



BỘ GIÁO DỤC VÀ ĐÀO TẠO

ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

KHOA KHOA HỌC MÁY TÍNH



ĐỒ ÁN MÔN HỌC

CS231 – NHẬP MÔN THỊ GIÁC MÁY TÍNH

ĐỀ TÀI

NHẬN DIỆN NGƯỜI CÓ HAY KHÔNG ĐEO KHẨU TRANG

Giảng viên hướng dẫn: ThS. Mai Tiến Dũng

Năm học: 2021 – 2022

Học kỳ: 2

Nhóm sinh viên thực hiện: Lê Quang Hùng

Tôn Anh Trúc

Mã số sinh viên: 20521363

20520944

Email: 20521363@gm.uit.edu.vn

20520944@gm.uit.edu.vn

Lớp: CS231.M22.KHCL



THÔNG TIN CHUNG

Đề tài:	Nhận diện người có hay không đeo khẩu trang.
Môn học:	CS231 – Nhập môn thị giác máy tính.
Lớp:	CS231.M22.KHCL.
Giảng viên hướng dẫn:	ThS. Mai Tiến Dũng.
Thời gian thực hiện:	Học kỳ 2. Năm học: 2021 – 2022.
Sinh viên thực hiện:	Tôn Anh Trúc – 20520944. Lê Quang Hùng – 20521363.
Nội dung đề tài:	<p>Nhận diện người có hay không đeo khẩu trang. Hay nói cách khác nhóm sẽ xác định xem trong hình có những người nào là có đeo khẩu trang, đeo khẩu trang sai hoặc không đeo khẩu trang, dựa vào đó tính được số lượng người ở mỗi trường hợp.</p> <p>Đưa ra kết quả thử nghiệm từ bộ dữ liệu mà nhóm đã thu thập. So sánh độ chính xác qua các phương pháp thực hiện khác nhau.</p>
Kế hoạch thực hiện:	<p>Tuần 1, 2: Thành lập nhóm. Chọn đề tài. Xây dựng kế hoạch thực hiện và phân công nhiệm vụ.</p> <p>Tuần 3, 4, 5, 6, 7: Khảo sát tình hình thực tế. Thu thập các thông tin liên quan và tìm nguồn tài liệu tham khảo. Tìm hiểu các kiến thức cần thiết cho quá trình thực hiện đề tài.</p> <p>Tuần 8, 9, 10, 11, 12: Tìm hiểu các công nghệ và công cụ có thể sử dụng trong đề tài. Thiết kế, xây dựng và viết chương trình cài đặt. Đánh giá và kiểm thử chương trình cài đặt.</p> <p>Tuần 13, 14: Chuẩn bị slide trình bày và tập duyệt</p>



báo cáo.

Tuần 15: Báo cáo đồ án. Tuần Dự trũ: Chính sửa báo cáo và chương trình cài đặt sau khi báo cáo.



LỜI CẢM ƠN

Nhóm xin chân thành gửi lời cảm ơn đến ThS. Mai Tiến Dũng – Giảng viên khoa Khoa học máy tính, Trường Đại học Công nghệ thông tin, Đại học Quốc gia thành phố Hồ Chí Minh, đồng thời là giảng viên giảng dạy lớp CS231.M22.KHCL – Môn Nhập môn thị giác máy tính, trong thời gian qua đã tận tình hướng dẫn và định hướng cho nhóm trong suốt quá trình thực hiện và hoàn thành đồ án.

Trong quá trình thực hiện đồ án nhóm đã cố gắng rất nhiều để hoàn thành đồ án một cách tốt nhất và hoàn thiện nhất, song cũng sẽ không tránh khỏi được những sai sót ngoài ý muốn. Nhóm mong rằng sẽ nhận được những lời nhận xét và những lời góp ý chân thành từ quý thầy/cô và các bạn trong quá trình thực hiện chương trình của nhóm để chương trình ngày càng hoàn thiện hơn. Mọi thắc mắc cũng như mọi góp ý của mọi người xin gửi email về một trong các địa chỉ email sau: 20520944@gm.uit.edu.vn (Tôn Anh Trúc), 20521363@gm.uit.edu.vn (Lê Quang Hùng). Mỗi ý kiến đóng góp sẽ là một nguồn động lực to lớn đối với nhóm để nhóm có thể cố gắng cải tiến chương trình ngày càng hoàn thiện và phát triển đồ án lên một mức cao hơn, nhóm cũng sẽ dựa vào đó để phát triển hơn những ưu điểm và cải thiện được phần nào đó những nhược điểm của chương trình. Hy vọng đề tài “Nhận diện người có hay không đeo khẩu trang” do nhóm thực hiện sẽ trở thành một công cụ hữu ích và có thể ứng dụng được trong lĩnh vực Thị giác máy tính và cũng như ở đời thực.

Thành phố Hồ Chí Minh, tháng năm 2022

Nhóm sinh viên thực hiện

Tôn Anh Trúc, Lê Quang Hùng



MỤC LỤC

	trang
Chương 1. TỔNG QUAN	6
1. Giới thiệu	6
2. Sơ lược về đề tài	17
3. Phát biểu bài toán	17
4. Mô hình tổng quát	19
5. Đối tượng nghiên cứu	20
Chương 2: CƠ SỞ LÝ THUYẾT	21
1. Lịch sử phát triển của YOLO	21
2. Cách thức YOLO nhận diện vật thể	51
Chương 3: GIẢI QUYẾT BÀI TOÁN	57
1. Hướng tiếp cận	57
2. Dataset	58
3. Train Model	62
YOLOv4-Tiny	62
YOLOv5	73
4. Test và đánh giá mô hình	78
Chương 4: KẾT LUẬN	92
1. Nhận xét về mô hình	92
2. Bài học kinh nghiệm	92
TÀI LIỆU THAM KHẢO	93



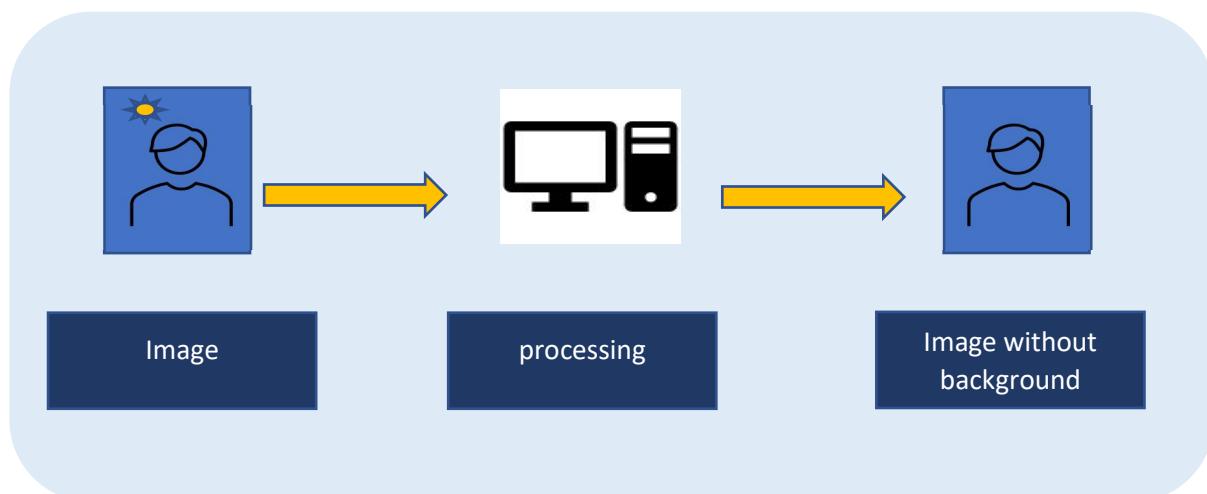
Chương 1. TỔNG QUAN

1. Giới thiệu

Khái niệm thị giác máy tính

Thị giác máy tính (Computer Vision) là một hình thức công nghệ mô tả khả năng của bộ máy có thể thu nhận và phân tích các dữ liệu trực quan, sau đó sẽ tiến hành đưa ra các quyết định về nó. Đơn giản thì thị giác máy tính là một công nghệ thuộc lĩnh vực khoa học máy tính và trí tuệ nhân tạo mà có tầm nhìn và khả năng xử lý nhận dạng như con người.

Mục đích của những người nghiên cứu về lĩnh vực này là tận dụng những kiến thức về thị giác của con người để phát triển những công cụ và kỹ thuật phù hợp vào hệ thống máy tính sao cho máy tính có thể giải quyết những vấn đề thực tế được đặt ra có liên qua đến thị giác máy tính. Tất cả có thể được thực hiện trên quy mô trên ảnh hay có thể là trên video.

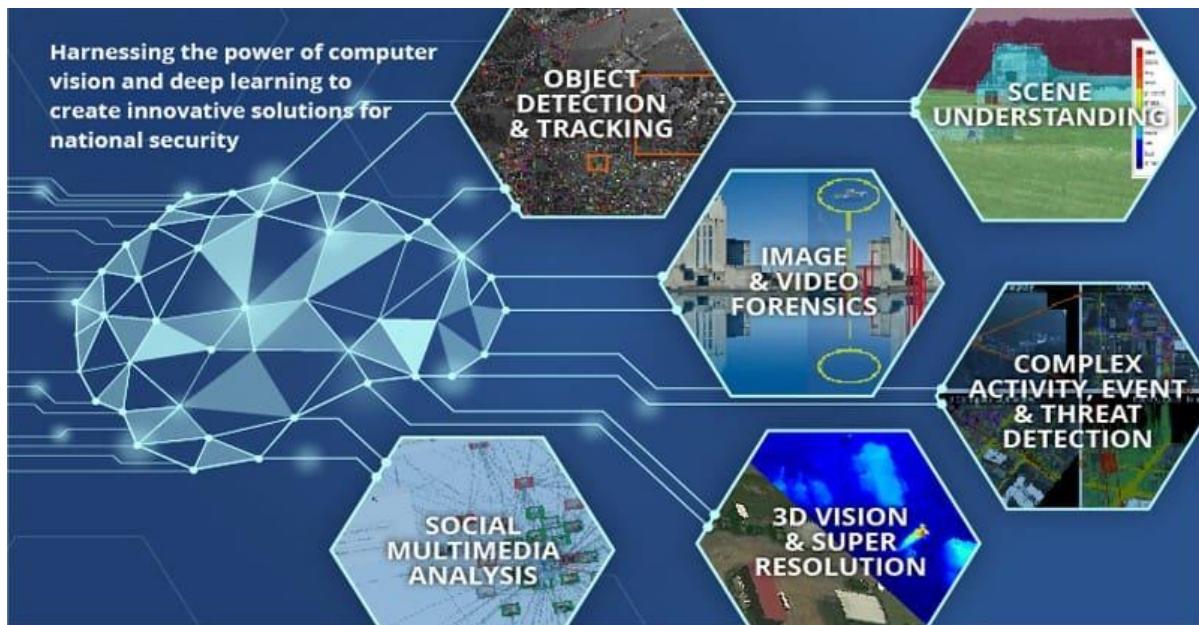


Mô phỏng quá trình áp dụng thị giác máy tính trong việc xóa background của hình ảnh (remove background)



Các lĩnh vực con của thị giác máy tính bao gồm tái cấu trúc cảnh, dò tìm sự kiện, theo dõi video, nhận diện bộ cục đối tượng, học, chỉ mục, đánh giá chuyển động và phục hồi ảnh.

Công nghệ này đề cập đến toàn bộ quá trình mô phỏng tầm nhìn trong một bộ máy phi sinh học của con người. Quá trình này bao gồm chụp ảnh, phát hiện và nhận dạng đối tượng, nhận biết bối cảnh tạm thời và sau đó nâng cấp sự hiểu biết về những sự kiện đã xảy ra trong một khoảng thời gian hợp lý.



Một số ứng dụng của thị giác máy tính hiện nay



Các cơ sở khoa học của thị giác máy tính

Thị giác: Là khả năng nhận và diễn giải thông tin từ ánh sáng đi vào mắt. Việc tri giác này còn được gọi là thị lực, sự nhìn. Những bộ phận khác nhau cấu thành thị giác được xem như là một tổng thể như là hệ thị giác và được tập trung nghiên cứu trong nhiều lĩnh vực khác nhau như tâm lý, khoa học nhận thức, khoa học thần kinh và sinh học phân tử.

Vấn đề chính của thị giác: Là những gì mà con người thấy được không phải chỉ là sự biến đổi của các kích thích ở võng mạc (tức là ảnh trên võng mạc). Vì vậy, những người có quan tâm đến dạng tri giác này đã cố gắng giải thích cách làm việc của tiếng trình xử lý hình ảnh để tạo ra cái mà chúng ta thực sự nhìn thấy.

Các nghiên cứu trước đây về thị giác: Có hai trường phái chính thời Hy Lạp cổ đưa ra các giải thích đầu tiên về cách hoạt động của việc nhìn trong cơ thể người.

- Trường phái thứ nhất là “lý thuyết phát xạ”: sự nhìn diễn ra khi các tia nhìn phát ra từ mắt bị chặn bởi các vật được nhìn thấy. Nếu nhìn một vật thì đó là do các tia nhìn đi từ mắt roi vào vật đó. Tuy nhiên, một ảnh khúc xạ cũng được nhìn thấy bởi các tia nhìn. Lúc này, tia nhìn đi từ mắt, xuyên qua không khí và rơi vào vật được nhìn thấy sau khi bị khúc xạ như là kết quả của sự chuyển động của các tia nhìn phát ra từ mắt.
- Trường phái thứ hai chủ trường về cách tiếp cận gọi là “sự dựa vào”, xem sự nhìn như là có một cái gì đó đại diện cho vật đi vào mắt. Với sự ủng hộ của các nhà truyền bá chính và những người kế tục, lý thuyết này dường như đã đạt đến một ít tri giác về bản chất của sự nhìn, nhưng ánh sáng chưa đóng vai trò gì trong lý thuyết này và nó chỉ là sự suy đoán mà không có các cơ sở thực nghiệm.

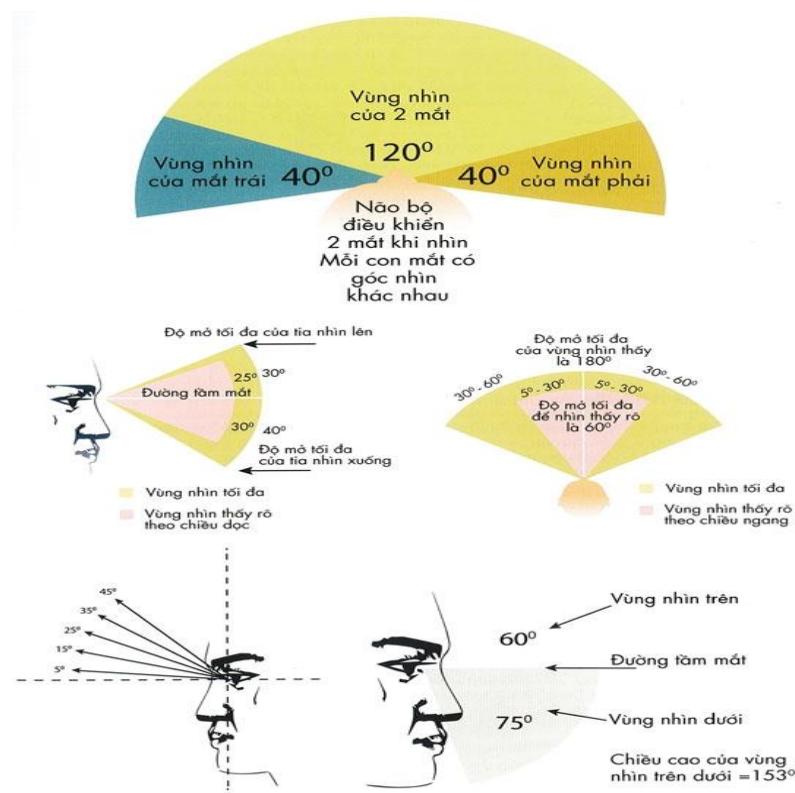
Suy luận vô thức:

- Nghiên cứu đầu tiên của Hermann von Helmholtz về thị giác trong thời hiện đại, ông khảo sát mắt người và kết luận rằng về mặt quan học thì mắt người khá kém cỏi. Theo ông thì tông tin thu được qua mắt kém chất lượng đến nỗi khó mà thấy gì được. Vì thế ông kết luận rằng sự nhìn chỉ có thể là kết quả của một dạng suy luận vô thức: một vấn đề về việc thừa nhận và kết luận từ những dữ liệu không đầy đủ, dựa trên những kinh nghiệm đã có.
- Sự suy luận này cần có những kinh nghiệm về thế giới. Các ví dụ đã được biết đến là:
 - o Ánh sáng đến từ phía trên
 - o Các vật thường không được nhìn từ phía dưới lên
 - o Các khuôn mặt được nhìn thấy (và nhận ra) từ phía bên phải
- Nghiên cứu về ảo giác (trường hợp mà quá trình suy luận là sai) đã đem lại những hiểu biết sâu hơn về các dạng suy luận mà hệ thị giác thực hiện.



