiết kế website giới thiệu sản phẩm và sử dụng Django để xử lý trang web, truyền hình ảnh đã xử lý lên website.

3. Nội dung và phạm vi đề tài:

a. Nội dung đề tài:

- Giới thiệu và phân biệt các khái niệm liên quan đến Trí Tuệ Nhân Tạo (Artificial Intelligence), Thị Giác Máy Tính (Computer Vision), Học Máy (Machine Learning), Học Sâu (Deep Learning).
- Nghiên cứu và triển khai thuật toán YOLOv3 bằng Pytorch:
    + Triển khai mô-đun phát hiện qua hình ảnh.
    + Triển khai mô-đun phát hiện qua video/camera.
- Kiểm thử mô hình.
- Nghiên cứu các chỉ số đánh giá mô hình.
- Xây dựng trang web hiển thị màn hình xử lý bằng Django.
- Hiển thị song song màn hình xử lý trên website và trên desktop.

b. Phạm vi đề tài:

- Bài toán phát hiện đối tượng (Object Detection).
- Thuật toán YOLOv3.
- Ứng dụng Django để xây dựng website giới thiệu sản phẩm và sử dụng Django để xử lý trang web, truyền hình ảnh đã xử lý lên website.

4. Công nghệ, công cụ và ngôn ngữ lập trình:

a. Công nghệ:

Python, OpenCV, Pytorch, Django, Colab Notebook, CUDA, Darknet.

b. Công cụ:

+ Một số thư viện mã nguồn mở của Python:

Opencv-python, pandas, numpy, django, torch,...

+ Visual Studio Code
+ Darknet: Open Source Neural Networks
+ Google Colab

c. Ngôn ngữ lập trình: Python

5. Các kết quả chính dự kiến sẽ đạt được và ứng dụng:

- Sử dụng camera tiến hành phát hiện các vị trí còn trống và đã đỗ trong thời gian thực. Hiển thị lên màn hình vị trí còn trống và đã đỗ, đếm những vị trí còn trống, đã đỗ, tổng các vị trí hiện có.
- Sử dụng Django để xây dựng website.
- Mô phỏng mô hình trực tiếp.
- Hoàn chỉnh cuốn báo cáo đề tài.
- Nắm được kiến trúc thuật toán YOLOv3 và có thể ứng dụng vào mọi đề tài liên quan.
- Nắm được các ưu, nhược điểm của thuật toán và các phương pháp tối ưu cho thuật toán để cải thiện mô hình nhận diện.
- Nắm được những quy trình trong huấn luyện và kiểm tra mô hình trong các mô hình Deep Learning.

6. Giáo viên và cán bộ hướng dẫn

Họ tên: PHẠM THỊ MIÊN

Đơn vị công tác: Bộ môn Công Nghệ Thông Tin – Trường Đại học Giao thông Vận tải phân hiệu tại TP HCM

Điện thoại: 0961170638 Email:

Ngày … tháng 03 năm 2021 **Đã giao nhiệm vụ TKTN**

BM Công Nghệ Thông Tin **Giáo viên hướng dẫn**

ThS. Phạm Thị Miên

Đã nhận nhiệm vụ TKTN

Sinh viên: NGUYỄN VĂN HIỆP Ký tên:

Điện thoại: 0973550322 Email: nguyenvanhiepcmg@gmail.com

LỜI CẢM ƠN

Qua thời gian học tập và rèn luyện tại trường Trường Đại học Giao thông Vận tải phân hiệu tại TP HCM, đến nay chúng em đã kết thúc khoá học 4 năm và hoàn thành đồ án tốt nghiệp. Trong thời gian học tập tại trường để có được kết quả hiện tại em xin chân thành cảm ơn:

Cảm ơn tập thể các thầy cô giáo Bộ môn Công Nghệ Thông Tin và các thầy cô thỉnh giảng đã giảng dạy, quan tâm và tạo điều kiện thuận lợi để chúng em học tập rèn luyện trong suốt thời gian qua, giúp chúng em trang bị những kiến thức, kỹ năng cần thiết cho công việc thực tế sau này. Cảm ơn thầy cô giáo Bộ môn cũng như Ban Giám Hiệu đã cho phép em thực hiện đề tài tốt nghiệp: ***NGHIÊN CỨU TRIỂN KHAI THUẬT TOÁN YOLOV3 VÀ XÂY DỰNG HỆ THỐNG HƯỚNG DẪN ĐẬU XE Ô TÔ PARKING VISION***.

Và cảm ơn thạc sĩ Phạm Thị Miên đã luôn quan tâm nhiệt tình hướng dẫn, giúp đỡ em trong quá trình thực hiện đồ án tốt nghiệp. Cô cũng luôn nhắc nhở, động viên mỗi khi em gặp khó khăn, nhờ vậy mà em đã hoàn thành tốt đồ án tốt nghiệp của mình đúng thời hạn được giao. Nếu không có những lời hướng dẫn, dạy bảo của cô thì em nghĩ bài báo cáo này của em sẽ rất khó có thể hoàn thiện được.

Em cũng xin gửi lời cảm ơn tới gia đình, bạn bè, những người đã động viên, giúp đỡ em rất nhiều trong thời gian học tập và làm đồ án tốt nghiệp.

Mặc dù đã cố gắng nỗ lực học hỏi không ngừng để hoàn thành đề tài, vậy nhưng thời gian thực hiện đồ án có hạn, kiến thức của em còn hạn chế. Do vậy, không tránh khỏi những thiếu sót, em rất mong nhận được những ý kiến đóng góp quý báu của thầy cô trong hội đồng bảo vệ đồ án tốt nghiệp để kiến thức của em được hoàn thiện hơn.

Đồng thời em xin cam đoan rằng nội dung đồ án của chính em nghiên cứu xây dựng nên, nếu có nội dung tham khảo đều được trích dẫn cụ thể, rõ ràng.

TP. Hồ Chí Minh, ngày 11 tháng 06 năm 2021

Sinh viên thực hiện

Nguyễn Văn Hiệp

NHẬN XÉT CỦA GIÁO VIÊN HƯỚNG DẪN

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

Tp. Hồ Chí Minh, ngày … tháng 07 năm 2021

Giáo viên hướng dẫn

ThS. Phạm Thị Miên

MỤC LỤC

DANH MỤC THUẬT NGỮ

STT	THUẬT NGỮ	Ý NGHĨA TIẾNG VIỆT	TỪ VIẾT TẮT	GHI CHÚ
1	You Only Look Once	Bạn Chỉ Nhìn Một Lần	YOLO	
2	You Only Look Once Version 3	Bạn Chỉ Nhìn Một Lần Phiên bản 3	YOLOv3	
3	Artificial Intelligence	Trí tuệ nhân tạo	AI	
4	Deep Learning	Học Sâu	DL	
5	Machine Learning	Học Máy	ML	
6	Computer Vision	Thị Giác Máy Tính	CV	
7	Internet of Things	Internet Vạn Vật	IOT	
8	Electronic Know Your Customer	Định Danh Khách Hàng Điện Tử	eKYC	
9	Optical Character Recognition	Nhận Dạng Ký Tự Quang Học	OCR	
10	Insight	Sự thật ngầm hiểu		
11	Framework	Bộ khung		
12		Cơ sở dữ liệu	CSDL	
13	Image Processing	Xử lý ảnh	XLA	
14	Picture Element	Điểm ảnh	pixel	
15	Mean squared error	Sai số toàn phương trung bình	MSE	
16	Responsive	Tương thích		
17	Package	Gói		
18	Graphical Processing Unit	Bộ Xử Lý Đồ Họa	GPU	
19	Central Processing Unit	Bộ Xử Lý Trung Tâm	CPU	
20	Training	Huấn luyện		
21	Open Source Computer Vision Library	Thư viện Thị giác Máy tính Nguồn mở	OpenCV	
22	Real-time	Thời gian thực		
23	Neural	Nơ-ron		
24	Convolutional Neural Network	Mạng Neural Tích Chập	CNN	
25	Weight	Trọng số		
26	Bias	Độ lệch		
27	Multi Layer Perceptron	Mạng Neural Nhiều Lớp	MLP	
28	Backpropagation	Lan truyền ngược		
29	Overfiting	Quá khớp		
30	Fully Connected Layer	Lớp Kết Nối Đầy Đủ		
31	Learning rate	Tốc độ học		

32	Momentum	Gia tốc học		
33	Gradient	Độ dốc		
34	Activation function	Hàm kích hoạt		
35	Feature	Đặc trưng		
36	Internal Covariate Shift	Chuyển dịch nội hiệp phương sai	ICS	
37	Pretrained model	Mô hình được đào tạo trước		
38	Transfer Learning	Học chuyển giao		
39	Extraction	Trích xuất		
40	Classify/Regress	Phân loại/hồi quy		
41	Skip connection	Kỹ thuật bỏ qua kết nối		
42	Feature Pyramid Networks	Mạng Kim Tự Tháp Đặc Trưng	FPN	
43	Object detection	Phát hiện đối tượng		
44	Bounding box	Hộp giới hạn	bbox	
45	Single Shot Detector	Máy dò ảnh đơn	SSD	
46	Fully Convolutional Network	Mạng Tích Chập Hoàn Toàn	FCN	
47	Feature map	Biểu đồ đặc trưng		
48	Stride	Bước nhảy		
49	Ground truth	Thực sự/chân lý		
50	Anchor	Điểm neo		
51	Offset	Độ lệch/khoảng trống		
52	Intersection Over Union	Giao trong hợp	IOU	
53	Predicted	Được dự đoán		
54	Frame	Khung hình		
55	Graphical User Interface	Giao diện đồ họa người dùng	GUI	
56	Frames per second	Số hình ảnh xuất hiện trên khung hình trong một giây	FPS	
57	Backbone	Lõi / xương sống		
58	Open source	Mã nguồn mở		
59	Gradient Descent	Giảm độ dốc	GD	
60	Dataset	Tập dữ liệu		

DANH MỤC CÁC BẢNG

DANH MỤC HÌNH ẢNH

CHƯƠNG 1. TỔNG QUAN ĐỀ TÀI

1.1 Sơ lược về Computer Vision và sự hỗ trợ của Deep Learning

1.1.1 Thị Giác Máy Tính (Computer Vision)

Thị Giác Máy Tính (Computer Vision) đề cập đến toàn bộ quá trình mô phỏng tầm nhìn của con người trong một bộ máy phi sinh học. Điều này bao gồm việc chụp ảnh ban đầu, phát hiện và nhận dạng đối tượng, nhận biết bối cảnh tạm thời giữa các cảnh và phát triển sự hiểu biết ở mức độ cao về những gì đang xảy ra trong khoảng thời gian thích hợp. Trong thực tế, một hệ thống cung cấp khả năng Thị Giác Máy Tính đáng tin cậy, chính xác và trong thời gian thực là một vấn đề đầy thách thức vẫn chưa được phát triển một cách đầy đủ.

Khi các hệ thống này được phát triển hoàn chỉnh, sẽ có vô số ứng dụng dựa vào Thị Giác Máy Tính như một thành phần chính. Những ví dụ điển hình đó là xe hơi tự lái, robot tự động, máy bay không người lái, thiết bị hình ảnh y tế thông minh hỗ trợ phẫu thuật và cấy ghép phẫu thuật phục hồi thị lực của con người.

Có thể thấy Thị Giác Máy Tính mang nhiều hứa hẹn rất lớn trong tương lai, tuy vậy nó là một hệ thống phức tạp và luôn là thách thức đối với các hệ thống máy tính. Một phần của sự phức tạp là do thực tế Thị Giác Máy Tính không phải là một nhiệm vụ duy nhất. Thay vào đó, nó là một chuỗi các nhiệm vụ không đơn giản mà mỗi yêu cầu sử dụng các thuật toán phức tạp và đủ sức mạnh tính toán để hoạt động trong thời gian thực. Ở cấp độ cao, các tác vụ phụ của Thị Giác Máy Tính là phát hiện và phân đoạn đối tượng, phân loại hình ảnh, theo dõi đối tượng, gắn nhãn hình ảnh với các mô tả có ý nghĩa (ví dụ như chú thích hình ảnh) và cuối cùng, hiểu ý nghĩa của toàn bộ bối cảnh.

1.1.2 Đóng góp từ Deep Learning

Mặc dù vẫn còn những trở ngại đáng kể trong con đường phát triển của Thị Giác Máy Tính đến "cấp độ con người", các hệ thống Deep Learning đã đạt được tiến bộ đáng kể trong việc xử lý một số nhiệm vụ phụ có liên quan. Lý do cho sự thành công này một phần dựa trên trách nhiệm bổ sung được giao cho các hệ thống Deep Learning. Điều hợp lý để nói rằng sự khác biệt lớn nhất với các hệ thống Deep Learning là chúng không còn cần phải được lập trình để tìm kiếm các đặc điểm cụ thể. Thay vì tìm kiếm các đặc điểm cụ thể bằng thuật toán được lập trình cẩn thận, các mạng lưới Neural bên trong các

hệ thống Deep Learning được đào tạo để có khả năng tự học. Ví dụ: nếu ô tô trong hình ảnh bị phân loại sai thành xe máy thì ta không tinh chỉnh các tham số hoặc viết lại thuật toán. Thay vào đó, ta tiếp tục đào tạo cho đến khi hệ thống làm cho đúng.

1.1.3 Một số ứng dụng chính

- Phân loại hình ảnh
- Phân loại hình ảnh cùng với việc bản địa hóa
- Phát hiện đối tượng
- Tái thiết hình ảnh
- Theo dõi đối tượng

Những tiến bộ trong hệ thống Deep Learning và sức mạnh tính toán đã giúp cải thiện tốc độ, độ chính xác và độ tin cậy tổng thể của hệ thống Thị Giác Máy Tính. Khi các mô hình Deep Learning cải thiện và sức mạnh tính toán trở nên dễ dàng hơn, chúng ta sẽ tiếp tục đạt được những tiến bộ và sự ổn định đối với các hệ thống tự vận hành có thể thực sự nắm bắt và phản ứng với những gì chúng cảm nhận.

1.2 Đặt vấn đề

1.2.1 Thực trạng

Cùng với sự phát triển vượt bậc về kinh tế xã hội, nhu cầu của con người ngày càng được nâng cao, do đó nhu cầu lưu thông hàng hoá và những đòi hỏi về đi lại ngày càng tăng. Nếu như trên không trung, máy bay là phương tiện chiếm ưu thế, thì trên mặt đất, ô tô và vận tải ô tô lại chiếm ưu thế về năng lực vận chuyển và khả năng cơ động.

"Theo Sở Giao thông vận tải Hà Nội, trên địa bàn Hà Nội hiện có khoảng 6,9 triệu phương tiện giao thông (ô-tô, xe máy), chưa kể lượng xe ngoại tỉnh ra, vào thành phố hằng ngày. Dự tính, với tốc độ tăng trưởng xe máy 7,66%/năm; ô-tô 16,15%/năm thì đến năm 2025 Hà Nội có 1,3 triệu ô-tô và 7,3 triệu xe máy; năm 2030 có 1,7 triệu ô-tô và 7,7 triệu xe máy. Thế nhưng, điều đáng nói là hệ thống giao thông tĩnh (bãi, điểm đỗ xe công cộng) hiện nay còn quá thiếu và yếu, khiến nhu cầu đỗ xe ngày càng cấp thiết.[1]" (*theo báo nhandan.vn số ra ngày 09/03/2021*)

[1] https://nhandan.vn/baothoinay-dothi/tim-co-che-cho-cac-diem-do-xe-637883/

Thực trạng cho ta thấy cùng với sự tăng lên của các phương tiện giao thông chỉ ở Hà Nội, với các thành phố lớn thì cũng đang gặp vấn đề tương tự, ô tô đang có tốc độ tăng trưởng về số lượng khá cao về số lượng, do đó nhu cầu đỗ xe cũng ngày càng cao. Việc tìm được một chỗ đỗ xe trong thời buổi hiện nay cũng đang còn khá khó khăn. Đôi khi, có thể có chỗ trống trong bãi đỗ xe nhưng người sử dụng sẽ không biết nó nằm ở đâu, hoặc là không biết trong bãi có còn chỗ trống hay không. Rất nhiều vụ đỗ xe trước nhà dân, vỉa hè, lòng đường, công viên các nơi không phép… dẫn đến nhiều bất cập, ví dụ tình trạng tắc nghẽn giao thông và mất mỹ quan đô thị gây cản trở đường đi của chủ nhà, thậm chí nguy hiểm cho các phương tiện giao thông khác.

1.2.2 Các loại hình bãi đỗ xe phổ biến hiện nay

- **Bãi đỗ xe truyền thống:** Hệ thống đỗ xe ô tô truyền thống có ở hầu hết các gia đình, văn phòng và trung tâm giải trí. Không gian đỗ xe riêng biệt được tạo ra để vào và ra trơn tru để tăng tổ chức và giảm lưu lượng. Chúng đòi hỏi ít năng lượng hơn với yêu cầu năng lượng duy nhất là chiếu sáng, chức năng không bị cản trở bởi sự cố mất điện. Nhà để xe truyền thống đang hoạt động suốt ngày đêm, một số bãi đỗ xe truyền thống có không gian hạn chế nên người người sử dụng phải tìm chỗ trống trước khi đỗ xe. Nếu chúng có thiết kế đơn giản, số vốn bỏ ra không lớn. Chiến lược bảo vệ xe hơi cũng có thể được cung cấp bằng cách giao nhiều lô cho các cá nhân cụ thể quản lý.

Hình 1.1 Bãi đỗ xe truyền thống tại siêu thị Emart Gò Vấp TP. HCM. (Ảnh Internet)

- **Hệ thống bãi đỗ xe tự động:** đòi hỏi ít diện tích hơn, loại bỏ trình điều khiển khỏi quá trình đỗ xe. Nó có thể là hoàn toàn tự động hoặc bán tự động. Chiếc xe phải được lái đến một điểm nhập cảnh nơi người lái và hành khách thoát khỏi xe. Sau đó, nó được di chuyển tự động hoặc bán tự động (với một số hỗ trợ cần thiết) đến không gian được phân bổ của nó. Hệ thống bãi đỗ xe tự động tối đa hóa không gian hạn chế, một lợi thế trong khu vực không gian hạn chế có sẵn.

Hình 1.2 Bãi đỗ xe tự động tại Đà Nẵng. (Ảnh Internet)

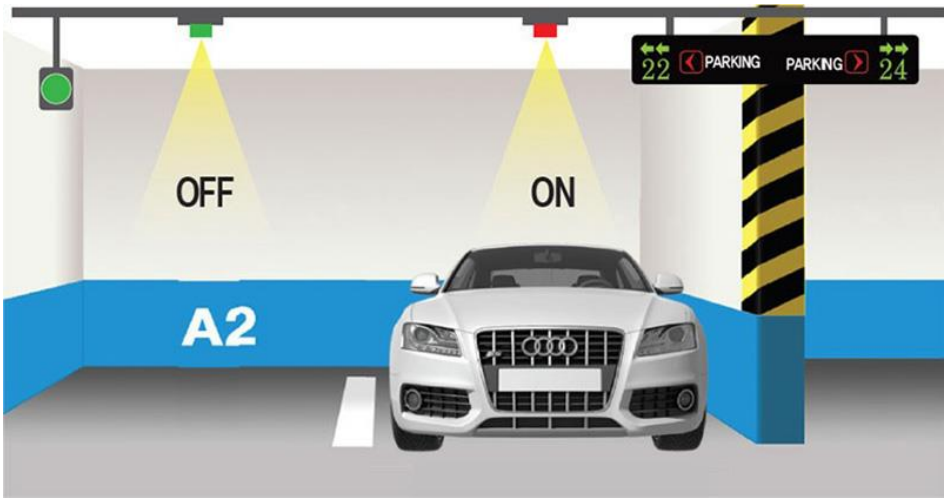
1.2.3 Hệ Thống Hướng Dẫn Bãi Đỗ Xe (Parking Guidance System - PGS)

Cho dù là bãi đỗ xe truyền thống hay tự động thì trong một bãi đỗ xe hoàn chỉnh không thể thiếu hệ thống hướng dẫn đỗ xe (Viết tắt là PGS), các loại hình phổ biến tại Việt Nam hiện nay:

- **Chỉ dẫn thủ công:** Đây là phương pháp chỉ dẫn truyền thống nhất, mọi công việc đều do con người trực tiếp tham gia vào làm, khi đó khi vào chỗ để xe thì người sử dụng sẽ được một nhân viên trong bãi xe hướng dẫn đỗ xe, hoặc nhiều khi đó cũng chính là người ghi vé. Phương pháp này tốn rất nhiều nhân công và thiếu chính xác, ví dụ như một bãi đỗ xe lớn thì nhiều khi họ sẽ không nắm được số lượng chỗ trống còn lại và vị trí nào còn trống trong bãi để chỉ dẫn cho người sử dụng được. Hơn nữa còn mang lại phiền toái khá nhiều cho người sử dụng khi phải mất thời gian tự đi tìm chỗ trống, dễ gây ra va chạm với các xe khác, tốn nhiên liệu tiêu thụ cho xe và tăng mức độ ô nhiễm

môi trường, làm tâm trạng người sử dụng trở nên chán nản và bực bội, gây ùn ứ các phương tiện đi vào bãi, có thể làm giảm doanh thu của bãi xe.

- **Hệ thống hướng dẫn thông minh bằng cảm biến:** Hệ thống hướng dẫn đỗ xe thông minh cung cấp sự tiện lợi trong việc hướng dẫn đỗ xe một cách tự động cho bãi đỗ xe ô tô. Người sử dụng dễ dàng nhận biết vị trí còn trống tại từng khu vực trong bãi đỗ xe khi họ đi vào bãi. Tại mỗi vị trí xe đỗ được gắn một thiết bị siêu âm để phát hiện vị trí đỗ là có xe hay không có xe. Tình trạng của các vị trí báo cáo về trung tâm theo thời gian thực. Máy tính chủ sẽ thu thập tất cả trạng thái của từng vị trí từ bộ siêu âm và điều khiển các bảng quang báo hiển thị để hướng dẫn người sử dụng dễ dàng tìm đến vị trí còn trống để đỗ xe.



Hình 1.3 Minh họa cho hệ thống hướng dẫn bằng cảm biến. (Ảnh Internet)

Tuy nhiên giá mỗi cảm biến rất đắt đỏ, tốn khá nhiều chi phí triển khai, việc bảo trì cũng diễn ra khá khó khăn và tốn kém khi phải thuê nhân viên bên ngoài về vì yêu cầu tính chuyên môn cao, và giá linh kiện đắt đỏ, hơn nữa không phải lúc nào thì người sử dụng cũng biết là trong bãi còn chỗ trống hay không, đến khi vào nhưng hết chỗ thì cũng gây ra nhiều phiền toái. Hệ thống này chủ yếu được triển khai trong nhà có mái che, hoặc hầm.

1.2.4 Giải pháp mới sử dụng công nghệ Computer Vision và Deep Learning:

Hệ thống hướng dẫn đậu xe thông minh Parking Vision

Để giải quyết những nhược điểm của các phương pháp trên thì giải pháp được đề xuất để giải quyết cũng như là đề tài của đồ án này đó là *nghiên cứu triển khai thuật toán YOLOv3 và xây dựng hệ thống hướng dẫn đậu xe ô tô Parking Vision*. Ta sẽ sử

dụng bài toán **phát hiện đối tượng** nằm trong lĩnh vực Computer Vision để phát hiện ra các vị trí đã đỗ và còn trống trong bãi, cùng với đó sử dụng công nghệ Deep Learning để huấn luyện và hỗ trợ cho Computer Vision trong việc kiểm tra dự đoán của mô hình. Một trong những sự kết hợp đó là thuật toán You Only Look Once (YOLO), là một mô hình CNN để phát hiện đối tượng mà một ưu điểm nổi trội là nhanh hơn nhiều so với những mô hình cũ. Thậm chí có thể chạy tốt trên những thiết bị IOT như Raspberry Pi[2]. Khi sử dụng phương pháp này thì có những ưu điểm sau:

- **Đối với chủ đầu tư:** Giúp chủ đầu tư quản lý bãi giữ xe một cách chuyên nghiệp, khoa học với các công nghệ hiện đại, đồng thời giảm chi phí chủ đầu tư (bao gồm chi phí đầu tư máy chủ, tên miền, màn hình hiển thị và camera đối với giải pháp này), thích hợp triển khai ở những nơi ngoài trời như bãi đỗ xe của siêu thị, trường học, nhà hàng... có thể bao quát tầm nhìn của toàn bộ xe.
- **Đối với người sử dụng:** Thuận tiện cho người sử dụng, có thể nắm bắt số lượng chỗ trống và vị trí chính xác trong bãi, giúp chủ xe giảm thời gian tìm kiếm, tiết kiệm nhiên liệu, giúp tâm trạng thoải mái hơn thay vì tự mình tìm xem vị trí nào còn trống. Nhờ đó có thể biết trước bãi còn chỗ đỗ xe hay không, có thể tự tin đến nơi đỗ xe. Hơn nữa người sử dụng có thể theo dõi tình trạng xe mình ở mọi lúc mọi nơi, sử dụng như một camera an ninh.
- **Đối với người quản lý:** Thuận tiện cho người quản lý bãi xe, giúp cho họ nắm bắt thông tin số lượng chỗ trống hiện tại, chính xác vị trí còn trống để chỉ dẫn cho các tài xế vào bãi, tạo điều kiện thuận lợi cho việc điều hành các phương tiện di chuyển.

1.3 Tình hình nguyên cứu

Hiện nay ở Việt Nam phương pháp hướng dẫn ô tô dựa vào Computer Vision còn chưa phát triển nhiều, có lẽ vì ngành Computer Vision ở Việt Nam chưa được ứng dụng sâu và rộng, đây là một ngành mới nổi ở Việt Nam và có nhiều triển vọng phát triển, hiện tại chỉ được áp dụng nhiều vào các giải pháp nhận diện khuôn mặt, nhận diện ký tự - OCR. Những sản phẩm của các giải pháp đó có thể kể đến như: Định danh khách hàng - eKYC, nhận dạng giấy tờ tùy thân, nhận dạng biển số xe. Nhưng công nghệ luôn phát triển không ngừng, và các bài toán đó đã được làm đi làm lại, cải tiến rất nhiều lần và cũng đã dần hoàn thiện, điều đó thúc đẩy mở ra các giải pháp khác cho chúng ta tìm

[2] Raspberry Pi là từ để chỉ các máy tính bo mạch đơn kích thước chỉ bằng một thẻ tín dụng.

hiểu. Ta nên có "insight" về xu thế phát triển công nghệ mới của tương lai và nghĩ ra giải pháp cho những vấn đề mới từ đó kiến tạo thế giới tiện dụng và tốt đẹp hơn.

Hiện tại dù YOLO đã ra tới phiên bản YOLOv5, nhưng phiên bản này đang còn nhiều tranh cãi về tốc độ cũng như độ chính xác, không phải do chính chủ tác giả phát hành. Phiên bản YOLOv4 cũng đã có rất nhiều cải tiến so với YOLOv3 nhưng hiện giờ YOLOv3 vẫn còn rất mạnh về cả tốc độ lẫn độ chính xác, hơn nữa với một người mới tìm hiểu về lĩnh vực này thì tôi chọn phiên bản YOLOv3 để tìm hiểu và triển khai, bởi vì phiên bản càng mới thì kiến trúc của thuật toán càng phức tạp, nếu quá phức tạp sẽ rất khó để tìm hiểu trong thời gian ngắn.

PyTorch là một package được xây dựng dựa trên Python để thay thế Numpy để tận dụng sức mạnh tính toán của GPU và cung cấp tính linh hoạt như một nền tảng phát triển Deep Learning. Pytorch là framework được phát triển bởi Facebook, Inc[3]. Đây là một ông lớn về công nghệ đầu tư rất nhiều nguồn lực cho việc phát triển Trí Tuệ Nhân Tạo. Pytorch được phát triển với giấy phép mã nguồn mở do đó nó tạo được cho mình một cộng đồng rất lớn. Một cộng đồng lớn đồng nghĩa với nhiều tài nguyên để học và các vấn đề của ta có thể đã có ai đó giải quyết và chia sẻ với cộng đồng. Pytorch cùng với Tensorflow[4] và Keras[5] là một trong những framework phổ biến được sử dụng trong các bài toán về Deep Learning hiện nay. Đặc biệt, trong các lĩnh vực nghiên cứu, hầu như các tác giả đều sử dụng Pytorch để triển khai bài toán của mình.

Django là một trong số những web framework bậc cao miễn phí, là mã nguồn mở được tạo ra bởi ngôn ngữ Python dựa trên mô hình MTV (gồm Model-Template-Views). Hiện framework này được phát triển, quản lý bởi Django Software Foundation. Django ra đời với mục tiêu hỗ trợ thiết kế các website phức tạp dựa trên những CSDL có sẵn. Nó hoạt động dựa theo nguyên lý 'cắm' các thành phần và và tái sử dụng để tạo nên các website với ít code, ít khớp nối, có khả năng phát triển và không bị trùng lặp. Lợi thế hàng đầu của Django là khả năng thiết kế, tạo lập website và các ứng dụng nhanh chóng. Vì vậy Django được sử dụng để thực hiện đồ án này.

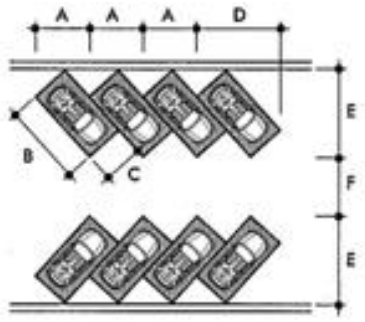
[3] Facebook, Inc. là một công ty truyền thông xã hội và công nghệ Mỹ có trụ sở tại Menlo Park, California.
[4] TensorFlow là một thư viện phần mềm mã nguồn mở dành cho máy học trong nhiều loại hình tác vụ nhận thức và hiểu ngôn ngữ.
[5] Keras là một thư viện phần mềm mã nguồn mở cung cấp giao diện Python cho các mạng nơ-ron nhân tạo.

1.4 Quá trình nguyên cứu

Để hiểu rõ hơn về nghiệp vụ liên quan tới bãi đỗ xe, tôi sử dụng các mẫu bãi đỗ xe truyền thống và phổ biến nhất trên internet để tìm hiểu và triển khai giúp ích cho quá trình thu thập hình ảnh và đánh nhãn cũng như phát hiện sau này được dễ dàng hơn.

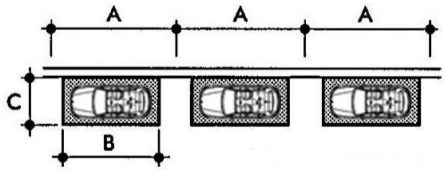
1.4.1 Các kiểu bãi đỗ xe ô tô thông dụng

- Bãi đỗ xe ô tô chéo góc 45 độ:



Hình 1.4 Bãi đỗ xe góc chéo. (Ảnh Internet)

- Bãi đỗ xe ô tô song song:



Hình 1.5 Bãi đỗ xe song song. (Ảnh Internet)

1.4.2 Các trạng thái có thể có trong một chỗ đỗ xe ô tô

Chỉ có 2 trạng thái chính, đó là:

- Trống: Chỗ đó chưa có ô tô nào được đỗ
- Đã đỗ: Chỗ đó đã có ô tô đỗ

1.4.3 Các điều kiện bên ngoài ảnh hưởng đến chất lượng hình ảnh

- Thời tiết: Mưa, nắng
- Ánh sáng: Nắng đỗ bóng, nắng gắt, nắng nhẹ, râm mát, trời tối bật đèn,...
- Các yếu tố gây nhiễu: Cây cối, bụi bặm bám trên camera,…

1.4.4 Giải quyết bài toán phát hiện đối tượng bằng thuật toán YOLOv3

Có 2 phương án để giải quyết bài toán phát hiện chỗ còn trống và đã đỗ với thuật toán YOLOv3, đó là:

- Thứ nhất, sử dụng tệp weights được tác giả thuật toán YOLOv3 huấn luyện sẵn với tập dữ liệu COCO sau đó phát hiện các vị trí của ô tô, và lấy tổng các vị trí hiện có trong bãi trừ cho số lượng vị trí ô tô đó. Tuy phương án này rút ngắn thời gian huấn luyện mô hình nhưng độ chính xác không cao, dễ bị nhiễu bởi các đối tượng khác bởi vì tập dữ liệu COCO được huấn luyện để nhận dạng lên đến 80 đối tượng, và trong lúc phát hiện có thể có ô tô đang di chuyển sẽ khiến thuật toán nhận diện nhầm. Hơn nữa các vị trí còn trống sẽ không được phát hiện, cho nên thuật toán sẽ không biết khi nào ô tô được đậu đúng vị trí, vì khi ô tô đậu bên ngoài vạch cũng được phát hiện. Cuối cùng phương án này chúng ta cần phải biết trước số lượng tổng các vị trí trong bãi. Đây là phương án tiếp cận quá nhiều bước.
- Thứ hai, ta sẽ huấn luyện mô hình để phát hiện được cả 2 trạng thái trống và đã đỗ trong bãi, sau đó tính tổng bằng cách lấy tổng các đối tượng ta phát hiện được. Ta sẽ chọn phương án này để triển khai.

1.5 Cấu trúc báo cáo đồ án tốt nghiệp

Cấu trúc đồ án được chia thành các chương như sau:

Chương 1: Tổng quan đề tài – Giới thiệu tổng quan về đề tài đồ án tốt nghiệp.

Chương 2: Cơ sở lý thuyết

- Phân biệt các khái niệm liên quan đến Trí Tuệ Nhân Tạo, Thị Giác Máy Tính, Học Máy, Học Sâu.
- Giới thiệu các công nghệ sử dụng.
- Các kiến thức liên quan đến thống kê học sử dụng trong bài làm.
- Tổng quan về xử lý ảnh.
- Tổng quan về Thị Giác Máy Tính và Học Sâu.

Chương 3: Mạng Neural Tích Chập (CNN)

- Giới thiệu Mạng Neural và Mạng Neural lan truyền ngược.
- Giới thiệu Mạng Neural Tích Chập.

- Giới thiệu bài toán phát hiện đối tượng.
- Giới thiệu thuật toán hồi quy logistic.
- Tìm hiểu về Gradient Descent.
- Tìm hiểu về Batch normalization.
- Giới thiệu về Transfer Learning.
- Giới thiệu thuật toán YOLOv3.
- Giới thiệu các chỉ số đánh giá thuật toán.

Chương 4: Triển khai thuật toán và thiết kế, xây dựng website

- Thu thập dữ liệu, tiền xử lý, gắn nhãn và huấn luyện thuật toán.
- Triển khai thuật toán YOLOv3 phát hiện qua hình ảnh, video, camera xem trực tiếp màn hình xử lý trên máy tính.
- Kiểm tra thuật toán.
- Xây dựng website giới thiệu sản phẩm và xem màn hình xử lý.

Kết luận và kiến nghị

- Đưa ra kết quả đạt được, những thứ còn tồn tại và hướng phát triển về thuật toán lẫn website trong tương lai.

Phụ lục

Tài liệu tham khảo.

CHƯƠNG 2. CƠ SỞ LÝ THUYẾT

2.1 Phân biệt các khái niệm chính

2.1.1 Trí Tuệ Nhân Tạo

Trí Tuệ Nhân Tạo (Artificial Intelligence – viết tắt là AI) có thể được định nghĩa như một ngành của khoa học máy tính liên quan đến việc tự động hóa các hành vi thông minh. AI là một bộ phận của khoa học máy tính và do đó nó phải được đặt trên những nguyên lý lý thuyết vững chắc, có khả năng ứng dụng được của lĩnh vực này. Có thể hiểu đó là trí tuệ của máy móc được tạo ra bởi con người. Trí tuệ này có thể tư duy, suy nghĩ, học hỏi,... như trí tuệ con người. Xử lý dữ liệu ở mức rộng lớn hơn, quy mô hơn, hệ thống, khoa học và nhanh hơn so với con người.

Trong lịch sử phát triển AI, các nhà nghiên cứu phân thành 4 hướng tiếp cận chính:

- Hành động như người (acting humanly)
- Suy nghĩ như người (thinking humanly)
- Suy nghĩ hợp lý (thinking rationally)
- Hành động hợp lý (acting rationally)

Một trong những hướng phát triển đó điển hình là bài kiểm tra Turing (Turing test), đây là hướng phát triển AI hành động như người. Bài kiểm tra này được Alan Turing[6] đề xuất vào năm 1950, mục đích để kiểm tra hệ thống máy tính đã đạt đến khả năng thông minh hay chưa. Bài test gồm một người đặt câu hỏi, một người trả lời câu hỏi và một máy tính phản hồi câu hỏi. Nếu trong quá trình trao đổi mà người đặt câu hỏi không thể phân biệt được người hay máy trả lời các câu hỏi này thì máy được xem là thông minh. Đến nay thì hướng tiếp cận này đã đạt được một số thành quả nhất định như:

- **Natural Language Processing**: Máy có khả năng đọc hiểu và giao tiếp bằng ngôn ngữ tự nhiên với người.

[6] Alan Mathison Turing OBE FRS là một nhà toán học, logic học và mật mã học người Anh, được xem là cha đẻ của ngành khoa học máy tính.

- **Knowledge Representation:** Máy có khả năng lưu trữ tri thức thông qua thị giác, thính giác, hay văn bản.
- **Automated Reasoning**: Máy có khả năng sử dụng tri thức đã lưu trữ để trả lời câu hỏi hay đưa ra kết luận hữu ích.
- **Machine Learning:** Máy có khả năng thích nghi với các điều kiện môi trường xung quanh để rút trích ra các nguyên lý từ tri thức thu nhận được phục vụ cho việc ra quyết định.
- **Computer Vision**: Máy có khả năng quan sát và xác định được các đối tượng xung quanh.
- **Robotics**: Máy có khả năng tương tác với đối tượng và di chuyển trong môi trường xung quanh.

Những gì chúng ta đang thực hiện với AI hiện nay nằm trong khái niệm "AI hẹp" (Narrow AI). Công nghệ này có khả năng thực hiện các nhiệm vụ cụ thể một cách tương tự, hoặc tốt hơn con người.

2.1.2 Học Máy

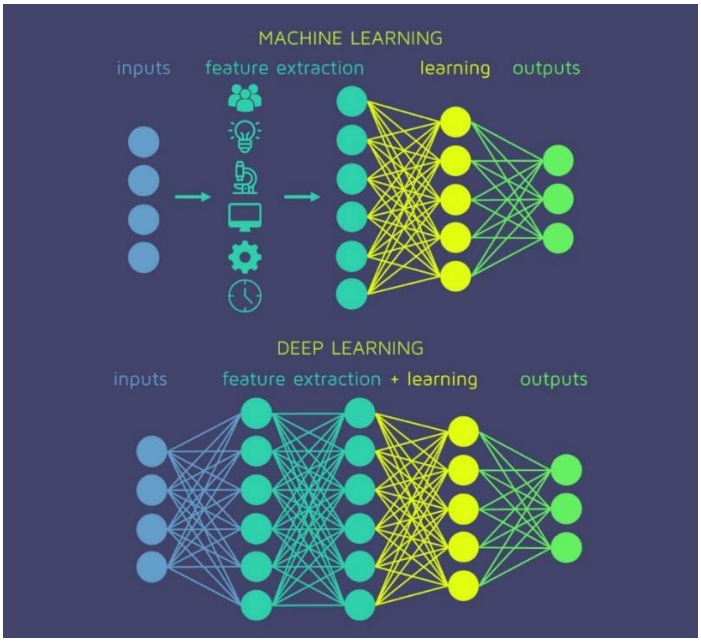
Học Máy (Machine Learning – viết tắt là ML) là một thuật ngữ rộng để chỉ hành động dạy máy tính cải thiện một nhiệm vụ mà nó đang thực hiện, là một thành phần trong hướng tiếp cận hành động như người. Cụ thể hơn, ML đề cập tới bất kỳ hệ thống mà hiệu suất của máy tính khi thực hiện một nhiệm vụ sẽ trở nên tốt hơn sau khi hoàn thành nhiệm vụ đó nhiều lần. Hay nói cách khác, khả năng cơ bản nhất của ML là sử dụng thuật toán để phân tích những thông tin có sẵn, học hỏi từ nó rồi đưa ra quyết định hoặc dự đoán về một thứ gì đó có liên quan. Thay vì tạo ra một ứng dụng với những hành động, hướng dẫn chi tiết để thực hiện một nhiệm vụ cụ thể, máy tính được "huấn luyện" bằng cách sử dụng lượng dữ liệu và các thuật toán để học cách thực hiện nhiệm vụ đó. Nếu không có ML, AI hiện tại sẽ bị hạn chế khá nhiều bởi nó mang lại cho máy tính sức mạnh để tìm ra mọi thứ mà không được lập trình rõ ràng.

Các phương pháp học ML thường được chia làm 3 nhóm:

- Supervised learning (Học có giảm sát): Sử dụng các giải thuật và hướng giải quyết như: Decision tree (cây quyết định), k-NN, naive bayes, SVM, neural network (mạng neural), Deep Learning (Học Sâu)…
- Unsupervised Learning (Học không giám sát): Sử dụng các giải thuật và hướng giải quyết như: k-means, hierachical clustering (phân cụm phân cấp).
- Reinforcement learning (Học củng cố): Bao gồm các phương pháp: Active (chủ động), passive (thụ động), generalization (tổng quát hóa).

2.1.3 Học Sâu

Học Sâu (Deep Learning – viết tắt là DL) chỉ là một phương pháp nằm trong hướng giải quyết học có giám sát của ML. Nó như là một loại ML với "neural network" sâu có thể xử lý dữ liệu theo cách tương tự như một bộ não con người có thể thực hiện.



Hình 2.1 Sự khác biệt giữa ML và DL

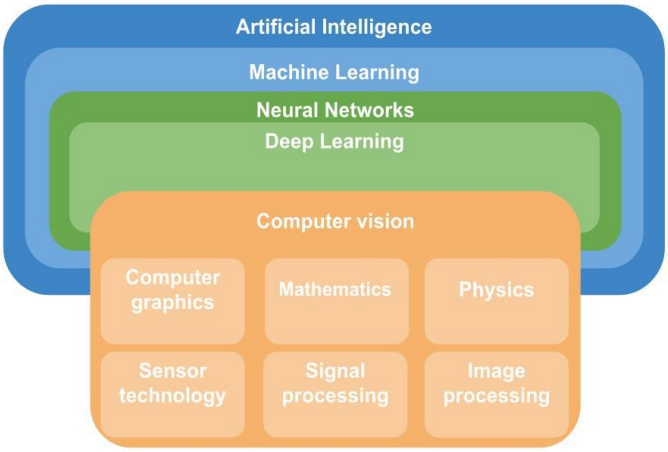
Tóm lại Deep learning là loại ML mà trong đó máy tự đào tạo chính nó. Deep Learning đòi hỏi rất nhiều dữ liệu đầu vào và sức mạnh tính toán hơn là ML, nó đã bắt đầu được triển khai bởi các công ty công nghệ lớn như Facebook, Amazon[7]. Trong đó, một trong những cái tên nổi tiếng nhất về ML là AlphaGo[8], một máy tính có thể chơi cờ vây với chính bản thân nó cho đến khi nó có thể dự đoán những đường đi nước bước chính xác nhất đủ để đánh bại nhiều nhà vô địch trên thế giới.

[7] Amazon.com, Inc. là một công ty công nghệ đa quốc gia của Mỹ có trụ sở tại Seattle, Washington
[8] AlphaGo là chương trình máy tính cờ vây do Google DeepMind phát triển tại London

2.1.4 Thị Giác Máy Tính

Thị Giác Máy Tính (Computer Vision – viết tắt là CV) là một lĩnh vực liên ngành của Trí Tuệ Nhân Tạo nhằm mục đích cung cấp cho máy tính và các thiết bị khác có khả năng tính toán hiểu biết ở mức độ cao từ cả hình ảnh và video kỹ thuật số, bao gồm chức năng thu thập, xử lý và phân tích hình ảnh kỹ thuật số. Đây là lý do tại sao Thị Giác Máy Tính, một phần, là một lĩnh vực phụ khác của Trí Tuệ Nhân Tạo, chủ yếu dựa vào Học Máy và các thuật toán Học Sâu để xây dựng các ứng dụng Thị Giác Máy Tính. Ngoài ra, Thị Giác Máy Tính bao gồm một số công nghệ hoạt động cùng nhau: Đồ họa máy tính (Computer graphics), Xử lý hình ảnh (Image processing), Xử lý tín hiệu (Signal processing), Công nghệ cảm biến (Sensor technology), Toán học (Mathematics) hoặc thậm chí Vật lý (Physics).

Từ các khái niệm trên ta có thể khái quát mối quan hệ giữa AI, ML, DL, CV thông qua sơ đồ sau:



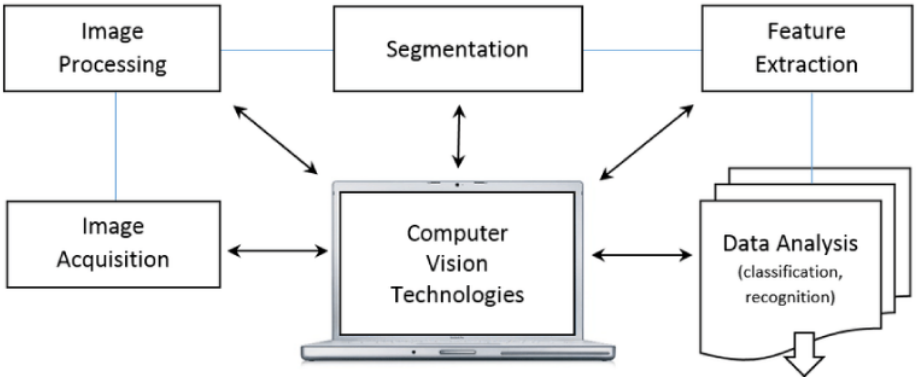
Hình 2.2 Sơ đồ mối quan hệ giữa AI, ML, DL, CV

2.2 Xử lý ảnh

2.2.1 Khái niệm

Xử lý ảnh (Image Processing), viết tắt là XLA, là đối tượng nghiên cứu của lĩnh vực Thị Giác Máy Tính, là quá trình biến đổi từ một ảnh ban đầu sang một ảnh mới với các đặc tính và tuân theo ý muốn của người sử dụng. Xử lý ảnh có thể gồm quá trình phân tích, phân lớp các đối tượng, làm tăng chất lượng, phân đoạn và tách cạnh, gán nhãn cho vùng hay quá trình biên dịch các thông tin hình ảnh của ảnh.

2.2.2 Các quá trình xử lý ảnh



Hình 2.3 Các giai đoạn chính trong xử lý ảnh. (Ảnh Internet)

- Thu nhận ảnh (Image acquisition): Đây là công đoạn đầu tiên mang tính quyết định đối với quá trình XLA. Ảnh đầu vào sẽ được thu nhận qua các thiết bị như camera, sensor, máy scanner,… và sau đó các tín hiệu này sẽ được số hóa. Các thông số quan trọng ở bước này là độ phân giải, chất lượng màu, dung lượng bộ nhớ và tốc độ thu nhận ảnh của các thiết bị.
- Tiền xử lý (image processing): Ở bước này, ảnh sẽ được cải thiện về độ tương phản, khử nhiễu, khử bóng, khử độ lệch,… với mục đích làm cho chất lượng ảnh trở lên tốt hơn nữa, chuẩn bị cho các bước xử lý phức tạp hơn về sau trong quá trình XLA. Quá trình này thường được thực hiện bởi các bộ lọc.
- Phân đoạn ảnh (segmentation): phân đoạn ảnh là bước then chốt trong XLA. Giai đoạn này phân tích ảnh thành những thành phần có cùng tính chất nào đó dựa theo biên hay các vùng liên thông. Tiêu chuẩn để xác định các vùng liên thông có thể là cùng màu, cùng mức xám… Mục đích của phân đoạn ảnh là để có một miêu tả tổng hợp về nhiều phần tử khác nhau cấu tạo lên ảnh thô.
- Trích xuất đặc trưng (feature extraction): Chuyển đổi dữ liệu ảnh thô thành một dạng thích hợp hơn cho việc xử lý. Để chuyển đổi chúng, tùy thuộc vào mục đích mà biểu diễn một vùng ảnh dưới dạng biên hoặc dưới dạng một vùng hoàn chỉnh gồm tất cả những điểm ảnh thuộc về nó. Ngoài ra chúng ta còn phải đưa ra một phương pháp mô tả dữ liệu đã được chuyển đổi đó sao cho những tính chất cần quan tâm đến sẽ được làm nổi bật lên, thuận tiện cho việc xử lý chúng.
- Phân tích dữ liệu (data analysis): Đây là bước cuối cùng trong quá trình XLA. Trong đó có quá trình nhận dạng ảnh có thể được nhìn nhận một cách đơn giản là việc gán nhãn cho các đối tượng trong ảnh. Giải thích/phân loại là công đoạn gán nghĩa cho một tập các đối tượng đã được nhận biết.

2.2.3 Một số hệ màu cơ bản

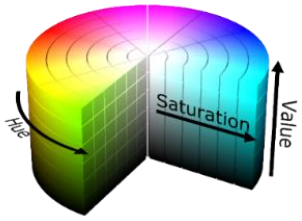
Không gian màu (Colour Space) được hiểu là các mô hình toán để miêu tả màu sắc. Mỗi không gian màu đều có một tác dụng và ứng dụng trong các bài toán khác nhau.

- **Hệ màu RGB**: Là không gian màu phổ biến dùng trong máy tính, máy ảnh, điện thoại và nhiều thiết bị kĩ thuật số khác nhau. Không gian màu này khá gần với cách mắt người tổng hợp màu sắc. Nguyên lý cơ bản là sử dụng 3 màu sắc cơ bản R (red - đỏ), G (green - xanh lục) và B (blue - xanh lam) để biểu diễn tất cả các màu sắc. Thông thường, trong mô hình 24 bit mỗi kênh màu sẽ sử dụng 8bit để biểu diễn, tức là giá trị R, G, B nằm trong khoảng 0 - 255. Bộ 3 số này biểu diễn cho từng điểm ảnh, mỗi số biểu diễn cho cường độ của một màu. Với mô hình màu 24bit thì số màu tối đa có thể tạo ra là 255 x 255 x 255 = 16581375 màu. Một điểm cần lưu ý là với các thư viện đọc ảnh và hiển thị ảnh như matplotlib[9], Pillow[10] thì các ảnh được đọc theo RGB tuy nhiên Opencv[11] đọc ảnh theo các kênh BGR.

Hình 2.4 Hệ màu RGB. (Ảnh Internet)

- Không gian màu HSV (còn gọi là HSB) là một cách tự nhiên hơn để mô tả màu sắc, dựa trên 3 số liệu:
  - H: (Hue) là phần màu của mô hình màu và được biểu thị dưới dạng một số từ 0 đến 360 độ.
  - S: (Saturation) Độ bão hòa là lượng màu xám trong màu, từ 0 đến 100 phần trăm. Một hiệu ứng mờ nhạt có thể có được từ việc giảm độ bão hòa về không để giới thiệu nhiều màu xám hơn.
  - B (hay V): (Bright hay Value) Độ sáng hay giá trị hoạt động kết hợp với độ bão hòa và mô tả độ sáng hoặc cường độ của màu sắc, từ 0-100 phần trăm, trong đó 0 là hoàn toàn đen và 100 là sáng nhất và cho thấy màu sắc nhất.

[9] Matplotlib là một thư viện vẽ đồ thị cho ngôn ngữ lập trình Python và phần mở rộng toán học số NumPy của nó
[10] Pillow là một phần của PIL- Python Image Library, thư viện xử lý ảnh của Python.
[11] OpenCV là một thư viện các chức năng lập trình chủ yếu nhắm vào thị giác máy tính thời gian thực.



Hình 2.5 Hệ màu HSV. (Ảnh Internet)

2.2.4 Biểu diễn ảnh

Ảnh trong thực tế là một ảnh liên tục cả về không gian và giá trị độ sáng. Để có thể xử lý ảnh bằng máy tính thì cần thiết phải tiến hành số hóa ảnh. Quá trình số hóa biến đổi các tín hiệu liên tục sang tín hiệu rời rạc thông qua quá trình lấy mẫu (rời rạc hóa về không gian) và lượng tử hóa các thành phần giá trị mà về nguyên tắc bằng mắt thường không thể phân biệt được hai điểm liền kề nhau. Các điểm như vậy được gọi là các pixel (Picture Element) hay các phần tử ảnh hoặc điểm ảnh. Ở đây cần phân biệt khái niệm pixel hay đề cập đến trong các hệ thống đồ họa máy tính. Để tránh nhầm lẫn ta gọi khái niệm pixel này là pixel thiết bị. Khái niệm pixel thiết bị có thể xém xét như sau: Khi ta quan sát màn hình (trong chế độ đồ họa), màn hình không liên tục mà gồm các điểm nhỏ, gọi là pixel. Mỗi pixel gồm một tập tọa độ (x, y) và màu.

Như vậy mỗi ảnh là tập hợp các điểm ảnh. Khi được số hóa nó thường được biểu diễn bởi mảng 2 chiều I(n,p): n là dòng và p là cột.

Về mặt toán học có thể xem ảnh là một hàm hai biến f(x,y) với x, y là các biến tọa độ. Giá trị số ở điểm (x,y) tương ứng với giá trị xám hoặc độ sáng của ảnh (x là các cột còn y là các hàng). Giá trị của hàm ảnh f(x,y) được hạn chế trong phạm vi của các số nguyên dương.

$$0 \leq f(x,y) \leq f_{max.}$$

Mô hình Vector: Ngoài mục đích tiết kiệm không gian lưu trữ, dễ dàng hiển thị và in ấn, các ảnh biểu diễn theo mô hình vector còn có ưu điểm cho phép dễ dàng lựa chọn, sao chép, di chuyển, tìm kiếm…Theo những yêu cầu này thì kỹ thuật biểu diễn vector tỏ ra ưu việt hơn. Trong mô hình này, người ta sử dụng hướng vector của các điểm ảnh lân cận để mã hóa và tái tạo lại hình ảnh ban đầu. Các ảnh vector được thu nhận trực tiếp từ các thiết bị số hóa hoặc được chuyển đổi từ các ảnh Raster thông qua các chương trình vector hóa. Một số định dạng như EPS, AI và PDF.

Mô hình Raster: là mô hình biểu diễn ảnh thông dụng nhất hiện nay. Ảnh được biểu diễn dưới dạng ma trận các điểm ảnh. Tùy theo nhu cầu thực tế mà mỗi điểm ảnh có thể được biểu diễn bởi một hay nhiều bit. Mô hình Raster thuận lợi cho việc thu nhận, hiển thị và in ấn. Các ảnh được sử dụng trong phạm vi của đề tài này cũng là các ảnh được biểu diễn theo mô hình Raster. Các file ảnh raster bao gồm JPEG, GIF và PNG.

2.2.5 Độ phân giải ảnh

Độ phân giải (Resolution) của ảnh là mật độ điểm ảnh được ấn định trên một ảnh số được hiển thị. Khoảng cách giữa các điểm ảnh phải được chọn sao cho mắt người vẫn thấy được sự liên tục của ảnh. Việc lựa chọn khoảng cách thích hợp tạo nên một mật độ phân giải và được phân bố theo trục x và y trong không gian hai chiều.

2.2.6 Phạm vi ứng dụng của xử lý ảnh

Xử lý ảnh đã đem lại nhiều ứng dụng trong nhiều lĩnh vực khác nhau: y học, khoa học hình hình sự, khí tượng thuỷ văn, quản lý, sản xuất...

2.3 Phần mềm Nicepage

Nicepage là một trình xây dựng trang web mới mẻ, khác biệt với các trình xây dựng phổ biến khác trên thị trường. Nó nhằm mục đích trả lại quyền tự do sáng tạo cho các nhà thiết kế web, với trình chỉnh sửa kéo và thả nâng cao cho phép nhà thiết kế web tự do sáng tạo tối đa.

Các tính năng chính:

- Tự do kéo thả các thành phần của trang web
- Hỗ trợ Responsive
- Hỗ trợ các hiệu ứng animation
- Hỗ trợ các khối tạo sẵn
- Hỗ trợ nhiều mẫu trang web tạo sẵn

Cung cấp miễn phí giới hạn tính năng và bản trả phí sử dụng nhiều tính năng hơn. Tuy nhiên với nhu cầu thiết kế trang web đơn giản với thời gian nhanh chóng và miễn phí nên tôi chọn phần mềm này để thiết kế trang web của mình.

2.4 Ngôn ngữ lập trình python

2.4.1 Giới thiệu về Python

Python là ngôn ngữ lập trình hướng đối tượng, cấp cao, thông dịch, mạnh mẽ, được tạo ra bởi Guido van Rossum. Nó dễ dàng để tìm hiểu và đang nổi lên như một trong những ngôn ngữ lập trình nhập môn tốt nhất cho người lần đầu tiếp xúc với ngôn ngữ lập trình. Python hoàn toàn tạo kiểu động và sử dụng cơ chế cấp phát bộ nhớ tự động. Python có cấu trúc dữ liệu cấp cao mạnh mẽ và cách tiếp cận đơn giản nhưng hiệu quả đối với lập trình hướng đối tượng. Cú pháp lệnh của Python là điểm cộng vô cùng lớn vì sự rõ ràng, dễ hiểu và cách gõ linh động làm cho nó nhanh chóng trở thành một ngôn ngữ lý tưởng để viết script và phát triển ứng dụng trong nhiều lĩnh vực, ở hầu hết các nền tảng.

2.4.2 Lý do Python được sử dụng cho các dự án về AI và Machine Learing

- **Python đơn giản và nhất quán:** Python cho phép các lập trình viên viết code ngắn gọn và dễ đọc, các lập trình viên sẽ tập trung được thời gian, trí lực vào giải quyết bài toán của Học Máy thay vì gặp rắc rối với kỹ thuật của ngôn ngữ. Và vì Python là ngôn ngữ có mục đích chung, nó có thể thực hiện một tập hợp các tác vụ Học Máy phức tạp và cho phép xây dựng các nguyên mẫu nhanh chóng, kiểm thử sản phẩm của mình cho mục đích học máy tốt hơn.
- **Python có nhiều lựa chọn về thư viện và framework:** Việc thực hiện các thuật toán AI và ML có thể rất khó và mất nhiều thời gian nên có một môi trường được cấu trúc tốt và được thử nghiệm tốt là rất quan trọng. Vì thế, để giảm thời gian phát triển dự án, các lập trình viên chuyển sang một số framework và thư viện của Python. Một framework / thư viện có thể được hiểu và các code được viết sẵn mà các lập trình viên có thể sử dụng ngay để giải quyết các tác vụ lập trình phổ biến. Quan điểm là "không phát minh lại bánh xe". Python có một kho công nghệ phong phú bao gồm rất nhiều thư viện cho trí tuệ nhân tạo và học máy: numpy, opencv, keras, TensorFlow,…
- **Python độc lập với nền tảng:** Độc lập với nền tảng tức là một ngôn ngữ lập trình hoặc framework cho phép các lập trình viên triển khai mọi thứ trên một máy và sử dụng chúng trên một máy khác mà không có bất kỳ thay đổi nào (hoặc chỉ thay đổi tối thiểu).
- **Cộng đồng Python rất lớn:** Trong Khảo sát Lập trình viên năm 2020 của StackOverflow[12], Python đứng thứ 4 trong top 10 ngôn ngữ lập trình phổ biến nhất, điều này có nghĩa là ta có thể tìm thấy các tài liệu học Python, cộng đồng hỗ trợ dễ dàng. Qua

[12] Số liệu lấy từ https://insights.stackoverflow.com/survey/2020

nhiều năm phát triển (từ lúc Python được sinh ra năm 1994) cộng đồng Python AI đã phát triển trên toàn thế giới, kể cả Việt Nam. Có nhiều diễn đàn, group tích cực trao đổi về kinh nghiệm và giải pháp liên quan đến Học máy. Đối với bất kỳ nhiệm vụ nào ta cần giải quyết, tỷ lệ khá cao là đã có người nào đó trên cộng đồng đã xử lý thành công.

2.4.3 Một số thư viện liên quan

- Numpy là một thư viện lõi phục vụ cho khoa học máy tính của Python, hỗ trợ cho việc tính toán các mảng nhiều chiều, có kích thước lớn với các hàm đã được tối ưu áp dụng lên các mảng nhiều chiều đó. Numpy đặc biệt hữu ích khi thực hiện các hàm liên quan tới Đại Số Tuyến Tính. Trong numpy, chiều của mảng gọi là *axes,* số chiều gọi là *rank.*

- Pandas trong python là một thư viện mã nguồn mở, hỗ trợ đắc lực trong thao tác dữ liệu. Đây cũng là bộ công cụ phân tích và xử lý dữ liệu mạnh mẽ của ngôn ngữ lập trình python. Thư viện này được sử dụng rộng rãi trong cả nghiên cứu lẫn phát triển các ứng dụng về khoa học dữ liệu. Thư viện này sử dụng một cấu trúc dữ liệu riêng là Dataframe. Pandas cung cấp rất nhiều chức năng xử lý và làm việc trên cấu trúc dữ liệu này. Chính sự linh hoạt và hiệu quả đã khiến cho pandas được sử dụng rộng rãi.

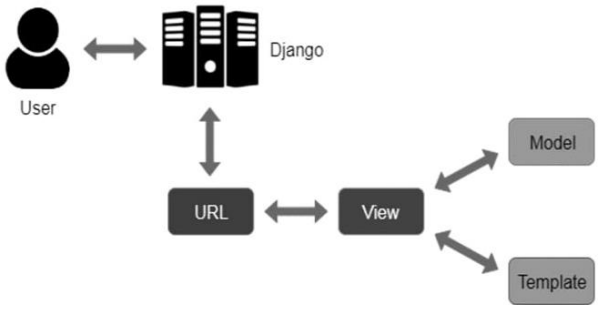
2.5 Django framework

2.5.1 Ưu điểm của Django:

1. Sử dụng ngôn ngữ lập trình Python
2. Sử dụng ORM, Mô hình MVC (MTV)
3. Hỗ trợ Multi-Site, đa ngôn ngữ
4. Nhiều tài liệu và cộng đồng hỗ trợ
5. Hỗ trợ giao diện quản lý Admin
6. Dễ dàng mở rộng

2.5.2 Mô hình MTV (Model-Template-Views) trong Django

Django sử dụng mô hình MTV tương tự như mô hình MVC (Model-View-Controller) trong các framework khác.



Hình 2.6 Mô hình MTV của Django. (Ảnh Internet)

Bảng 2.1 Mô tả chức năng các thành phần trong mô hình MTV.