Lý thuyết Gestalt:

- Các nhà tâm lý học cấu trúc hình thức (Gestalt psychologists) đã đưa ra nhiều câu hỏi nghiên cứu mà các nhà khoa học về thị giác thời nay đang nghiên cứu.
- Các định luật cấu trúc hình thức về sự tổ chức đã dẫn đến nghiên cứu về cách thức mà con người nhận thức các thành phần thị giác như là các mẫu thức hoặc các toàn thể được tổ chức thay cho nhiều phần riêng biệt khác nhau. Theo thuyết này thì có 6 yếu tố chính để xác định xem chúng ta ghép nhóm các đồ vật theo sự nhìn: sự gần gũi về không gian, sự tương tự, sự đóng kín, sự đối xứng, sự quen thuộc và sự liên tục.



Các vấn đề liên quan đến thị giác



Các lĩnh vực của thị giác máy tính

Xử lý hình ảnh:

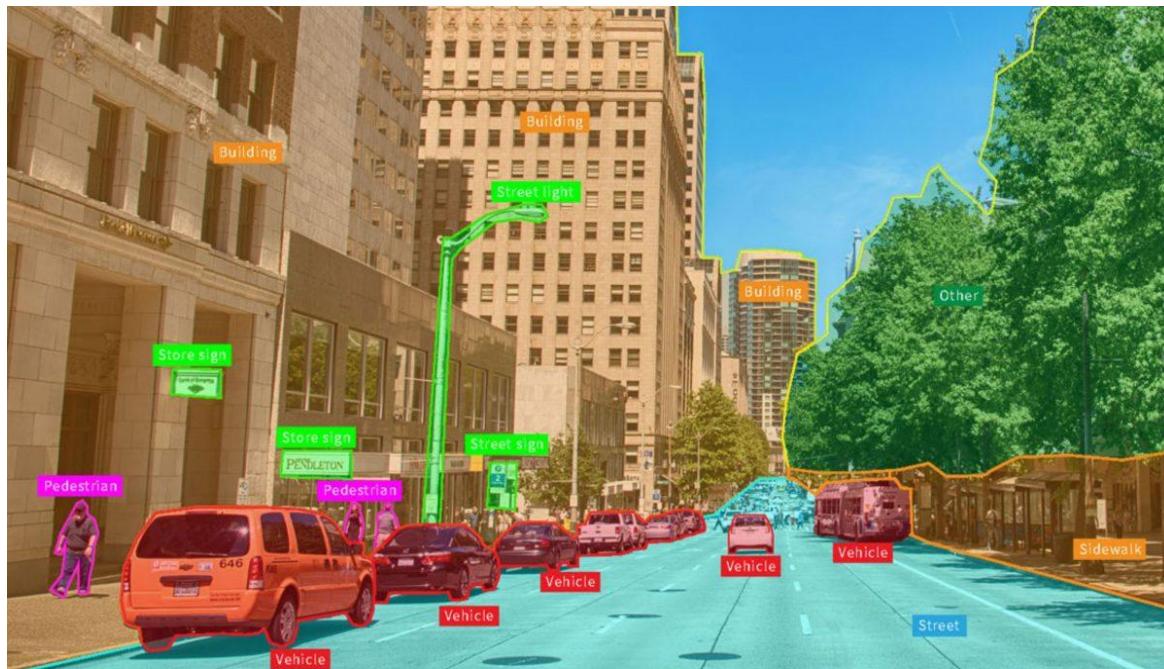
- Đây là một trong những mảng quan trọng nhất trong kỹ thuật thị giác máy tính, làm tiền đề cho nhiều nghiên cứu sau này. Nó là một lĩnh vực mang tính khoa học và công nghệ. Xử lý ảnh là một ngành khoa học mới mẽ so với nhiều ngành khoa học khác nhưng tốc độ phát triển của nó rất mạnh mẽ, kích thích các trung tâm nghiên cứu.
- Hai nhiệm vụ cơ bản của xử lý ảnh là nâng cao chất lượng thông tin hình ảnh và xử lý số liệu cung cấp cho các quá trình khác trong đó có việc ứng dụng thị giác vào điều khiển. Xử lý ảnh trước đây chủ yếu được sử dụng làm nâng cao chất lượng ảnh (gia tăng chất lượng ảnh quang học trong mắt người quan sát). Thời gian gần đây, phạm vi ứng dụng xử lý ảnh mở rộng không ngừng, có thể nói hiện không có lĩnh vực khoa học nào không sử dụng các thành tựu của công nghệ xử lý ảnh kỹ thuật số.



Xử lý hình ảnh (Image Processing)

Nhận diện mẫu: Giải thích các kỹ thuật khác nhau để phân loại mẫu.

Quang trắc: Liên quan đến việc thu thập các số đo chính xác từ hình ảnh.



Một ví dụ trong lĩnh vực nhận diện mẫu (Pattern Recognition)

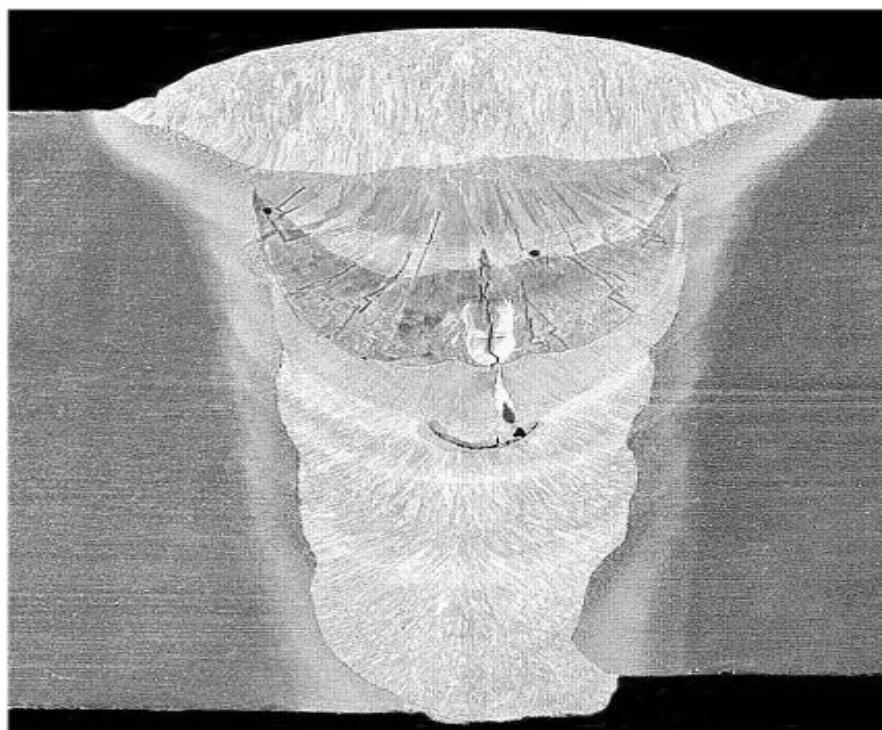
Ứng dụng của thị giác máy tính

Hiện nay công nghệ thị giác máy tính có rất nhiều ứng dụng vào đời sống thường ngày của chúng ta, chẳng hạn như máy chơi game có thể nhận ra cử chỉ của con người hoặc camera ở điện thoại di động có thể tự động tập trung vào các đối tượng chính ở trong bối cảnh mà chúng ta cần chụp hoặc là cần quay video. Nó đang tác động đến nhiều lĩnh vực trong cuộc sống của chúng ta.

Một số ứng dụng cơ bản nhất của thị giác máy tính hiện nay: Phát hiện các khiếm khuyết, trình học tự động, vận hành tự động, xử lý dữ liệu, lĩnh vực y tế, nhận diện khuôn mặt, ngân hàng, bán lẻ.

- Phát hiện các khiếm khuyết: Ứng dụng vào việc kiểm tra vết nứt kim loại, lỗi sơn, bản in xấu,...

Với thị giác máy tính thì chúng ta có thể kiểm tra tất cả các lỗi nhỏ nhất có kích thước nhỏ hơn 0,05mm.



Kiểm tra vết nứt trong mối hàn

- Trình đọc tự động: Một trong những ứng dụng nổi bật hiện nay là ứng dụng vào việc tự động dịch trong ứng dụng Google translate khi camera điện thoại được trỏ vào vùng văn bản của bất kỳ ngôn ngữ nào.
Sử dụng thuật toán nhận dạng ký tự (OCR) để trích xuất thông tin, cụ thể là nhận dạng ký tự quang học, cho phép một bản dịch chính xác sau đó chuyển thành lớp phủ lên văn bản thực.



Dịch trực tiếp bằng hình ảnh



- Vận hành tự động: Xe không người lái, máy bay không người lái, robot,... Trong lĩnh vực này phụ thuộc rất nhiều vào Computer vision và Deep learning.

Cụ thể như trong xe tự hành: Công nghệ AI phân tích dữ liệu thu thập được từ hàng triệu người lái xe, học hỏi từ hành vi lái xe để tự động tìm làn đường, ước tính độ cong đường, phát hiện các mối nguy hiểm và giải thích các tín hiệu và tín hiệu giao thông.



Xe tự hành

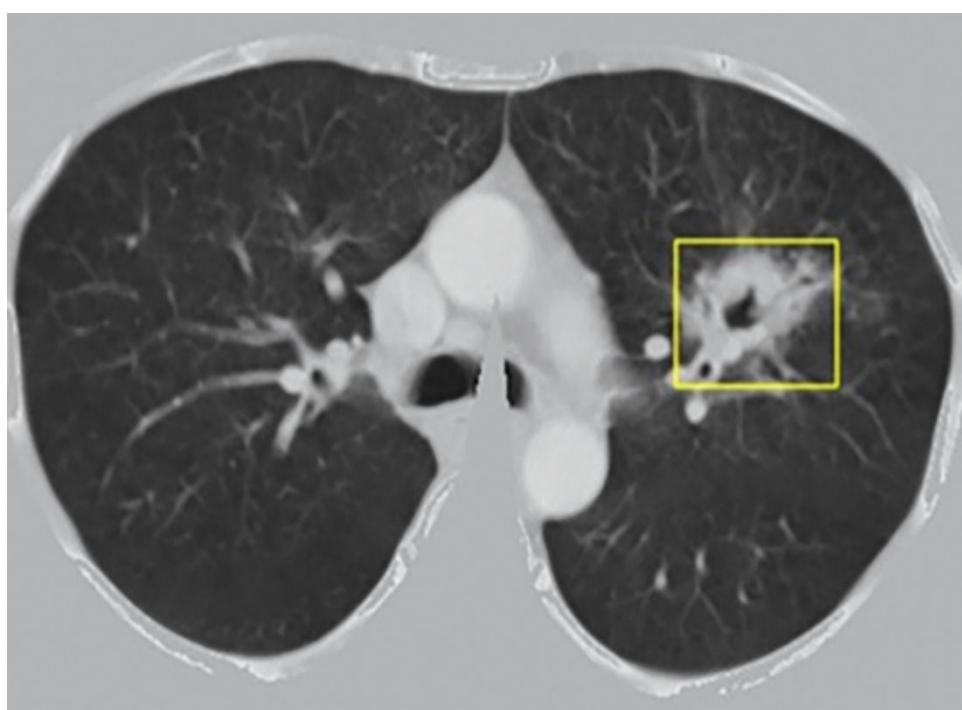
- Xử lý dữ liệu: Nhận dạng và tổ chức thông tin.

Các công cụ Computer vision và mô hình Deep learning đã được đưa vào nghiên cứu, đòi hỏi khối lượng dữ liệu lớn được gán nhãn. Khi các thuật toán Deep learning phát triển, chúng chủ yếu thay thế quy trình gắn thẻ thủ công thông qua một phương pháp tiếp cận được gọi là nghiên cứu dữ liệu đán đóng - thu thập theo thời gian thực tự động và gắn thẻ dữ liệu do các chuyên gia tạo ra và từ đó máy học sẽ bắt đầu quy trình nhận dạng các đối tượng.



Hệ thống thông tin

- Lĩnh vực y tế: Nhận dạng mẫu và xử lý hình ảnh.
Hình ảnh y khoa đã trở thành một phần thiết yếu trong cách thức làm việc của các chuyên gia trong lĩnh vực y tế, hướng đến các công cụ chẩn đoán tốt hơn và tăng đáng kể khả năng đưa ra các hành động hiệu quả hơn.



Chẩn đoán té bào ung thư

- Nhận diện khuôn mặt:
Khái niệm này xuất hiện từ năm 2011 khi Google chứng minh rằng có thể tạo ra một máy dò tìm khuôn mặt chỉ bằng những hình ảnh không được gắn



nhãn. Họ đã thiết kế một hệ thống có thể tự học để phát hiện hình ảnh con mèo mà không cần giải thích với hệ thống là con mèo trông như thế nào.

Ngày nay, điện thoại thông minh có thể sử dụng máy ảnh chất lượng cao để nhận dạng. Thị giác máy tính đang được sử dụng trong lĩnh vực an ninh để tìm kiếm tội phạm, dự đoán sự di chuyển khẩn cấp của đám đông,...



Nhận diện khuôn mặt

- Ngân hàng: Các ứng dụng nhận dạng hình ảnh sử dụng học máy để phân loại và trích xuất dữ liệu phục vụ cho việc giám sát quá trình xác thực các tài liệu như thẻ căn cước hoặc giấy phép lái xe có thể được sử dụng để cải thiện trải nghiệm của khách hàng từ xa và tăng cường bảo mật.



Nhận diện khách hàng thông qua việc quét hình ảnh



- Bán lẻ: Thị giác máy tính đang được sử dụng trong các cửa hàng ngày càng nhiều, đặc biệt là giúp cải thiện trải nghiệm của khách hàng, Pinterest Lens là một công cụ tìm kiếm sử dụng thị giác máy tính để phát hiện các đối tượng. Bằng cách sử dụng ứng dụng điện thoại thông minh trong các cửa hàng, chúng ta có thể hình dung một sản phẩm trông như thế nào và nhận được các sản phẩm khác liên quan đến nó.



Quét mã QR của sản phẩm

Những hạn chế của thị giác máy tính

Các hệ thống thị giác máy tính hiện tại thực hiện tốt việc phân loại hình ảnh và bản địa hóa các đối tượng trong ảnh, khi chúng được đào tạo đầy đủ với các ví dụ. Nhưng ở phần cốt lõi của chúng, các thuật toán học sâu cung cấp sức mạnh cho các ứng dụng thị giác máy tính chính là việc đổi chiều các mẫu pixel. Chúng không hiểu những gì đang diễn ra trong các hình ảnh.

Con người có thể khai thác kiến thức rộng lớn về thế giới của mình để lập đầy những lỗ hổng khi họ đối mặt với một tình huống mà họ chưa từng thấy trước đây. Không giống như con người, các thuật toán thị giác máy tính cần phải được hướng dẫn kỹ lưỡng về các loại đối tượng mà chúng phải phát hiện. Ngay khi môi trường của chúng chứa những thứ đi chệch khỏi các ví dụ đã được đào tạo, chúng bắt đầu hành động theo những cách phi lý, chẳng hạn như không phát hiện ra các phương tiện khẩn cấp dừng đỗ ở những vị trí khác thường.

Hiện tại, giải pháp duy nhất để giải quyết những vấn đề này là đào tạo các thuật toán AI trên với ngày càng nhiều các ví dụ, với hi vọng lượng dữ liệu bổ sung sẽ bao quát mọi tình huống mà AI sẽ gặp phải. Nhưng những kinh nghiệm cho thấy, nếu không có sự nhận thức theo tình huống, sẽ luôn có những góc khuất trong những tình huống hiếm hoi làm rối loạn thuật toán AI.



Mặc dù thị giác máy tính có những hạn chế nhất định nhưng đường như trí thông minh thị giác không dễ tách rời khỏi phần còn lại của trí thông minh, đặc biệt là kiến thức chung, sự trừu tượng và kỹ năng ngôn ngữ.

2. Sơ lược về đề tài

Tình hình dịch bệnh hiện nay:

Tình hình dịch bệnh Covid tuy đã ổn định hơn, nhưng vẫn kéo dài dai dẳng chưa chấm dứt.

Phòng bệnh còn hơn chữa bệnh, phương pháp hiệu quả và đơn giản nhất là đeo khẩu trang.

Ở những nơi công cộng, lượng người đông, kiểm soát bằng sức người rất khó nên phải cần có phương pháp nhận diện người đeo khẩu trang tự động để dễ dàng theo dõi hơn.

3. Phát biểu bài toán

Đầu vào (Input)

Đối tượng cần để nhận diện là con người. Đầu vào là một bức hình có thể là một người, một vài người hoặc là rất nhiều người.

Đầu ra (Output)

Bức hình đã trải qua tiền xử lý là gán nhãn và đưa ra kết quả người đeo khẩu trang, có đeo khẩu trang nhưng đeo sai hoặc không đeo khẩu trang.

Ví dụ minh họa

Để minh họa một cách trực quan mô hình của bài toán, chúng tôi xin đưa ra một ví dụ như sau:

Cho trước đầu vào là một ảnh gồm có 2 người như sau:



Sau quá trình gán nhãn thì ta có đầu ra là ảnh như sau:



hình ảnh được trích xuất từ kết quả detect bằng YOLOv5



4. Mô hình tổng quát

Object detection là gì?

Phân loại ảnh (image classification): liên quan đến việc gán nhãn cho hình ảnh.

Định vị vật thể (object localization): liên quan đến việc vẽ một hộp giới hạn (Bounding Box) xung quanh một hoặc nhiều đối tượng trong hình ảnh nhằm khoanh vùng đối tượng.

Phát hiện đối tượng (object detection): là nhiệm vụ khó khăn hơn và là sự kết hợp của cả hai nhiệm vụ trên: Vẽ một Bounding Box xung quanh từng đối tượng quan tâm trong ảnh và gán cho chúng một nhãn. Kết hợp cùng nhau, tất cả các vấn đề này được gọi là object recognition hoặc object detection.

Như thế nào là nhận dạng đối tượng?

Nhận dạng đối tượng là một thuật ngữ chung để mô tả một tập hợp các nhiệm vụ thị giác máy tính có liên quan đến việc xác định đối tượng trong ảnh kỹ thuật số.

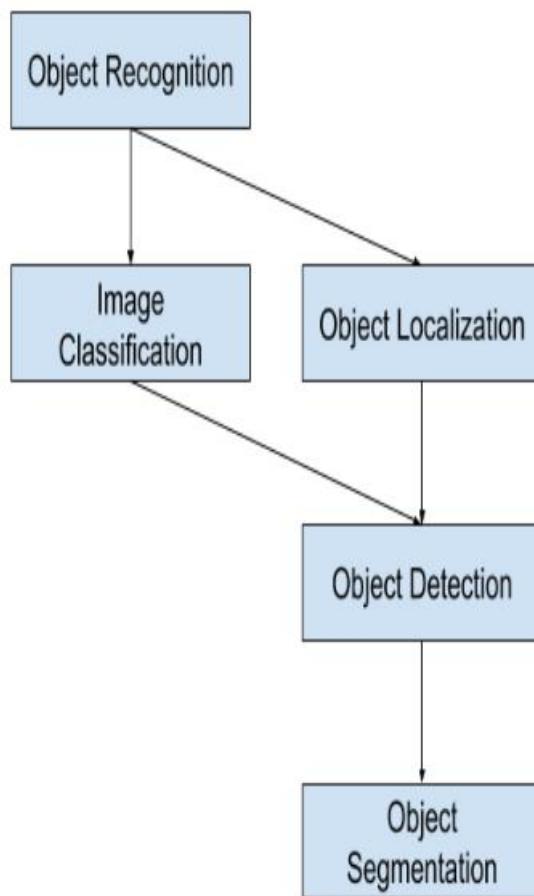
Phân loại hình ảnh liên quan đến việc dự đoán lớp của một đối tượng trong một hình ảnh. Định vị vật thể đề cập đến việc xác định vị trí của một hoặc nhiều đối tượng trong một hình ảnh và vẽ Bounding Box xung quanh chúng. Chúng ta có thể phân biệt giữa ba nhiệm vụ thị giác máy tính cơ bản trên thông qua 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.

Một số định nghĩa khác cũng rất quan trọng trong Computer Vision là phân đoạn đối tượng (object segmentation), trong đó các đối tượng được nhận dạng bằng cách làm nổi bật các pixel cụ thể của đối tượng thay vì Bounding Box. Và Image Captioning kết hợp giữa các kiến trúc mạng CNN và LSTM để đưa ra các lý giải về hành động hoặc nội dung của một bức ảnh.



Bên dưới là sơ đồ tổng hợp các tác vụ của Computer Vision .



Sơ đồ các mối liên hệ giữa các tác vụ trong Computer Vision

Chúng ta cũng có thể hiểu object recognition tương tự như object detection theo một cách tương đối nào đó. Gần đây thì Object Recognition đã trở thành một phần của cuộc thi ILSVRC, một trong những cuộc thi nhận diện ảnh lớn nhất hành tinh.

Điểm khác biệt nữa trong các mô hình image classification so với Object Recognition đó là mô hình image classification có hàm loss function chỉ dựa trên sai số giữa nhãn dự báo và nhãn thực tế trong khi object detection đánh giá dựa trên sai số giữa nhãn dự báo và sai số khung hình dự báo so với thực tế.

5. Đối tượng nghiên cứu

Đối tượng nghiên cứu của đề tài là mô hình nhận diện người có hay không đeo khẩu trang.

Đề tài sử dụng 2 mô hình YOLO khác nhau nhằm mục đích so sánh và cải thiện một số vấn đề còn mắc phải trong quá trình gán nhãn.



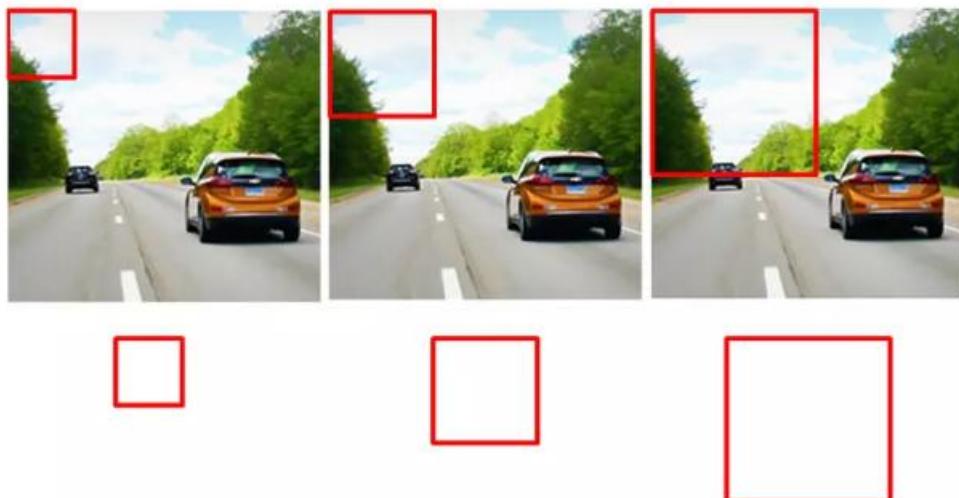
Chương 2: CƠ SỞ LÝ THUYẾT

1. Lịch sử phát triển của YOLO

Mở đầu

Object Detection là một bài toán phổ biến trong Computer Vision. Mục tiêu của Object Detection là xác định và phân loại các object (vật thể) tồn tại trong ảnh, là một bài toán multitask, thực hiện Classification và Regression (Localization) đồng thời. Ở những thời kì đầu tiên trong lĩnh vực này, các nghiên cứu tập trung vào việc sử dụng các hand-crafted feature (đặc điểm nhận dạng do con người nghĩ ra) như SIFT, HOG. Tuy nhiên, cách làm như vậy khá là hạn chế về độ chính xác và khả năng áp dụng lên đa dạng các object cần nhận dạng.

Từ năm 2012, CNN nổi lên như một thế lực trong các bài toán về Classification sử dụng đâu vào là ảnh, và cũng dần dần được đưa vào sử dụng trong Object Detection. Một phương pháp khá nổi tiếng đó chính là Sliding Window (Hình 1). Sử dụng một cửa sổ $S \times S$, trượt dần trên ảnh đầu vào. Ảnh lấy ra nằm trong vùng có cửa sổ trượt sẽ được đưa qua một mạng Classification để nhận diện xem trong vùng vừa lấy ra có object nào. Cách làm này vừa chậm mà Bounding Box của vật thể trong ảnh còn không tốt.

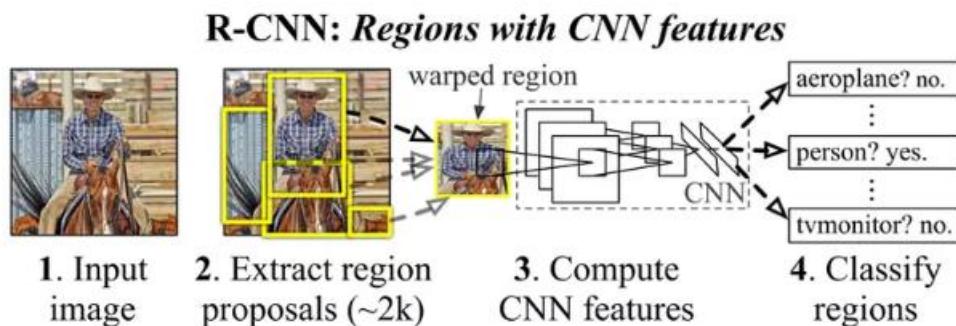


Hình 1. Phương pháp Sliding Window áp dụng trong Object Detection. Sử dụng một cửa sổ trượt, lấy ra từng vùng trên bức ảnh và phân loại vùng đó xem có object gì trong đó

Năm 2013 đánh dấu sự ra đời của một mạng neural cục kì nổi tiếng trong Object Detection: R-CNN. Trước tiên R-CNN sẽ xác định các vùng có tiềm năng tồn tại object, các vùng có tiềm năng này sau đó sẽ được đưa vào một



CNN để thực hiện Classification xem vùng đó là object nào, đồng thời cải thiện Bounding Box cho object (Hình 2). Họ nhà R-CNN tiếp tục phát triển mạnh mẽ với 2 nghiên cứu sau đó là Fast R-CNN và Faster R-CNN, cải thiện tốc độ cũng như là độ chính xác của mô hình. Tuy nhiên, họ nhà R-CNN vẫn khá là chậm với phương pháp hai pha: 1 pha xác định vùng có tiềm năng là object, 1 pha phân loại và cải tiến Bounding Box của object.



Hình 2. Phương pháp 2 pha của họ nhà R-CNN. Pha đầu tiên dùng để xác định những vùng có tiềm năng là object. Pha thứ hai dùng để phân loại và cải tiến bounding box cho object

Vì vậy, YOLO ra đời với mục tiêu là một mạng Object Detection có tốc độ cực nhanh với phương pháp một pha của mình.

YOLOv1

Phương pháp một pha

YOLO sẽ đưa ảnh đầu vào qua một CNN, tạo ra một feature map $S \times S$, gọi là grid. YOLO thực hiện detect object tại mỗi cell (ô) trong $S \times S$ grid đó. Tức là, thay vì có một bước tìm ra các vùng có khả năng tồn tại object, thì YOLO sẽ thực hiện detect trên toàn bộ ảnh, khởi đầu từ vùng gì cả.

Một cell sẽ predict ra B Bounding Box và xác suất cho C class. Một Bounding Box sẽ mang 5 thông tin: tâm của Bounding Box (x, y), chiều dài và rộng của Bounding Box (w, h) và độ tự tin. Độ tự tin dùng để xác định xem trong Bounding Box đó có tồn tại object hay không. Nếu trong Bounding Box đó không tồn tại object, ta bỏ qua luôn toàn bộ giá trị prediction khác của Bounding Box và Classification, nhằm bỏ qua những nơi không tồn tại object. Việc sử dụng thông tin độ tự tin này cũng phần nào giống pha thứ nhất trong phương pháp hai pha, lọc ra những nơi tồn tại object.

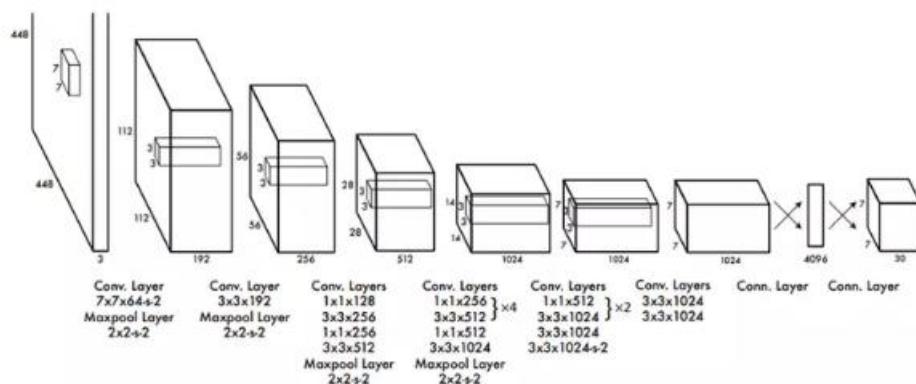
Vậy, trong một cell, ta sẽ predict ra một tensor có $B \times 5 + C$ phần tử, với B là số lượng Bounding Box, 5 là 5 thông tin trong Bounding Box gồm: trung tâm box, chiều dài rộng box và độ tự tin và C là số class. Và trong một feature map gồm $S \times S$ cell, tensor dự đoán từ mạng sẽ có độ dài $S \times S \times (B \times 5 + C)$



Kiến trúc mạng

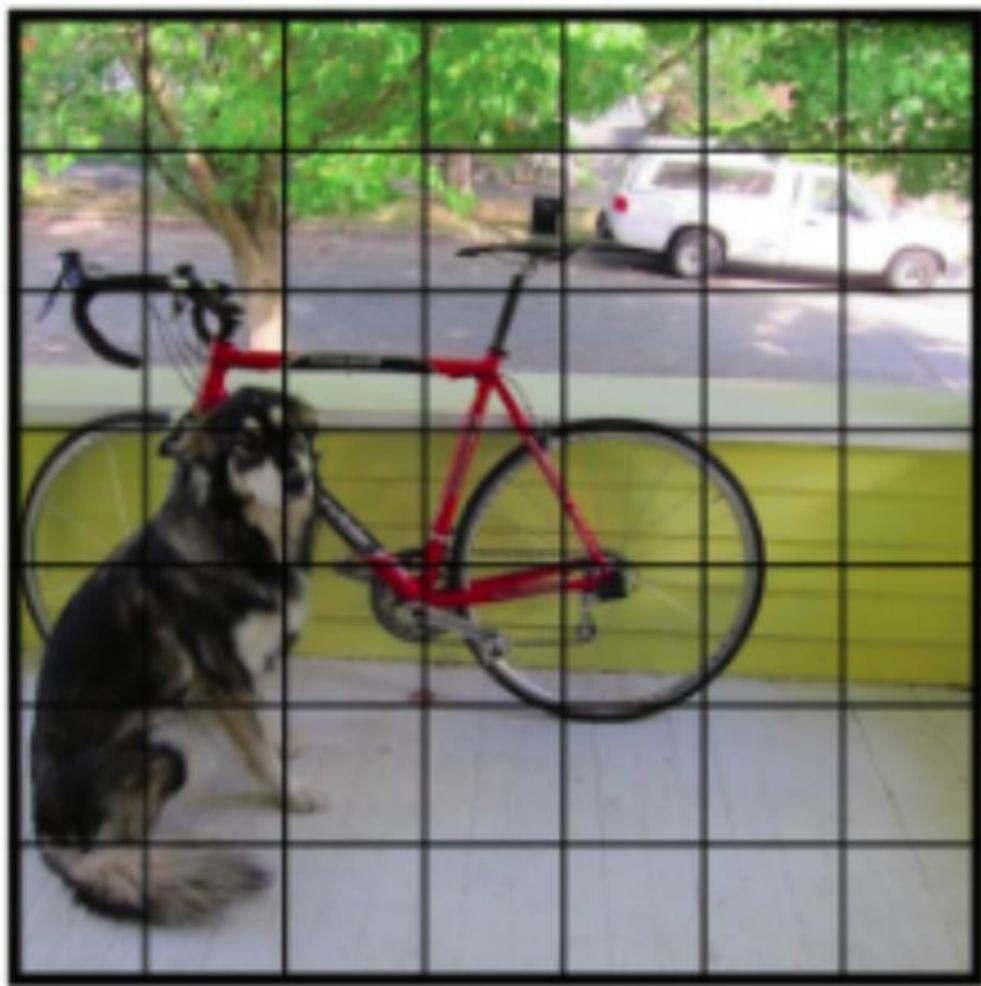
YOLOv1 sử dụng 24 lớp chập: bao gồm lớp (1×1) gọi là Reduction layers (dùng để giảm kích thước ảnh), lớp chập (3×3) gọi là Convolution layers. Xen kẽ giữa 24 lớp chập là các lớp max pooling và kết thúc bằng 2 lớp fully connected. Kết quả là một ma trận 3 chiều có dạng $7 \times 7 \times 30$.