Thành phần	Mô tả
Model	Nơi thiết kế ra những table cho database, từ đó Django ORM đã cung cấp những phương thức xử lý, nghiệp vụ lên database
Template	Là những mẫu được thiết kế ra và xử lý output ra mã html/css cho trang web
Views	Các hàm để xử lý khi có request từ người dùng

2.5.3 Cấu trúc Project Django

- **__init__.py**: Đây là 1 file cơ bản trong Python dùng để biến folder chứa nó thành package, giúp import các tệp.
- **setttings.py**: Đây là file cấu hình dự án. (VD: cấu hình database, đặt múi giờ, cài thêm thư viện, ...).
- **urls.py**: Đây là tệp giúp tạo các đường dẫn URL của trang web để liên kết các webpage lại với nhau.
- **wsgi.py**: Đây là tệp giúp chúng ta triển khai website lên server.

2.6 Pytorch Framework

2.6.1 Tensor trong Pytorch

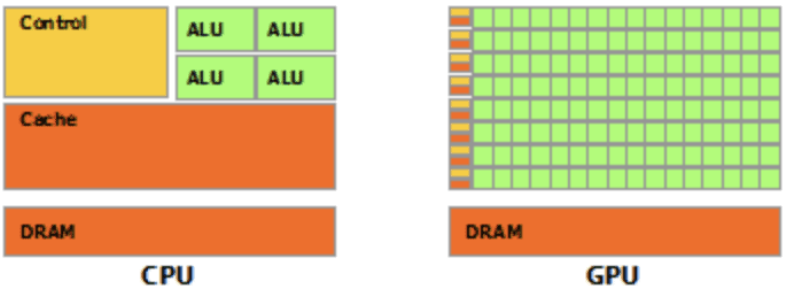
Tensor cũng giống như numpy array nhưng được chuyển sang tensor để sử dụng tính toán trên GPU. Các phép toán, biến đổi và các hoạt động cơ bản của numpy đều có

thể thực hiện được trên Tensor. Việc chuyển đổi từ numpy array sang tensor và ngược lại được Pytorch hỗ trợ rất linh hoạt bằng lệnh *from_numpy()* và *numpy()*.

2.6.2 CUDA tensor

a) Bộ Xử Lý Đồ Họa (GPU)

GPU là một bộ xử lý có khả năng xử lý tốt các phép tính chuyên biệt. Điều này trái ngược với bộ xử lý trung tâm (CPU), là bộ xử lý có khả năng xử lý tốt các tính toán chung. CPU là bộ xử lý cung cấp năng lượng cho hầu hết các tính toán điển hình trên các thiết bị điện tử. GPU có thể nhanh hơn nhiều so với CPU, Tuy nhiên, không phải luôn luôn là như vậy. Tốc độ của GPU so với CPU phụ thuộc vào loại tính toán được thực hiện. Loại tính toán phù hợp nhất cho GPU là loại tính toán có thể được thực hiện song song. Về mặt kiến trúc, sự khác biệt chính giữa CPU và GPU là CPU thường có số lõi hạn chế để thực hiện các phép toán số học. Ngược lại, một GPU có thể có hàng trăm và hàng nghìn lõi.



Hình 2.7 So sánh kiến trúc CPU và GPU. (Ảnh Internet)

b) Tính toán song song

Tính toán song song là một loại tính toán mà bằng một phép tính cụ thể được chia thành các phép tính nhỏ hơn độc lập có thể được thực hiện đồng thời. Các phép tính kết quả sau đó được tổng hợp lại hoặc đồng bộ hóa để tạo thành kết quả của phép tính lớn ban đầu. Số lượng tác vụ mà một tác vụ lớn hơn có thể được chia thành tùy thuộc vào số lượng lõi chứa trên một phần cứng cụ thể. Lõi là các đơn vị thực sự thực hiện tính toán trong một bộ xử lý nhất định và CPU thường có bốn, tám hoặc mười sáu lõi trong khi GPU có thể có hàng nghìn.

Như vậy chúng ta có thể kết luận rằng tính toán song song được thực hiện bằng GPU và các tác vụ phù hợp nhất để giải quyết bằng GPU là các tác vụ có thể được thực

hiện song song. Nếu tính toán có thể được thực hiện song song, chúng ta có thể tăng tốc tính toán của mình bằng cách sử dụng các phương pháp lập trình song song và GPU.

Tuy nhiên việc chuyển các tác vụ tính toán tương đối nhỏ sang GPU sẽ không làm tăng tốc và thậm chí có thể làm chậm hơn. GPU hoạt động tốt cho các tác vụ có thể được chia thành nhiều tác vụ nhỏ hơn và nếu một tác vụ tính toán đã nhỏ, chúng ta sẽ không thu được nhiều lợi ích bằng cách chuyển tác vụ đó sang GPU. Vì lý do này, việc chỉ sử dụng CPU khi mới bắt đầu thường được ưu tiên hơn và khi chúng ta giải quyết các vấn đề phức tạp hơn thì lúc đó hãy nghĩ đến việc sử dụng GPU nhiều hơn.

c) CUDA

CUDA là nền tảng lập trình tính toán và mô hình lập trình được phát triển bởi Nvidia[13] cho việc tính toán nói chung trên chính các nhân GPU của hãng. CUDA cho phép lập trình viên tăng tốc các ứng dụng tính toán bằng cách khai thác sức mạnh của GPU cho các phần tính toán song song.

d) CUDA trong Deep Learning

Deep learning có yêu cầu tính toán tốc độ rất cao. Ví dụ, để huấn luyện cho mô hình về dịch thuật Google Translate năm 2016, Google Brain và Google Translate đã thử nghiệm hàng trăm lần, mỗi lần kéo dài một tuần để chạy TensorFlow sử dụng GPU. Họ đã mua 2000 server được trang bị GPU từ Nvidia. Nếu không có GPU, việc training sẽ mất hàng tháng thay vì trong 1 tuần training.

Hầu hết các Deep Learning framework sử dụng thư viện cuDNN cho việc tính toán Deep Neural Network. Thư viện này cũng rất quan trọng trong việc training của các mô hình Deep Learning. Khi CUDA và cuDNN được cải tiến liên tục qua các phiên bản, các ứng dụng Deep Learning sẽ tiếp tục được hưởng lợi về mặt tốc độ.

e) CUDA tensor trong PyTorch

Thư viện CUDA của PyTorch cho phép theo dõi GPU đang sử dụng và khiến bất kỳ bộ tensor nào được tạo được tự động gán cho thiết bị đó, để đưa một tensor lên một thiết bị bất kì ta sử dụng hàm *to(), cuda()*. Sau khi một tensor được cấp phát, ta có thể thực hiện các thao tác với nó và kết quả cũng được gán cho cùng một thiết bị. Kiểm tra thiết bị có hỗ trợ CUDA không bằng lệnh *torch.cuda.is_available()*.

[13] nVIDIA, một tập đoàn đa quốc gia, chuyên về phát triển bộ xử lý đồ họa và công nghệ chipset cho các máy trạm, máy tính cá nhân, và các thiết bị di động

- **Thực thi không đồng bộ:**

Các hoạt động của GPU là không đồng bộ theo mặc định để cho phép thực hiện song song số lượng lớn hơn các phép tính. Các hoạt động không đồng bộ nói chung là vô hình đối với người dùng vì PyTorch tự động đồng bộ hóa dữ liệu được sao chép giữa CPU và GPU hoặc GPU và GPU.

Một điểm cần lưu ý về việc sử dụng các hoạt động không đồng bộ hay đồng bộ là với các phép đo thời gian. Với các hoạt động không đồng bộ, các phép đo sẽ không chính xác. Để giải quyết vấn đề này trong khi vẫn bật chế độ không đồng bộ có thể gọi *torch.cuda.synchronize()* trước khi đo hoặc có thể sử dụng *torch.cuda.Event* để ghi lại thời gian.

- **PyTorch Autograd:**

Chức năng của autograd: Tự động tính toán đạo hàm trên toàn bộ các toán tử của tensor. Nó là một framework được định nghĩa trong quá trình chạy, có nghĩa rằng quá trình lan truyền ngược được xác định khi mà code được chạy, và do đó mỗi vòng lặp có thể có kết quả thay đổi tham số theo lan truyền ngược khác nhau.

Theo dõi lịch sử của tensor torch: torch.tensor là package khởi tạo các tensor torch. Mỗi một tensor torch sẽ có 1 thuộc tính là *.requires_grad*, nếu ta set thuộc tính này về *True*, các toán tử triển khai trên tensor sẽ được theo dõi. Khi kết thúc quá trình lan truyền thuận (hoặc quá trình tính toán output) ta có thể gọi *.backward()* và mọi tính toán gradient sẽ được tự động thực hiện dựa trên lịch sử đã được lưu lại. Các gradient cho tensor này sẽ được tích lũy và xem tại thuộc tính *.grad*.

Ngoài ra để ngăn tensor lưu lại lịch sử (và sử dụng bộ nhớ), chúng ta cũng có thể bao quanh code block triển khai tensor với hàm *with torch.no_grad():* nó rất hữu ích trong trường hợp đánh giá mô hình khi sử dụng lệnh *model.eval()* bởi vì khi thuộc tính *requires_grad = True* thì mô hình sẽ có thể được cập nhật tham số. Nhưng quá trình đánh giá mô hình sẽ không cần cập nhật tham số nên chúng ta không cần áp dụng gradient lên chúng. Đơn giản là set *requires_grad = False*.

2.6.3 Cấu trúc của một mô hình Học Sâu trong PyTorch

Chúng ta có thể tạo một mô hình là một lớp bằng cách kế thừa **nn.Module**, điều này cho phép tạo một mô hình học sâu dưới dạng một lớp. Lớp này có 2 hàm bắt buộc phải có đó là **init** và **forward**:

- **init** là một hàm khởi tạo nhận vào các biến, tham số giúp ta có thể khởi tạo các biến, hàm của đối tượng được khởi tạo. Vì class này kế thừa **nn.Module** nên khi khởi tạo một đối tượng mới của class thì phải khởi tạo lớp kế thừa nên trong hàm init luôn phải có *super().__init__()* . Ngoài ra hàm khởi tạo này chúng ta có thể khởi tạo các layer custom sử dụng trong mô hình, tạo backbone bằng cách tải mô hình pretrained, khởi tạo và thực thi một số hàm khác.
- **forward** là hàm nhận vào là dữ liệu input ban đầu. Dữ liệu sẽ đi lần lượt qua từng layer của mô hình và trả về output của mô hình.

2.7 Giới thiệu OpenCV

2.7.1 Khái niệm

OpenCV viết tắt cho Open Source Computer Vision Library. OpenCV là thư viện nguồn mở hàng đầu cho Computer Vision và Machine Learning, và hiện có thêm tính năng tăng tốc GPU cho các hoạt động theo real-time.

OpenCV có một cộng đồng người dùng khá hùng hậu hoạt động trên khắp thế giới bởi nhu cầu cần đến nó ngày càng tăng theo xu hướng chạy đua về sử dụng Computer Vision của các công ty công nghệ.

2.7.2 Ứng dụng của OpenCV

- Hình ảnh street view
- Kiểm tra và giám sát tự động
- Robot và xe hơi tự lái
- Phân tích hình ảnh y học
- Tìm kiếm và phục hồi hình ảnh/video
- Phim – cấu trúc 3D từ chuyển động
- Nghệ thuật sắp đặt tương tác

2.7.3 Cấu trúc dữ liệu ảnh trong OpenCV

Cấu trúc dữ liệu hình ảnh trong OpenCV được tổ chức dưới dạng một ma trận (số học) 3 chiều. Thứ tự các chiều của ma trận được sắp xếp theo chiều cao, chiều rộng và kênh màu (height, width, channel). Mỗi phần tử trong ma trận này có kiểu dữ liệu là số nguyên (0-255) hoặc số thực (0-1) mô tả giá trị của mức sáng (intensity).

Trong Xử Lý Ảnh, mức sáng của ảnh chạy trong miền giá trị 0-255. Màu đen mang giá trị 0, màu trắng mang giá trị 255. Màu xám đậm sẽ có giá trị lệch về 0 nhiều, và màu xám nhạt lệch về phía 255. Do đó, đối với ảnh xám ta chỉ cần một kênh màu để mô tả và lưu trữ, đối với ảnh màu được biểu diễn theo hệ màu BGR được giới thiệu ở trên, và ta cần tới 3 kênh màu để lưu trữ.

OpenCV (phiên bản Python) sử dụng cấu trúc dữ liệu Numpy, do đó ảnh đọc lên bằng câu lệnh *cv2.imread()* sẽ trả ra một numpy array. Các thao tác ta thực hiện sẽ xử lý trên numpy array. Do đó, ta thoải mái sử dụng các hàm có trong thư viện numpy lên ma trận ảnh.

2.8 Darknet: Open Source Neural Networks

Darknet là một framework open source chuyên biệt về phát hiện đối tượng được viết bằng ngôn ngữ C và CUDA. Các mô hình được huấn luyện trên Darknet nhanh hơn, đồng thời Darknet dễ cài đặt và hỗ trợ tính toán CPU và GPU. Cộng đồng sử dụng Darknet đông đảo, được hỗ trợ nhiệt tình. Để không tốn thời gian xây dựng lại mô-đun huấn luyện thuật toán YOLOv3 thì tôi sử dụng thư viện này để tận dụng những mã nguồn có sẵn để thực hiện huấn luyện mô hình song song với quá trình triển khai thuật toán, và tinh chỉnh các thông số trong lúc huấn luyện.

2.9 Google Colab

Google Colaboratory (gọi tắt là Google Colab hay Colab) là một sản phẩm của Google Research. Colab dựa trên Jupyter Notebook, người dùng có thể viết và thực thi đoạn mã python thông qua trình duyệt và đặc biệt rất phù hợp với Data Analysis, Machine Learning và giáo dục.

Sự phát triển mạnh mẽ của Machine learning và Deep learning trong những năm gần đây không chỉ bởi các thuật toán, các mô hình tân tiến liên tiếp ra đời mà còn bởi sự phát triển không ngừng của phần cứng, đặc biệt là GPU. Việc tính toán về toán học cho Deep Learning trên CPU có thể mất hàng tháng! Nhưng những tính toán này có thể được gán cho GPU để thực hiện nhanh hơn. Việc huấn luyện Neural Network trên CPU không được khuyến nghị. GPU cần thiết cho việc tính toán ở mức độ cao.

Như chúng ta đều biết, GPU rất đắt tiền. Ta có thể giải quyết vấn đề này bằng Colab, nó cung cấp một loạt các GPU miễn phí có sức mạnh to lớn về hiệu suất.

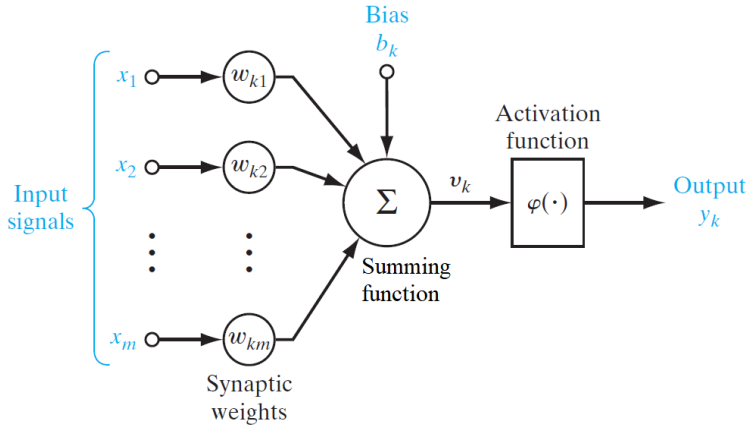
CHƯƠNG 3. MẠNG NEURAL TÍCH CHẬP

3.1 Giới thiệu về Mạng Neural và Mạng Neural lan truyền ngược

3.1.1 Định nghĩa

Mạng Neural Nhân Tạo, Artificial Neural Network (ANN) là một mô hình xử lý thông tin phỏng theo cách thức xử lý thông tin của các hệ neural sinh học. Nó được tạo nên từ một số lượng lớn các phần tử (neural) kết nối với nhau thông qua các liên kết (trọng số liên kết) làm việc như một thể thống nhất để giải quyết một vấn đề cụ thể nào đó. Một mạng neural nhân tạo được cấu hình cho một ứng dụng cụ thể (nhận dạng mẫu, phân loại dữ liệu,...) thông qua một quá trình học từ tập các mẫu huấn luyện. Về bản chất học chính là quá trình hiệu chỉnh trọng số liên kết giữa các neural. Các neural đơn lẻ được gọi là các Perceptron.

3.1.2 Cấu tạo Mạng Neural



Hình 3.1 Cấu trúc của một neural nhân tạo được gán nhãn k

Các thành phần cơ bản của một neural nhân tạo bao gồm:

- Tập các đầu vào: Là các tín hiệu vào (input signals) của neural, các tín hiệu này thường được đưa vào dưới dạng một vector **N** chiều.

- Tập các liên kết: Mỗi liên kết được thể hiện bởi một trọng số liên kết – Synaptic weight. Trọng số liên kết giữa tín hiệu vào thứ j với neural k thường được kí hiệu là $\mathbf{w_{kj}}$. Thông thường, các trọng số này được khởi tạo một cách ngẫu nhiên ở thời điểm khởi tạo mạng và được cập nhật liên tục trong quá trình học mạng.

- Hàm tổng (Summing function) thường dùng để tính tổng của tích các đầu vào với trọng số liên kết của nó.

• Bias (còn gọi là một độ lệch) là sự sai khác giữa trung bình dự đoán của mô hình chúng ta xây dựng với giá trị chính xác đang cố gắng để dự đoán.

• Một hàm kích hoạt (activation function) dùng để đưa các tín hiệu đầu ra của neural vào một miền giá trị nhất định hoặc vào một tập hợp các giá trị cố định.

• Đầu ra: Là tín hiệu đầu ra của neural, với mỗi neural sẽ có tối đa là một đầu ra.

Xét về mặt toán học, cấu trúc của một neural **k**, được mô tả bằng biểu thức sau:

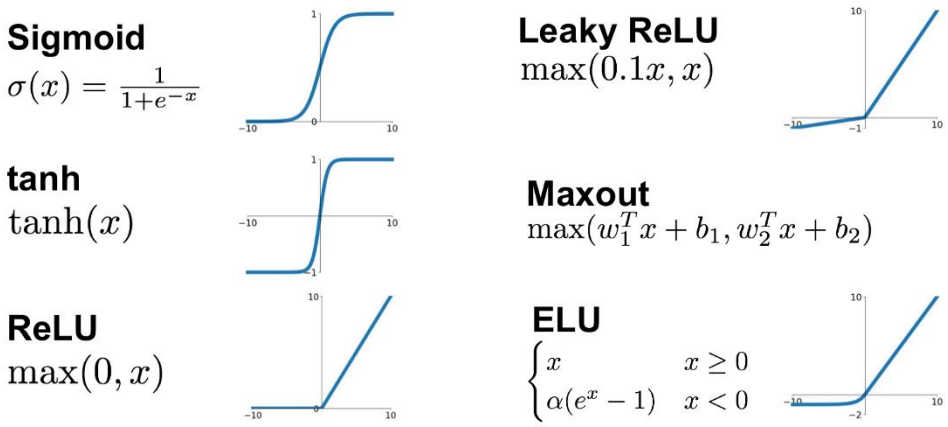
$$u_k = \sum_{j=1}^{m} w_{kj} x_j$$
$$v_k = u_k + b_k$$
$$y_k = \varphi(v_k)$$

Trong đó: $x_1, x_2, ..., x_m$: là các tín hiệu vào; $(w_{k1}, w_{k2}, ..., w_{km})$ là các trọng số liên kết của neural thứ **k**; u_k là hàm tổng; b_k là độ lệch; $\varphi(.)$ là hàm kích hoạt và y_k là tín hiệu đầu ra của neural có nhãn **k**.

Cách thức kết nối các neural trong mạng xác định kiến trúc của mạng. Các neural trong mạng có thể kết nối đầy đủ (fully connected), hoặc kết nối cục bộ (partially connected).

3.1.3 Một số hàm kích hoạt phổ biến

Hàm kích hoạt (activation function) mô phỏng tỷ lệ truyền xung qua axon của một neuron thần kinh. Trong một mạng nơ-ron nhân tạo, hàm kích hoạt đóng vai trò là thành phần phi tuyến tại output của các nơ-ron. Trong bài viết này, chúng ta sẽ cùng tìm hiểu các hàm kích hoạt phổ biến nhất và các ưu, nhược điểm của chúng.



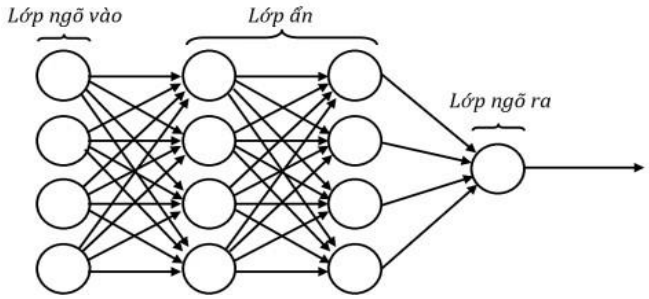
Hình 3.2 Các hàm kích hoạt phổ biến

3.1.4 Multi-layers Perceptron (MLP)

a) Giới thiệu mạng MLP

Mạng nhiều lớp truyền thẳng (MLP: Multi Layer Perceptron) là mạng có **n** (**n** ≥ 2) lớp (thông thường lớp đầu vào không được tính đến): Trong đó gồm một lớp đầu ra (lớp thứ **n**) và (**n**-1) lớp ẩn, bao gồm:

- Lớp ngõ vào (input layer), cung cấp dữ liệu từ ngoài vào mạng.
- Lớp ngõ ra (output layer) có nhiệm vụ thực hiện tính toán và truyền thông tin ra bên ngoài.
- Lớp ẩn (hidden layer) thực hiện nhận thông tin từ các lớp phía trước, tính toán và chuyển thông tin đến các lớp phía sau.



Hình 3.3 Cấu trúc mạng đa lớp MLP

Hoạt động của mạng MLP như sau: Tại lớp ngõ vào các neural nhận tín hiệu vào xử lý (tính tổng trọng số, gửi tới hàm kích hoạt rồi cho ra kết quả (là kết quả của hàm kích hoạt); kết quả này sẽ được truyền tới các neural thuộc lớp ẩn thứ nhất; các neural tại đây tiếp nhận như là tín hiệu đầu vào, xử lý và gửi kết quả đến lớp ẩn thứ 2, quá trình tiếp tục cho đến khi các neural thuộc lớp ra cho kết quả.

b, Huấn luyện mạng MLP

Các phương pháp học phổ biến:

Học là quá trình thay đổi hành vi của các vật theo một cách nào đó làm cho chúng có thể thực hiện tốt hơn trong tương lai.

Một mạng neural được huyấn luyện sao cho với một tập các vector đầu vào X, mạng có khả năng tạo ra tập các vector đầu ra mong muốn Y của nó. Tập X được sử dụng cho huấn luyện mạng được gọi là tập huấn luyện (training set). Các phần tử x thuộc X được gọi là các mẫu huấn luyện (training example). Quá trình huấn luyện bản chất là

sự thay đổi các trọng số liên kết của mạng. Trong quá trình này, các trọng số của mạng sẽ hội tụ dần tới các giá trị sao cho với mỗi vector đầu vào x từ tập huấn luyện, mạng sẽ cho ra vector đầu ra y như mong muốn.

Có ba phương pháp học phổ biến:

- Học có giám sát: Là quá trình học có sự tham gia giám sát của một "thầy giáo". Cũng giống như việc ta dạy một em nhỏ các chữ cái. Ta đưa ra một chữ "a" và bảo với em đó rằng đây là chữ "a". Việc này được thực hiện trên tất cả các mẫu chữ cái. Sau đó khi kiểm tra ta sẽ đưa ra một chữ cái bất kì (có thể viết hơi khác đi) và hỏi em đó đây là chữ gì? Như vậy với học có giám sát, số lớp cần phân loại đã được biết trước. Nhiệm vụ của thuật toán là phải xác định được một cách thức phân lớp sao cho với mỗi vector đầu vào sẽ được phân loại chính xác vào lớp của nó.
- Học không giám sát: Là việc học không cần có bất kỳ một sự giám sát nào. Trong bài toán học không giám sát, tập dữ liệu huấn luyện được cho dưới dạng: D = {(x_1, x_2, ..., x_N)}, với (x_1, x_2, ..., x_N) là vector đặc trưng của mẫu huấn luyện. Nhiệm vụ của thuật toán là phải phân chia tập dữ liệu D thành các nhóm con, mỗi nhóm chứa các vector đầu vào có đặc trưng giống nhau. Như vậy với học không giám sát, số lớp phân loại chưa được biết trước, và tùy theo tiêu chuẩn đánh giá độ tương tự giữa các mẫu mà ta có thể có các lớp phân loại khác nhau.
- Học tăng cường: Là sự tổ hợp của cả hai mô hình trên. Phương pháp này cụ thể như sau: với vector đầu vào, quan sát vector đầu ra do mạng tính được. Nếu kết quả được xem là "tốt" thì mạng sẽ được thưởng theo nghĩa tăng các trọng số kết nối lên; ngược lại mạng sẽ bị phạt, các trọng số kết nối không thích hợp sẽ được giảm xuống. Do đó học tăng cường là học theo nhà phê bình (critic), ngược với học có giám sát là học theo thầy giáo (teacher).

Học có giám sát trong các mạng neural:

Học có giám sát có thể được xem như việc xấp xỉ một ánh xạ: **X→ Y**, trong đó **X** là tập các vấn đề và **Y** là tập các lời giải tương ứng cho vấn đề đó. Các mẫu **(x, y)** với **x = (x_1, x_2, . . ., x_n) ∈ X, y = (yl, y2, . . ., ym) ∈ Y** được cho trước. Học có giám sát trong các mạng neural thường được thực hiện theo các bước sau:

- Bước 1: Xây dựng cấu trúc thích hợp cho mạng neural, chẳng hạn có

(**n** + 1) neural vào (**n** neural cho biến vào và 1 neural cho ngưỡng $\mathbf{x_0}$), **m** neural đầu ra, và khởi tạo các trọng số liên kết của mạng.

- Bước 2: Đưa một vector **x** trong tập mẫu huấn luyện **X** vào mạng
- Bước 3: Tính vector đầu ra **o** của mạng
- Bước 4: So sánh vector đầu ra mong muốn **y** (là kết quả được cho trong tập huấn luyện) với vector đầu ra **o** do mạng tạo ra; nếu có thể thì đánh giá lỗi.
- Bước 5: Hiệu chỉnh các trọng số liên kết theo một cách nào đó sao cho ở lần tiếp theo khi đưa vector **x** vào mạng, vector đầu ra **o** sẽ giống với y hơn.
- Bước 6: Nếu cần, lặp lại các bước từ 2 đến 5 cho tới khi mạng đạt tới trạng thái hội tụ. Việc đánh giá lỗi có thể thực hiện theo nhiều cách, cách dùng nhiều nhất là sử dụng lỗi tức thời: **Err = (o - y)**, hoặc **Err = |o - y|**; sai số toàn phương trung bình **(MSE): Err = $(o- y)^2/2$.**

Có hai loại lỗi trong đánh giá một mạng neural. Thứ nhất, gọi là lỗi rõ ràng (apparent error), đánh giá khả năng xấp xỉ các mẫu huấn luyện của một mạng đã được huấn luyện. Thứ hai, gọi là lỗi kiểm tra (test error), đánh giá khả năng tổng quá hóa của một mạng đã được huấn luyện, tức khả năng phản ứng với các vector đầu vào mới. Để đánh giá lỗi kiểm tra chúng ta phải biết đầu ra mong muốn cho các mẫu kiểm tra.

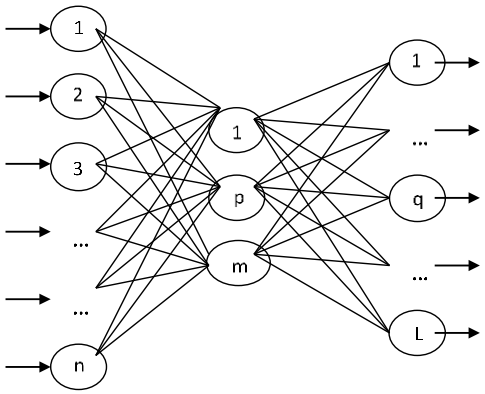
Thuật toán tổng quát ở trên cho học có giám sát trong các mạng neural có nhiều cài đặt khác nhau, sự khác nhau chủ yếu là cách các trọng số liên kết được thay đổi trong suốt thời gian học. Trong đó tiêu biểu nhất là thuật toán lan truyền ngược.

3.1.5 Mạng lan truyền ngược (Backpropagation Network)

Mạng neural lan truyền ngược có mô hình như sau:

Mạng có 3 lớp:

- Lớp vào (Input Layer) – số node vào là số thuộc tính của đối tượng cần phân lớp.
- Lớp ra (Output Layer) – Số node ra là số đặc điểm cần hướng tới của đối tượng (giá trị ra cần hướng đến – học có giám sát).
- Lớp ẩn (Hidden Layer) – Số node ẩn thường là không xác định trước, nó thường là do kinh nghiệm của người thiết kế mạng, nếu số node ẩn quá nhiều mạng sẽ cồng kềnh, quá trình học sẽ chậm, còn nếu số node ẩn quá ít làm mạng học không chính xác.



Hình 3.4 Mô hình mạng neural lan truyền ngược

Các neural ở các lớp trong thường được kết nối đầy đủ với tất cả các neural lớp ngoài, trên mỗi đường kết nối giữa 2 neural ở 2 lớp khác nhau có 1 trọng số mạng (weight). Các trọng số này đóng vai trò là các giá trị ẩn số mà mạng cần phải tìm ra (học) sao cho với các giá trị đầu vào, thông qua mạng ta nhận được kết quả xấp xỉ với đầu ra mong muốn tương ứng của mẫu học.

3.1.5 Một số vấn đề lưu ý khi xây dựng mạng MLP

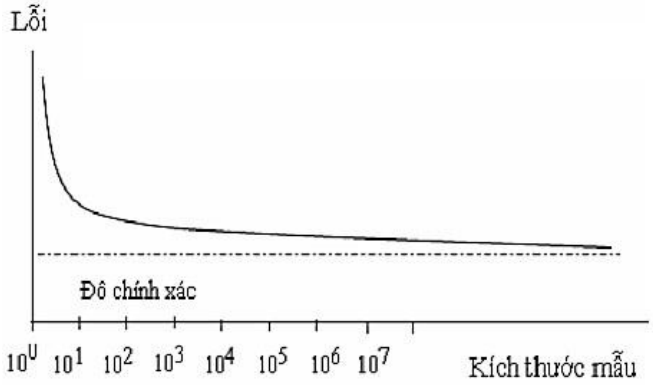
a) Xác định kích thước mẫu (dữ liệu đưa vào huấn luyện):

Hai yếu tố quan trọng ảnh hưởng đến kích thước mẫu:

- Dạng hàm đích: khi hàm đích càng phức tạp thì kích thước mẫu cần tăng.
- Nhiễu: khi dữ liệu bị nhiễu (thông tin sai hoặc thiếu thông tin) kích thước mẫu cần tăng.

Đối với mạng truyền thẳng, cho hàm đích có độ phức tạp nhất định, kèm một lượng nhiễu nhất định thì độ chính xác của mô hình luôn có một giới hạn nhất định. Nói cách khác độ chính xác của mô hình là hàm theo kích thước tập mẫu.