Với mỗi $1 \times 1 \times 30$ trong $7 \times 7 \times 30$ được gọi là một tensor, vậy đối với $7 \times 7 \times 30$ thì ta sẽ có $7 \times 7 = 49$ tensors.



Hình 3. Kiến trúc mạng của YOLOv1

YOLO chia một bức ảnh thành $S \times S$ grid cell, thường là 7×7 . Trọng tâm của vật thể được tìm trong các grid cell đó (được học thông qua việc gán nhãn và đào tạo). Nếu trọng tâm của đối tượng nằm trong grid cell bất kỳ, thì nó (grid cell chứa trọng tâm của đối tượng) sẽ chịu trách nhiệm tìm vật thể đó.

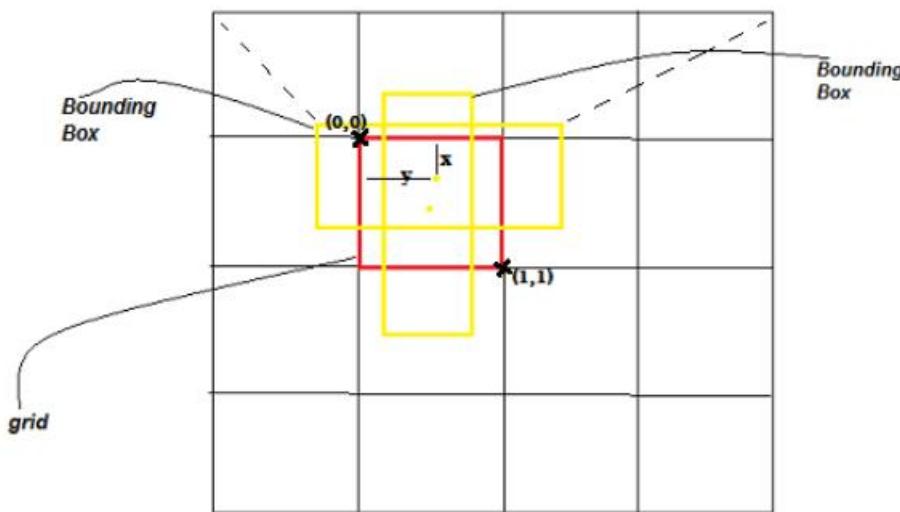


S \times S grid on input

YOLOv1 chia bức ảnh thành SxS grid cell

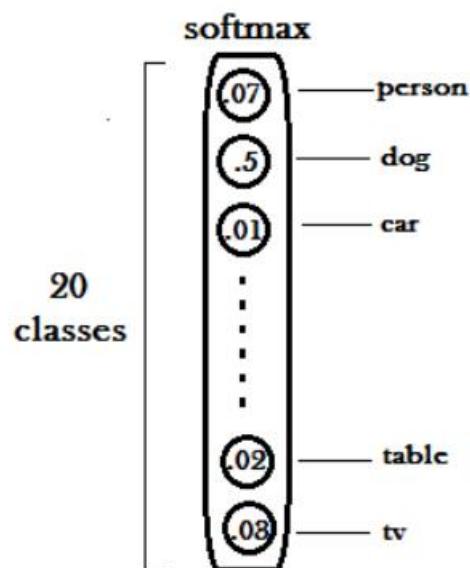
Vậy làm sao để những grid cell đó tìm vật thể?

- Các grid cell này sẽ phải đưa ra ($2 \times B + C$) giá trị dự đoán, trong đó:
 - B bounding box (tức là hộp chứa đối tượng)- đối với YOLOv1 thì $B=2$. Với mỗi B sẽ có 5 giá trị nhỏ nữa là : x (trọng tâm đối tượng theo phương x- đối chiếu với grid cell chứa nó), y (trọng tâm đối tượng theo phương y – đối chiếu với grid cell chứa nó), w (độ rộng của vật thể- đối chiếu với kích thước của toàn bộ bức ảnh), h (độ cao của vật thể – đối chiếu với kích thước của toàn bộ bức ảnh) và **confidence** (độ tự tin- xác suất đó là vật thể).
 - Từ 2 bounding box B, ta dựa vào **confidence** của bounding box nào lớn hơn và IOU so với ground truth box (nhãn được gán bằng tay) lớn hơn thì được xuất ra là vật thể.



Trọng tâm vật thể

C sẽ bằng số classes và có giá trị là xác suất của trọng tâm vật thể rơi vào grid cell đó(đối với YOLOv1 thì số classes là 20, bởi vì model YOLOv1 là một model đã được train sẵn và chúng ta không thể đào tạo hay sửa chữa được nên chỉ có thể lấy ra dùng).



Giá trị của C

Với mỗi grid cell thì tối đa sẽ chỉ phát hiện được 1 đối tượng, vậy nên đối với YOLOv1 thì sẽ xảy ra những vấn đề sau:

- Phát hiện được tối đa 49 đối tượng(nếu thuật toán chia ảnh thành 7×7).



- Nếu một grid cell chứa trọng tâm của nhiều đối tượng, thì model cũng chỉ phát hiện được một đối tượng có độ chính xác cao nhất.
- Nếu vật thể nào đó quá lớn so với bức ảnh, đồng nghĩa với việc vật thể đó sẽ có nhiều trọng tâm ở nhiều grid cell khác nhau. Vậy nên vật thể đó sẽ được phát hiện nhiều lần.

Loss function

Một hàm loss thì luôn tồn tại ít nhất 2 yếu tố để có thể tính được: prediction (sự dự đoán của mạng) và target (cái chúng ta cần đạt được). Ta đã biết được prediction của YOLOv1 gồm những gì, trình bày ở phần "Phương pháp một pha" phía trên. Ta sẽ tìm hiểu YOLOv1 gồm những hàm loss gì, và target của chúng được chọn như nào.

$$\begin{aligned} & \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \\ & + \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} \left[(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2 \right] \\ & + \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} (C_i - \hat{C}_i)^2 \\ & + \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{noobj}} (C_i - \hat{C}_i)^2 \\ & + \sum_{i=0}^{S^2} \mathbb{1}_i^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2 \end{aligned}$$

Phía trên là toàn bộ hàm loss của YOLOv1, nhìn có vẻ rất đáng sợ nhưng ta hãy cùng chia nhỏ nó ra.



Confidence Loss

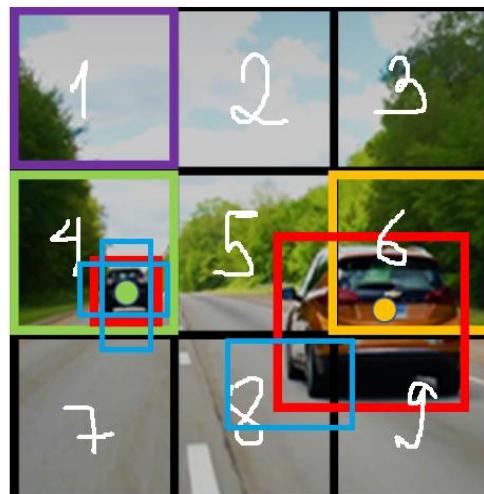
Xét Confidence Loss (hàm loss dành cho độ tự tin):

$$\begin{aligned} & + \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2 \\ & + \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2 \end{aligned}$$

Đây là loss của độ tự tin trên toàn bộ tấm ảnh, gồm $S \times S$ cell và mỗi cell gồm B Bounding Box. Ta sẽ chia nhỏ hàm loss này hơn nữa. Xét một Bounding Box trong một cell, ta sẽ thu được hàm loss : $(C_i - \hat{C}_i)^2$. Target cho hàm loss này, \hat{C}_i sẽ là giá trị IoU của Bounding Box được dự đoán so với ground truth Bounding Box.

Tại sao Confidence Loss lại gồm 2 phần? Xét phần trên của Confidence Loss, phần này sẽ phụ trách cho cell thứ i mà tồn tại object trong cell đó. Một cell thì gồm $B (=2)$ Bounding Box, lặp qua toàn bộ B Bounding Box được dự đoán trong cell thứ i , nếu Bounding Box thứ j cho kết quả IoU với ground truth Bounding Box là lớn nhất thì $\mathbb{1}_{ij}^{\text{obj}} = 1$, còn lại thì bằng 0. Xét phần dưới. Phần dưới của Confidence Loss, phần này sẽ phụ trách cho cell thứ i mà **không** tồn tại object trong cell đó, còn lại thì tương tự như phần trên.

Ta có thể dễ dàng thấy là số cell **không** tồn tại object sẽ nhiều hơn rất nhiều so với số cell tồn tại object, vì vậy, phần loss cho cell không tồn tại object sẽ lớn hơn nhiều và lúc này, $\lambda_{\text{noobj}} = 0.5$ đóng vai trò là một hệ số làm nhỏ đi phần loss cho cell không tồn tại object xuống.



Hình 4. Một object sẽ được coi là tồn tại trong một cell nếu trung tâm của object nằm trong cell đó. Xét một cell tồn tại object là cell 4, tồn tại 2 Bounding Box. Lúc này Confidence Loss trong cell 4 sẽ được tính theo công thức đầu của Confidence Loss với C_i là IoU lớn nhất giữa toàn bộ các Bounding Box được dự đoán (xanh) với ground truth Bounding Box (đỏ), còn \hat{C}_i sẽ do mạng dự đoán ra. Xét một cell không tồn tại object là cell 8, nhưng vẫn tồn tại một Bounding Box được dự đoán (xanh). Lúc này ở cell 8, Confidence Loss sẽ được tính theo công thức dưới của Confidence Loss với C_i và \hat{C}_i tương tự

Classification Loss

Công thức của Classification Loss như sau:

$$+ \sum_{i=0}^{S^2} \mathbb{1}_i^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2$$

Classification target cc của hàm loss sẽ là một vector one-hot, giống như các bài toán Classification thông thường khác. Tuy nhiên, Classification Loss chỉ được tính khi cell đó tồn tại object, còn nếu cell không tồn tại object thì Classification Loss sẽ =0

Localization (Regression) Loss

Công thức của Regression Loss như sau:

$$\begin{aligned} & \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \\ & + \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} \left[(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2 \right] \end{aligned}$$

Phần này thì cũng giống như Confidence Loss, lặp qua từng cell và từng Bounding Box trong cell, nếu cell đó tồn tại object thì thực hiện tính Loss, còn không thì Loss = 0. Điều thú vị ở đây là căn bậc 2 ở trong phần Loss dành cho chiều dài và chiều rộng của Bounding Box. Ý tưởng

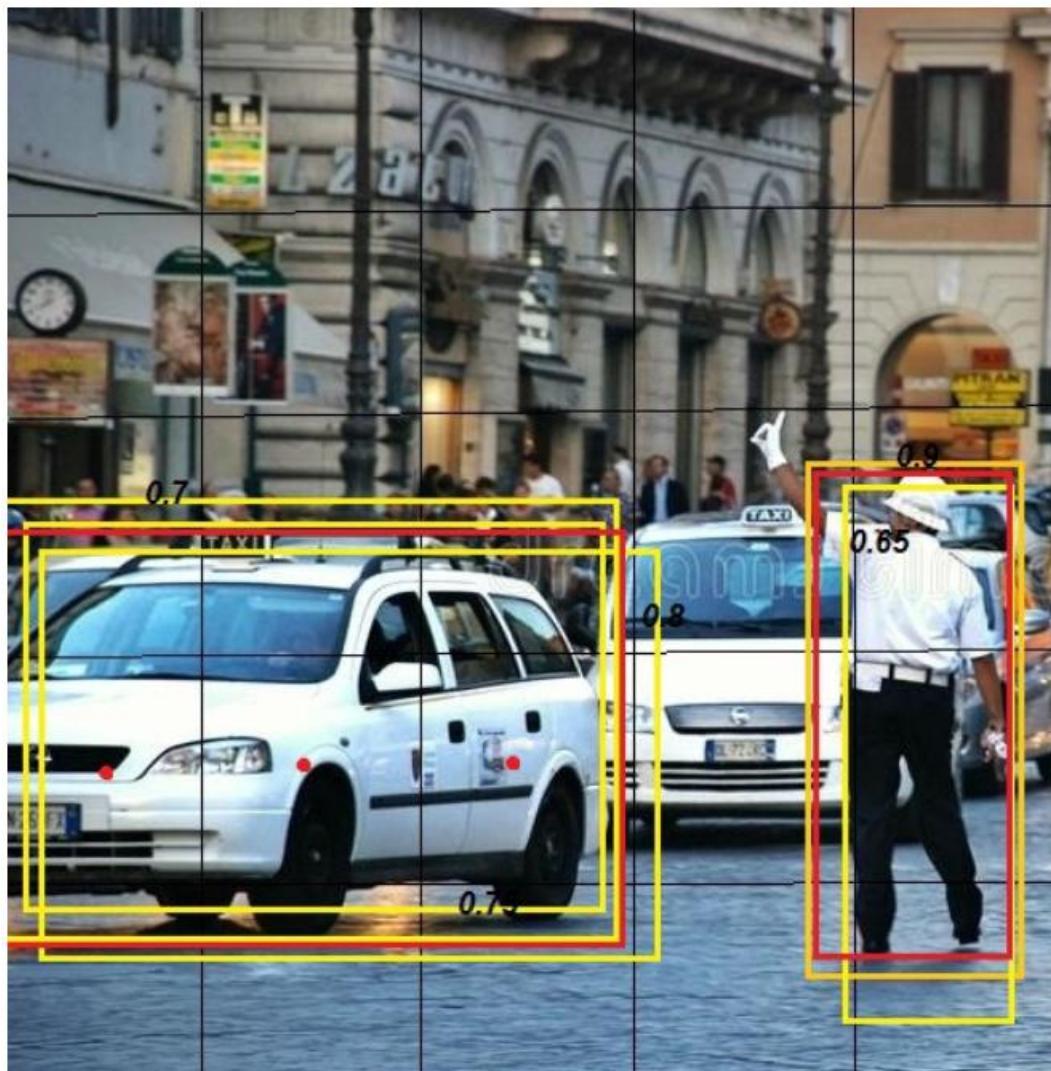


của việc này nằm ở việc Bounding Box có độ lớn khác nhau thì nên được đối xử khác nhau. Tức là, việc sai một vài pixel ở Bounding Box to sẽ không nghiêm trọng bằng việc sai một vài pixel ở Bounding Box nhỏ.

Lấy ví dụ lần nữa cho dễ hiểu nhé. Xét 1 ground truth Bounding Box có chiều rộng w_i là 500, Bounding Box từ dự đoán của model \hat{w}_i là 505, lúc này, loss tại chiều rộng sẽ là 5. Xét tiếp 1 ground truth Bounding Box có chiều rộng w_i là 20, Bounding Box từ dự đoán của model \hat{w}_i là 25, lúc này, loss tại chiều rộng vẫn là 5. Với một Bounding Box có chiều rộng tới tận 500, việc lệch đi 5 không nên có sự ảnh hưởng ngang với việc một Bounding Box có chiều rộng là 20 và lệch đi 5. Vì vậy, lấy căn bậc 2 của giá trị này sẽ làm giảm độ ảnh hưởng của Bounding Box lớn khi có lệch nhỏ.

Non-maximal suppression (NMS)

Như đã trình bày ở trên, sẽ có nhiều vật thể lớn và trọng tâm của nó có thể rơi vào nhiều grid cell cùng một lúc, sẽ dẫn tới việc một vật thể sẽ được dự đoán nhiều lần... Cho nên tác giả đã sử dụng NMS để loại bỏ những trường hợp như thế này. Cách thực hiện:



Minh họa NMS

1. Loại bỏ những đối tượng có $\text{confidence } C < \text{C-threshold}$ (ngưỡng này chúng ta có thể re-setup).
2. Sắp xếp những đối tượng có $\text{confidence } C$ theo thứ tự giảm dần.
3. Chọn bounding box của những vật thể có $\text{confidence } C$ cao nhất và xuất ra kết quả.
4. Loại bỏ những bounding box có $\text{IOU} < \text{IOU-threshold}$ (ngưỡng này chúng ta có thể re-setup)
5. Quay lại bước 3 cho tới khi mọi kết quả được kiểm tra.



Hạn chế của YOLOv1

- YOLOv1 chỉ có thể predict được một object trong một cell, nếu có 2 object tồn tại trong cùng một cell, hoặc nếu phải predict từng object trong một nhóm object, YOLOv1 sẽ gặp rất nhiều khó khăn.
- Với việc chỉ sử dụng 2 Bounding Box, YOLOv1 sẽ gặp khó khăn với những object có tỉ lệ khác so với trong training.
- Regression Loss của YOLOv1 chưa tốt, dẫn đến việc đưa ra Bounding Box không tốt.