Trong học có giám sát, underfitting xảy ra khi mô hình không thể mô tả được các mẫu cơ bản của dữ liệu. Các mô hình này thường có bias cao và variance (phương sai) thấp. Hiện tượng này dễ xảy ra khi lượng dữ liệu huấn luyện quá ít hoặc người phân tích dữ liệu cố gắng mô tả các dữ liệu phức tạp bằng các mô hình đơn giản như hồi quy tuyến tính hay hồi quy logistic. Do đó muốn tránh gặp hiện tượng này, ta cần thu thập thêm dữ liệu mẫu và tăng độ phức tạp của mô hình (tăng thêm 1 vài layer, node).



Hình 3.5 Ảnh hưởng của kích thước mẫu trong quá trình huấn luyện mạng

b) Xác định số neural tầng ẩn

Câu hỏi chọn số lượng neural trong tầng ẩn của một mạng MLP thế nào là khó, nó phụ thuộc vào bài toán cụ thể và vào kinh nghiệm của nhà thiết kế mạng. Có nhiều đề nghị cho việc chọn số lượng neural tầng ẩn **h** trong một mạng MLP. Chẳng hạn **h** phải thỏa mãn **h>(p-1)/(n+2),** trong đó **p** là sốlượng mẫu huấn luyện và **n** là số lượng đầu vào của mạng. Càng nhiều nút ẩn trong mạng, thì càng nhiều đặc tính của dữ liệu huấn luyện sẽ được mạng nắm bắt, nhưng thời gian học sẽ càng tăng.

c, Hiện tượng overfitting

Xảy ra khi mạng được luyện quá khớp (quá sát) với dữ liệu huấn luyện (kể cả nhiễu), nên nó sẽ trả lời chính xác những gì đã được học, còn những gì không được học thì nó không quan tâm. Như vậy mạng sẽ không có được khả năng tổng quát hóa. Vấn đề overfitting xảy ra vì mạng có năng lực quá lớn. Có 3 cách để hạn chế bớt năng lực của mạng:

- Hạn chế số nút ẩn
- Sử dụng các phương pháp tăng dữ liệu (Data Augmentation), Sử dụng GAN (Generative Adverserial Network) trong học không giám sát.
- Ngăn không cho mạng sử dụng các trọng số lớn
- Giới hạn số bước luyện

Khi mạng được luyện, nếu ta phát hiện ra thời điểm mạng đạt đến trạng thái tốt nhất, ta có thể ngừng tiến trình luyện trước khi hiện tượng overfitting xảy ra. Ta biết rằng, chỉ có thể để đánh giá mức độ tổng quát hóa của mạng bằng cách kiểm tra mạng

trên các mẫu nó không được học. Ta thực hiện như sau: chia mẫu thành tập mẫu huấn luyện và tập mẫu kiểm tra. Luyện mạng với tập mẫu huấn luyện nhưng định kỳ dừng lại và đánh giá sai số trên tập mẫu kiểm tra. Khi sai số trên tập mẫu kiểm tra tăng lên thì overfitting đã bắt đầu và ta dừng tiến trình luyện.

3.2 Mạng Neural Tích chập

3.2.1 Giới thiệu Mạng Neural Tích chập

Những năm gần đây, ta đã chứng kiến được nhiều thành tựu vượt bậc trong ngành Thị giác máy tính (Computer Vision). Các hệ thống xử lý ảnh lớn như Facebook, Google hay Amazon đã đưa vào sản phẩm của mình những chức năng thông minh như phát hiện khuôn mặt người dùng, phát triển xe hơi tự lái hay drone (thiết bị bay không người lái) giao hàng tự động.

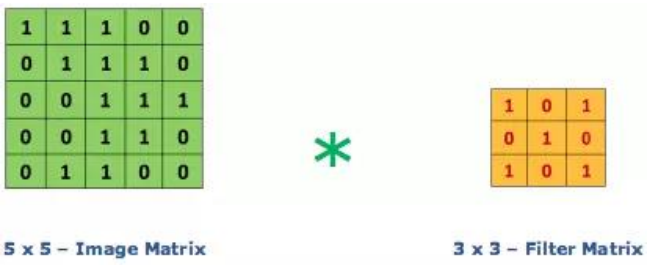
Convolutional Neural Network (CNN – Mạng nơ-ron tích chập) là một trong những mô hình Deep Learning tiên tiến giúp cho chúng ta xây dựng được những hệ thống thông minh với độ chính xác cao như hiện nay.

3.2.2 Các thành phần của một Mạng Neural Tích chập

a) Convolution Layer (Lớp tích chập)

Tích chập được sử dụng đầu tiên trong xử lý tín hiệu số (Signal processing). Nhờ vào nguyên lý biến đổi thông tin, các nhà khoa học đã áp dụng kĩ thuật này vào xử lý ảnh và video số.

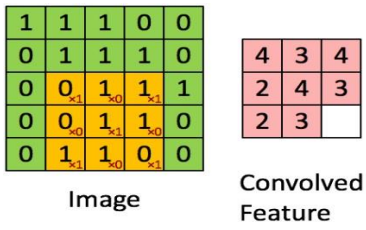
Để dễ hình dung, ta có thể xem tích chập như một cửa sổ trượt (sliding window) áp đặt lên một ma trận. Ta có thể theo dõi cơ chế của tích chập qua những hình ảnh minh họa bên dưới.



Hình 3.6 Bên trái là ma trận ảnh, bên phải là ma trận filter

Ma trận bên trái là một bức ảnh đen trắng. Mỗi giá trị của ma trận tương đương với một điểm ảnh (pixel), 0 là màu đen, 1 là màu trắng.

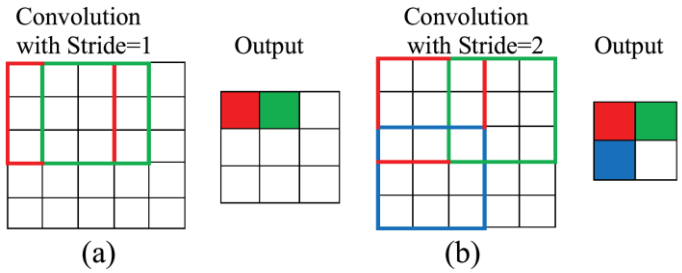
Sliding window còn có tên gọi là kernel, filter hay feature detector. Ở đây, ta dùng một ma trận filter 3×3 nhân từng thành phần tương ứng (element-wise) với ma trận ảnh bên trái. Giá trị đầu ra do tích của các thành phần này cộng lại. Kết quả của tích chập là một ma trận (convoled feature) sinh ra từ việc trượt ma trận filter và thực hiện tích chập cùng lúc lên toàn bộ ma trận ảnh bên trái. Như hình minh hoạ bên dưới:



Hình 3.7 Minh họa tích chập

Stride (bước nhảy)

Stride là số pixel thay đổi trên ma trận đầu vào. Khi stride là 1 thì ta di chuyển các kernel 1 pixel. Khi stride là 2 thì ta di chuyển các kernel đi 2 pixel và tiếp tục như vậy. Hình dưới là lớp tích chập hoạt động với stride 1 và stride 2.

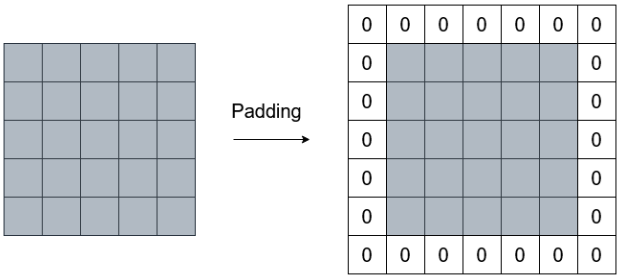


Hình 3.8 Tích chập với bước nhảy (a) là 1 và (b) là 2

Padding (đường viền)

Đôi khi kernel không phù hợp với hình ảnh đầu vào. Ta có 2 lựa chọn:

- Chèn thêm các số 0 vào 4 đường biên của hình ảnh (padding).
- Cắt bớt hình ảnh tại những điểm không phù hợp với kernel.



Hình 3.9 Minh hoạ padding khi thêm các số 0 vào biên của ảnh

Feature map (biểu đồ đặc trưng): Thể hiện kết quả mỗi lần ma trận filter quét qua input. Mỗi lần quét như thế sẽ xảy ra quá trình tính toán.

b) Nonlinear activation function (hàm kích hoạt phi tuyến)

Trong một mạng nơ-ron nhân tạo, hàm kích hoạt đóng vai trò là thành phần phi tuyến tại output của các nơ-ron.

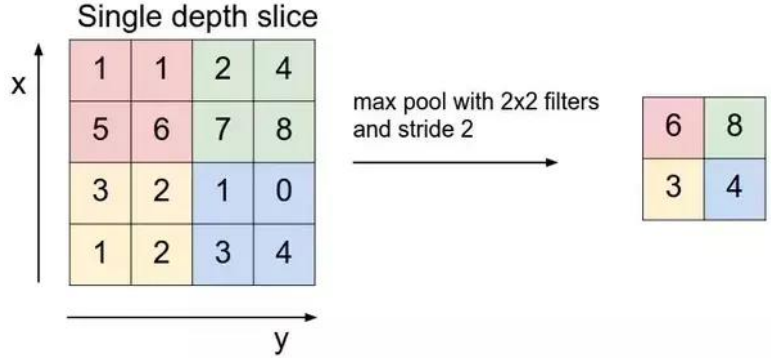
Có thể xem phần ***3.1.3 Một số hàm kích hoạt phổ biến*** để biết thêm chi tiết vì chức năng như nhau.

c) Pooling Layer (lớp tổng hợp)

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

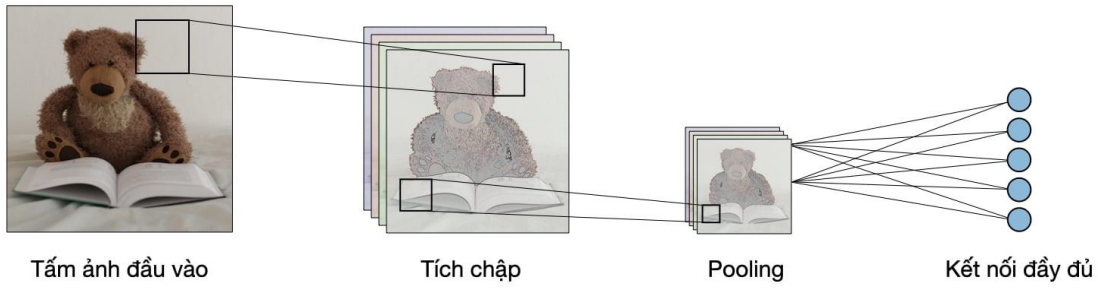
- Max Pooling
- Average Pooling
- Sum Pooling

Max pooling lấy phần tử lớn nhất từ ma trận đối tượng, hoặc lấy tổng trung bình (average pooling), tổng tất cả các phần tử trong map gọi là sum pooling.



Hình 3.10 Hàm max pooling với stride bằng 2

d) Fully connected layer (lớp kết nối đầy đủ)



Hình 3.11 Minh hoạ quá trình phân loại ảnh bằng CNN

Dùng để đưa ra kết quả. Ví dụ, sau khi các lớp convolutional layer và pooling layer đã nhận được các ảnh đã truyền qua nó, thì lúc đó ta sẽ thu được kết quả là mô hình đã đọc được khá nhiều thông tin về ảnh. Vì vậy, để liên kết các đặc điểm đó lại và cho ra output chúng ta dùng fully connected layer.

Ngoài ra, ở fully connected layer, thì khi có được các dữ liệu hình ảnh, chúng sẽ chuyển nó thành các mục có sự phân chia chất lượng. Giống như kiểu chia nó thành các phiếu bầu và sau đó sẽ đánh giá đề bầu cho hình ảnh đạt chất lượng tốt.

3.2.3 Mô hình Mạng Neural Tích chập

Mạng CNN gồm nhiều lớp Convolution chồng lên nhau, sử dụng các hàm kích hoạt phi tuyến để kích hoạt các trọng số. Mỗi một lớp sau khi được kích hoạt sẽ cho ra kết quả trừu tượng cho các lớp tiếp theo. Mỗi layer kế tiếp chính là thể hiện kết quả của layer trước đó.

Thông qua quá trình training, các lớp CNN tự động học các giá trị được thể hiện qua các lớp filter.

CNN có tính bất biến và tính kết hợp cục bộ (Location Invariance and Compositionality). Với cùng một đối tượng, nếu đối tượng này được chiếu theo các góc độ khác nhau (translation, rotation, scaling) thì độ chính xác của thuật toán sẽ bị ảnh hưởng đáng kể. Pooling layer sẽ cho ta tính bất biến đối với phép dịch chuyển (translation), phép quay (rotation) và phép co giãn (scaling).

Tính kết hợp cục bộ cho ta các cấp độ biểu diễn thông tin từ mức độ thấp đến mức độ cao và trừu tượng hơn thông qua convolution từ các filter. Đó là lý do tại sao CNN cho ra mô hình với độ chính xác rất cao. Cũng giống như cách con người nhận

biết các đối tượng trong tự nhiên, ta phân biệt được một con chó với một con mèo nhờ vào các đặc trưng từ mức độ thấp (có 4 chân, có đuôi) đến mức độ cao (dáng đi, hình thể, màu lông).

Cấu trúc cơ bản của CNN gồm 3 phần chính: Local receptive field, shared weights and bias, pooling:

- **Local receptive field:** hay còn gọi là các trường cục bộ. Tác dụng của lớp này chính là nó giúp chúng ta tách lọc các dữ liệu, thông tin của ảnh và chọn được những vùng ảnh có giá trị sử dụng nhất.

- **Shared weights and bias** (trọng số và độ lệch chia sẻ): Làm giảm tối đa số lượng các tham số là tác dụng chính của yếu tố này trong mạng CNN hiện nay. Bởi trong mỗi convolution có những feature map khác nhau, mỗi feature map lại giúp nhận diện một vài feature trong ảnh.

- **Pooling layer:** Đây gần như là lớp cuối cùng trước khi cho ra kết quả. Vì vậy, để có được kết quả dễ hiểu và dễ dùng nhất thì pooling layer sẽ có tác dụng làm đơn giản hóa thông tin đầu ra.Tức là, sau khi hoàn tất các quá trình tính toán và quét các lớp thì sẽ đi đến pooling layer để giảm lược bớt những thông tin không cần thiết, sau đó cho ra kết quả mà chúng ta mong muốn.

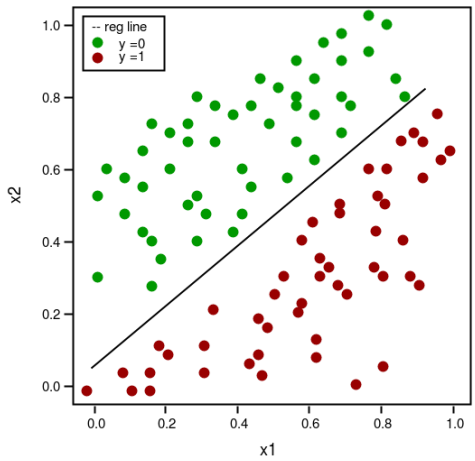
3.3 Hồi quy logistic (logistic regression)

3.3.1 Định nghĩa

Hồi quy logistic là một phương pháp phân tích thống kê được sử dụng để dự đoán giá trị dữ liệu dựa trên các quan sát trước đó của tập dữ liệu, thuộc thuật toán học có giám sát, đây là thuật toán đơn giản nhưng lại rất hiệu quả trong bài toán phân loại (Classification), được áp dụng trong bài toán phân loại nhị phân (Binary classification) tức ta sẽ có hai output, hoặc có thể gọi là hai nhãn.

Mục đích của hồi quy logistic là ước tính xác suất của các sự kiện, bao gồm xác định mối quan hệ giữa các tính năng từ đó đự đoán xác suất của các kết quả, nên đối với hồi quy logistic ta sẽ có:

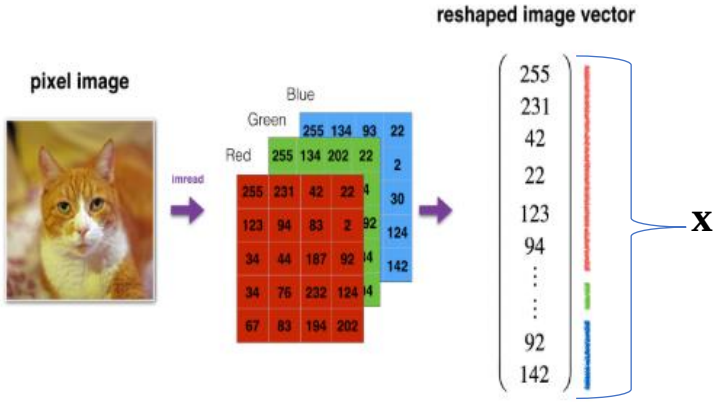
- **Input:** dữ liệu input (ta sẽ coi có hai nhãn là 0 và 1).
- **Output:** Xác suất dữ liệu input rơi vào nhãn 0 hoặc nhãn 1.



Hình 3.12 Minh hoạ dataset hồi quy logistic với 2 nhãn 0 và 1

3.3.2 Thiết lập bài toán trong phân loại ảnh

Trong bài toán này sẽ có hai nhãn là 0 và 1, input sẽ là bức ảnh. Như ta đã biết một bức ảnh màu sẽ có ba kênh màu là R, G, B với một bức ảnh có kích cỡ là $w*h$ thì sẽ có tất cả $w*h*3$ điểm ảnh (nhân với 3 vì có ba kênh màu), số điểm ảnh chính là số feature của vectơ đầu vào **x**. Ở hình dưới ta có thể thấy từ một bức ảnh màu ta sẽ reshape về một vectơ cột **x** để làm input cho mô hình logistic.



Hình 3.13 Minh hoạ quá trình reshape ảnh thành vector

Với mỗi bức ảnh ta sẽ có một input **x** nên với **m** bức ảnh ta sẽ có **m** input, việc lấy **m** bức ảnh làm input là bởi vì bài toán học máy có giám sát cần nhiều dữ liệu đầu vào để huấn luyện mô hình, một bức ảnh là quá ít để cho mô hình có thể học được.

3.3.3 Xây dựng mô hình

a) Kí hiệu

Xét trên một điểm dữ liệu ta có input **x** = [x_1;x_2;…;x_n] sẽ là một vector cột, ta sẽ ngăn cách các feature ($\mathbf{x_i}$) bằng dấu ";". input **x** sẽ có dạng như bên dưới:

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

Xét trên toàn bộ bộ dữ liệu (**m** điểm dữ liệu) ta sẽ có một vector hàng **X** = [$\mathbf{x}^{(1)}$,$\mathbf{x}^{(2)}$,…,$\mathbf{x}^{(m)}$] với mỗi cột là một điểm dữ liệu $\mathbf{x}^{i}$ ngăn cách nhau bởi dấu ",":

$$\mathbf{X} = [\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(m)}] = \begin{bmatrix} x_1^{(1)} & x_1^{(2)} & \ldots & x_1^{(m)} \\ x_2^{(1)} & x_2^{(2)} & \ldots & x_2^{(m)} \\ \vdots & \vdots & \ddots & \vdots \\ x_n^{(1)} & x_n^{(2)} & \ldots & x_n^{(m)} \end{bmatrix}$$

Ta có thể thấy **X** có được bằng cách xếp thành cột các các $\mathbf{x^{(i)}}$**.**

Bộ trọng số **w = [w_1;w_2;…;w_n]** cũng sẽ là một vector cột, ta sẽ ngăn cách các trọng số $\mathbf{w_i}$ bằng dấu ";" bộ trọng số **w** sẽ có dạng như dưới:

$$\mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix}$$

Ta có chuyển vị của vector **w** sẽ là một vector hàng, các $\mathbf{w_i}$ ngăn cách nhau bằng dấu ",":

$$\mathbf{w}^T = [w_1, w_2, \ldots, w_n]$$

Xét trên một điểm dữ liệu ta đặt $\mathbf{z = w^T x + b}$ (**b** gọi là bias), và $\mathbf{a = \sigma(z)}$**:**

$$z = \mathbf{w}^T\mathbf{x} + b$$
$$a = \sigma(z)$$

Xét trên toàn bộ tập dữ liệu ta đặt **Z = [$\mathbf{z^{(1)}}$,$\mathbf{z^{(2)}}$,…,$\mathbf{z^{(m)}}$]** và **A = [$\mathbf{a^{(1)}}$,$\mathbf{a^{(2)}}$,…,$\mathbf{a^{(m)}}$]**, đây đều là các vector hàng với **m** là số điểm dữ liệu:

$$\mathbf{Z} = [z^{(1)}, z^{(2)}, \ldots, z^{(m)}]$$
$$\mathbf{A} = [a^{(1)}, a^{(2)}, \ldots, a^{(m)}]$$

b) Sigmoid function

Hồi quy logistic sử dụng hàm kích hoạt Sigmoid để kiểm soát đầu ra.

Có thể xem phần ***3.1.3 Một số hàm kích hoạt phổ biến*** để biết chi tiết.

Tính chất:

- Là một hàm số liên tục và nhận giá trị trong khoảng (0;1)
- Chính vì là hàm liên tục nên hàm sigmoid sẽ có đạo hàm tại mọi điểm.

Đạo hàm:

$$\frac{\partial \sigma(x)}{\partial x} = \frac{e^{-x}}{(1+e^{-x})^2} = \left(\frac{1}{1+e^{-x}}\right)\left(1-\frac{1}{1+e^{-x}}\right) = (\sigma(x))(1-\sigma(x))$$

c) Mô hình

Quan hệ giữa nhiều biến độc lập $\mathbf{x_i}$ với biến mục tiêu (biến phụ thuộc), phương trình tổng quát để ước lượng biến mục tiêu **y**:

$$E(y) = w_n x_n + w_{n-1} x_{n-1} + \cdots + w_1 x_1 + w_0 = \mathbf{w}^T \mathbf{x} + w_0$$

Trong đó: $\mathbf{w_0}$ là giá trị ước lượng của y khi tất cả các xi đều đạt giá trị 0.

E(y) trong phương trình logistic regression là xác suất để kết luận giá trị của biến **y** không phải giá trị thực của biến **y**:

$$E(y) = P(y = 0||1|x_1, x_2, \ldots, x_n)$$

Áp dụng hàm sigmoid để chuyển giá trị $\mathbf{w^T x + w_0}$ thành xác suất để kết luận giá trị của biến **y** từ đó để xác định được nhãn của input **x**:

$$P = \frac{1}{1 + e^{-(\mathbf{w}^T \mathbf{x} + w_0)}}$$

Đây chính là mô hình của logistic regression.

d) Loss function (hàm mất mát):

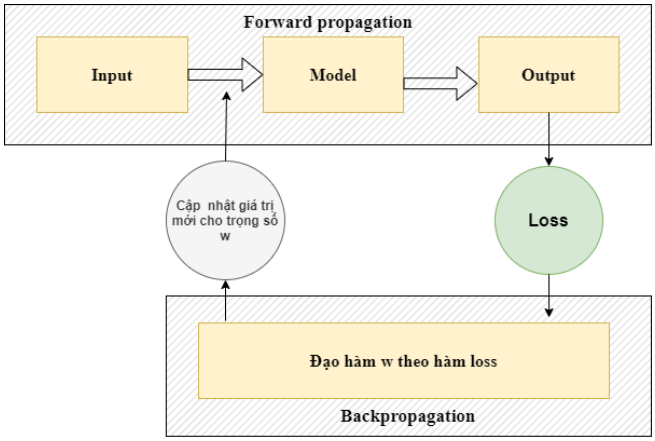
Hàm mất mát trong thuật toán học có giám sát (supervised learning) nói chung là hàm số để đo sự sai khác giữa giá trị thực tế và giá trị mô hình dự đoán, sự sai khác có thể sai khác về giá trị thực (như trong hồi quy tuyến tính) hoặc sai khác về nhãn (như trong bài toán phân loại) việc định nghĩa sự sai khác như nào là tùy vào từng bài toán cụ thể để từ đó xây dựng hàm mất mát.

Logistic regression sử dụng hàm mất mát **Binary Cross-Entropy:**

$$\mathcal{L}(\mathbf{w}) = -\sum_{i=1}^{m}(y^{(i)}\log a^{(i)} + (1-y^{(i)})\log(1-a^{(i)}))$$

e) Backpropagation (lan truyền ngược):

Ngược với Forward propagation là Backpropagation, đây là quá trình tính toán ngược trong thuật toán. Ngược ở đây có nghĩa là chúng ta sẽ từ đi từ loss function để điều chỉnh bộ trọng số **w** nhằm giảm được giá trị loss ở các lần tính toán xuôi tiếp theo.



Hình 3.14 Mô tả quá trình cập nhật trọng số w

Xét tại một điểm dữ liệu, ta có hàm mất mát như sau:

$$J(\mathbf{w}) = -(y^{(i)}\log a^{(i)} + (1-y^{(i)})\log(1-a^{(i)}))$$

Ta có đạo hàm:

$$\begin{aligned}\frac{\partial J(\mathbf{w})}{\partial \mathbf{w}} &= -\left(y^{(i)}\frac{\mathbf{x}^{(i)}a^{(i)}(1-a^{(i)})}{a^{(i)}} + (1-y^{(i)})\frac{-\mathbf{x}^{(i)}a^{(i)}(1-a^{(i)})}{1-a^{(i)}}\right)\\ &= -(y^{(i)}\mathbf{x}^{(i)}(1-a^{(i)}) - \mathbf{x}^{(i)}a^{(i)}(1-y^{(i)}))\\ &= -(\mathbf{x}^{(i)}(y^{(i)}-a^{(i)}))\\ &= \mathbf{x}^{(i)}(a^{(i)}-y^{(i)})\end{aligned}$$

Vậy ta có công thức cập nhật cho **w**:

$$\mathbf{w} = \mathbf{w} - \eta\frac{\partial J(\mathbf{w})}{\partial \mathbf{w}} = \mathbf{w} - \eta\mathbf{x}^{(i)}(a^{(i)}-y^{(i)})$$

Nếu biểu diễn tổng quát cho toàn bộ dữ liệu ta có:

$$\mathbf{w} = \mathbf{w} - \eta \frac{\mathbf{X}(\mathbf{A} - \mathbf{Y})^T}{m}$$

Trong đó: **η**: là tốc độ học (learning rate)

– $\mathbf{y^{(i)}}$: giá trị đúng của input $\mathbf{x^{(i)}}$, $\mathbf{y^{(i)}}$ nhận giá trị 0 hoặc 1.

– $\mathbf{a^{(i)}}$: giá trị mô hình dự đoán ứng với input $\mathbf{x^{(i)}}$.

– **m**: là số điểm dữ liệu.

– **X**: dạng biểu diễn tất cả điểm dữ liệu, ở phần kí hiệu mình đã chỉ rõ.

– **A**: dạng biểu diễn giá trị mô hình dự đoán được cho tất cả điểm dữ liệu.

– **Y**: dạng biểu diễn nhãn của tất cả điểm dữ liệu.

3.4 Gradient Descent

3.4.1 Giới thiệu Gradient Descent

Gradient Descent là một thuật toán tối ưu được sử dụng trong các bài toán Machine Learning và Deep Learning (thường là các bài toán tối ưu lồi (Convex Optimization) với mục tiêu là tìm một tập các biến nội tại (internal parameters) cho việc tối ưu mô hình. Nếu như nói Gradient Descent là cốt lõi của các thuật toán Machine Learning thì đạo hàm chính là cốt lõi của Gradient Descent. Trong đó:

● Gradient: là tỷ lệ độ nghiêng của đường. Về mặt toán học, Gradient của một hàm số là đạo hàm của hàm số đó tương ứng với mỗi biến của hàm. Đối với hàm số đơn biến, chúng ta sử dụng khái niệm Derivative (đạo hàm) thay cho Gradient.

● Descent: là từ viết tắt của descending, nghĩa là giảm dần.

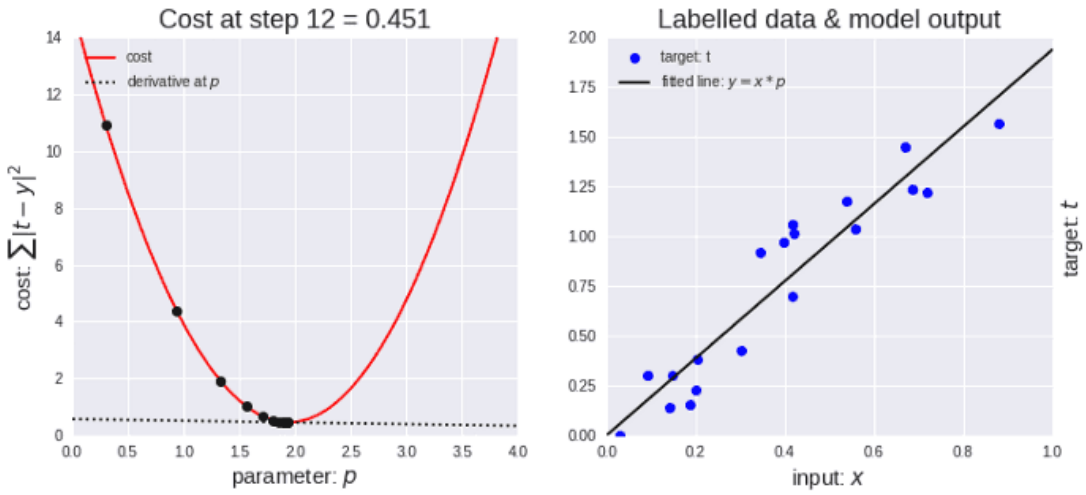
Gradient Descent có nhiều dạng khác nhau như Stochastic Gradient Descent (SGD), Mini-batch SDG. Nhưng về cơ bản thì đều được thực thi như sau:

- Bước 1: Khởi tạo biến nội tại.
- Bước 2: Đánh giá mô hình dựa vào biến nội tại và hàm mất mát.
- Bước 3: Cập nhật các biến nội tại theo hướng tối ưu hàm mất mát.
- Bước 4: Lặp lại bước 2, 3 cho tới khi thỏa điều kiện dừng.

Công thức cập nhật cho Gradient Descent có thể được viết là:

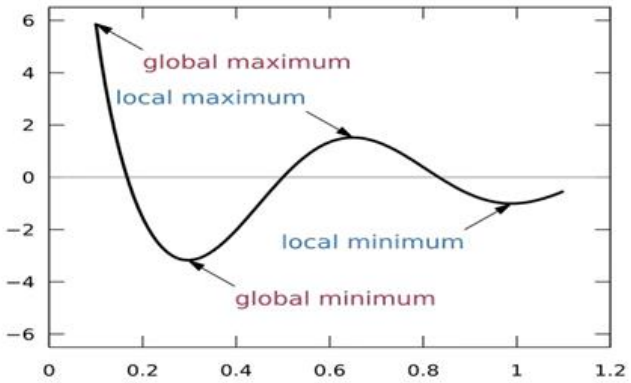
$$\theta \leftarrow \theta - \eta \nabla_\theta f(\theta)$$

Trong đó: θ là tập các biến cần cập nhật, η là tốc độ học (learning rate), $\nabla_\theta f(\theta)$ là Gradient của hàm mất mát f theo tập θ.