YOLOv2

Sau khi error analysis cho YOLOv1, nhóm tác giả nhận thấy rằng YOLOv1 gặp vấn đề về localization khá nhiều (Bounding Box không tốt), hơn nữa, Recall của YOLOv1 cũng khá là thấp (Phát hiện được ít vật thể), đã đề cập đến ở phần 1 của series. Vì vậy, trong YOLOv2 nhóm tác giả tập trung vào cải thiện 2 vấn đề này mà vẫn giữ được độ chính xác tốt.

Kiến trúc mạng

Type	Filters	Size/Stride	Output
Convolutional	32	3×3	224×224
Maxpool		$2 \times 2/2$	112×112
Convolutional	64	3×3	112×112
Maxpool		$2 \times 2/2$	56×56
Convolutional	128	3×3	56×56
Convolutional	64	1×1	56×56
Convolutional	128	3×3	56×56
Maxpool		$2 \times 2/2$	28×28
Convolutional	256	3×3	28×28
Convolutional	128	1×1	28×28
Convolutional	256	3×3	28×28
Maxpool		$2 \times 2/2$	14×14
Convolutional	512	3×3	14×14
Convolutional	256	1×1	14×14
Convolutional	512	3×3	14×14
Convolutional	256	1×1	14×14
Convolutional	512	3×3	14×14
Maxpool		$2 \times 2/2$	7×7
Convolutional	1024	3×3	7×7
Convolutional	512	1×1	7×7
Convolutional	1024	3×3	7×7
Convolutional	512	1×1	7×7
Convolutional	1024	3×3	7×7
Convolutional	1000	1×1	7×7
Avgpool		Global	1000
Softmax			

Bảng 1. Kiến trúc mạng của YOLOv2

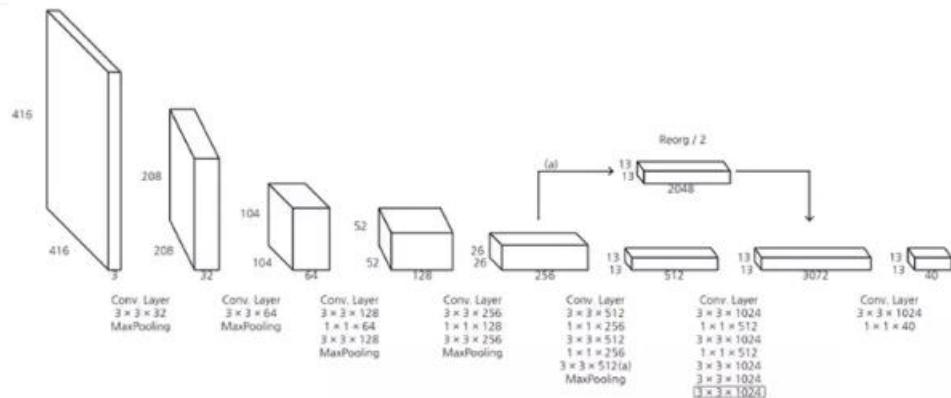
Darknet-19. Kiến trúc mạng của YOLOv2 được thể hiện ở bảng 1, được gọi là Darknet-19 vì có 19 lớp Convolution (Conv). Phần này



mình sẽ nói về những thay đổi trong kiến trúc cũng như là cách training của YOLOv2.

Batch Norm. Kiến trúc của YOLOv2 đã được thêm vào đó những lớp BatchNorm để việc training nhanh hơn và ổn định hơn. Với việc thêm vào BatchNorm, DropOut được loại bỏ khỏi model mà không sợ bị overfitting.

High-res Classifier. Backbone của YOLOv2 được pretrained trên ImageNet. Trong YOLOv1, backbone được train trên ImageNet với kích thước ảnh 224×224 , lúc train detection với toàn bộ model thì lại sử dụng kích thước ảnh 448×448 . Việc chuyển đổi ngọt như vậy khiến model phải vừa học Object Detection lại còn vừa phải thích ứng với kích thước ảnh mới. Vì vậy, trong YOLOv2, backbone trước tiên được finetune trên ImageNet với kích thước ảnh 448×448 trong vòng 10 epochs, rồi mới chuyển sang dạng Object Detection.



Hình 1. Kiến trúc của YOLOv2 với một skip-connection

Fine-grained Features. Thay vì predict trên 7×7 grid feature map như YOLOv1 thì YOLOv2 predict trên một 13×13 grid feature map, việc này sẽ khiến YOLOv2 predict những object nhỏ tốt hơn. Hơn nữa, YOLOv2 cũng sử dụng một skip-connection để kết hợp thông tin từ feature map ở layer trước đó vào feature map cuối (Hình 1).

Multi-scale training. Trước đó, YOLOv2 chỉ train với kích thước ảnh 448×448 . Sau khi áp dụng Anchor Box (sẽ nói ở phần sau), YOLO đổi kích thước ảnh thành 416×416 . Tuy nhiên, YOLOv2 muốn model có thể detect tốt với nhiều kích thước ảnh khác nhau, vì vậy, cứ mỗi 10 batches, YOLOv2 lại thay đổi kích thước ảnh đầu vào một lần. Sở dĩ điều này có thể thực hiện được vì kiến trúc mạng của YOLOv2 hoàn toàn tạo từ các lớp Conv và có hệ số suy giảm là 32. Do đó, kích thước ảnh đầu vào của YOLOv2 thay đổi với kích thước là bội số của 32, được lấy trong khoảng $\{320, 352, \dots, 608\}$.

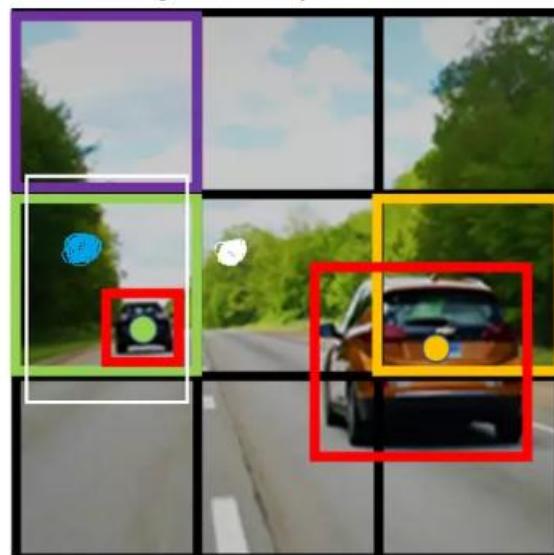


Những thay đổi khác

Áp dụng Anchor Box. Trong YOLOv2, tác giả áp dụng Anchor Box được sử dụng trong Faster R-CNN. Lúc này, kích thước ảnh đầu vào được chuyển từ 448×448 thành 416×416 vì tác giả muốn feature map thu được ở lớp cuối cùng là số lẻ (với kích thước ảnh 448×448 thì feature map ở lớp cuối là 14×14) để luôn có một ô trung tâm của feature map. Ý tưởng này đến từ việc các ảnh trong dataset COCO thường có một vật ở giữa ảnh, vì vậy, việc có một ô trung tâm của feature map để Anchor Box có thể dễ dàng lấy được luôn vật đó. Sử dụng Anchor Box, YOLOv2 bị giảm đi 0.3 mAP nhưng bù lại, Recall tăng lên. Tức là việc sử dụng Anchor Box khiến YOLOv2 phát hiện được nhiều vật thể hơn, nhưng bù lại, khả năng phát hiện chính xác lại kém đi.

Trong các mô hình 2 pha (họ nhà R-CNN), việc Anchor Box hoạt động rất tốt vì pha thứ nhất đã bao gồm việc tối ưu vị trí cho Bounding Box từ Anchor Box, còn trong YOLO thì không có. Do vậy, việc có các Anchor Box đẹp được sinh ra ngay từ lúc đầu khá là quan trọng. YOLOv2 đã thêm vào bước chọn các chỉ số cho Anchor Box được sinh ra ngay từ lúc đầu thông qua thuật toán k-means.

Thêm vào đó, việc tối ưu vị trí Bounding Box từ Anchor Box sinh ra được sử dụng trong Faster R-CNN đó chính là: model sẽ dự đoán độ dịch chuyển của Anchor Box gọi là t_x, t_y để suy ra vị trí của tâm Bounding Box x, y thông qua một công thức biến đổi. YOLOv2 nhận thấy việc này là không phù hợp, nên đã thêm vào một số giới hạn. YOLOv2 cũng vẫn dự đoán độ dịch chuyển t_x, t_y, t_w, t_h và Objectness Score t_o , nhưng, lúc này, t_x, t_y bị giới hạn giá trị của chúng trong khoảng $[0, 1][0, 1]$, việc này giới hạn vị trí tâm x, y của Bounding Box khi thực hiện phép biến đổi với t_x, t_y tức là khi predict t_x, t_y tại một grid cell sẽ không khiến cho tâm x, y bị ra ngoài grid cell đó. Nếu vẫn còn khó hiểu, có thể xem Hình 2 để hiểu rõ hơn công thức giới hạn được áp dụng cho t_x, t_y của YOLOv2.



Hình 2. Việc dự đoán t_x, t_y mà không kèm theo công thức giới hạn, thông qua phép biến đổi để tìm tâm cho Bounding Box từ Anchor Box (màu trắng) sẽ có thể khiến tâm nằm ngoài grid cell mà sinh ra Anchor Box đó. Với việc có công thức giới hạn của YOLOv2, tâm của Bounding Box (màu xanh) sẽ nằm trong grid cell mà sinh ra Anchor Box.

Tổng kết

Những cải tiến của YOLOv2 bao gồm 2 phần chính:

- Sử dụng một kiến trúc mạng mới và cách training mới.
- Áp dụng và thay đổi Anchor Box cho phù hợp.



YOLOv3

YOLOv3 là một bản nâng cấp đáng giá của YOLOv2 đồng thời giải thích những gì còn chưa rõ của YOLOv2.

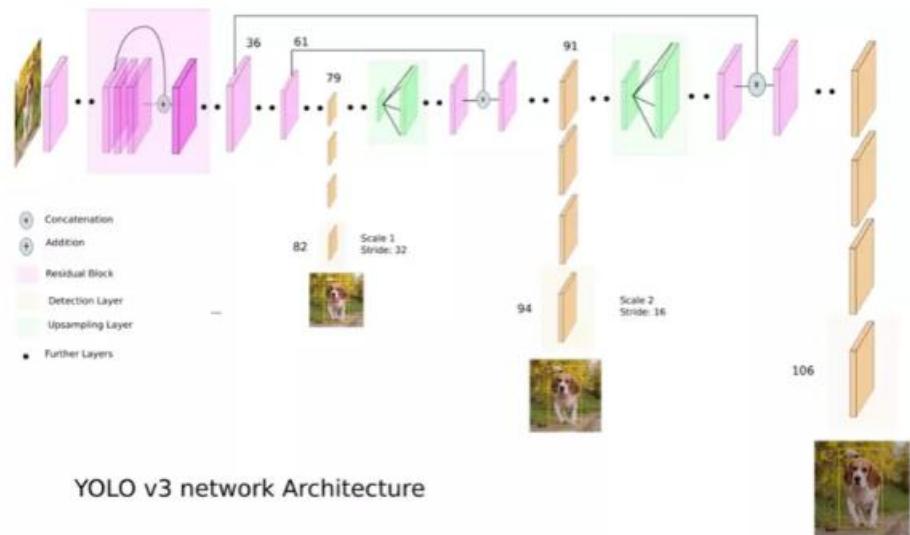
Kiến trúc mạng

Backbone. YOLOv3 xây dựng một backbone mới, gọi là Darknet-53. Backbone của YOLOv1 thì sử dụng 1×1 Convolution (gọi là Bottleneck) của Inception Network, lên YOLOv2 thì áp dụng thêm BatchNorm, sang YOLOv3 thì áp dụng thêm skip-connection từ ResNet, gọi là một Residual Block (Bảng 2).

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

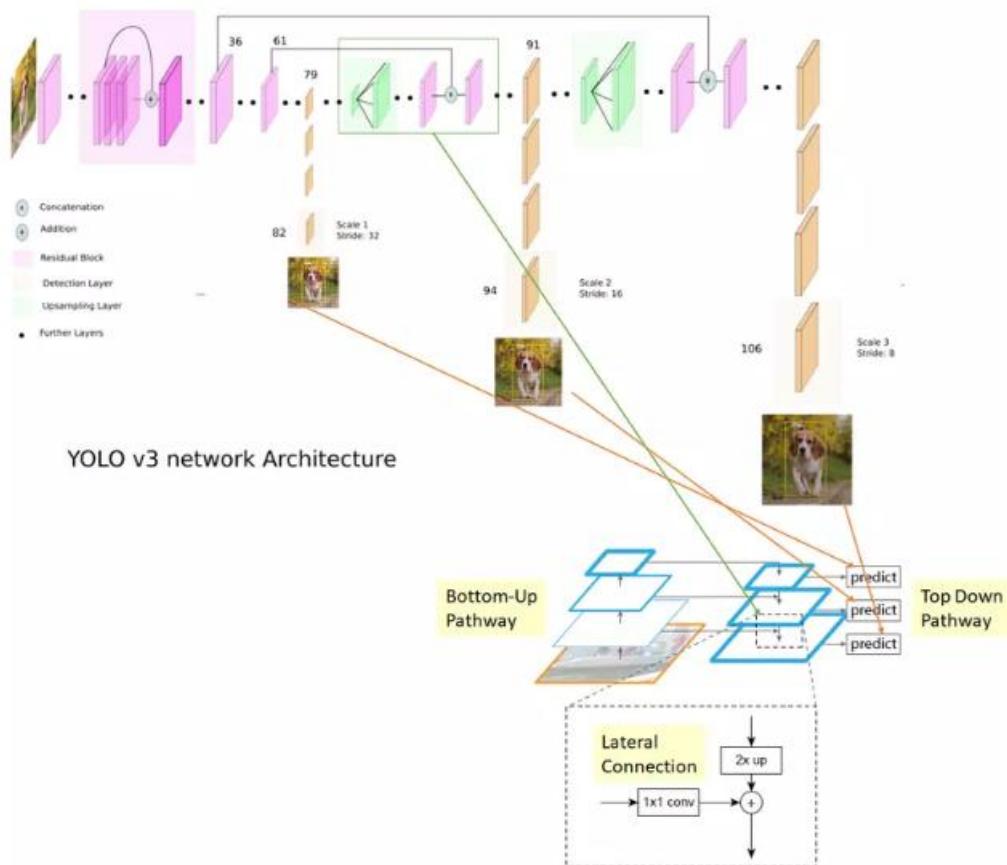
Bảng 2. Kiến trúc backbone của YOLOv3

Neck. Từ các phiên bản YOLO trước, phát hiện vật thể nhỏ luôn là một điểm yếu. Dù trong YOLOv2 đã sử dụng skip-connection từ layer trước đó để đưa thông tin từ feature map có kích thước lớn hơn vào feature map đằng sau, nhưng điều đó là không đủ. YOLOv3 là một sự nâng cấp cho vấn đề này, áp dụng Feature Pyramid Network, thực hiện phát hiện object ở 3 scale khác nhau của feature map (Hình 3).



Hình 3. Kiến trúc của YOLOv3 với Feature Pyramid Network

Đã đưa vào các lớp như: residual block, up sampling, skip connection



Làm rõ Hình 3



Những thay đổi khác

Xét một grid cell trong feature map, vector dự đoán của YOLO sẽ dự đoán $t_x, t_y, t_w, t_h, t_o, \hat{y}_1, \dots, \hat{y}_C$ với C là tổng số class.

Classification prediction

Các phiên bản YOLO trước sử dụng hàm *softmax* ở output của classification. Nhưng từ YOLOv3, output của classification được chuyển thành *sigmoid*. Điều này là vì với một số dataset, ví dụ như Open Images Dataset, có một số object sẽ được phân tán 2 class (class person và class woman).