Hình 3.15 Minh hoạ Gradient Descent

Tối ưu hàm mất mát là việc tìm các điểm tối ưu mà ở đó hàm mất mát đạt cực đại (maximum) hoặc cực tiểu (minimum). Nếu hàm mất mát không phải là hàm lồi thì sẽ có các local maximum hoặc local minimum points bên cạnh các global maximum hoặc global minimum points như hình bên dưới. Mục tiêu của GD là tìm được các global minimum points. Tuy nhiên trong các bài toán tối ưu lồi áp dụng GD thì các local minimum points của hàm mất mát cũng chính là global minimum points của nó.



Hình 3.16 Minh hoạ các điểm cực đại và cực tiểu trong Gradient Descent

Điều kiện dừng của GD có thể là:

- Kết thúc tất cả các epochs đã được định sẵn.

- Giá trị của hàm mất mát đủ nhỏ và độ chính xác của mô hình đủ lớn.
- Hàm mất mát có giá trị không thay đổi sau một số lần hữu hạn epochs.

Các bài toán trong thực tế áp dụng GD thường khó tìm được các điểm global minimum, đa phần rơi vào các điểm local minimum hoặc không phải các điểm tối ưu (không thể hội tụ), tuy nhiên chúng ta vẫn có thể chấp nhận các kết quả của GD trả về khi mô hình đã đủ tốt.

3.4.2 Phân biệt Epoch, Batch size và Iterations

a) Epoch

Một Epoch được tính là khi chúng ta đưa tất cả dữ liệu vào mạng neural network 1 lần. Khi dữ liệu quá lớn, chúng ta không thể đưa hết mỗi lần tất cả tập dữ liệu vào để huấn luyện được. Buộc lòng chúng ta phải chia nhỏ tập dữ liệu ra thành các batch (size nhỏ hơn). Ta bắt buộc phải dùng hơn 1 epoch vì chúng ta đang dùng thuật toán tối ưu là Gradient Descent. Thuật toán này đòi hỏi chúng ta phải đem toàn bộ dữ liệu qua mạng một vài lần để tìm được kết quả tối ưu. Vì vậy, dùng 1 epoch thật sự không đủ để tìm được kết quả tốt nhất.

b) Batch Size

Batch size là số lượng mẫu dữ liệu trong một batch. Ở đây, khái niệm batch size và số lượng batch (number of batch) là hoàn toàn khác nhau. Như đã nói ở trên, chúng ta không thể đưa hết toàn bộ dữ liệu vào huấn luyện trong 1 epoch, vì vậy chúng ta cần phải chia tập dữ liệu thành các phần, mỗi phần có kích thước là batch size.

c) Iterations

Iterations là số lượng batchs cần để hoàn thành 1 epoch. Ví dụ chúng ta có tập dữ liệu có 20,000 mẫu, batch size là 500, vậy chúng ta cần 40 lần lặp (iteration) để hoàn thành 1 epoch.

3.4.3 Momentum

ý tưởng cơ bản của momentum là tăng gia tốc học khi cùng hướng với chiều của gradient và giảm tốc học khi ngược hướng với gradient. Khi momentum của GD đủ lớn thì các biến cập nhật có thể "vượt" qua các điểm local optimal để hướng đến các điểm

global. Một tham số quan trọng khi sử dụng momentum là γ, γ trong thực nghiệm thường được chọn là 0.9, hoặc ban đầu chọn γ = 0.5 tới khi ổn định và tăng dần lên 0.9.

3.4.4 Vanishing và Exploding Gradient

Trong lan truyền ngược (backpropagation) thuật toán sẽ từ output layer đi ngược trở lại input layer, tính toán gradient của hàm mất mất tương ứng với các biến nội tại (weight, bias) cho các hidden layers rồi dùng GD để cập nhật lại các biến này. Thuật toán được mong đợi sẽ hội tụ sau một số lần hữu hạn epochs. Thực tế khi tiến hành training với backpropagation thì gradient của hàm mất mất sẽ nhỏ dần do tiến hành nhân các số hạng nhỏ liên tiếp với nhau, nếu mô hình đủ "sâu" (nhiều hidden layers) thì giá trị gradient sẽ tiến dần đến 0 sau một số layers nhất định và làm cho mô hình không thể hội tụ dẫn đến không thể cập nhật được các biến nội tại như mong đợi. Hiện tượng này gọi là Vanishing Gradient.

Tuy nhiên gradient cũng có khả năng lớn dần trong quá trình backpropagation do nhân nhiều số hạng lớn liên tiếp nhau dẫn tới các giá trị cập nhật quá lớn và cũng làm cho mô hình không thể hội tụ. Hiện tượng này gọi là Exploding Gradient.

Có 2 nguyên nhân chính dẫn tới các hiện tượng trên là do việc khởi tạo các biến nội tại (weight initialization) và việc chọn activation function cho các layers. Có nhiều kỹ thuật khác nhau để giảm thiểu 2 hiện tượng này như Xavier and He Initialization Techniques, Batch Normalization và Gradient Clipping…

3.5 Batch normalization

3.5.1 Giới thiệu

Batch normalization là một trong các phương thức normalization được sử dụng phổ biến trong mô hình Deep Learning. Nó cho phép đào tạo nhanh hơn và ổn dịnh các mạng neural sâu bằng cách ổn định sự phân bố của các đầu vào các layer trong quá trình huấn luyện. Cách tiếp cận này chủ yếu liên quan đến Internal Covariate Shift (ICS). Để cải thiện việc huấn luyện mô hình, điều quan trọng là phải giảm ICS bằng cách kiểm soát means (giá trị trung bình) và variances (phương sai) của dữ liệu đầu vào các layer, nôm na là điều chỉnh phân phối của dữ liệu cho đồng bộ trên toàn mô hình. Batch normalization là một phương thức chuẩn hóa các hàm kích hoạt trong mạng qua một mini-batch theo kích thước được định nghĩa trước đó. Với mỗi feature, batch

normalization tính toán trung vị (median) và phương sai của feature đó trong một mini-batch. Sau đó, nó trừ đi giá trị trung bình và chia cho độ lệch chuẩn của mini-batch đó.

Công thức sẽ được biểu diễn như sau:

$$\mu_{\mathrm{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i // mini-batch mean$$

$$\sigma_{\mathrm{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i-\mu_{\mathrm{B}})^2 \quad // mini-batch variance$$

Điều gì xảy ra nếu ta tăng cường độ trọng số làm cho mô hình trở nên tốt hơn. Để giải quyết vấn đề này, chúng ta có thể thêm γ và β để chúng có thể học scale và shift với tham số tương ứng. Được mô tả như sau:

Input: Giá trị của x trên một mini-batch: $\{B\}=\{x_1 \ldots x_m\}$ Các tham số cần học: γ,β

Output: $\{y_i = BN_{\gamma,\beta}(x_i)\}$

$$\hat{x}_i \leftarrow \frac{x_i-\mu_{\mathrm{B}}}{\sqrt{\sigma_{\mathrm{B}}^2+\epsilon}} // normalize$$

$$y_i \leftarrow \gamma\hat{x}_i+\beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) // scale and shift$$

3.5.2 Lợi ích của batch normalization

- Làm giảm ICS và tăng tốc độ huấn luyện cho mô hình Deep Learning.
- Cách tiếp cận này làm giảm sự phụ thuộc của gradients vào tỉ lệ của các tham số hoặc giá trị ban đầu của chúng, dẫn đến learning rate cao hơn mà không có nguy cơ phân kỳ.
- Batch normalization có thể sử dụng các chế độ phi tuyến bão hòa bằng cách ngăn mạng khỏi bị kẹt trong các chế độ bão hòa.

3.6 Transfer Learning

Việc nghiên cứu khoa học, đưa ra những ý tưởng mới thì quan trọng nhất là **không làm lại những gì đã được làm rồi mà không làm tốt hơn được** vì thời gian sẽ không cho phép sự lãng phí như vậy xảy ra. Đặc biệt là trong Deep Learning, một ngành phát triển nhanh đến chóng mặt hiện nay, những ý tưởng mình nghĩ ra chắc gì đã chưa có ai làm? Deep Learing lan tỏa đến mọi lĩnh vực, vì thế cái quan trọng là sử dụng những

công việc sẵn có để tạo nên một mô hình mới tốt hơn, vì chính việc này đã rất khó khăn và tốn thời gian rồi chứ không nói đến nghiên cứu lại từ đầu mọi thứ.

a) Model (mô hình)

Chúng ta gọi những mô hình đi kèm weights được chia sẻ công khai là một pretrained model. Mô hình mới sử dụng một phần hay toàn bộ pretrained model như một phần của nó để học một task (công việc) mới được gọi là Transfered Model.

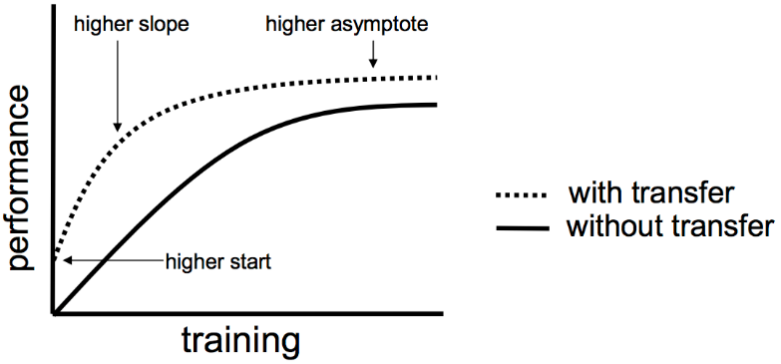
b) Source Tasks và Target Tasks

Những Pretrained Model như vậy thường được huấn luyện trên một hoặc một vài tập dữ liệu nhất định, tương thích và cho accuracy (độ chính xác) cao với một task hoặc nhiều tasks (multi-task deep learning) nào đó mà nó được huấn luyện. Chúng ta gọi các tasks mà pretrained model đó được huấn luyện để thực hiện là **source tasks**. Nhiệm vụ của chúng ta là tạo ra một mô hình mới để thực hiện một hoặc nhiều tasks nào đó. Những tasks cần được thực hiện của mô hình này có thể trùng hoặc không trùng với tasks mà pretrained model được huấn luyện, chúng ta gọi tasks này là **target tasks**.

c) Transfer Learning

Transfer Learning trong Deep Learning là một kỹ thuật mà trong đó:

- Một pretrained model đã được train trên source tasks cụ thể nào đó, khi đó một phần hay toàn bộ pretrained model có thể được tái sử dụng phụ thuộc vào nhiệm vụ của mỗi layer trong mô hình đó.
- Một mô hình mới sử dụng một phần hay toàn bộ pretrained model để học một target tasks và tùy vào nhiệm vụ của mỗi layer mà mô hình mới có thể thêm các layer khác dựa trên pretrained model sẵn có.



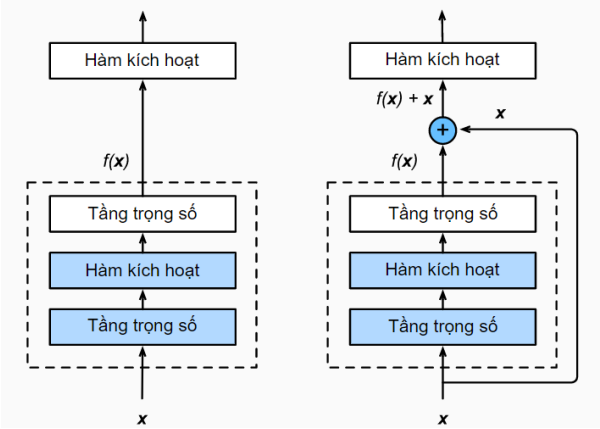
Hình 3.17 So sánh tương quan hiệu quả của model train từ đầu và transfered model

b) Fine-tuning

Fine-tuning được hiểu là lấy một pre-trained model, tận dụng 1 phần hoặc toàn bộ các layer, thêm/sửa/xoá 1 vài layer/nhánh để tạo ra một model mới. Thường các layer đầu của model được freeze (đóng băng) lại - tức weight (trọng số) các layer này sẽ không bị thay đổi giá trị trong quá trình train. Lý do bởi các layer này đã có khả năng trích xuất thông tin mức trừu tượng thấp, khả năng này được học từ quá trình training trước đó. Ta freeze lại để tận dụng được khả năng này và giúp việc train diễn ra nhanh hơn (model chỉ phải update weight ở các layer cao). Có rất nhiều các Object detection model được xây dựng dựa trên các Classifier model. Ví dụ mô hình YOLOv3 được xây dựng với backbone là Darknet 53.

3.7 Một số kiến trúc sử dụng

3.1.1 ResNet



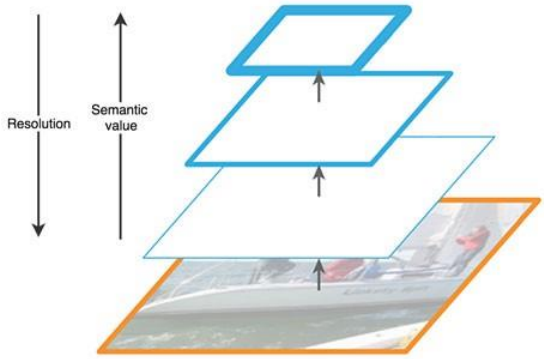
Hình 3.18 Sự khác biệt giữa một khối thông thường (trái) và một khối phần dư (phải)

ResNet đưa ra là sử dụng kết nối "tắt" đồng nhất để xuyên qua một hay nhiều lớp. Một khối như vậy được gọi là một Residual Block (khối phần dư). Ví dụ, ký hiệu đầu vào là **x**. Giả sử ánh xạ lý tưởng muốn học được là **f(x)**, và được dùng làm đầu vào của hàm kích hoạt. Phần nằm trong viền nét đứt bên trái phải khớp trực tiếp với ánh xạ **f(x)**. Điều này có thể không đơn giản nếu chúng ta không cần khối đó và muốn giữ lại đầu vào **x**. Khi đó, phần nằm trong viền nét đứt bên phải chỉ cần tham số hoá độ lệch khỏi giá trị **x**, bởi vì ta đã trả về **x + f(x)** (được gọi là một skip connection). Trên thực tế, ánh xạ phần dư thường dễ tối ưu hơn, vì chỉ cần đặt **f(x) = 0**.

Được sử dụng để ngăn chặn hiện tượng Vanishing Gradient.

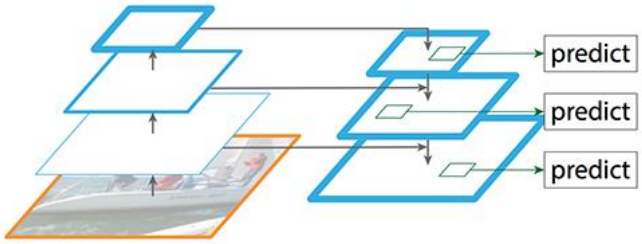
3.1.2 Feature Pyramid Networks (FPN)

Dò tìm các đối tượng có kích thước nhỏ là một vấn đề đáng được giải quyết để nâng cao độ chính xác. Và FPN là mô hình mạng được thiết kế ra dựa trên khái niệm pyramid (kim tự tháp) để giải quyết vấn đề này. Mô hình FPN kết hợp thông tin của mô hình theo hướng bottom-up kết hợp với top-down để dò tìm đối tượng (trong khi đó, các thuật toán khác chỉ thường sử dụng bottom-up). Khi chúng ta ở bottom và đi lên (up), độ phân giải sẽ giảm, nhưng giá trị ngữ nghĩa sẽ tăng lên. Xem hình mô phỏng bên dưới.



Hình 3.19 Quan hệ tương quan giữa độ phân giải và giá trị ngữ nghĩa

Trong quá trình xây dựng lại các layer từ top xuống bottom, chúng ta sẽ gặp một vấn đề khá nghiêm trọng là bị mất mát thông tin của các đối tượng. Ví dụ một đối tượng nhỏ khi lên top sẽ không thấy nó, và từ top đi ngược lại sẽ không thể tái tạo lại đối tượng nhỏ đó. Để giải quyết vấn đề này, chúng ta sẽ tạo các kết nối (skip connection) giữa các reconstruction layer và các feature map để giúp quá trình detector dự đoán các vị trí của đối tượng thực hiện tốt hơn (hạn chế tốt nhất việc mất mát thông tin).



Hình 3.20 Thêm các skip connection giữa feature map và reconstruction layer

3.1.3 Darknet 53

DarkNet-53 là một mạng nơ-ron phức tạp có độ sâu 53 lớp. Vì các mạng neural sâu (deep neural network) rất khó đào tạo, và với độ sâu ngày càng tăng, đôi khi độ chính xác của mạng bị bão hòa dẫn đến lỗi huấn luyện cao hơn. Để giải quyết vấn đề này, residual block đã được sử dụng. Sự khác biệt về kiến trúc giữa khối tích chập thông

thường và residual block là việc bổ sung skip connection. Skip connection mang đầu vào cho các lớp sâu hơn.

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

Hình 3.21 Kiến trúc của mô hình Darknet 53

3.8 Object detection (phát hiện đối tượng)

3.8.1 Giới thiệu

Phát hiện đối tượng là một thuật ngữ chung để mô tả một tập hợp các nhiệm vụ Thị Giác Máy Tính có liên quan liên quan đến việc xác định các đối tượng trong ảnh kỹ thuật số. Phân loại hình ảnh liên quan đến việc dự đoán lớp (class) của một đối tượng trong một hình ảnh. Định vị đối tượng đề cập đến việc xác định vị trí của một hoặc nhiều đối tượng trong một hình ảnh và vẽ bounding box (hộp giới hạn) xung quanh chúng. Phát hiện đối tượng kết hợp hai nhiệm vụ trên và thực hiện cho một hoặc nhiều đối tượng trong hình ảnh. Chúng ta có thể phân biệt giữa ba nhiệm vụ Thị Giác Máy Tính cơ bản trên thông qua input và output của chúng như sau:

Phân loại hình ảnh: Dự đoán nhãn của một đối tượng trong một hình ảnh.

- Input: Một hình ảnh với một đối tượng, chẳng hạn như một bức ảnh.

- Output: Nhãn lớp (ví dụ: Một hoặc nhiều số nguyên được ánh xạ tới nhãn lớp).

Định vị đối tượng: Xác định vị trí hiện diện của các đối tượng trong ảnh và cho biết vị trí của chúng bằng bounding box.

- Input: Một hình ảnh có một hoặc nhiều đối tượng, chẳng hạn như một bức ảnh.

- Output: Một hoặc nhiều bounding box được xác định bởi tọa độ tâm, chiều rộng và chiều cao.

Phát hiện đối tượng: Xác định vị trí hiện diện của các đối tượng trong bounding box và nhãn của các đối tượng nằm trong một hình ảnh.

- Input: Một hình ảnh có một hoặc nhiều đối tượng, chẳng hạn như một bức ảnh.

- Output: Một hoặc nhiều bounding box và nhãn cho mỗi bounding box.

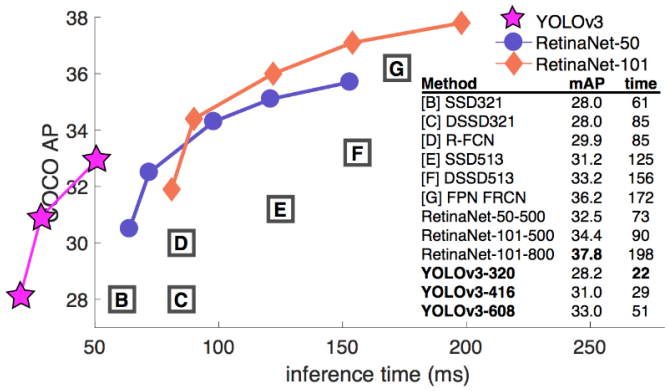
Lịch sử hình thành, phát triển và đặc điểm cấu trúc của các thuật toán object detection bao gồm 3 nhóm chính:

- Họ các mô hình RCNN (Region-Based Convolutional Neural Networks) giải quyết các nhiệm vụ định vị đối tượng và phát hiện đối tượng.
- Họ các mô hình SSD (Single Shot Detector) mô hình phát hiện đối tượng thời gian thực dựa trên kiến trúc VGG-16.
- Họ các mô hình YOLO (You Only Look Once), là một nhóm kỹ thuật thứ ba để phát hiện đối tượng được thiết kế để phát hiện đối tượng theo thời gian thực.

3.8.2 Lựa chọn mô hình

Trong phạm vi đồ án này, tôi chỉ quan tâm đến họ các mô hình thứ 3 là YOLOv3, bởi vì những lý do sau:

- Về mặt tốc độ và độ chính xác: Các mô hình họ RCNN tuy độ chính xác cao nhưng tốc độ không thể thực hiện theo thời gian thực, vậy ta loại bỏ RCNN.



Hình 3.22 So sánh tốc độ của các mô hình phát hiện đối tượng tại thời điểm YOLOv3 mới ra mắt (2018)[14]

Ta có thể thấy thì tốc độ của YOLOv3 vượt xa so với các phiên bản SSD, và độ chính xác (mAP) cũng cao hơn ở chế độ suy luận mô hình.

- YOLOv3 có FPN sử dụng các feature map nhiều thông tin hơn để cải thiện độ chính xác, có thể phát hiện đối tượng nhỏ với độ chính xác cao. SSD có layer ở bottom không được sử dụng để phát hiện đối tượng. Vì những layter này có độ phân giải cao nhưng giá trị ngữ nghĩa của chúng lại không đủ cao (thấp) nên những nhà nghiên cứu bỏ chúng đi để tăng tốc độ xử lý. Cho nên, SSD chỉ sử dụng các layer ở lớp trên, và do đó sẽ không phát hiện được các đối tượng có kích thước nhỏ.
- YOLOv3 sử dụng batch normaliation và gradient descent để chuẩn hoá và tối ưu dữ liệu.
- YOLOv3 sử dụng nhiều data augmentation trong quá trình training.
- YOLOv3 Sử dụng 9 anchors (điểm neo) để phát hiện đối tượng.

3.9 YOLOv3

3.9.1 Giới thiệu YOLO

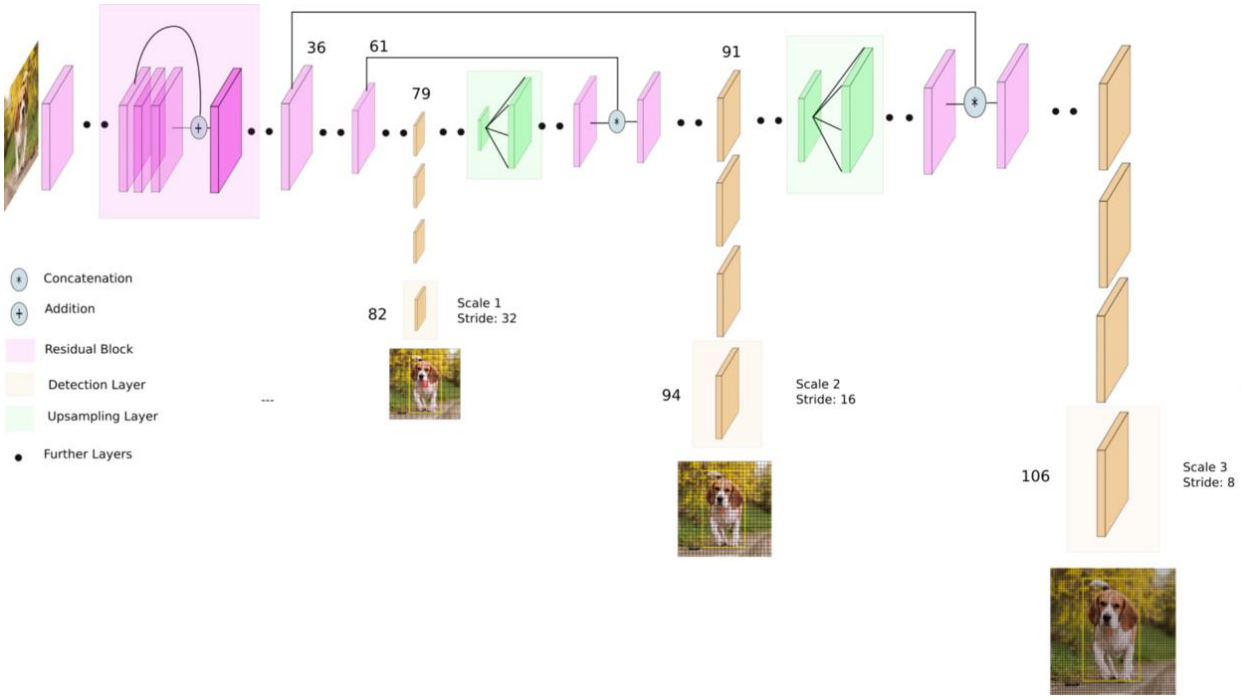
YOLO - You Only Look Once (tạm dịch Bạn Chỉ Nhìn Một Lần) là một trong những mô hình phát hiện đối tượng tốt nhất ở thời điểm hiện tại, phiên bản của mô hình này đều có những cải tiến rất đáng kể sau mỗi phiên bản. Sau 3 phiên bản của tác giả chính Joseph Redmon là YOLOv1 đến v3, tính đến thời điểm hiện tại có thêm một paper YOLOv4 của Alexey Bochkovskiy được dẫn link trực tiếp từ repository gốc của Joseph

[14] https://pjreddie.com/darknet/yolo/

Redmon và YOLOv5 đang được phát triển. Nhiều hệ thống phát hiện đối tượng cần phải "nhìn" qua ảnh nhiều hơn một lần để có thể phát hiện tất cả các đối tượng trong ảnh, hoặc nó phải trải qua hai giai đoạn để phát hiện đối tượng. YOLO không cần phải trải qua những quy trình nhàm chán này. Nó chỉ cần "nhìn" một lần vào hình ảnh để phát hiện tất cả các đối tượng và đó là lý do tại sao họ chọn tên You Only Look Once và đó thực sự là lý do tại sao YOLO là một mô hình rất nhanh. Trong phạm vi đồ án này chỉ giới thiệu phiên bản YOLOv3.

3.9.2 YOLOv3

a) Kiến trúc của YOLOv3



Hình 3.23 Kiến trúc YOLOv3

YOLOv3 có thể gọi là thuật toán hay mô hình (nó là một mô hình Deep Learning hoàn chỉnh gồm nhiều thuật toán khác nhau), sử dụng một biến thể của Darknet, ban đầu là mạng 53 lớp được huấn luyện trên tập dữ liệu Imagenet. Đối với nhiệm vụ phát hiện, 53 lớp khác được xếp chồng lên nó, tạo nên một kiến trúc mạng tích chập hoàn toàn (Fully Convolutional Network - FCN) gồm 106 lớp, với các residual block (khối phần dư), skip connection (bỏ qua kết nối) và các lớp upsampling, tương tự với mô hình FPN và ResNet. Không có lớp tổng hợp nào được sử dụng và một lớp tích chập có bước nhảy bằng 2 được sử dụng để downsampling cho các biểu đồ đặc trưng (feature map). Điều này giúp ngăn ngừa mất các đặc trưng cấp thấp thường được cho là do các lớp tổng hợp.

YOLOv3 bỏ qua ba lớp cuối cùng của Darknet 53 (lớp avgpool, lớp full connected, lớp softmax) vì các lớp này chủ yếu được sử dụng để phân loại hình ảnh. Trong nhiệm vụ phát hiện đối tượng, YOLOv3 sử dụng Darknet 53 chỉ để trích xuất các đặc trưng hình ảnh nên ba lớp này sẽ không cần thiết.

Là một FCN, kích thước của hình ảnh đầu vào luôn bất biến và có thể chỉnh các giá trị này trong tệp cấu hình mạng. Một vấn đề lớn là nếu chúng ta muốn xử lý hình ảnh theo batch (hình ảnh theo batch có thể được xử lý song song bởi GPU, tăng tốc độ xử lý), chúng ta cần có tất cả hình ảnh có chiều cao và chiều rộng cố định. Điều này là cần thiết để nối nhiều hình ảnh thành một batch lớn (ghép nhiều PyTorch tensor thành một để thuận tiện cho quá trình triển khai).

Các lớp downsampling hình ảnh bằng một hệ số được gọi là bước nhảy (stride).

b) Phát hiện đối tượng ở 3 tỷ lệ kích thước khác nhau

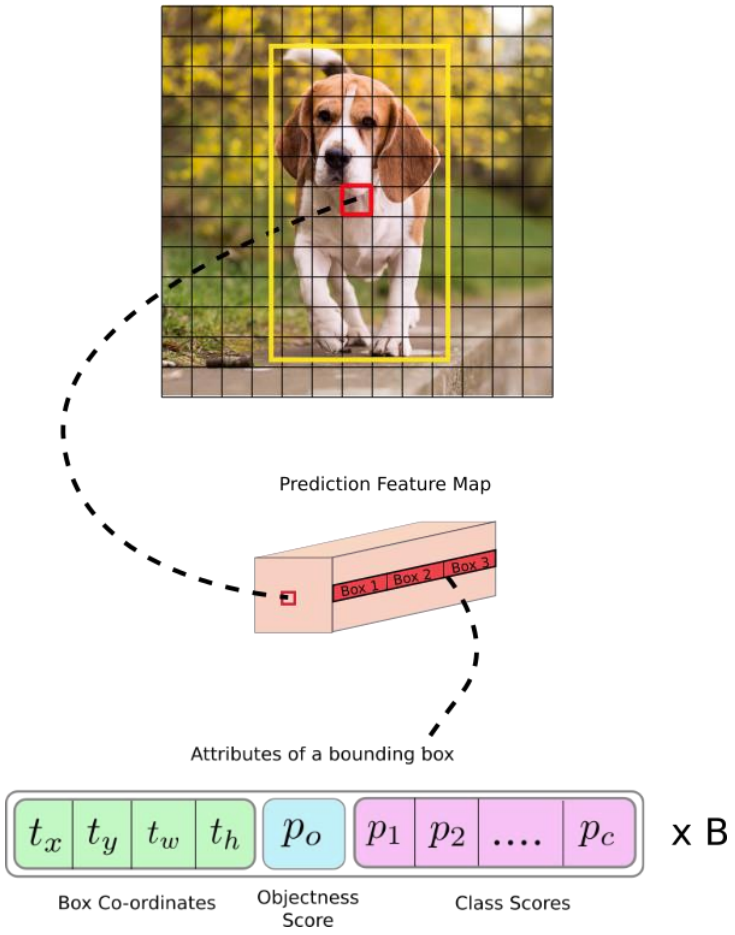
Tính năng nổi bật nhất của YOLOv3 là nó thực hiện phát hiện ở ba tỷ lệ kích thước khác nhau. Trong YOLO v3, việc phát hiện được thực hiện bằng cách áp dụng các kernel phát hiện 1 x 1 lên các biểu đồ đặc trưng có ba kích thước khác nhau tại ba vị trí khác nhau trong mạng.

Lưu ý, mặc dù thuật ngữ chính xác để mô tả một đơn vị trong biểu đồ đặc trưng sẽ là một neural, nhưng ở đây gọi nó là một ô (cell) làm cho nó trực quan hơn trong ngữ cảnh bài toán này.

Hình dạng (shape) của kernel phát hiện là **1 x 1 x (B x (5 + C))**. Ở đây **B** là số hộp giới hạn (bounding box) mà một ô (cell) trên feature map có thể dự đoán, "**5**" là bởi có **4** thuộc tính hộp giới hạn (toạ độ, kích thước tâm) và độ tin cậy của một đối tượng (hay **objectness score**), và **C** là số lớp (bao gồm cả **class scores**) cần dự đoán. Ví dụ, trong YOLO v3 được đào tạo trên dataset COCO, có **B = 3** và **C = 80**, do đó shape của kernel là **1 x 1 x 255**. Biểu đồ đặc trưng do kernel này tạo ra có chiều cao và chiều rộng giống hệt biểu đồ đặc trưng trước đó và có một dãy các thuộc tính phát hiện được gọi là độ sâu (**depth**) như đã mô tả ở trên. YOLOv3 dự đoán 3 hộp giới hạn cho mỗi ô.