Bounding Box prediction

Giữ nguyên ý tưởng Anchor Box với k-means từ YOLOv2, YOLOv3 làm rõ cách chọn Anchor Box của mình. Ở một grid cell trong feature map, YOLOv3 tạo ra 9 Anchor Box (YOLOv2 là 5), cứ mỗi 3 Anchor Box sẽ thuộc về một scale. Ở YOLOv2 có nhắc đến hàm biến đổi để biến đổi Anchor Box được tạo ra thành Bounding Box cho object. Mình sẽ làm rõ công thức đây ở đây. Gọi 4 giá trị dịch chuyển mà YOLOv3 predict ra là t_x, t_y, t_w, t_h . Như đã nói ở trên, YOLOv2 giới hạn t_x, t_y ở trong khoảng $[0,1][0,1]$, việc giới hạn này được áp dụng sử dụng hàm *sigmoid* σ . Xét một grid cell (c_x, c_y) trong feature map, Anchor Box được sinh ra có chiều dài, chiều rộng là (p_w, p_h) , với các giá trị dự đoán t_x, t_y, t_w, t_h công thức biến đổi có dạng:

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

với (b_x, b_y) là tâm của Bounding Box, (b_w, b_h) là chiều dài, chiều rộng của Bounding Box. Để hiểu tại sao chúng ta phải giới hạn t_x, t_y thì một lần nữa hãy nhìn lại Hình 2 ở trên. Với $c_x = 0$ và $c_y = 1$, nếu không giới hạn thì $t_x + c_x$ sẽ khiến tâm b_x của Bounding Box vượt ra ngoài cái grid cell $c_x = 0, c_y = 1$ đó, sang hòn grid cell $c_x = 1, c_y = 1$.



Loss function

Loss function của YOLOv3 khá là khác so với các phiên bản YOLO trước nhưng lại không hề được đề cập đến trong paper.

$$\sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{I}_{i,j}^{obj} \sum_{k=1}^C BCE(y_k, \hat{y}_k)$$

Xét Classification Loss, với việc áp dụng *sigmoid* trong classification, Classification Loss của YOLOv3 sẽ áp dụng Binary Cross Entropy chứ không còn là squared loss nữa. Tiếp tục thực hiện chia nhỏ hàm loss này cho dễ hiểu nhé. Xét một grid cell có tồn tại object, lặp qua C giá trị trong vector Classification, tính BCE Loss giữa classification target y_k và classification prediction \hat{y}_k . Xét toàn bộ Anchor Box $\sum_{j=0}^B$ ở trên toàn bộ grid cell $\sum_{j=0}^B$, ta sẽ có được công thức của Classification Loss như trên.

$$\begin{aligned} & \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{I}_{i,j}^{obj} (-\log(\sigma(t_o))) \\ & + \lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{I}_{i,j}^{noobj} (-\log(1 - \sigma(t_o))) \end{aligned}$$

Xét Confidence Loss, ta có thể thấy thay vì là squared loss như YOLOv1, YOLOv3 đã áp dụng Binary Cross Entropy Loss. Nhưng không chỉ mỗi hàm loss là thay đổi, cả Confidence target cũng thay đổi. Confidence target trong YOLOv1 là IoU score giữa Bounding Box được dự đoán và ground truth Bounding Box, còn trong YOLOv3, confidence target =1 nếu IoU score giữa một Anchor Box trong số 9 Anchor Box sinh ra với ground truth Bounding Box là lớn nhất, và được đưa vào vé trên của Confidence Loss để tính. Với những Anchor Box mà có IoU score



giữa chúng với ground truth Bounding Box nhỏ hơn một threshold là 0.5 thì chúng sẽ được đưa vào đưa vào về dưới của Confidence Loss. Còn những Anchor Box mà có IoU score lớn hơn threshold 0.5 nhưng lại không phải là Anchor Box được chọn ở trong vé trên thì chúng sẽ không được đưa vào hàm loss.

$$\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{I}_{i,j}^{obj} \left((t_x - \hat{t}_x)^2 + (t_y - \hat{t}_y)^2 + (t_w - \hat{t}_w)^2 + (t_h - \hat{t}_h)^2 \right)$$

Còn Regression Loss thì gần như tương đương, chỉ bỏ đi căn bậc 2 khi ở phần chiều dài và chiều rộng.

Nhận dạng ở các tỉ lệ ảnh khác nhau

Đối với YOLOv1, chúng ta đã biết thuật toán này chia bức ảnh thành các grid cell với kích thước $S \times S$ với $S = 3,5$ hoặc 7. Còn đối với YOLOv2 thì kích thước này là 13. Nhưng đối với các kích thước đó, vẫn chưa đủ để có thể tìm kiếm những đối tượng có kích thước nhỏ trong bức hình. Vậy nên YOLOv3 đã xử lý bằng cách là nhận dạng 3 lần trên một bức ảnh với kích thước khác nhau.

Giả sử ta có bức ảnh đầu vào có kích thước : 416×416 :

- Tại lớp chap thứ 82: Bức ảnh được chia thành các grid cell với kích thước 13×13 (bức ảnh đã được chia với 32). Tại đây, các grid cell sẽ có trách nhiệm tìm các vật thể có kích thước lớn trong bức hình.
- Tại lớp chap thứ 94: Bức ảnh được chia thành các grid cell với kích thước 26×26 (bức ảnh được chia với 16). Và có trách nhiệm tìm các vật thể có kích thước trung bình.
- Tương tự, tại lớp chap 106, bức ảnh được chia thành các grid cell với kích thước 52×52 (ảnh được chia với 8) và có trách nhiệm tìm các vật thể có kích thước bé.
- Bức ảnh sau khi được chia thành các grid cell, thì được gọi là Down sample images(ảnh giảm mẫu)



Detections at:

82, 94, 106 layers

Downsamples input:



quá trình tìm kiếm vật thể

Tensor

Ở YOLOv1 ta đã biết, kết quả cuối cùng của model là SxS các tensor (sau 24 lớp chập và các lớp khác, kết thúc là 2 lớp fully connected) , với mỗi tensor có kích thước $1 \times 1 \times 30$, còn ở YOLOv2 thì là $13 \times 13 \times 125$.

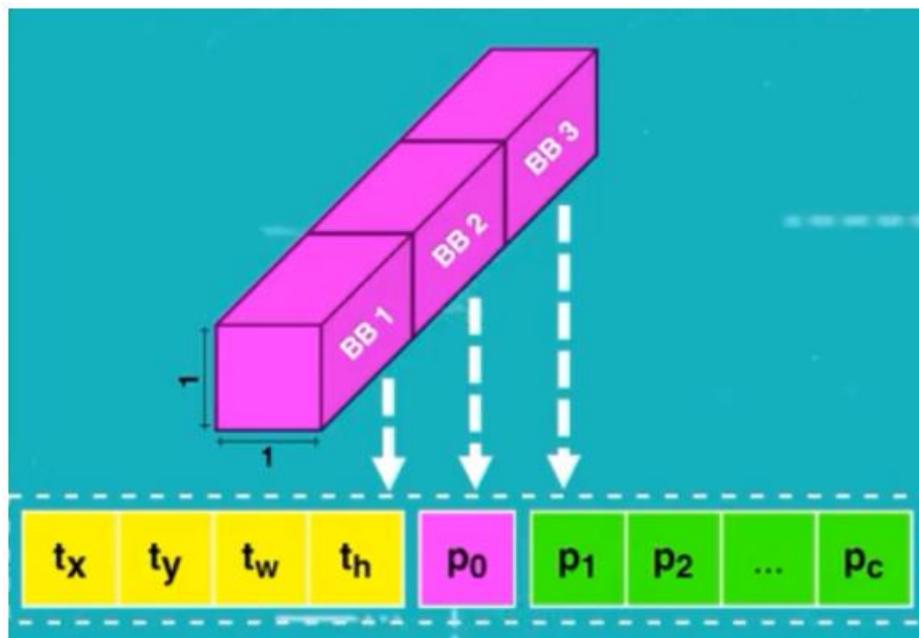
Sang tới YOLOv3 thì kết quả đầu ra lại được thay đổi thành SxSx255, trong đó:

- S lần lượt được gán với các giá trị: 13, 26, 52. Và
- Giá trị 255 = B x (5+C) với B = 3 là số lượng bounding box, và C = 80 là số lớp vật thể.
- Mỗi bounding box chứa 5 giá trị bao gồm: x, y, w, h và p (xác suất grid cell **có chứa vật thể**) như đã giải thích ở YOLOv1.
- Để tính ra kết quả cuối cùng thì mỗi bounding box sẽ nhân giá trị P lần lượt với P₁, P₂, ..., P_C (là xác suất grid



cell đó chứa các vật thể cụ thể) và lựa chọn ra kết quả cao nhất.

- Với mỗi bức ảnh được nhận dạng, thì YOLOv3 sẽ sinh ra $(13 \times 13 + 26 \times 26 + 52 \times 52) \times 3 = 10,647$ bounding box. Từ đó sẽ tính xác suất và xuất ra kết quả là những vật thể có xác suất cao nhất.



kích thước của tensor

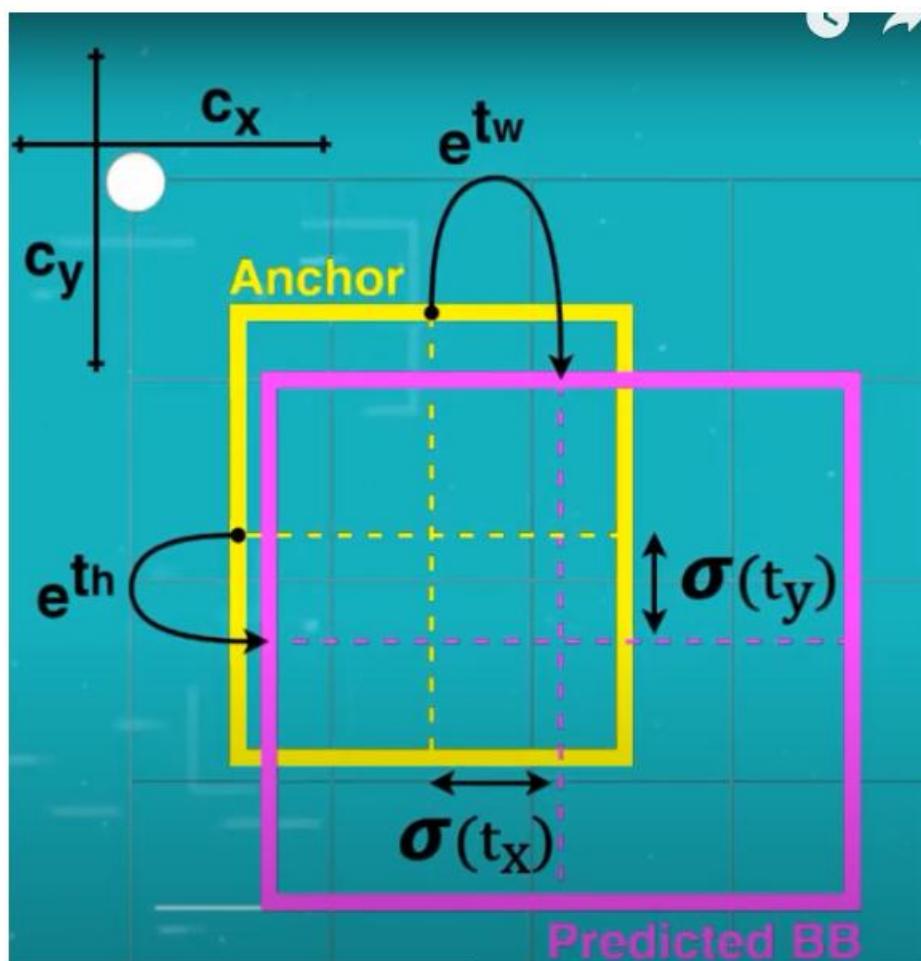
Anchor box

Anchor box thực ra là các bounding box nhưng được tạo sẵn (chứ không phải kết quả của quá trình nhận dạng – prediction).

Với mỗi grid cell sẽ 9 anchor box khác nhau với kích thước:

- Grid cell 13×13 : $(116 \times 90), (156 \times 198), (373 \times 326)$.
- Grid cell 26×26 : $(30 \times 61), (62 \times 45), (59 \times 119)$.
- Grid cell 52×52 : $(10 \times 13), (16 \times 30), (33 \times 23)$.

Trong quá trình đào tạo sẽ kết hợp với thuật toán K-mean cluster (thuật toán phân cụm). Cùng với ground truth để tính ra sai sót giữa ground truth và anchor box bằng cách điều chỉnh các giá trị x,y,w,h, từ đó học được các đặc điểm của vật thể.



quá trình huấn luyện để tinh chỉnh kích thước của anchor box sao cho giống với vật thể nhất

Tổng kết

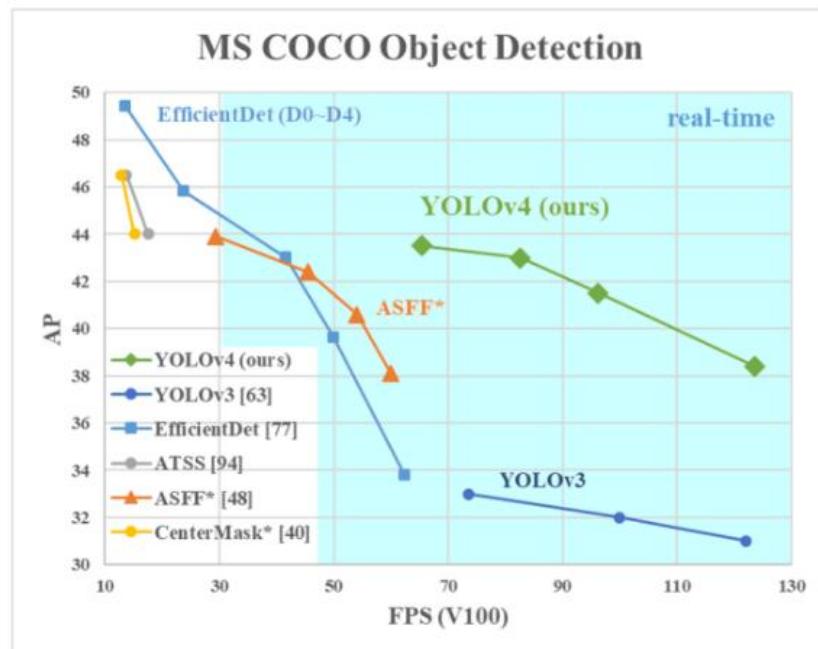
Những cải tiến của YOLOv3 bao gồm:

- Một backbone mới: kết hợp skip-connection vào trong backbone, tăng số lớp Convolution lên.
- Thêm Feature Pyramid Network, thực hiện predict tại 3 scale.
- Hàm loss mới.



YOLOv4

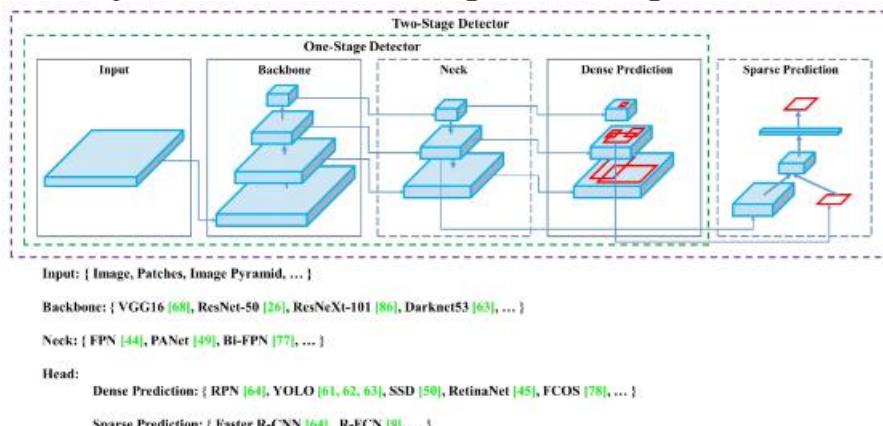
YOLOv4 có nhiều sự cải tiến đặc biệt giúp tăng độ chính xác và tốc độ hơn đối với người anh em YOLOv3 trên cùng tập dataset COCO và trên GPU V100.



so sánh YOLOv4 và YOLOv3

Cấu trúc của v4 được tác giả chia làm bốn phần:

- Backbone
- Neck
- Dense prediction - sử dụng các one-stage-detection như YOLO hoặc SSD.
- Sparse Prediction – sử dụng các two-stage-detection như RCNN.



cấu trúc YOLOv4



Backbone – Trích xuất đặc trưng

Lựa chọn Backbone

Mạng backbone cho nhận dạng vật thể thường được đào tạo trước (pre-train) thông qua bài toán phân loại ImageNet. Pre-train có nghĩa là trọng số của mạng đã được điều chỉnh để xác định các đặc điểm liên quan trong một hình ảnh, mặc dù chúng sẽ được tinh chỉnh trong nhiệm vụ mới là phát hiện đối tượng. Tác giả xem xét sử dụng các backbone:

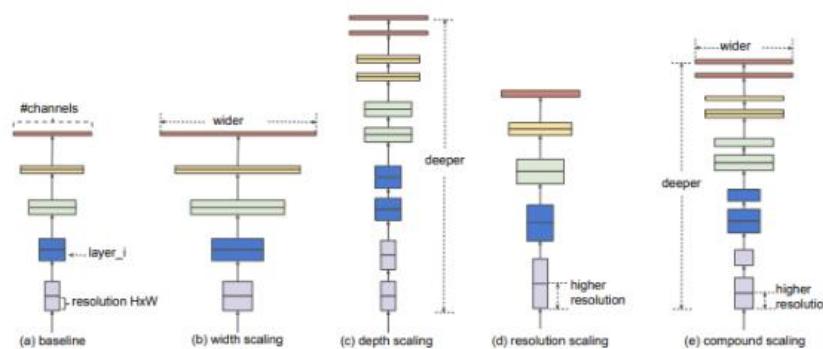
- CSPResNext50
- CSPDarknet53
- EfficientNet-B3

Backbone model	Input network resolution	Receptive field size	Parameters	Average size of layer output (WxHxC)	BFLOPs (512x512 network resolution)	FPS (GPU RTX 2070)
CSPResNext50	512x512	425x425	20.6 M	1058 K	31 (15.5 FMA)	62
CSPDarknet53	512x512	725x725	27.6 M	950 K	52 (26.0 FMA)	66
EfficientNet-B3 (ours)	512x512	1311x1311	12.0 M	668 K	11 (5.5 FMA)	26

các Backbone được sử dụng trong YOLOv4

DenseNet được thiết kế để kết nối các lớp trong mạng nơ-ron phức tạp nhằm mục đích: để giảm bớt vấn đề gradient biến mất (khó có thể sao chép tín hiệu đã bị thất thoát trong một mạng rất sâu), để tăng cường lan truyền tính năng, khuyến khích mạng sử dụng lại các tính năng và giảm số lượng thông số mạng.