Mỗi ô của feature map dự đoán một đối tượng thông qua một trong các hộp giới hạn của nó nếu tâm của đối tượng nằm trong trường tiếp nhận (receptive field) của ô đó. (Trường tiếp nhận là vùng của hình ảnh đầu vào có thể nhìn thấy đối với ô). Điều này

liên quan đến cách YOLO được huấn luyện, trong đó chỉ có một hộp giới hạn chịu trách nhiệm phát hiện bất kỳ đối tượng nhất định nào. Đầu tiên, chúng ta phải xác định chắc chắn hộp giới hạn này thuộc về ô nào. Để làm điều đó, ta chia hình ảnh đầu vào thành một lưới có kích thước bằng với kích thước của biểu đồ đặc trưng cuối cùng. YOLO v3 đưa ra dự đoán ở ba tỷ lệ kích thước, bằng cách downsampling kích thước của hình ảnh đầu vào tại các lớp lần lượt với 32, 16 và 8.



Hình 3.24 Minh hoạ một ô chịu trách nhiệm phát hiện tại kích thước 13 x 13

Ở đây ta lấy ví dụ bằng ảnh có kích thước 416 x 416. Lần phát hiện đầu tiên được thực hiện bởi lớp thứ 82. Đối với 81 lớp đầu tiên, hình ảnh được mạng downsampling sao cho lớp thứ 81 có stride là 32. Lúc này, biểu đồ đặc trưng sẽ có kích thước 13 x 13. Một lần phát hiện được tạo ở đây bằng cách sử dụng kernel phát hiện 1 x 1, tạo nên biểu đồ đặc trưng phát hiện có shape là 13 x 13 x 255.

Sau đó, ô (trên hình ảnh đầu vào) chứa tâm của hộp ground truth của một đối tượng được chọn làm ô chịu trách nhiệm dự đoán đối tượng. Trong ảnh trên, nó là ô được đánh dấu màu đỏ, chứa trung tâm của hộp ground truth (được đánh dấu màu vàng).

Bây giờ, ô này có thể dự đoán ba hộp giới hạn. Cái nào sẽ được gán cho nhãn ground truth? Để hiểu được điều đó, chúng ta phải tìm hiểu khái niệm neo (anchors) ở phần sau.

Tiếp theo, feature map từ lớp 79 phải được tích chập một vài lớp trước khi upsampling 2 lần đến kích thước 26 x 26. Feature map này sau đó được ghép độ sâu (depth) với feature map từ lớp 61. Sau đó, feature map mới được ghép nối phải được tích chập một vài lớp 1 x 1 để hợp nhất các đặc trưng từ lớp trước đó. Sau đó, phát hiện thứ hai được thực hiện bởi lớp thứ 94, tạo nên feature map có shape là 26 x 26 x 255.

Cuối cùng, quy trình tương tự lại được thực hiện một lần nữa, trong đó feature map từ lớp 91 phải được tích chập một vài lớp trước khi được nối theo độ sâu với feature map từ lớp 36. Giống như trên, một vài lớp tích chập 1 x 1 theo sau để kết hợp thông tin từ trước lớp (36). Sau đó, phát hiện cuối cùng ở lớp thứ 106, tạo ra feature map có kích shape là 52 x 52 x 255.

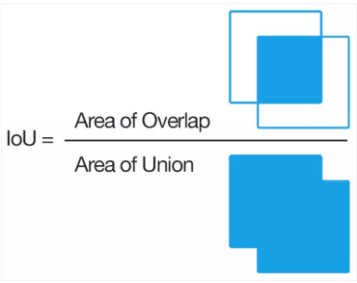
Kích thước 13 x 13 chịu trách nhiệm phát hiện các vật thể lớn, trong khi kích thước 52 x 52 phát hiện các vật thể nhỏ, với kích thước 26 x 26 phát hiện các vật thể trung bình. Khắc phục nhược điểm của các phiên bản YOLO trước đó.

b) Anchor Boxes

Dự đoán chiều rộng và chiều cao của hộp giới hạn trên thực tế dẫn đến gradients không ổn định trong quá trình huấn luyện. Thay vào đó, YOLOv3 dự đoán các phép biến đổi không gian logarit, hoặc đơn giản là độ lệch (offset) của các hộp giới hạn mặc định được xác định trước gọi là neo (anchor). Sau đó, các biến đổi này được áp dụng cho các hộp neo để thu được dự đoán. YOLO v3 dự đoán ba hộp giới hạn trên mỗi ô (3 neo) ở cả 3 tỷ lệ vì vậy YOLOv3 dự đoán tổng cộng 10647 bounding box. Hộp giới hạn chịu trách nhiệm phát hiện đối tượng sẽ là hộp neo có **IoU** cao nhất với hộp ground truth. Các hộp neo này được thực hiện bằng cách chạy phân cụm **K-mean** trên tập dữ liệu.

c) Intersection Over Union (IOU)

IoU (Intersection Over Union) là hàm đánh giá độ chính xác của object detector trên tập dữ liệu cụ thể. IoU được tính bằng:



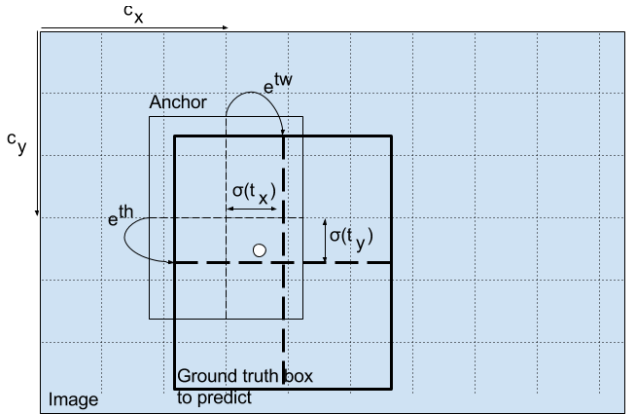
Trong đó Area of Overlap là diện tích phần giao nhau giữa predicted bounding box với grouth truth box, còn Area of Union là diện tích phần hợp giữa predicted bounding box với hộp grouth truth. Nếu IOU > 0.5 thì prediction được đánh giá là tốt.

d) Bounding Box Prediction (dự đoán hộp giới hạn)

Các công thức sau đây mô tả cách chuyển đổi đầu ra của mạng để có được các dự đoán hộp giới hạn:

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

b_x, b_y, b_w, b_h lần lượt là tọa độ tâm (x, y) và chiều rộng, chiều cao bouding box dự đoán. t_x, t_y, t_w, t_h là đầu ra của mạng. c_x và c_y là tọa độ trên cùng bên trái của lưới ảnh. p_w và p_h là kích thước hộp neo. YOLOv3 dự đoán tọa độ tâm thông qua một hàm **sigmoid**. Điều này buộc giá trị đầu ra phải nằm trong khoảng từ 0 đến 1 (để nằm trong phạm vi một ô). Sau đó, ta thêm các tọa độ trên cùng bên trái c_x, c_y để dự đoán tọa độ thực của hộp giới hạn.



Hình 3.25 Minh hoạ các dự đoán toạ độ bounding box dựa qua hộp neo

Kích thước của hộp giới hạn được dự đoán bằng cách áp dụng phép biến đổi không gian logarit cho đầu ra và sau đó nhân với kích thước hộp neo. Trong kết quả của

dự đoán, b_w và b_h, được chuẩn hóa theo chiều cao và chiều rộng của hình ảnh. (Nhãn huấn luyện được chọn theo cách này).

Đối với **objectness score (điểm đối tượng)**: Trong YOLO v3 đưa ra điểm cho các đối tượng cho mỗi hộp giới hạn. Nó sử dụng hồi quy logistic để dự đoán điểm đối tượng để mang tính khách quan, sử dụng hàm kích hoạt leaky ReLU.

Đối với **class scores**: YOLOv3 không sử dụng softmax; thay vào đó sử dụng logistic classifier rời rạc (trong lúc huấn luyện sử dụng hàm mất mát binary cross-entropy), vốn dùng để phân biệt các đối tượng bao hàm nhau ví dụ như con người và phụ nữ, sử dụng softmax cho dự đoán chỉ giả định rằng mỗi hộp có đúng một lớp, điều này thường không đúng.

e) Thresholding (ngưỡng)

Ta lọc các hộp dựa trên điểm đối tượng của chúng, các hộp có điểm dưới ngưỡng sẽ bị bỏ qua.

f) Thuật toán Non-Maximum Suppression

Sau khi thực hiện object detection một ảnh qua mạng YOLOv3, chúng ta sẽ thu được rất nhiều proposals (hộp giới hạn đề xuất). Ở trạng thái này, có rất nhiều proposals là bounding box cho một đối tượng duy nhất, điều này dẫn tới việc dư thừa. Vì vậy, chúng ta cần giữ một hộp có điểm tin cậy cao nhất và loại bỏ các hộp giới hạn khác có tỷ lệ trùng lặp cao và điểm thấp. Kỹ thuật này được gọi là NMS.

Kỹ thuật này hoạt động theo ba bước:

- Bước 1: Xác định xem số hộp giới hạn có lớn hơn 0 hay không. Nếu không, hãy kết thúc quá trình.
- Bước 2: Chọn ô giới hạn có độ tin cậy cao nhất và lấy nó ra.
- Bước 3: Tính IoU giữa hộp giới hạn đã chọn và các hộp giới hạn còn lại. Loại bỏ tất cả các hộp giới hạn có giá trị IoU cao hơn giá trị ngưỡng. Chuyển sang bước 1.

g) Giải thích hàm mất mát

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

- Mất mát của việc dự đoán lớp - Classifycation loss được tính theo hàm BCE (binary cross entropy).
- Mất mát của dự đoán tọa độ tâm, chiều dài, rộng của boundary box (x, y ,w, h) - Localization loss được tính theo hàm lỗi toàn phương trung bình MSE.
- Mất mát của việc dự đoán bounding box đó chứa đối tượng so với nhãn thực tế tại ô vuông đó – Confidence loss được tính theo hàm BCE.

$$loss(object) = \lambda_{coord} \sum_{i=0}^{K\times K} \sum_{j=0}^{M} I_{ij}^{obj} (2 - w_i \times h_i)[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2] +$$
$$\lambda_{coord} \sum_{i=0}^{K\times K} \sum_{j=0}^{M} I_{ij}^{obj} (2 - w_i \times h_i)[(w_i - \hat{w}_i)^2 + (h_i - \hat{h})^2] -$$

Mất mát của dự đoán Bounding Box

$$\sum_{i=0}^{K\times K} \sum_{j=0}^{M} I_{ij}^{obj} [\hat{C}_i \log(C_i) + (1 - \hat{C}_i) \log(1 - C_i)] -$$
$$\lambda_{noobj} \sum_{i=0}^{K\times K} \sum_{j=0}^{M} I_{ij}^{noobj} [\hat{C}_i \log(C_i) + (1 - \hat{C}_i) \log(1 - C_i)] -$$

Mất mát của dự đoán đối tượng

$$\sum_{i=0}^{K\times K} I_{ij}^{obj} \sum_{c \in classes} [\hat{p}_i(c) \log(p_i(c)) + (1 - \hat{p}_i(c)) \log(1 - p_i(c))]$$

Mất mát của dự đoán lớp

Hình 3.26 Hàm mất mát của YOLOv3

Trong đó: λ_{coord}, $\lambda_{noobject}$: Là trọng số thành phần trong paper gốc tác giả lấy giá trị lần lượt là 5 và 0.5.

- K×K: Là tỷ lệ ô lưới tại ảnh đó (với một ảnh kích thước 416 thì có 3 tỷ lệ là 13×13, 26×26 và 52×52).
- M: Số bounding box cần dự đoán tại một ô, ở YOLOv3 là bằng 3.
- $I_{ij}^{obj} = 1$: Nếu box thứ j của ô thứ i có chứa object. Vì huấn luyện cần các image với ground-truth (vị trí của các objects) nên YOLO biết điểm trung tâm của từng object rơi vào ô nào trong grid K×K.
- $I_{ij}^{noobj} = 1$: Nếu box thứ j của ô thứ i không chứa object.
- $(2 – w_i \times h_i)$: Hệ số tỷ lệ để tăng tổn thất của hộp nhỏ thứ i.
- (x_i, y_i): Là toạ độ tâm của grouth-truth bounding box thứ i và $(\hat{x}_i, \hat{y}_i)$ là tọa độ tâm của predicted bounding box thứ i.

- (w_i, h_i): Là kích thước của grouth-truth bounding box thứ i và ($\hat{w}_i$, $\hat{h}_i$) là kích thước của predicted bounding box thứ i.
- C_i: Điểm tin cậy của ô i. Đối với các hộp j của ô i nơi object tồn tại C_i luôn = 1. Và ngược lại.
- $\hat{C}_i$: Điểm tin cậy dự đoán của ô i. $\hat{C}_i = P(contain\ object) * IoU\ (predict\ bbox, ground\ truth\ bbox)$.
- $p_i(c)$: Xác suất có điều kiện, có hay không ô i có chứa một đối tượng của lớp $c \in classes$. Chú ý $p_i(c)$ luôn = 1 nếu đúng lớp c với ground-truth, ngược lại thì $p_i(c)$ luôn = 0.
- $\hat{p}_i(c)$: Xác suất có điều kiện dự đoán của ô i.
- classes : Các lớp đối tượng cần được nhận dạng, ví dụ chó, mèo, oto…

3.9 Một số chỉ số đánh giá thuật toán phát hiện đối tượng

3.9.1 Precision và Recall

Quy ước: Giả sử chia một số kết quả khi kiểm tra mô hình phát hiện đối tượng trong một bức ảnh thành 3 nhóm được hiểu như sau:

- False negative (**FN**): Mô hình báo không có đối tượng, nhưng trong ảnh có.
- False positive (**FP**): Mô hình báo có đối tượng, nhưng trong ảnh không có.
- True positive (**TP**): Mô hình báo có đối tượng và trong ảnh cũng có.

- **Precision**: Đánh giá độ tin cậy của kết luận đưa ra (bao nhiêu % lời kết luận của mô hình là chính xác).

$$\boldsymbol{Precision} = \frac{\boldsymbol{TP}}{\boldsymbol{TP} + \boldsymbol{FP}}$$

- **Recall**: Đánh giá khả năng tìm kiếm toàn bộ các ground truth của mô hình (bao nhiêu % positive samples mà mô hình nhận diện được).

$$\boldsymbol{Recall} = \frac{\boldsymbol{TP}}{\boldsymbol{TP} + \boldsymbol{FN}}$$

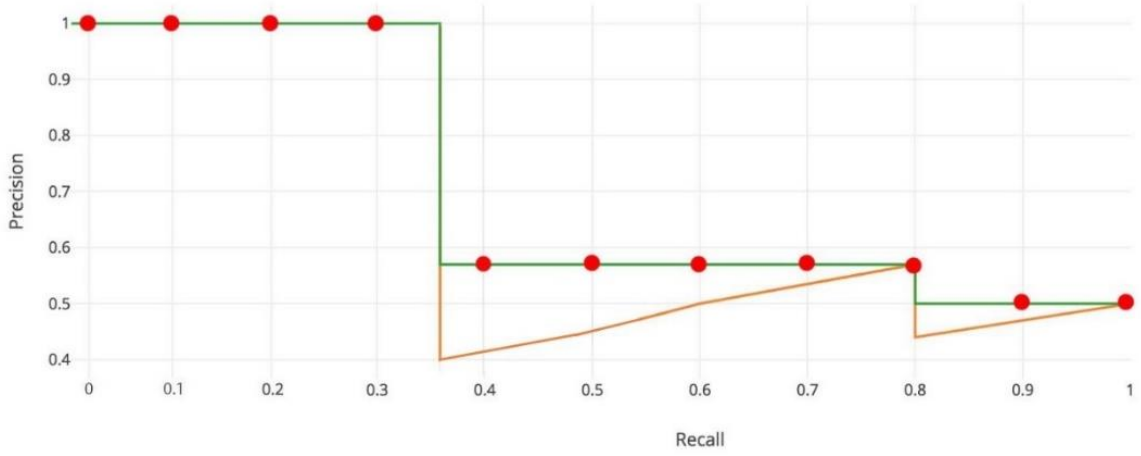
- **IoU**: ta đã tìm hiểu ở phần trước.
- **Precision-Recall Curve**: Đường biểu biễn mối liên hệ giữa precision và recall.

3.9.2 AP và mAP

Với Precision-Recall Curve, **Area Under the Curve (AUC)** còn có một tên khác là **Average precision (AP).** Giả sử có **N** ngưỡng (ngưỡng thường là IoU) để tính precision và recall, với mỗi ngưỡng cho một cặp giá trị precision, recall là $\mathbf{P_n}$, $\mathbf{R_n}$, **n = 1, 2, …, N**. Precision-Recall curve được vẽ bằng cách vẽ từng điểm có toạ độ **(Rn,Pn)** trên trục toạ độ và nối chúng với nhau. AP được xác định bằng:

$$AP=\sum n(R_n-R_n-1)P_n$$

Ở đó $\mathbf{(R_n-R_n-1)P_n}$ chính là diện tích hình chữ nhật có chiều rộng $\mathbf{(R_n-R_n-1)}$ và chiều cao $\mathbf{P_n}$, đây cũng gần với cách tính tích phân dựa trên cách tính diện tích của từng hình chữ nhật nhỏ.



Hình 3.27 Giá trị AP là giá trị phía dưới đường biểu diễn Precision-Recall Curve

mAP (mean Average Precision) là trung bình của AP được tính cho tất cả các lớp. Trong một vài trường hợp, mAP chính là AP.

Kí hiệu:

- **AP@[0.5:0.05:0.95]** : Giá trị AP trung bình của các AP khi IoU có giá trị (0.5, 0.55, 0.6, …, 0.95)
- **AP@0.5**: Giá trị AP khi IoU = 0.5
- **AP (small/medium/large)**: AP cho small/medium/large object (object có kích thước: nhỏ hơn 32×32; từ 32×32 đến 96×96; lớn hơn 96×96).

Mối quan hệ giữa precision – recall giúp mAP đánh giá được về độ chính xác của classification task. Precision – recall thay đổi khi ngưỡng IoU thay đổi, do đó, tại một giá trị IoU xác định, có thể đo hoặc so sánh độ tốt của các mô hình.

CHƯƠNG 4. THIẾT KẾ VÀ XÂY DỰNG CHƯƠNG TRÌNH

4.1 Hiện trạng tổ chức

Các vấn đề mà một hệ thống hướng dẫn bãi đỗ xe PGS cần đó là:

- Phát hiện những vị trí còn trống và đã đỗ trong bãi.
- Đếm những vị trí còn trống và đã đỗ.

4.2 Yêu cầu hệ thống

4.2.1 Yêu cầu chức năng

- Giới thiệu sản phẩm công nghệ hiện tại.
- Thông tin liên hệ.
- Hiển thị video camera theo thời gian thực tại bãi đỗ xe bao gồm:
  - Hiển thị những phát hiện gồm nhãn và vị trí còn trống, đã đỗ.
  - Hiển thị số lượng vị trí còn trống, đã đỗ và tổng các vị trí hiện có.

4.2.2 Yêu cầu phi chức năng

- Độ chính xác của mô hình cần tinh chỉnh phù hợp để quá trình nhận diện diễn ra thuận lợi trong thời gian dài.
- Giao diện và hiệu ứng đẹp, đơn giản, thân thiện dễ sử dụng với mọi đối tượng người dùng.
- Cần chú thích rõ ràng, dễ hiểu với người sử dụng.
- Thông tin giới thiệu các lợi ích và chức năng ngắn gọn, dễ hiểu, đầy đủ.
- Tốc độ của video camera phải ổn định có thể nhìn thấy rõ các vị trí phát hiện được và số lượng vị trí đó trong bãi.
- Thông tin giới thiệu ngắn gọn rõ ràng.
- Video camera phải hiển thị chính xác bất kể điều kiện thời tiết và sánh sáng khác nhau làm nhiễu hình ảnh.
- Nhãn của các vị trí còn trống và đã đỗ có màu khác nhau để người dùng dễ phân biệt.

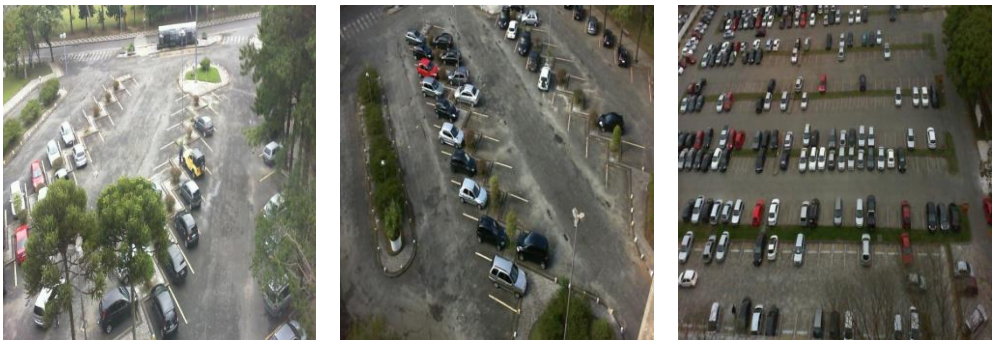
4.3 Huấn luyện mô hình

Huấn luyện mô hình là bước đầu tiên trong một trình phát hiện đối tượng, lúc này mô hình sẽ được học có giám sát và tự rút trích ra các đặc trưng thành một tập các trọng số (weights) từ tập dữ liệu đầu vào.

4.3.1 Thu thập dữ liệu

a) Tập dữ liệu lớn

Đối với tập dữ liệu lớn dùng để phát hiện nhiều đối tượng trong một ảnh, số lượng ảnh rất lớn phù hợp để huấn luyện mô hình YOLOv3. Ta tải tập dữ liệu này được chia sẻ công khai trên Roboflow, đây là trang web chia sẻ những tập dữ liệu được gán nhãn sẵn và đầy đủ các định dạng cần thiết cho huấn luyện các mô hình CNN. Tập dữ liệu đó tên là PKLot Dataset[15], sau đó tải với định dạng YOLO Darknet để phù hợp với công cụ Darknet ta sẽ dùng để huấn luyện. Định dạng này bao gồm các ảnh và tệp văn bản trong đó ảnh và tệp annotation chứa nhãn có định dạng *txt* cùng tên với nhau, tập dữ liệu này bao gồm 8169 ảnh khác nhau với kích thước 640 x 640.



Hình 4.1 Một số hình ảnh trong tập dữ liệu PKLot Dataset

b) Tập dữ liệu mô phỏng

Đối với tập dữ liệu mô phỏng dùng để mô phỏng mô hình trực tiếp, tự thu thập và gán nhãn cho ảnh, cũng như kiểm tra xem với dữ liệu ít ỏi thì mô hình còn chính xác hay không. Để dữ liệu chính xác và tiện trong quá trình gán nhãn, ta phải thu thập dữ liệu từ một góc nhìn nhất định trong suốt quá trình và các điều kiện ánh sáng khác nhau từ camera, tổng cộng ảnh thu thập được là 202 ảnh với kích thước 1024 x 768.

Tuy nhiên để tối ưu tốc độ và độ chính xác của mô hình lúc huấn luyện thì ta sẽ dùng công cụ Photoshop[16] với tính năng Actions để resize hàng loạt các ảnh về kích

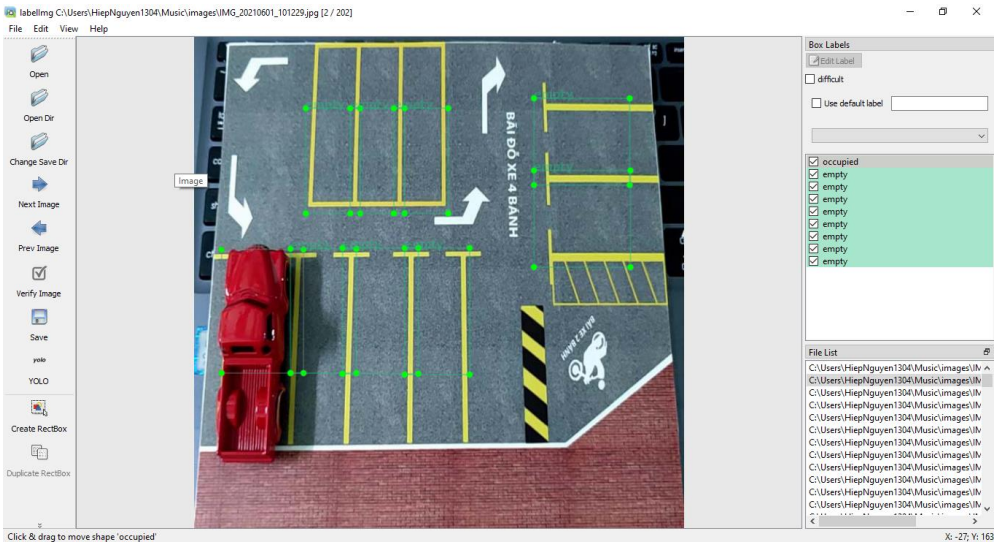
[15] https://public.roboflow.com/object-detection/pklot
[16] Photoshop là một phần mềm chỉnh sửa đồ họa được phát triển và phát hành bởi hãng Adobe Systems

thước 640 x 640, và giảm chất lượng lẫn kích thước tệp ảnh xuống thấp (tuy vậy vẫn giữ được những đặc trưng ban đầu) giảm áp lực học cho mô hình.

Hình 4.2 Dữ liệu mô phỏng tự thu thập

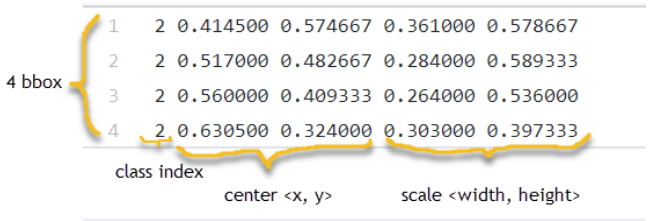
4.3.2 Gán nhãn cho ảnh

Hiện tại có rất nhiều các công cụ mã nguồn mở khác nhau được sử dụng để gán nhãn cho mô hình YOLO, tuy nhiên sau khi dùng thử một số phần mềm thì tôi sử dụng công cụ *labelImg*[17] của **pypi**.



Hình 4.3 Gán nhãn bằng labelImg

Bên dưới là ví dụ nội dung của một tệp ***annotation.txt*** của một ảnh:



Hình 4.4 Nội dung một tệp annotation

[17] https://pypi.org/project/labelImg/

Nội dung của file annotation sẽ bao gồm:

<class-index> <center-x> <center-y> <bbox-width> <bbox-height>

Trong đó: các giá trị *<center-x> <center-y> <bbox-width> <bbox-height>* là tâm và kích thước width (chiều rộng), height (chiều cao) của bounding box đã được chuẩn hóa bằng cách chia cho chiều rộng và chiều cao của ảnh, do đó các giá trị ngày luôn nằm trong khoảng [0, 1]. <class-index> là giá trị index đánh dấu các classes. Trong trường hợp một ảnh có nhiều bounding box thì tệp annotation sẽ gồm nhiều dòng, mỗi một bounding box là một dòng.

4.3.3 Cấu hình các tệp cần thiết

a) Tạo tệp train/validation

Ta sẽ tạo ra 2 tệp ***train.txt*** và ***val.txt*** chứa dữ liệu đường dẫn tới các file hình ảnh nằm trong tập train (chứa dữ liệu huấn luyện) và validation (chứa dữ liệu kiểm thử) bằng một đoạn script viết bằng Python. Khi đó đối với mỗi ảnh thì đường dẫn của nó nằm trên một dòng khác nhau.

```
data/images/IMG_20210601_101240.jpg
data/images/IMG_20210601_101247.jpg
data/images/IMG_20210601_101309.jpg
data/images/IMG_20210601_101319.jpg
data/images/IMG_20210601_101324.jpg
```

Hình 4.5 Nội dung trong tệp train.txt

b) Tạo file object name

Đây là tệp chứa tên các lớp mà chúng ta sẽ huấn luyện mô hình. Trên file này, thứ tự các tên class cần phải đặt đúng với index của nó trong các tệp nhãn của vật thể, ta tạo tệp ***obj.names*** và ghi tên 2 lớp ta cần huấn luyện là **empty** (còn trống) và **occupied** (đã đỗ), với mỗi lớp nằm trên mỗi dòng khác nhau.

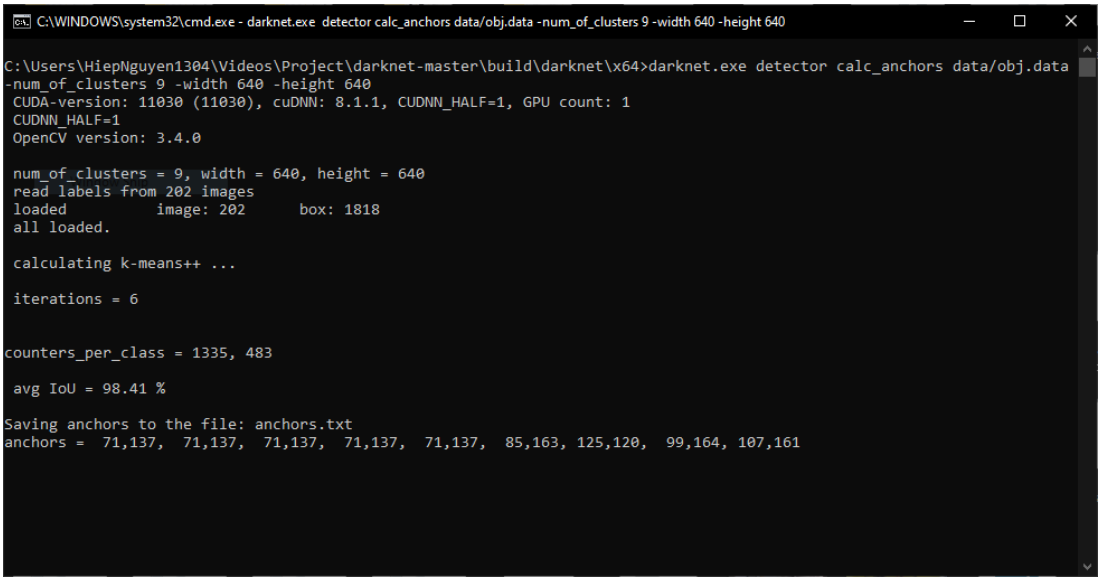
c) Tạo tệp config data

Tệp config data (tên tệp là ***obj.data***) sẽ khai báo một số thông tin như:

- Số lượng các lớp
- Đường dẫn tới các tệp **train.txt**, **val.txt**
- Đường dẫn tới file **obj.names**
- Thư mục backup mô hình huấn luyện.

d) Tạo anchor

Ta sẽ tạo anchor mới cho tập dữ liệu mô phỏng bằng thuật toán phân cụm k-mean bằng đoạn script được tác giả cung cấp sẵn, kết quả phân tích tạo ra kết quả như sau:



Hình 4.6 Kết quả 9 điểm neo mới được sinh ra từ thuật toán k-mean

e) Chỉnh sửa tệp config model

Để sử dụng mô hình Darknet, ta clone repository của tác giả Joseph Redmon tại địa chỉ *https://github.com/pjreddie/darknet* về máy.