EfficientNet được thiết kế bởi Google Brain để chủ yếu nghiên cứu vấn đề mở rộng quy mô của mạng nơ-ron tích chập. Có rất nhiều quyết định mà chúng ta có thể đưa ra khi mở rộng ConvNet của mình bao gồm kích thước đầu vào, tỉ lệ chiều rộng, tỉ lệ chiều sâu và mở rộng tất cả những điều trên. EfficientNet cho rằng có một điểm hoàn hảo, có thể tối ưu cho tất cả các thông số đó và thông qua tìm kiếm, họ đã tìm thấy điểm đó.



Efficient Net

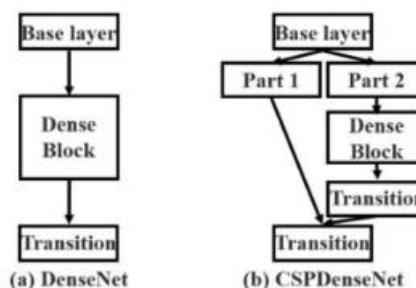


Efficient Net vượt trội hơn các mạng khác có cùng kích thước về phân loại hình ảnh. Tuy nhiên, tác giả của YOLOv4 cho rằng các mạng khác có khả năng hoạt động tốt hơn trong cài đặt để phát hiện đối tượng nên quyết định thử nghiệm với tất cả 3 mạng CNN ở trên. Và cuối cùng thì tác giả chọn mạng CSPDarknet53 là backbone cho model.

Cấu trúc của CPSDarknet53

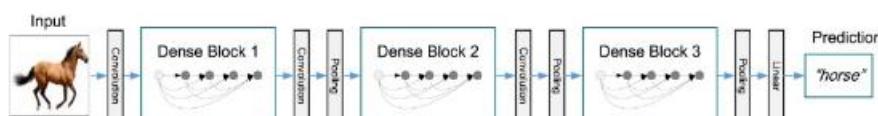
CSPDarknet53 được cấu tạo từ CSP, Darknet53.

CSP(Cross-Stage-Partial connections) có nguồn gốc từ kiến trúc DenseNet sử dụng đầu vào trước đó và nối nó với đầu vào hiện tại trước khi chuyển vào Dense layer. Nó có nhiệm vụ chia đầu vào của khối thành 2 phần, một phần sẽ qua các khối chập, và phần còn lại thì không (đi thẳng tới cuối khối). sau đó hai phần sẽ được cộng lại và đưa vào khối tiếp theo. Ý tưởng ở đây là loại bỏ các nút thắt tính toán trong DenseNet và cải thiện việc học bằng cách chuyển phiên bản chưa chỉnh sửa của feature maps.



Mô tả cấu trúc CSP

DenseNet (Dense connected convolutional network) là một trong những network mới nhất cho visual object recognition. Nó cũng gần giống Resnet nhưng có một vài điểm khác biệt. DenseNet có cấu trúc gồm các dense block và các transition layers. Được stack dense block-transition layers-dense block-transition layers như hình vẽ. Với CNN truyền thống nếu chúng ta có L layer thì sẽ có L connection, còn trong DenseNet sẽ có $L(L+1)/2$ connection (tức là các lớp phía trước sẽ được liên kết với tất cả các lớp phía sau nó).



Cấu trúc DenseNet



Darknet53: Yolov4 sử dụng **CSPDarknet53** để làm backbone vì theo tác giả, **CSPDarknet53** có độ chính xác trong task object detection cao hơn so với ResNet; và mặc dù ResNet có độ chính xác trong task classification cao hơn, hạn chế này có thể được cải thiện nhờ hàm activation Mish và một vài kỹ thuật sẽ được đề cập phía dưới.

Neck – Tổng hợp đặc trưng

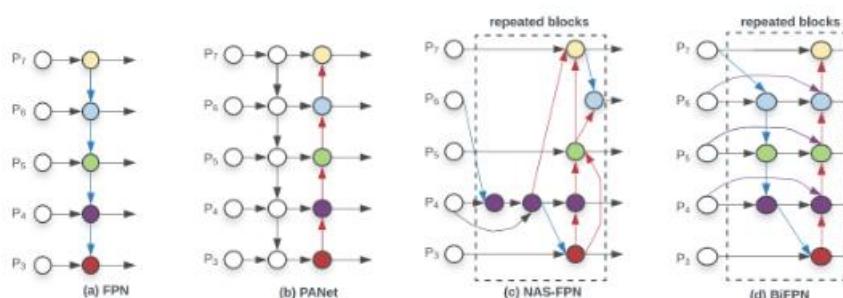
Neck có nhiệm vụ trộn và kết hợp các features map đã học được thông qua quá trình trích xuất đặc trưng (backbone) và quá trình nhận dạng (YOLOv4 gọi là Dense prediction).

Với mỗi lần thực hiện detect với các kích thước ảnh rescale khác nhau tác giả đã thêm các luồng đi từ dưới lên và các luồng đi từ trên xuống vào cùng nhau theo từng hoặc được nối với nhau trước khi đưa vào head, từ đó lớp nhận dạng sẽ chứa thông tin phong phú hơn.

Tác giả của YOLOv4 đã cho phép tùy biến sử dụng các cấu trúc cho phần Neck như là :

- FPN
- PAN
- NAS-FPN
- BiFPN
- ASFF
- SFAM
- SSP

Một số cấu trúc điển hình:



Một số cấu trúc sử dụng cho phần cổ (Neck)



YOLOv5

YOLOv5 được công bố gần đây với những so sánh ban đầu cho thấy độ chính xác tương đương YOLOv4 và có tốc độ nhanh hơn khi thực hiện dự đoán, tuy nhiên vẫn có rất nhiều hoài nghi về độ tin cậy của những so sánh này vì YOLOv5 mới được ra mắt trên GitHub chứ chưa có bài báo chính thức nào cả, thậm chí cũng không được công nhận bởi tác giả của các bản YOLO trước đó. Tuy nhiên, nó vẫn được cộng đồng đón nhận.

Tác giả là Glenn Jocher cũng là người phát minh ra phương pháp tăng dữ liệu Mosaic và được Alexey Bochkovsky thừa nhận trong bài báo YOLOv4 (Bochkovskiy, et al., 2020). Tuy nhiên, mô hình YOLOv5 của ông đã gây ra nhiều tranh cãi trong cộng đồng thị giác máy tính vì tên gọi và những cải tiến của nó.

Mặc dù được phát hành sau YOLOv4 một tháng, thời gian bắt đầu nghiên cứu YOLOv4 và YOLOv5 đã khá gần (tháng 3 - tháng 4 năm 2020). Để tránh va chạm, Glenn quyết định đặt tên cho phiên bản YOLO của mình là YOLOv5. Vì vậy, về cơ bản, cả hai nhà nghiên cứu đều áp dụng những sáng tạo tiên tiến nhất trong lĩnh vực thị giác máy tính vào thời điểm đó. Điều đó làm cho kiến trúc của YOLOv4 và YOLOv5 rất giống nhau và nó khiến nhiều người không hài lòng với cái tên YOLOv5 (thứ 5 của YOLO) khi nó không chứa đựng nhiều cải tiến vượt trội so với phiên bản YOLOv4 trước đó. Bên cạnh đó, Glenn đã không công bố bất kỳ bài báo nào cho YOLOv5, khiến nhiều người nghi ngờ hơn về YOLOv5.

Tuy nhiên, YOLOv5 sở hữu lợi thế về kỹ thuật. YOLOv5 được viết bằng ngôn ngữ lập trình Python thay vì C như các phiên bản trước. Điều đó làm cho việc cài đặt và tích hợp trên các thiết bị IoT dễ dàng hơn. Ngoài ra, cộng đồng PyTorch cũng lớn hơn cộng đồng Darknet, đồng nghĩa với việc PyTorch sẽ nhận được nhiều đóng góp và tiềm năng phát triển hơn trong tương lai. Do được viết bằng 2 ngôn ngữ khác nhau trên 2 framework khác nhau nên so sánh hiệu suất giữa YOLOv4 và YOLOv5 rất khó chính xác. Nhưng sau một thời gian, YOLOv5 đã chứng tỏ hiệu suất cao hơn YOLOv4 trong một số hoàn cảnh nhất định và phần nào tạo được niềm tin trong cộng đồng thị giác máy tính bên cạnh YOLOv4.

Nano					
YOLOv5n					
4 MB _{FP16} 6.3 ms _{V100} 28.4 mAP _{coco}	14 MB _{FP16} 6.4 ms _{V100} 37.2 mAP _{coco}	41 MB _{FP16} 8.2 ms _{V100} 45.2 mAP _{coco}	89 MB _{FP16} 10.1 ms _{V100} 48.8 mAP _{coco}	166 MB _{FP16} 12.1 ms _{V100} 50.7 mAP _{coco}	
Small					
YOLOv5s					
Medium					
YOLOv5m					
Large					
YOLOv5l					
XLarge					
YOLOv5x					

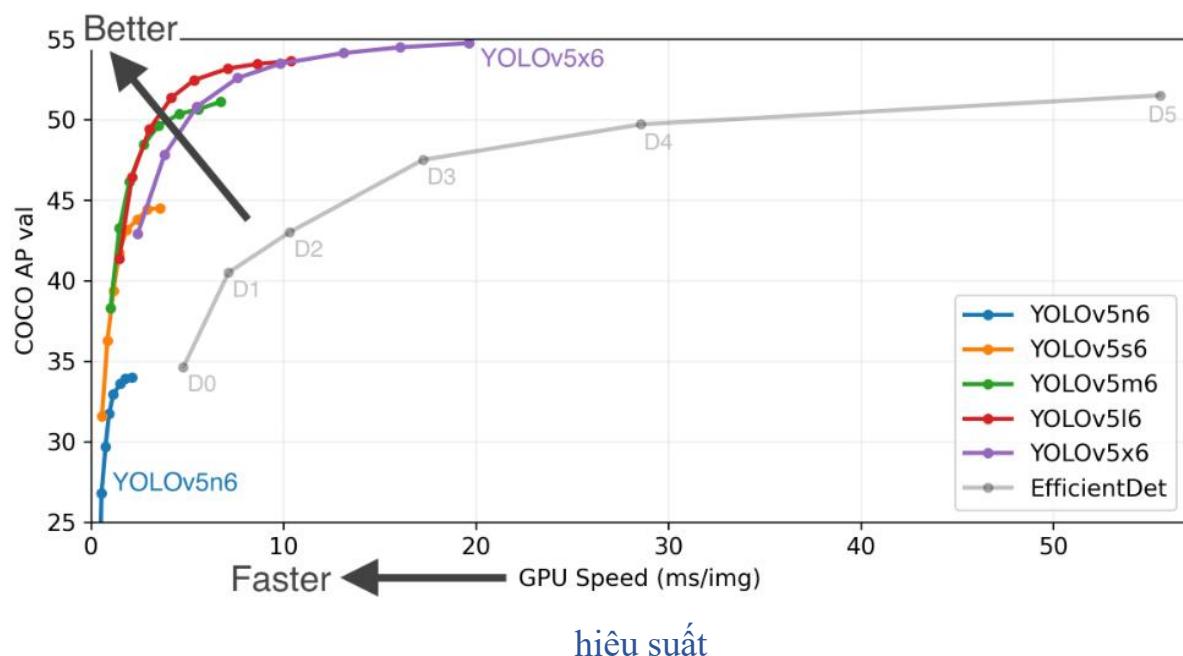
một số model trong YOLOv5



Với mỗi model của YOLOv5, độ phức tạp tăng dần từ Nano -> XLarge.
Đồng thời thời gian cần để train một model hoặc là độ chính xác cũng tăng từ
Nano -> XLarge.

Model	size (pixels)	mAP ^{val} 0.5:0.95	mAP ^{val} 0.5	Speed CPU b1 (ms)	Speed V100 b1 (ms)	Speed V100 b32 (ms)	params (M)	FLOPs @640 (B)
YOLOv5n	640	28.0	45.7	45	6.3	0.6	1.9	4.5
YOLOv5s	640	37.4	56.8	98	6.4	0.9	7.2	16.5
YOLOv5m	640	45.4	64.1	224	8.2	1.7	21.2	49.0
YOLOv5l	640	49.0	67.3	430	10.1	2.7	46.5	109.1
YOLOv5x	640	50.7	68.9	766	12.1	4.8	86.7	205.7
YOLOv5n6	1280	36.0	54.4	153	8.1	2.1	3.2	4.6
YOLOv5s6	1280	44.8	63.7	385	8.2	3.6	12.6	16.8
YOLOv5m6	1280	51.3	69.3	887	11.1	6.8	35.7	50.0
YOLOv5l6	1280	53.7	71.3	1784	15.8	10.5	76.8	111.4
YOLOv5x6 + TTA	1536	55.0	72.7	3136	26.2	19.4	140.7	209.8
		55.8	72.7	-	-	-	-	-

phân tích cụ thể



hiệu suất



Pytorch

PyTorch là một Neural Network Dynamic Framework. Nói đến tính động Dynamic thì cũng sẽ có các Framework được gọi là Static Neural Network Framework như Tensorflow, Keras, Theano ... và điểm khác biệt cơ bản giữa chúng nằm ở hai điểm sau:

- Trong các **Static Framework** thì đồ thị tính toán (computation graph) được định nghĩa trước, sau đó được compile và tiến hành ném các mẫu dữ liệu qua đồ thị để tính toán. Chúng ta có thể tưởng tượng điều này giống như một người làm đường ống nước. Trước tiên anh ta sẽ phải xây dựng đường ống và các van liên kết các đường ống (tương tự như xây dựng kiến trúc một mạng nơ ron) và sau đó anh ta mới đổ nước vào đường ống (tương tự như việc fit dữ liệu vào mô hình)
- Còn đối với một **Dynamic Framework** thì sao. Nó sẽ không thực hiện việc khởi tạo graph trước mà sẽ tiến hành chạy ngay khi chúng ta fit dữ liệu vào trong mô hình. Sẽ không có các thao tác biên dịch khiến cho việc tính toán được thực hiện nhanh chóng.

Để hiểu được Pytorch là gì, đầu tiên chúng ta phải biết khái niệm Framework trong lĩnh vực phần mềm. **Framework** là thuật ngữ Tiếng anh mang nghĩa “**bộ khung**”. Nó là những đoạn code chức năng cơ bản được các nhà lập trình viết sẵn, tạo nên một tập hợp các thư viện lập trình, thư viện phần mềm, các trình biên dịch, diễn dịch và cả API nữa cung cấp các tiện ích cơ bản giúp lập trình viên thuận tiện và nhanh chóng hơn trong việc xây dựng các phần mềm ứng dụng. Framework ra đời giúp cho nhà lập trình giải quyết được rất nhiều vấn đề từ lập trình nói chung đến lập trình web, app cụ thể nói riêng.

Một cách đơn giản và dễ hiểu hơn, hãy tưởng tượng framework là một khung nhà thô đã được làm sẵn trên một nền móng cơ bản và vững chắc, còn chúng ta chính là những kỹ sư, việc xây dựng và thiết kế nội thất cho căn nhà như thế nào phụ thuộc hoàn toàn vào chúng ta.