Ta chỉnh sửa tệp ***yolov3.cfg*** tại thư mục **cfg**, đây là tệp cấu hình của YOLOv3, chứa thông tin về số lớp, filter, các lớp tích chập, các thông số học,… Ta chỉnh sửa lại như sau:

- Tại dòng 6: Thay đổi *batch=64.* Nghĩa là: batch = số ảnh (cả tệp annotation) được đưa vào huấn luyện trong một batch.
- Tại dòng 7: Thay đổi *subdivisions=16*. Trong một batch được chia thành nhiều block, mỗi block chứa batch/subdivisions ảnh được đưa vào GPU xử lý tại một thời điểm. Weights của mô hình được update sau mỗi batch.
- Tại các dòng 8, 9: Thay *width, height* thành *640* đối với tập dữ liệu lớn và *416* đối với tập dữ liệu mô phỏng (đây là giá trị trung bình mang giá trị ngữ nghĩa cao kể cả sau này khi tăng hay giảm đi kích thước ảnh và phù hợp cho cấu hình máy tính hiện tại, trong đó giá trị kích thước càng cao thì độ chính xác càng cao và ngược lại).

- Tại các dòng 610, 696, 783: Thay thành *classes=2* là số lượng classes (lớp) chúng ta huấn luyện.

- Tại các dòng 603, 689, 776: Thay số lượng *filter=21*. Đây chính là layer cuối cùng của base network. Do đó chúng có output shape thay đổi theo số lượng classes theo đúng công thức của bài trước đó là: *(n_classes + 5)x3 = (2+5)x3 = 21*.

- *max_batches*: tại dòng 20 là số lượng batch tối đa để huấn luyện mô hình YOLO. Theo như ý kiến của tác giả thì *max_batches = n_classes × 2000*, và không được nhỏ hơn số lượng ảnh. Đối với tập dữ liệu ảnh có số lượng ảnh gần 9000 nên ta đặt *max_batches = 9000*, và đối với tập dữ liệu mô phỏng thì *max_batches = 4000*.

- *burn_in*: Tại dòng 19 là số lượng batch ban đầu được giữ sao cho learning_rate rất bé. Giá trị này sẽ tăng dần từ 0 đến learning_rate. Sau đó learning_rate sẽ được giữ ổn định. Thực nghiệm cho thấy thiết lập learning_rate bé ở những steps đầu sẽ giúp cho thuật toán hội tụ nhanh hơn. Đối với tập dữ liệu lớn *max_batches* chỉ là 9000 nên cần điều chỉnh giảm *burn_in = 200*, còn đối với tập dữ liệu mô phỏng *burn_in = 80*.

- *steps*: Tại dòng 22. Theo ý kiến của tác giả, thì số lượng *steps* chiếm 80% và 90% của *max_batches,* đối với tập dữ liệu lớn điều chỉnh về steps=7200,8100. Còn đối với tập dữ liệu mô phỏng *steps*=3200,3600. Đây là các vị trí step mà chúng ta sẽ bắt đầu giảm dần learning_rate vì thuật toán đã đạt tới điểm hội tụ nên không cần thiết lập learning_rate quá cao.

- Tại các dòng 609, 695, 782: Thay thành *anchors = 71,137, 71,137, 71,137, 71,137, 71,137, 85,163, 125,120, 99,164, 107,161* (chỉ tập dữ liệu mô phỏng).

Ngoài những thuộc tính được chỉnh sửa như trên thì còn nhiều thuộc tính khác được để giá trị mặc định như: *momentum=0.9, learning_rate=0.001, channels=3,…*

4.3.4 Huấn luyện mô hình

a) Tiến hành huấn luyện

Tiếp theo ta sẽ tải toàn bộ mô hình Darknet lên Google Drive sau đó dùng Google Colab mount dữ liệu qua thư mục ở Google Drive (vì dữ liệu lưu trên Colab chỉ lưu được khá ngắn, và vì quá trình train lâu nên dễ bị lỗi gián đoạn làm dừng quá trình train, dễ bị mất dữ liệu khi bị ngừng session) và tiến hành quá trình *make* các tệp cần thiết trên mô hình Darknet. Cuối cùng ta huấn luyện mô hình bằng cú pháp:

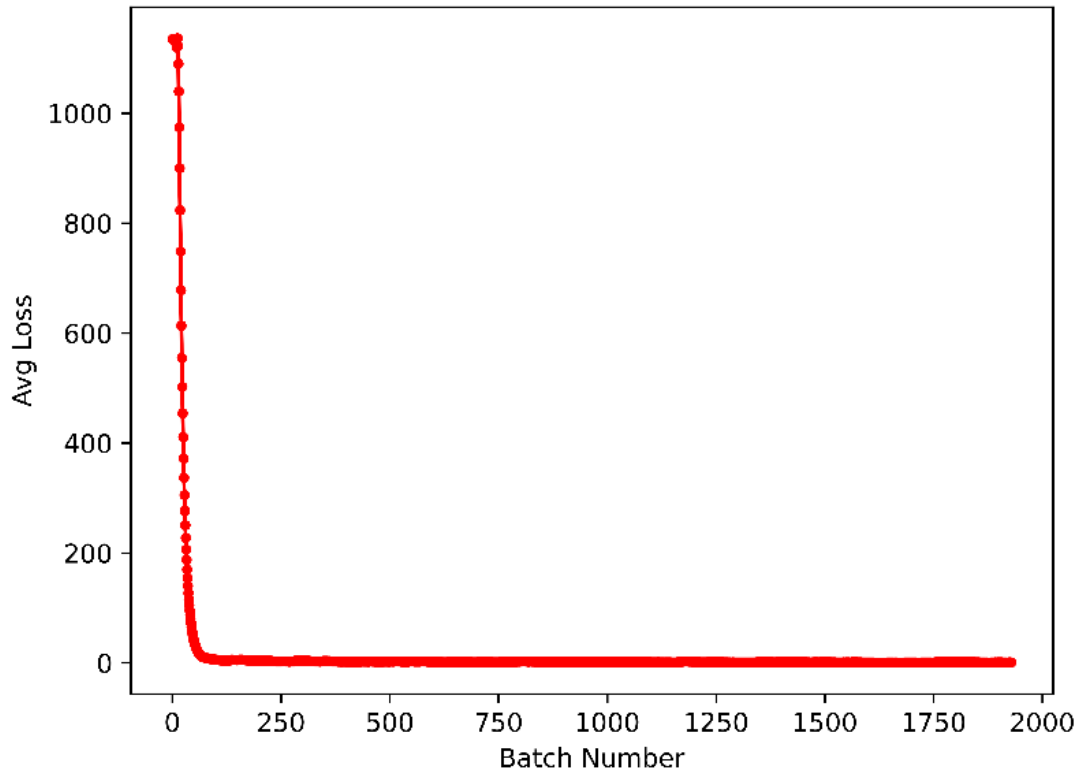
!./darknet detector train [tệp data config] [tệp model config] [pretrain-model weights] > [tệp lưu log]

Trong đó: Tệp *data config, model config* đã được mô tả ở trên.

- *pretrain-model weights*: Chúng ta sẽ sử dụng pretrain model darknet53.conv.74 được huấn luyện từ bộ dữ liệu ImageNet để khởi tạo các trọng số sử dụng *Fine-tuning*.
- *tệp lưu log*: Tệp lưu giữ các giá trị output của mô hình.

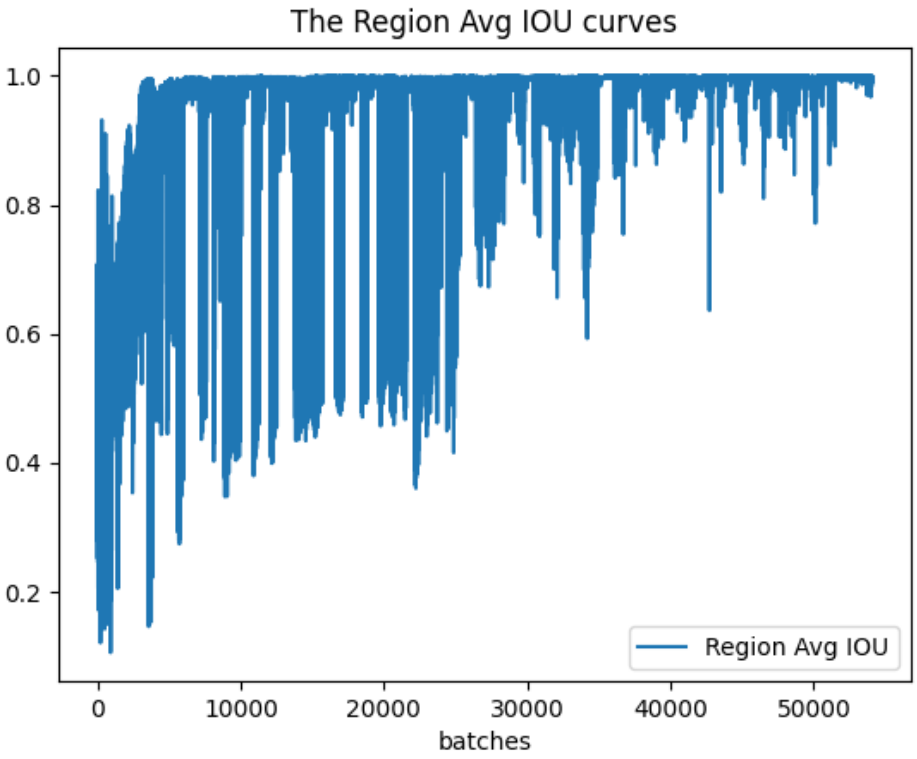
Bất cứ khi nào quá trình huấn luyện bị dừng đột ngột, ta có thể train tiếp từ điểm checkpoint được lưu từ tệp *.weights* hay *.backup* cuối cùng trong thư mục **backup** được sinh ra sau mỗi 100 batch cập nhật sau 1000 vòng lặp, bằng cú pháp sau:

!./darknet detector train [tệp data config] [tệp model config] backup/[last weights] > [tệp lưu log]



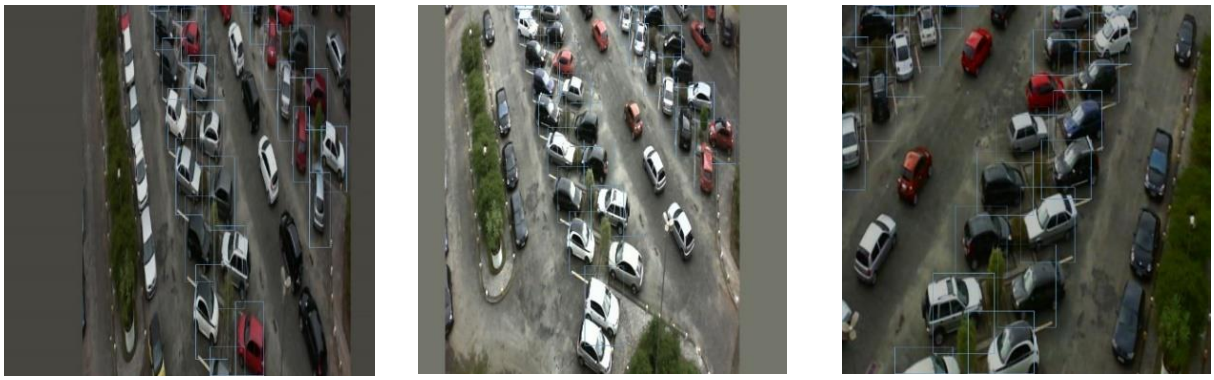
Hình 4.7 Trực quan hoá hàm mất mát sau gần 2000 batch của tập dữ liệu mô phỏng

Ta có thể thấy sau khoảng 80 batch thì thuật toán đã bắt đầu hội tụ đúng như giá trị *burn_in* trong cấu hình tệp *yolov3.cfg*. Sau khoảng vài ngàn lần lặp thì giá trị *avg_loss* không thấy tăng và đã rất nhỏ thì ta có thể dừng quá trình train sớm bởi vì lúc này thuật toán đã hội tụ và giá trị hàm mất mát đã tiệm cận 0.



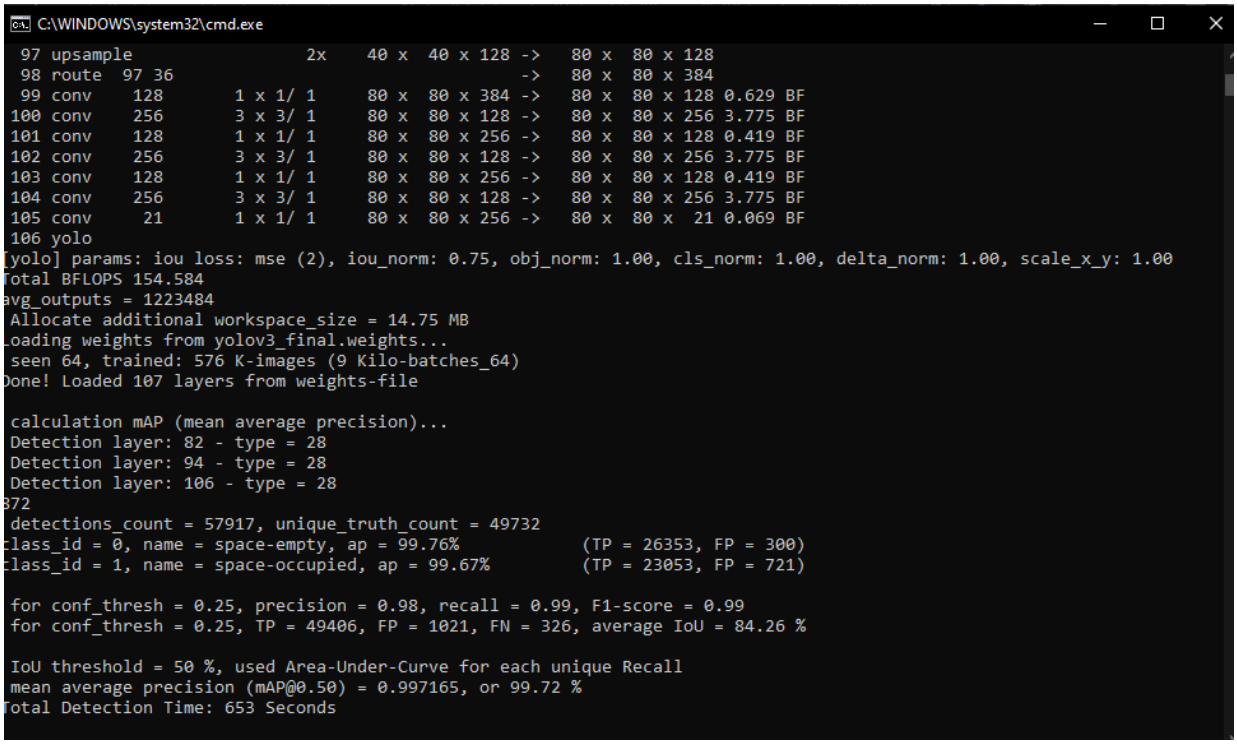
Hình 4.8 Trực quan hoá Region Avg IOU của tập dữ liệu mô phỏng

Region Avg IOU: Cho biết IOU trung bình của hình ảnh trong subdivision hiện tại, ta có thể thấy thì chỉ số IOU dần chính xác hơn ở các số batch lớn hơn bởi vì quá trình học diễn ra thuận lợi và đúng với dự định ban đầu.



Hình 4.9 Các phương pháp data augmentation được sử dụng trong quá trình học

Tạo ra nhiều mẫu huấn luyện hơn bằng cách sử dụng các thông số *saturation, exposure, hue* (trong hệ màu HSV), và trong đó có thông số *random* sẽ huấn luyện mạng ở các độ phân giải khác nhau, từ đó tăng độ chính xác của mô hình.



Hình 4.10 Kết quả tính toán mAP@50 = 99.72% trên 872 ảnh của tập dữ liệu lớn

Tệp **weights**: Một số định dạng của các tệp lưu trọng số đó là *.weights, .backup* hay có thể là số như tệp pretrained model *darknet53.conv.74* như ở trên đã giới thiệu. Về bản chất thì các loại định dạng tệp này chỉ dùng để phân biệt chúng với nhau vì chúng đều là những tệp nhị phân. Nhưng tệp pretrained chỉ lưu giữ kiến trúc backbone, được sử dụng 1 lần cho kỹ thuật fine-tuning. Cấu trúc tệp *.weight* bao gồm:

- Tham số phiên bản major (kiểu int, 4 byte)
- Tham số số phiên bản minor (kiểu int, 4 byte)
- Bản sửa đổi tham số số phiên bản (kiểu int, 4 byte)
- Số lượng ảnh được huấn luyện đã nhìn thấy (kiểu size_t, loại 8 byte trong hệ điều hành 64 bit)
- Tham số lớp chuyển đổi: trọng số chập (kiểu float, mỗi tham số 4 byte).

b) Một số thông tin về output của mô hình

- Với tập dữ liệu lớn tổng cộng mô hình cần dự đoán số bouding box tối đa ở 3 tỷ lệ là: (20*20 + 40*40 + 80*80) * 3 = 25200 mỗi ảnh; với mỗi tỷ lệ dự đoán tổng cộng 5 + 2 = 7 tham số đối với mỗi đối tượng trên mỗi bounding box trong mỗi ảnh. Ta sử dụng batch = 9000, với mỗi batch ta có 64 ảnh, vậy số bounding box tối đa cần dùng để dự đoán trong toàn bộ quá trình huấn luyện là 25200 * 9000 *

64 = 1.45152E10 (với E10 = 10^{10}). Số ảnh được tạo ra trong quá trình huấn luyện tối đa là 9000 * 64 = 576000 ảnh.

- Với tập dữ liệu mô phỏng tổng cộng mô hình cần dự đoán số bouding box tối đa ở 3 tỷ lệ là: (13*13 + 26*26 + 52*52) * 3 = 10647 mỗi ảnh; với mỗi tỷ lệ dự đoán tổng cộng 5 + 2 = 7 tham số đối với mỗi đối tượng trên mỗi bounding box trong mỗi ảnh. Ta sử dụng batch = 4000, với mỗi batch ta có 64 ảnh, vậy số bounding box tối đa cần dùng để dự đoán trong toàn bộ quá trình huấn luyện là 10647 * 4000 * 64 = 2.725632E9 (với E9 = 10^9). Số ảnh được tạo ra trong quá trình huấn luyện tối đa là 4000 * 64 = 256000 ảnh.
- Ví dụ output của quá trình huấn luyện có dạng:

"19: 0.783273, 0.827796 avg, 0.000020 rate, 1.650999 seconds, 339 images

Region 82 Avg IOU: 0.759600, Class: 0.809470, Obj: 0.732717, No Obj: 0.002799, .5R: 1.000000, .75R: 0.500000, count: 4

Region 94 Avg IOU: 0.699416, Class: 0.917663, Obj: 0.226457, No Obj: 0.000643, .5R: 1.000000, .75R: 0.333333, count: 6

Region 106 Avg IOU: 0.667185, Class: 0.959919, Obj: 0.089371, No Obj: 0.000099, .5R: 1.000000, .75R: 0.000000, count: 1"

Giải thích:

- *19:* cho biết số lần lặp của quá trình huấn luyện hiện tại;
- *0.783273* là giá trị mất mát tổng thể (hàm mất mát);
- *0.827796* avg là giá trị mất mát trung bình;
- *0.000020 rate* thể hiện tốc độ học (learning rate) hiện tại.
- *1.650999 seconds* cho biết tổng thời gian huấn luyện theo batch hiện tại;
- *339 images* là tổng số ảnh được tạo ra tính tới hiện tại.
- *Region 82 Avg IOU: 0.759600* cho biết IOU trung bình của hình ảnh trong subdivision hiện tại (bằng 75%), ở lớp thứ 82, dự đoán đối tượng có kích thước nhỏ.
- *Region 94 Avg IOU: 0.699416* Cho biết IOU trung bình của hình ảnh trong subdivision hiện tại (bằng 69%), ở lớp thứ 94, dự đoán đối tượng có kích thước trung bình.

- *Region 106 Avg IOU: 0.667185* cho biết IOU trung bình của hình ảnh trong subdivision hiện tại (bằng 66%), ở lớp thứ 106, dự đoán đối tượng có kích thước lớn.
- *Class: 0.809470* là tỷ lệ chính xác phân loại nhãn của đối tượng.
- *Obj: 0.732717* là giá trị trung bình của điểm đối tượng.
- *No Obj: 0.002799* là giá trị trung bình của điểm không có đối tượng.
- *.5R: 1.000000* là tỷ lệ các mẫu positive có IOU lớn hơn 50%.
- *.75R: 0.500000* là tỷ lệ các mẫu positive có IOU lớn hơn 75%.
- *count: 4* cho biết số lượng hình ảnh huấn luyện có các mẫu positive thực sự trong tất cả các hình của subdivision hiện tại.

4.4 Triển khai thuật toán YOLOv3 bằng Pytorch và xử lý website

4.4.1 Triển khai thuật toán YOLOv3

a) Xây dựng backbone

Trong phạm vi bài báo cáo này vì để bài báo cáo không quá dài nên tôi chỉ tóm tắt quá trình triển khai thuật toán và không đi vào chi tiết. Quá trình triển khai thuật toán bằng Pytorch có sử dụng một số lớp cơ bản như ***nn.Module***, ***nn.Sequential*** và ***torch.nn.parameter***. Trong đó các mạng neural sẽ được xây dựng dựa trên package ***torch.nn*** và ***nn.Module*** sẽ bao gồm các layers và một phương thức **forward (input)** để trả ra kết quả **output**. ***nn.Sequential*** là một lớp container (chứa) mở rộng lớp cơ sở ***nn.Module*** và cho phép chúng ta compose (dàn xếp, kết hợp) các module với nhau. ***torch.nn.parameter*** khi được sử dụng với ***nn.Module*** sẽ tự động được thêm vào danh sách các tham số của nó.

Cấu trúc thư mục mô hình của mô hình YOLOv3 được triển khai như sau:

- **colors**: Chứa tệp nhị phân định dạng màu
  pallete
- **config**: Chứa tệp cấu hình mạng
  yolov3.cfg
- **data:** Chứa tệp danh sách tên lớp
  parking.names
- **outputs**: Chứa ảnh đã được xử lý khi dùng module phát hiện ảnh

- **util**: Chứa kiến trúc backbone của mô hình

  __init__.py

  image_processor.py

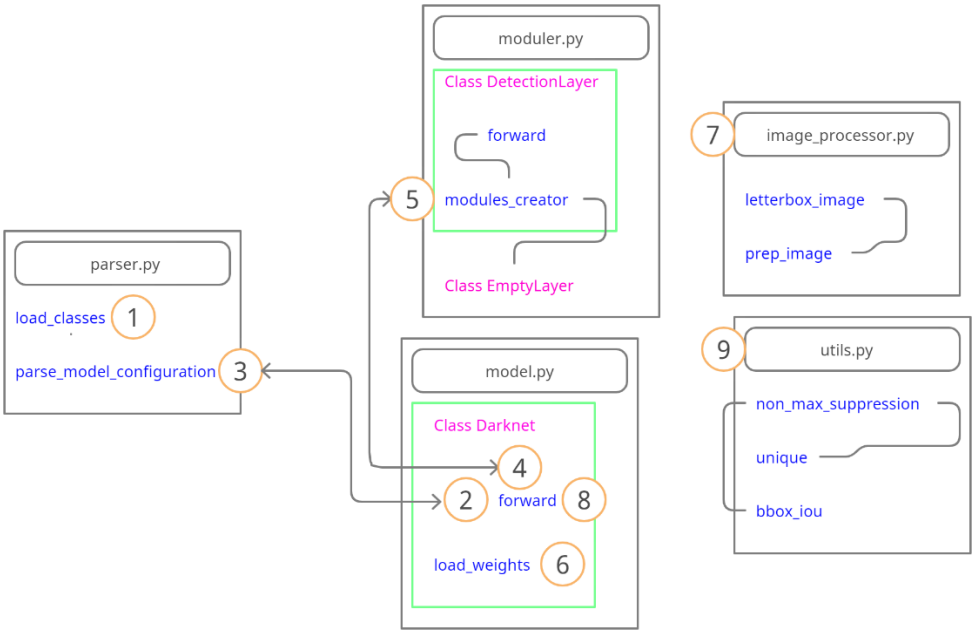
  model.py

  moduler.py

  parser.py

  utils.py

- **weights**: Chứa tệp weights của mô hình



Hình 4.11 Sơ đồ quá trình xử lý trong backbone

b) Xây dựng trình phát hiện kiểm tra suy luận của mô hình

Đối với module phát hiện đối tượng trong ảnh:

Ta tạo tệp *images.py*, đầu tiên ta import các module cần thiết ở phần backbone và thư viện opencv, sử dụng module ArgParse của Python để nhận vào đối số đường dẫn tới tệp hoặc thư mục ảnh cần phát hiện. Khởi tạo các siêu tham số như *outputs, batch_size, confidence, nms_thesh, classes* tương ứng với thư mục sẽ xuất ảnh, kích thước batch, độ tự tin, ngưỡng của NMS, các lớp cần phát hiện. Sau đó tải tệp cấu hình và tệp weights. Kiểm tra xem máy có hỗ trợ CUDA không, nếu có thì đặt mô hình vào CUDA để GPU xử lý, gắn một số cờ (flag) cần thiết ở các vị trí khác nhau để đo tốc độ đọc ảnh. Dùng opencv đọc hình ảnh từ đĩa hoặc hình ảnh từ một thư mục, nếu sai đường

dẫn hay tên tệp thì thông báo lỗi và thoát ra khỏi chương trình. Các đường dẫn của ảnh được lưu trữ trong một danh sách được gọi là *imlist*. Tạo batch và bắt đầu vòng lặp phát hiện cho toàn bộ số ảnh. Sử dụng hàm Counter từ module collections của Python để đếm các đối tượng được nhận diện được để tiến hành đếm các chỗ trống và đã đỗ. Còn một cách nữa để đếm các đối tượng đó là lưu các đối tượng nhận diện được vào một list, sau đó dùng vòng lặp để đếm tần suất xuất hiện của các đối tượng để tính tổng. Tiếp đó vẽ bounding box, in ra nhãn và xuất ảnh bằng *cv2.imwrite*. Nếu không có bất kì phát hiện nào thì thông báo cho người dùng và thoát khỏi chương trình. Cuối cùng in ra những thông tin log tổng hợp về quá trình phát hiện, sử dụng tính năng đồng bộ hoá của CUDA để tính thời gian chạy.

Kết quả dưới đây được chạy trên laptop với GPU NVIDIA GeForce 940MX (2GB VRAM), CPU Intel i5-8250U (8GB RAM). Đây là lần đầu quá trình phát hiện cho tệp ảnh được chạy nên tốc độ có thể chậm hơn một chút so với chạy các lần sau nếu có nhiều ảnh vì mô hình mới được khởi tạo lần đầu.

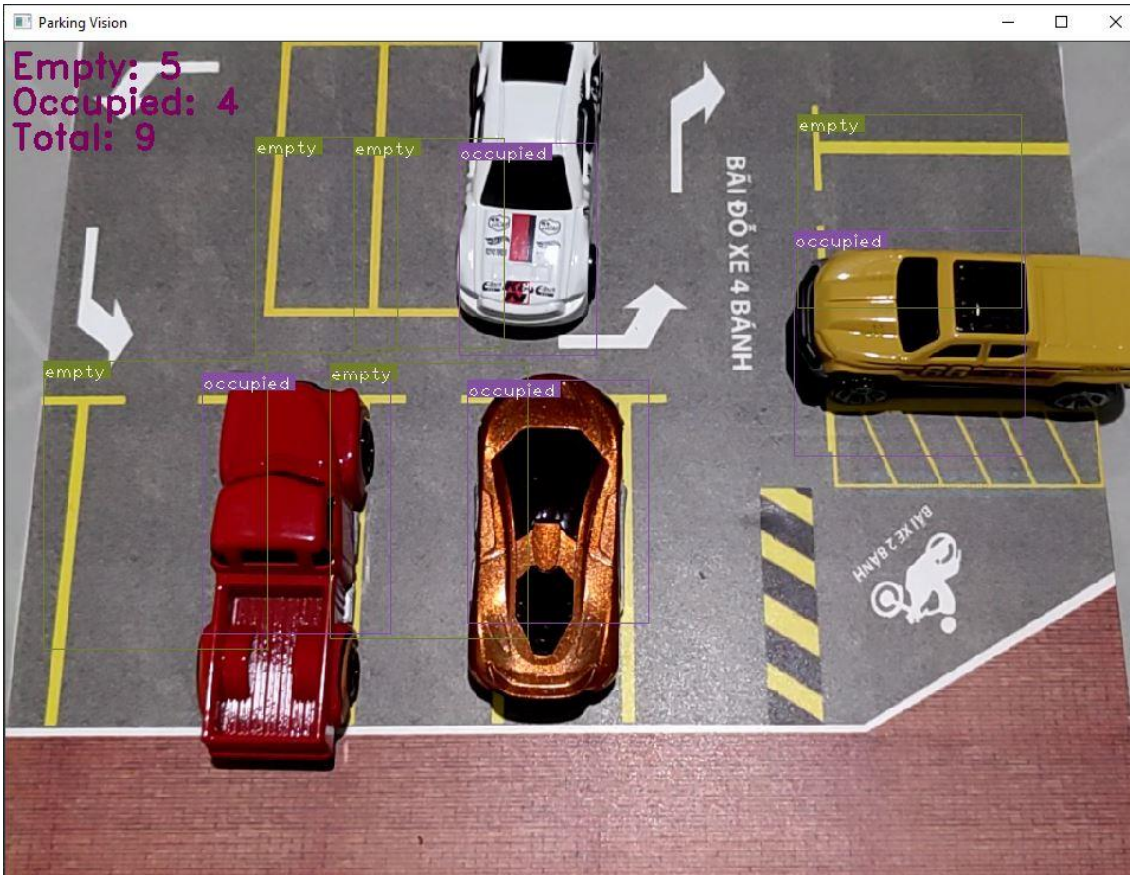
```
Đang load cấu hình và trọng số mô hình.....
Mô hình load thành công!
test.jpg được nhận diện trong 1.875 giây
empty: 24
occupied: 16
Tổng các vị trí: 40
------------------------------------------------------------------------
Tổng Kết
------------------------------------------------------------------------
Công việc                          : Thời gian xử lý (tính bằng giây)

Đọc địa chỉ                        : 0.001
Loading batch                      : 0.045
Nhận diện (1 ảnh)                  : 1.914
Xử lý đầu ra                       : 0.000
Vẽ hộp giới hạn                    : 0.079
Thời gian trung bình mỗi ảnh       : 2.045
```

Hình 4.12 Output của trình phát hiện ảnh

Đối với module phát hiện đối tượng trong video/camera:

Đối với trình phát hiện đối tượng trong video/camera thì mã nguồn không thay đổi nhiều, thay vì lặp qua thư mục thông qua các batch, thì ở đây ta sẽ làm việc với *frame* (khung hình) của video, vì bản chất thì video cũng chỉ là một dãy các *frame* hay có thể xem là các bức ảnh được nối tiếp nhau trong thời gian nhất định.

Ta tạo tệp *cam.py* và gọi nó là một *Python App*, đầu tiên ta tiến hành đọc video/camera bằng opencv, ta lặp qua các khung hình giống như cách đã làm với thư mục ảnh. Các mã được đơn giản hoá ở nhiều nơi vì không cần phải xử lý theo batch mà chỉ có một ảnh (khung hình) tại một thời điểm. Sau mỗi lần lặp lại ta theo dõi số lượng khung hình (frame) bằng biến *frames*, sau đó ta chia số này cho thời gian trôi qua kể từ khung hình đầu tiên để in tỷ lệ khung hình trên giây (frames per second - FPS) của video. Cuối cùng ta sử dụng *cv2.imshow* để hiện khung hình này lên Python App với những bounding box được vẽ trên đó, vòng lặp sẽ liên tục cho đến khi người dùng dừng chương trình.



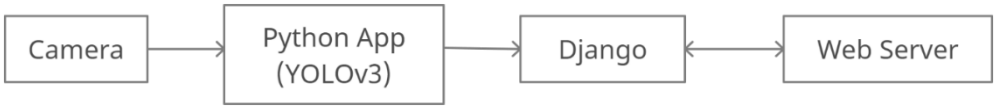
Hình 4.13 Trình phát hiện đối tượng chạy trên Python App

4.4.2 Xử lý website

Đầu tiên ta sẽ tạo trước các template gồm các tệp HTML được chia thành 4 tệp đó là*: home.html* chứa nội dung trang chủ và giới thiệu sản phẩm, *camera.html* chứa phần stream camera và hình ảnh trả về từ hàm **video_feed()**, *contact.html* chứa thông tin liên hệ và cuối cùng là *base.html* chứa header và footer chung của các trang trên. Về phần xử lý stream có các phương án xử lý khác nhau. Phương án đầu tiên ta xử lý tệp

view.py của Django bằng hàm **stream()** với mã gần giống hoàn toàn tệp *cam.py* của thuật toán YOLOv3, tạo hình ảnh và sử dụng phương thức *yield* để trả lại các frame ảnh, tiếp đó dùng hàm **video_feed()** với phương thức *StreamingHttpResponse* để stream các frame ảnh đó lên website. Tuy vậy phương án này tồn tại một lỗ hổng lớn đó là tiêu tốn quá nhiều tài nguyên hệ thống, vì với mỗi yêu cầu từ người dùng thì Web Server phải chạy lại từ đầu gây lãng phí, điều đó dẫn đến vấn đề là không cần một cuộc tấn công từ chối dịch vụ (DDOS)[18] và với một vài người dùng truy cập vào website thì website của chúng ta cũng tự sập vì tài nguyên hệ thống không đủ để đáp ứng cho thuật toán xử lý nặng như YOLOv3 với mỗi yêu cầu lặp lại như vậy. Điều đó cần một phương án giải quyết thứ 2 đó là tách YOLOv3 ra khỏi Django, sử dụng Python App của tệp *cam.py* chạy trước để liên tục sinh ra các hình ảnh từ các *frame,* Django liên tục nhận các hình ảnh đó và hiển thị lên website bằng các hàm tương tự với phương án đầu nói trên, sử dụng *gzip* với hàm **video_feed()** nén các tệp để website tải nhanh hơn. Phương án này đáp ứng cả 2 yêu cầu đó là vừa chạy được Python App song song với Django giúp người quản lý có thể vừa xem được hình ảnh trực tiếp từ Python App và người dùng có thể xem trên website. Cuối cùng thêm các URL đã được định nghĩa từ hàm *view.py* vào tệp *url.py* của Django.

Quá trình xử lý trong Django biểu diễn bằng sơ đồ sau:



Hình 4.14 Quá trình xử lý trong Django

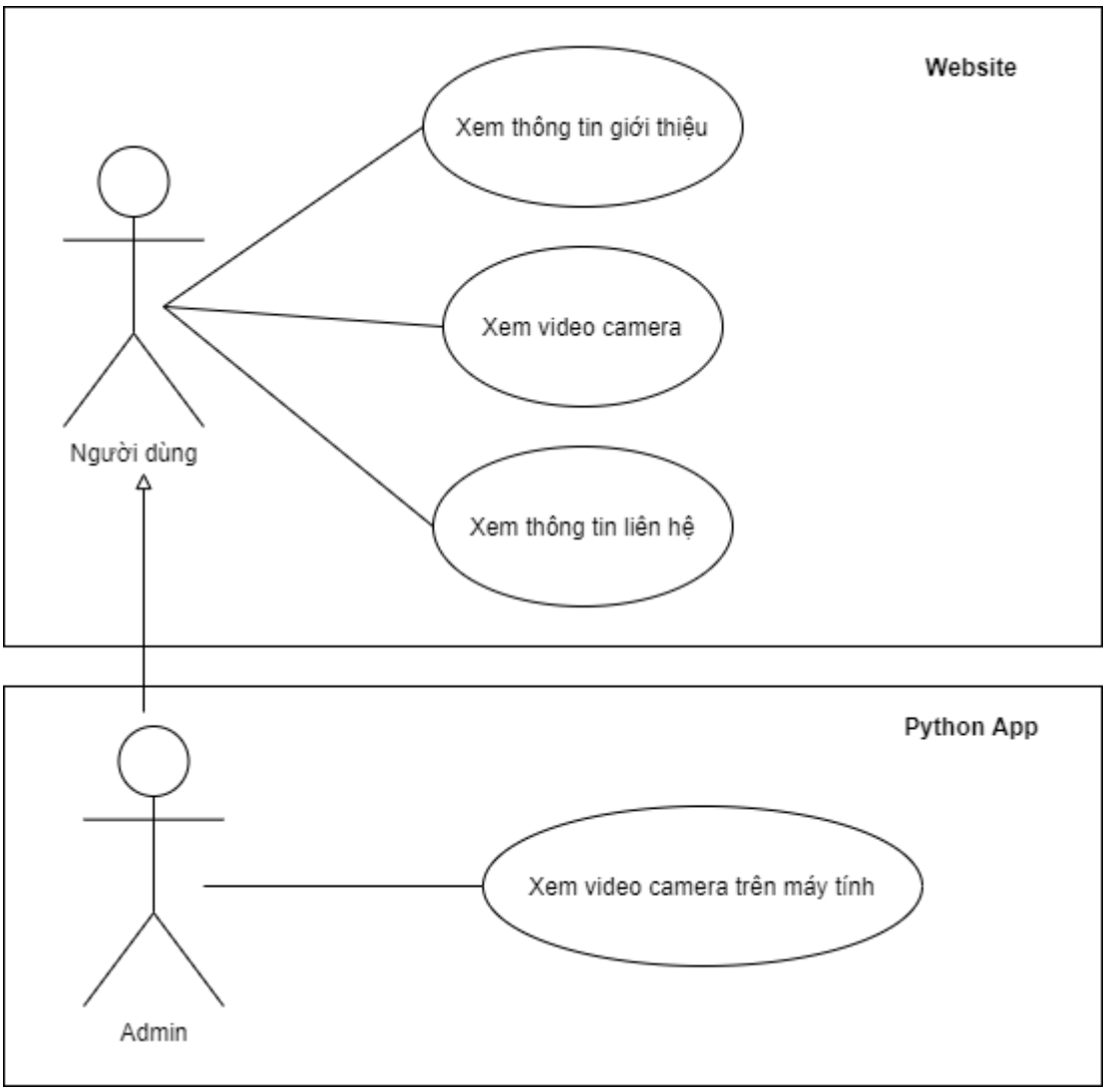
4.5 Sơ đồ use case của hệ thống

Biểu đồ usecase mô tả các chức năng của hệ thống, từ đó chỉ ra những công việc mà hệ thống cần làm để đáp ứng nhu cầu của người sử dụng.

Người quản trị (Admin) có thể chạy Website và Python App song song với nhau và có thể xem được cả 2.

Người dùng có thể xem thông tin giới thiệu, video camera và thông tin liên hệ.

[18] DDOS (Denial Of Service) là hình thức tấn công từ chối dịch vụ khá phổ biến, nó khiến cho máy tính mục tiêu không thể xử lý kịp các tác vụ và dẫn đến quá tải.

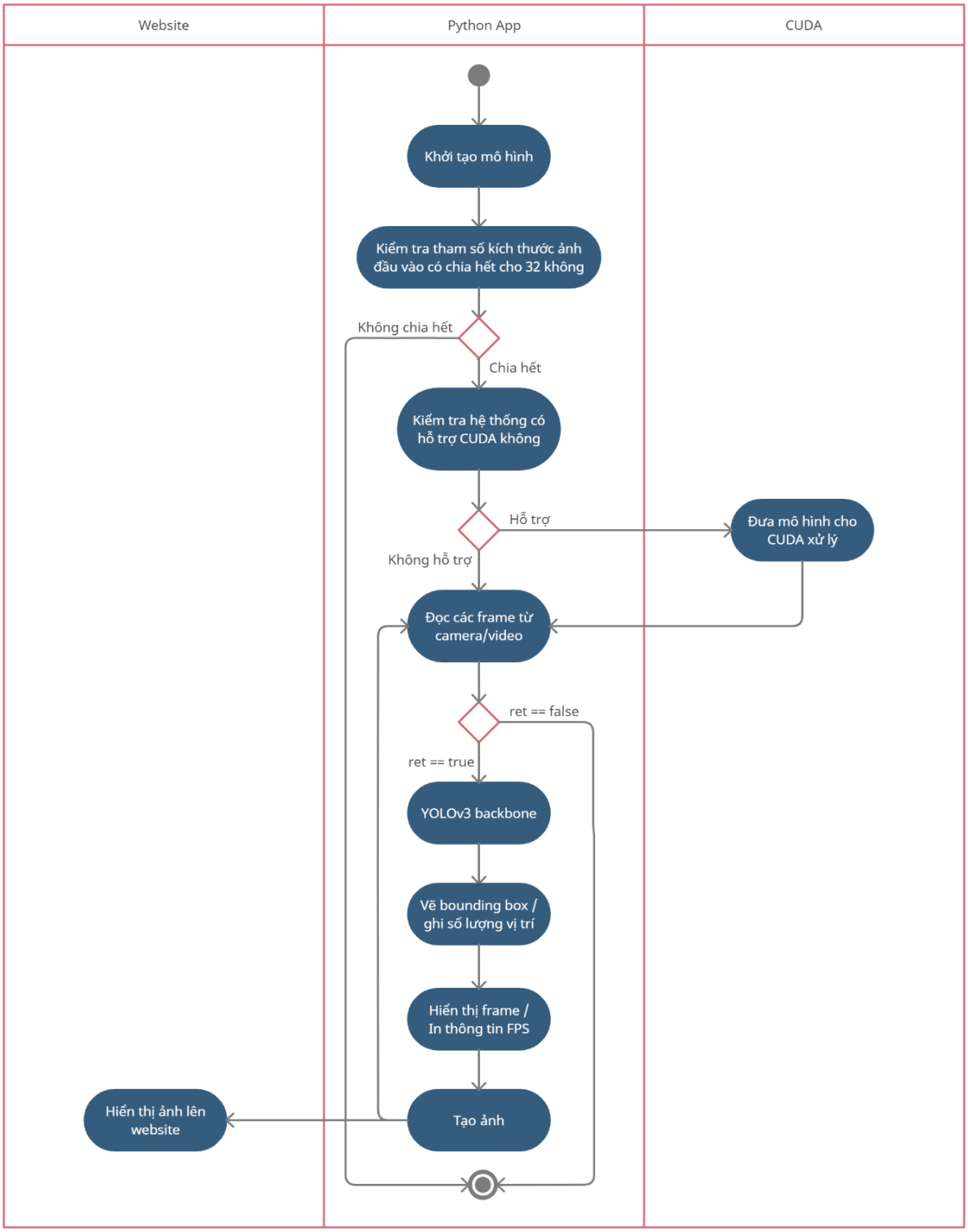


Hình 4.15 Sơ đồ use case hệ thống

4.6 Sơ đồ hoạt động

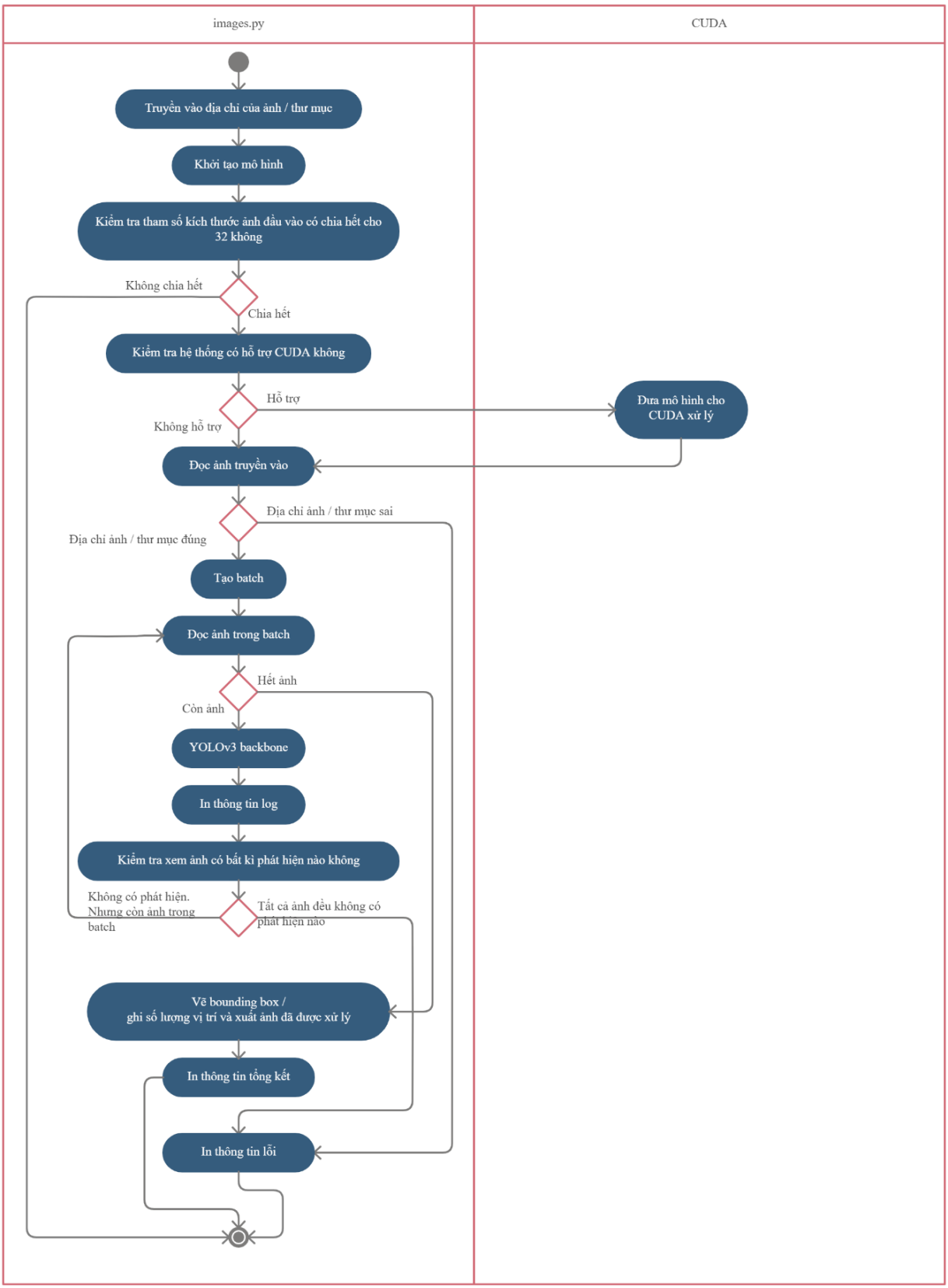
Sơ đồ hoạt động là một mô hình logic được dùng để mô hình hoá cho các hoạt động trong một quy trình nghiệp vụ. Nó chỉ ra luồng đi từ hoạt động này sang hoạt động khác trong một hệ thống. Nó đặc biệt quan trọng trong việc xây dựng mô hình chức năng của hệ thống và nhấn mạnh tới việc chuyển đổi quyền kiểm soát giữa các đối tượng.

4.6.1 Sơ đồ hoạt động truyền video lên website và Python App



Hình 4.16 Sơ đồ hoạt động truyền ảnh lên website

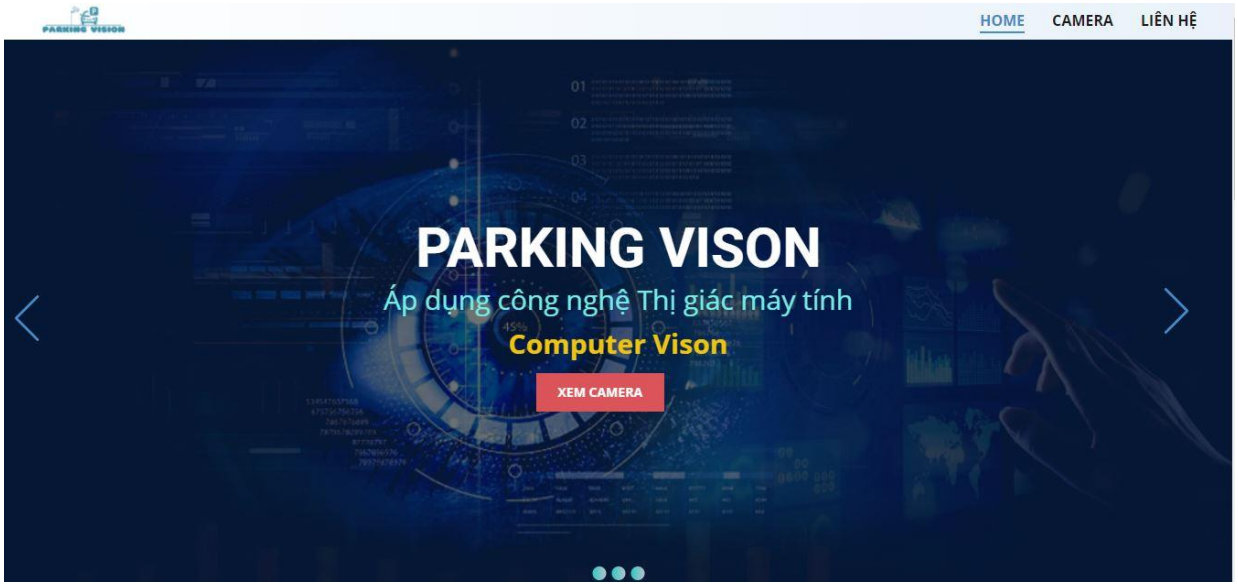
4.6.1 Sơ đồ hoạt động xử lý ảnh / thư mục ảnh



Hình 4.17 Sơ đồ hoạt động xử lý ảnh / thư mục ảnh

4.7 Xây dựng giao diện chương trình

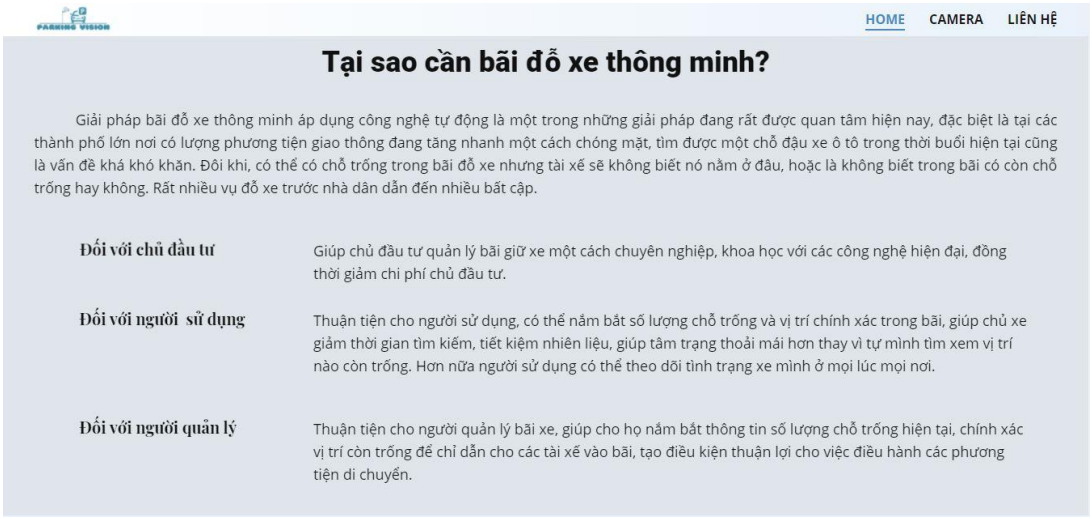
4.7.1 Giao diện trang chủ website – giới thiệu sản phẩm



Hình 4.18 Giao diện trang chủ

Các thông tin giới thiệu sản phẩm được thiết kế trên cùng một trang với trang chủ, thông tin ngắn gọn súc tích.

Hình 4.19 Giao diện giới thiệu sản phẩm dạng slide

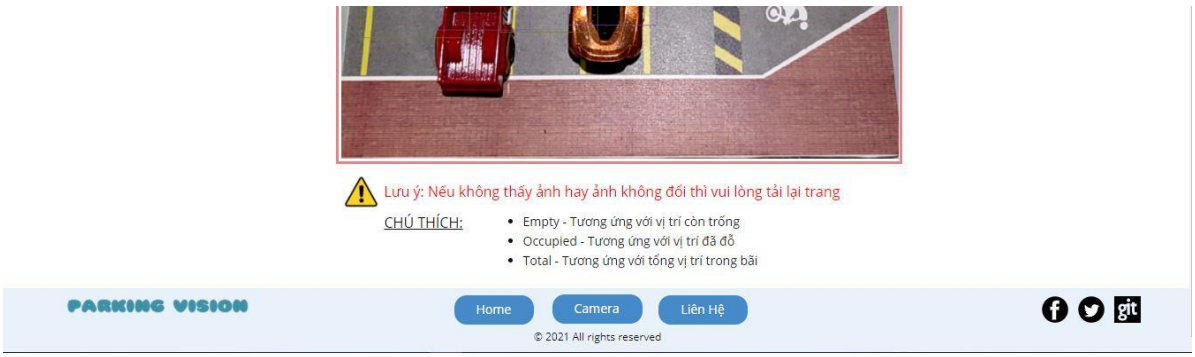


Hình 4.20 Giao diện giới thiệu sản phẩm

4.7.2 Giao diện xem video camera theo thời gian thực

Sau khi người dùng chọn chức năng xem camera thì sẽ gửi yêu cầu cho máy chủ xử lý và trả về dữ liệu video đang diễn ra tại bãi đỗ xe theo thời gian thực.

Hình 4.21 Giao diện xem camera theo thời gian thực



Hình 4.22 Giao diện chú thích cho camera theo thời gian thực

4.7.3 Giao diện thông tin liên hệ

Chứa thông tin liên hệ của chúng ta.

Hình 4.23 Giao diện thông tin liên hệ

KẾT LUẬN

Kết quả đạt được

Sau khi tìm hiểu, nguyên cứu, phân tích, thực hiện và kiểm nghiệm trên thực tế, đề tài đã được những yêu cầu ở mục tiêu đã đề ra:

- Xây dựng được hệ thống hướng dẫn bãi đỗ xe ô tô với các chức năng tiện lợi và cần thiết như:

  - Cho phép người dùng theo dõi trực tiếp tình hình hiện tại tại bãi đỗ xe trên website. Người quản lý có thể xem trực tiếp tình hình trên màn hình Python App cho tốc độ cao và dễ quan sát hơn.
  - Admin có thể chạy song song cả Python App và website. Và có thể dùng module xử lý ảnh để kiểm tra tính chính xác của mô hình khi huấn luyện.

- Bộ phát hiện đối tượng có thể phát hiện ở các điều kiện thời tiết, ánh sáng với độ chính xác cao, có thể chạy ổn định chuyển đổi linh hoạt giữa CPU và GPU. Website có giao diện thân thiện, tính tương thích với điện thoại, máy tính cao, dễ sử dụng, đồng thời website cũng thực hiện tốt trên hầu hết các trình duyệt web như: Chrome, Edge,…

Nhược điểm

+ Do website được xử lý theo dạng lấy kết quả từ quá trình sinh ảnh từ Python application và sau đó hiển thị lên bằng thẻ html *img* không cần tải lại trang và quá trình này diễn ra liên tục nên có thể có một số thời điểm website sẽ lấy ảnh đúng lúc ảnh chưa sinh ra khiến website lỗi không hiện ảnh.

+ Module phát hiện vật thể trong ảnh / thư mục chưa ghi được tệp khi đối số truyền vào là thư mục.

+ Website còn ít chức năng và Python App không được viết bởi các framework hỗ trợ lập trình GUI nên giao diện đơn giản và không có chức năng.

+ Chưa chạy module nhận diện qua camera/video tại các bãi đỗ xe thật.

+ Đây là mô hình khá nặng nên yêu cầu máy chủ có cấu hình cao và tích hợp GPU. Đồng thời mã nguồn có thể còn chưa tối ưu và trong lớp upsampling sử dụng kỹ thuật Nearest-Neighbor để đổi lại tốc độ xử lý nên độ chính xác sẽ không cao bằng Bilinear.

+ Cần thu thập khá nhiều dữ liệu ảnh về đối tượng cần nhận diện và xử lý chúng tốn khá nhiều thời gian sau đó huấn luyện để đạt được sự khái quát hoá cao.

Hướng phát triển

- Khắc phục những nhược điểm trên.
- Trang website có thể thêm chức năng đặt trước chỗ để xe.
- Tìm kiếm đối tác ngành Robotics kết hợp với mô hình trên để có thể hướng dẫn cho các xe tự hành có thể tự nhận diện được vị trí chưa đỗ trong bãi đỗ xe.
- Thêm khả năng nhận diện các phương tiện khác như xe máy, container,…
- Phân quyền thêm cho website bằng một trang chỉ cho admin truy cập để xem stream, lấy dữ liệu thu được lưu vào cơ sở dữ liệu.

PHỤ LỤC

Phụ lục 1: Hướng dẫn cài đặt

- Tải tệp weights tại địa chỉ:

https://drive.google.com/drive/folders/1sne3rveUK3j0QYPTrdc96MB6OT_5z0jf?usp=sharing

Sau đó đặt tệp weights tại thư mục **weights**.

- Tinh chỉnh các siêu tham số *confidence, nms_thesh* ở mức vừa đủ nhận diện được tất cả các đối tượng, càng tăng các thông số này thì độ chính xác càng cao. Chỉnh siêu tham số *inp_dim* ở độ phân giải vừa đủ với cấu hình máy tính (thường trung bình là 416). Thay đổi tên tệp weights tại phương thức *model.load_weights()* với **yolov3.weights** là tệp weights cho bộ dữ liệu mô phỏng, **bigyolov3.weights** là tệp weights cho bộ dữ liệu lớn. Đối với module nhận diện ảnh, có thể kiểm tra bằng những hình ảnh tại thư mục **imgs**.
- Cài đặt các môi trường cần thiết trong tệp **requirements.txt**.

Phụ lục 2: Hướng dẫn sử dụng

- Đối với module nhận diện ảnh, chạy theo lệnh sau để kiểm tra:

  python images.py --images [tên ảnh]

- Đối với module nhận diện qua camera, chạy bằng lệnh sau để kiểm tra:
  python cam.py
- Đối với website, chạy bằng lệnh sau để kiểm tra:
  python manage.py runserver
- Website sau khi triển khai, kiểm tra tại địa chỉ:
  https://parkingvision.herokuapp.com/
  (Do máy chủ web cần cấu hình cao và camera để chạy liên tục nên không thể triển khai cùng được, vì vậy đồ án này chỉ triển khai trước phần web Front End, phần demo trực tiếp sẽ sử dụng **ngrok** để tạo đường hầm (tunnel) từ localhost ra internet).
- Xem video demo chạy website và 2 module nhận diện qua camera và ảnh tại địa chỉ website: https://youtu.be/IHwrNHjOy_Y

TÀI LIỆU THAM KHẢO

[1]. Joseph Redmon & Ali Farhadi, *YOLOv3: An Incremental Improvement*, University of Washington, 2018.

[2]. Samuel Ordonia, *Detecting Cars in a Parking Lot using Deep Learning*, A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science, San Jose State University, 2019.

[3]. Arepalli Rama Venkata Naga Sai, *Car Parking Occupancy Detection using YOLOv3*, A thesis submitted in partial fulfillment of the requirements for the degree of Master of Engineering in Microelectronics and Embedded Systems, Asian Institute of Technology, 2019.

[4]. Lê Thị Thu Hằng, nghiên cứu về mạng neural tích chập và ứng dụng cho bài toán nhận dạng biển số xe, Luận văn thạc sĩ trường Đại học Công nghệ, Đại học Quốc gia Hà Nội, 2016.

[5]. Mykel J. Kochenderfer & Tim A. Wheeler, *Algorithms for Optimization*, The MIT Press, 2019.

[6]. Andrew Ng, *Machine Learning Yearning*, 2018.

[7]. Alberto Fernández Villán, *Mastering OpenCV 4 with Python*, Packt, 2019.

[8]. Nguyễn Quang Hoan, Giáo Trình Xử Lý Ảnh, Học Viện Công Nghệ Bưu Chính Viễn Thông, 2006.

[9]. Vũ Hữu Tiệp's Blog, "*Machine Learning cơ bản*", https://machinelearningcoban.com, truy cập ngày 06 tháng 07 năm 2021.

[10]. How to implement a YOLO (v3) object detector from scratch in PyTorch, https://blog.paperspace.com/how-to-implement-a-yolo-object-detector-in-pytorch/, truy cập ngày 06 tháng 07 năm 2021.

[11]. Logistic regression, http://maitrongnghia.com/2020/04/logistic-regression/, truy cập ngày 06 tháng 07 năm 2021.

[12]. Normalization and Normalization Techniques in Deep Learning, https://viblo.asia/p/normalization-and-normalization-techniques-in-deep-learning-QpmleJyn5rd, truy cập ngày 06 tháng 07 năm 2021.

[13]. Hướng dẫn tất tần tật về Pytorch để làm các bài toán về AI, https://viblo.asia/p/huong-dan-tat-tan-tat-ve-pytorch-de-lam-cac-bai-toan-ve-ai-YWOZrNkNZQ0, truy cập ngày 06 tháng 07 năm 2021.

[14]. Deep Learning - Cách làm việc với CUDA trong PyTorch, https://gpuhub.net/deep-learning-cach-lam-viec-voi-cuda-trong-pytorch/, truy cập ngày 06 tháng 07 năm 2021.

[15]. Wikipedia, *"Sai số toàn phương trung bình"*, https://vi.wikipedia.org/wiki/Sai_số_toàn_phương_trung_bình, truy cập ngày 06 tháng 07 năm 2021.

[16]. How to train YOLOv3 on the custom dataset, https://thebinarynotes.com/how-to-train-yolov3-custom-dataset/, truy cập ngày 06 tháng 07 năm 2021.

[17]. Training AlekseyAB YOLOv3 on own dataset in Google Colab, https://vovaprivalov.medium.com/training-alekseyab-yolov3-on-own-dataset-in-google-colab-8f3de8105d86, truy cập ngày 06 tháng 07 năm 2021.

[18]. Django documentation, https://docs.djangoproject.com/en/3.2/, truy cập ngày 06 tháng 07 năm 2021.

[19]. Tiêu chuẩn thiết kế bãi đỗ xe ô tô mới nhất 2021, https://bilparking.com.vn/article/kich-thuoc-tieu-chuan-bai-do-xe-oto-2020, truy cập ngày 06 tháng 07 năm 2021.

[20]. Xử lý dữ liệu với Pandas trong Python, https://codelearn.io/sharing/xu-ly-du-lieu-voi-pandas-trong-python, truy cập ngày 06 tháng 07 năm 2021.