Pytorch chính là một **framework hỗ trợ Deep Learning** được phát triển bởi **Facebook**. (Bên cạnh Amazon, Google hay Apple, Facebook được biết đến là đơn vị công nghệ đầu tư rất nhiều nguồn lực cho việc phát triển trí tuệ nhân tạo).

Những cải tiến chính trong YOLOv5

YOLOv5 khác với tất cả các bản phát hành trước vì, vì đây là một triển khai PyTorch chứ không phải là một nhành từ Darknet gốc. Tương tự như YOLOv4, YOLOv5 có backbone CSP và cổ PA-NET. Các cải tiến chính bao gồm tăng dữ liệu khám và neo hộp giới hạn tự động học (auto learning bounding box anchors).



Tranh cãi trong cộng đồng học máy

Việc phát hành YOLOv5 đã thu hút nhiều sự chú ý và dây ra các cuộc thảo luận sôi nổi trên các nền tảng cộng đồng học máy. Điều này chủ yếu là do một số dữ kiện trên một bài báo được xuất bản bởi nhóm Roboflow liên quan đến YOLOv5.

Bài báo đó có tiêu đề ‘YOLOv5 is Here’ đã được xuất bản vào ngày 10 tháng 6 năm 2021 trên blog Roboflow, nêu rõ một sự kiện quan trọng. Tiếp theo là một số trích dẫn từ bài đăng trên blog đó của Joseph Nelson và Jacob Solawetz

“Chạy Tesla P100, chúng tôi thấy thời gian suy luận lên đến 0,007 giây cho mỗi hình ảnh, nghĩa là 140 khung hình mỗi giây (FPS)! Ngược lại, YOLO v4 đạt được 50 FPS sau khi được chuyển đổi sang cùng thư viện Ultralytics PyTorch.”

“YOLO v5 nhỏ. Cụ thể, tệp trọng số cho YOLO v5 là 27 megabyte. Tệp trọng số của chúng tôi cho YOLO v4 (với kiến trúc Darknet) là 244 megabyte. YOLO v5 nhỏ hơn gần 90% so với YOLO v4.”



2. Cách thức YOLO nhận diện vật thể

Classification + Localization -> Detection

Image Classification là nơi một thuật toán được áp dụng cho một hình ảnh để dự đoán lớp của đối tượng, ví dụ: xe ô tô.

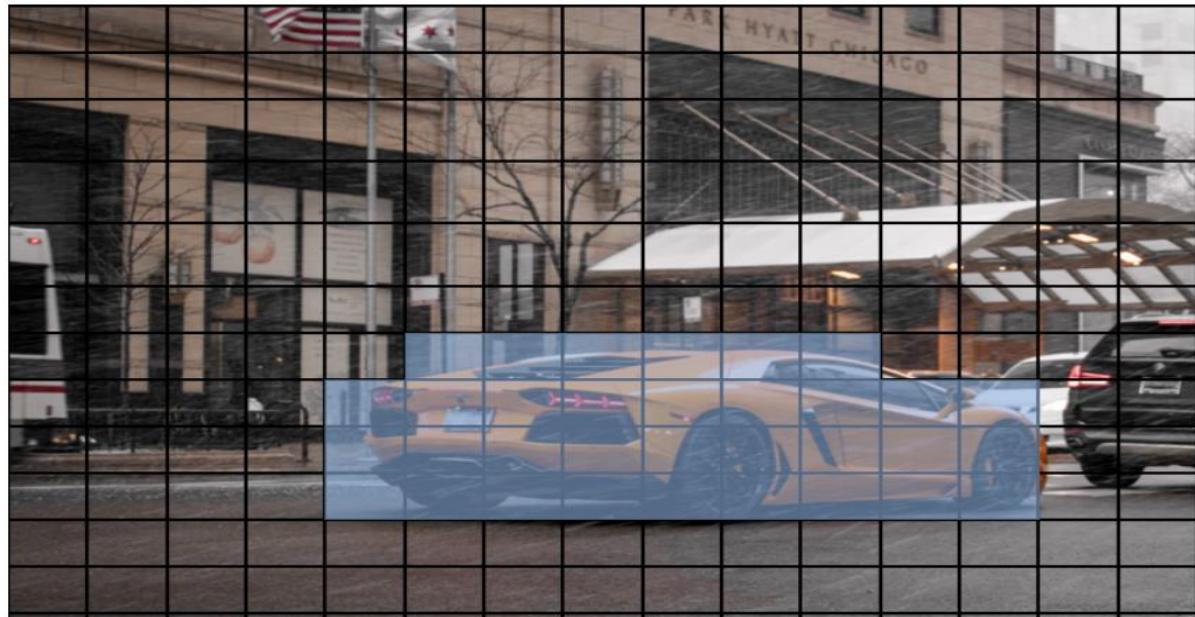
Classification with localization không chỉ dự đoán lớp của đối tượng mà còn tìm ra vị trí của đối tượng bằng cách vẽ một Bounding Box xung quanh đối tượng.

Detection liên quan đến cả Classification và Localization và có thể nhận diện hơn một đối tượng và thậm chí nhiều hơn một lớp.



Bounding Box

Một nhiệm vụ phân loại tiêu chuẩn sẽ liên quan đến một hình ảnh chạy qua Convnet với nhiều lớp trong đó các đặc trưng vector được đưa vào một đơn vị softmax, ví dụ như xuất ra lớp được dự đoán (các đối tượng mà thuật toán đang cố gắng phát hiện, ví dụ như ô tô, cây cối, người đi bộ). Các mô hình phát hiện đối tượng như YOLO hoạt động bằng cách chia hình ảnh thành một grid cell ô trong đó mỗi grid cell chịu trách nhiệm dự đoán một bounding box nếu tâm của bounding box nằm trong ô đó. Sau đó, nó sẽ xuất ra lớp dự đoán, tọa độ của bounding box như hình dưới đây:



các ô có màu xanh dương dự đoán rằng tâm của bounding box nằm trong ô

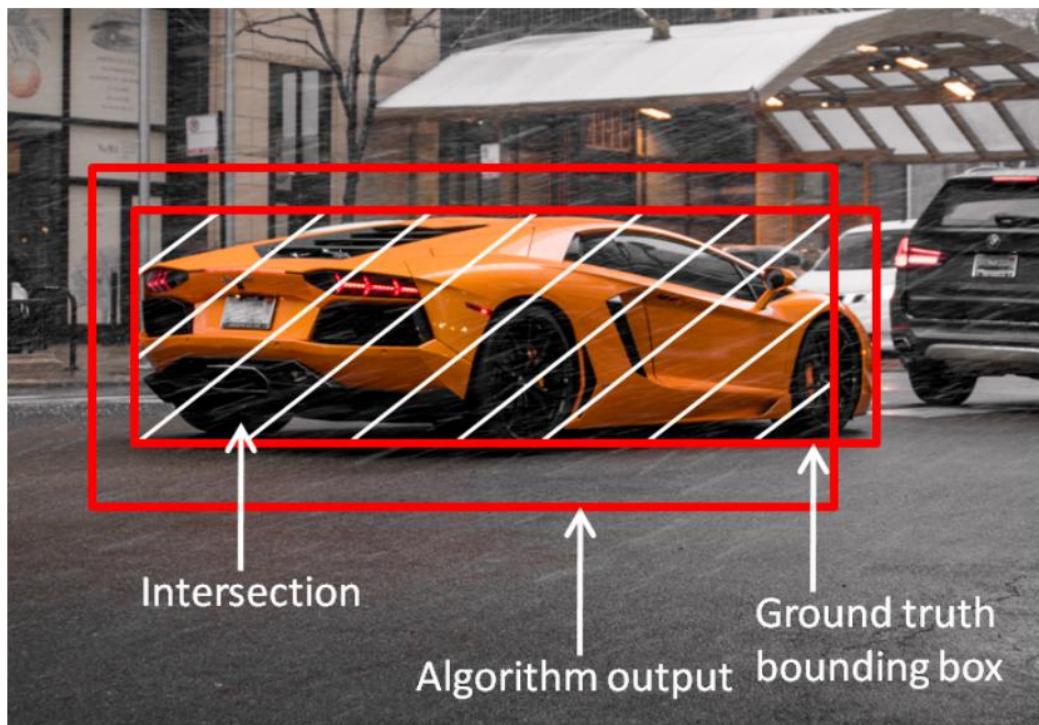


hình ảnh hiển thị các biến được liên kết với bounding box

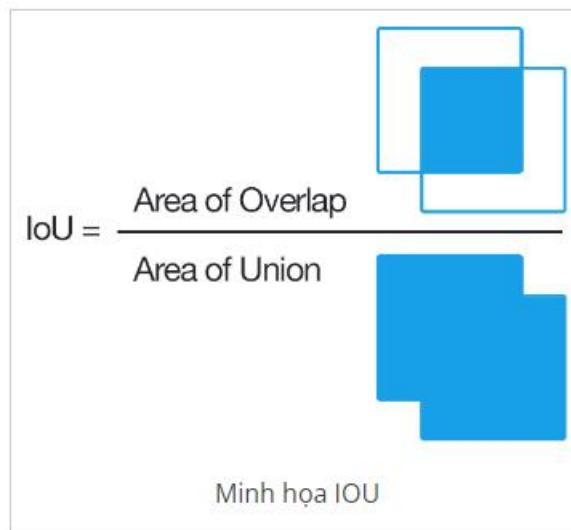
Intersection over Union (IoU)

Khi thuật toán xuất ra các bounding box localizing object được phát hiện, làm thế nào để ta biết liệu thuật toán hoạt động tốt hay không? Thì đây là lúc IoU phát huy tác dụng. Nói chung, IoU là chỉ số đánh giá được sử dụng để đo độ chính xác của phát hiện đối tượng trên tập dữ liệu cụ thể. Chỉ số này thường được gặp trong các Object Detection Challenge. IOU thường được đánh giá hiệu năng của các bộ phát hiện đối tượng như HOG + Linear SVM và mạng nơ ron tích chập (R-CNN, FastR-CNN, YOLO,...).

Để áp dụng được IoU để đánh giá cần: Algorithm Bounding Box và Ground Truth Bounding Box



Tỉ lệ này là tỉ lệ đo lường mức độ giao nhau giữa hai bounding box để nhằm xác định hai bounding box có bị đè chòng lên nhau không. Tỉ lệ này được tính dựa trên phần diện tích giao nhau giữa 2 bounding box với phần tổng diện tích giao nhau và không giao nhau giữa chúng.

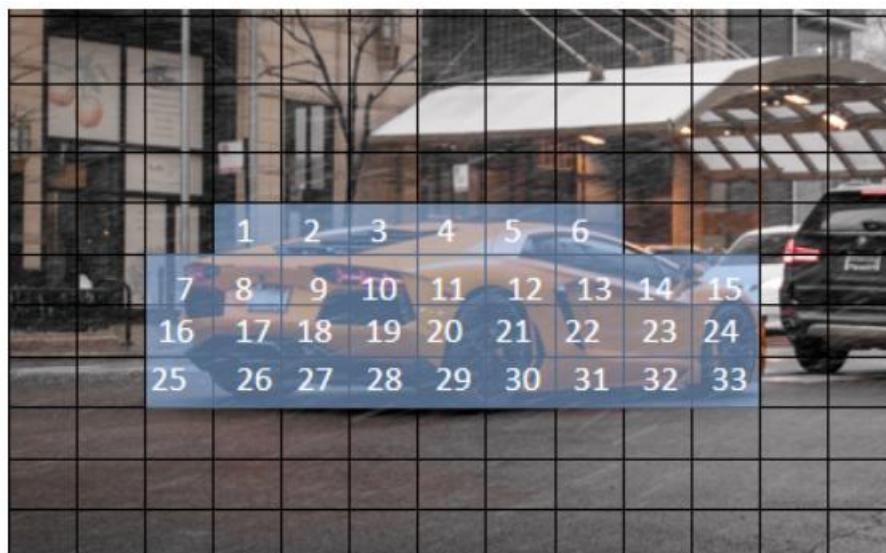


Nguồn IoU là khoảng 0,5. IoU có giá trị $\geq 0,5$ được coi là dự đoán đúng.



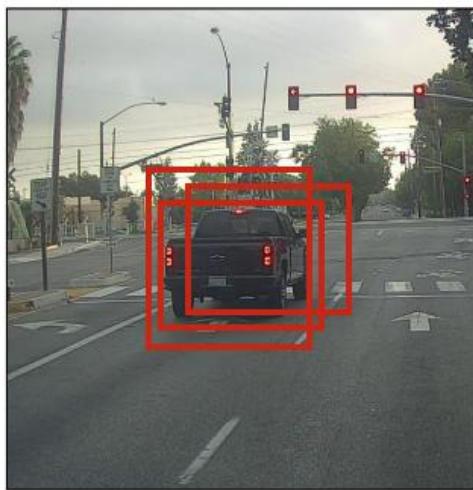
Non-max Suppression (NMS)

Với tham chiếu đến hình ảnh bounding box bên dưới, các ô có nhãn 1-33 đều dự đoán rằng tâm của bounding box nằm trong ô của chúng.



Điều này sẽ dẫn đến thuật toán sẽ phát hiện đối tượng nhiều lần thay vì chỉ một lần. Đây là lúc mà tính năng NMS xuất hiện để đảm bảo thuật toán phát hiện mỗi đối tượng chỉ một lần.

Before non-max suppression



Non-Max
Suppression



After non-max suppression



Các bước của non-max suppression:

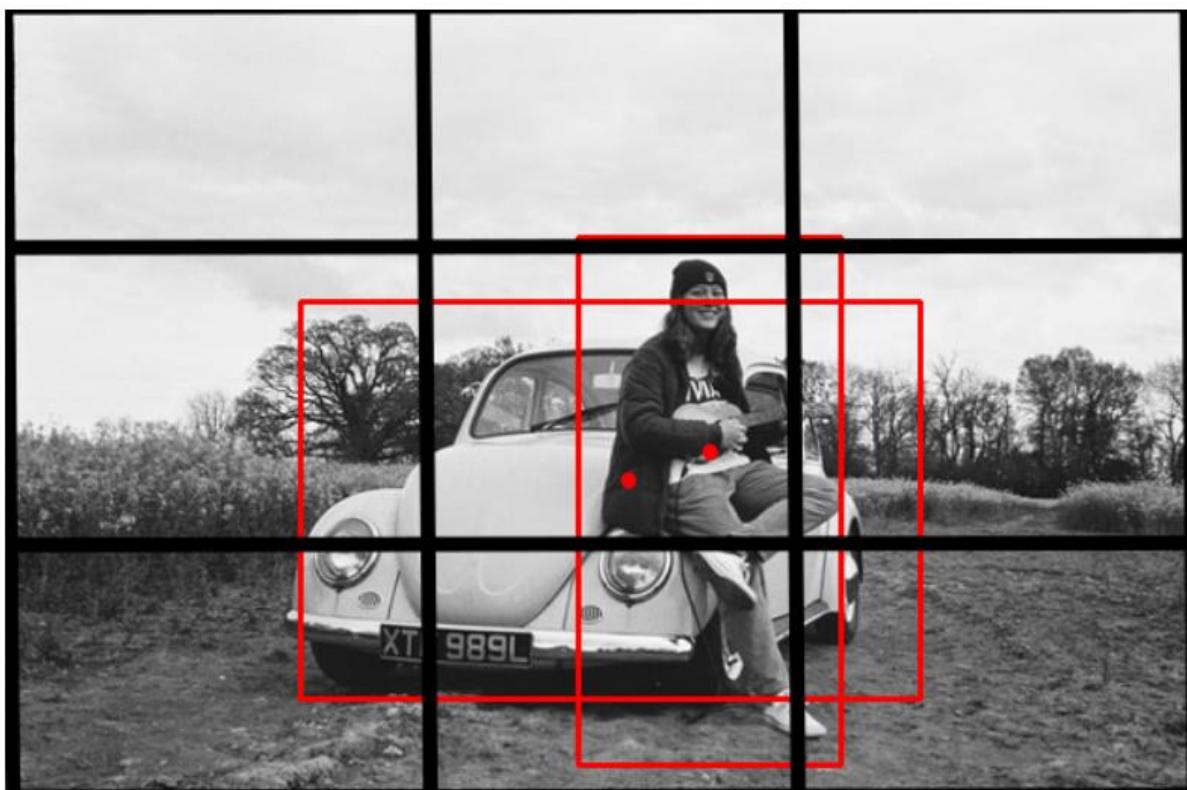
- Step 1: Đầu tiên YOLO sẽ tìm cách giảm bớt số lượng các bounding box bằng cách lọc bỏ toàn bộ những bounding box có xác suất chứa vật thể nhỏ hơn một ngưỡng threshold nào đó, thường là 0.5.
- Step 2: Đối với các bounding box giao nhau, non-max suppression sẽ lựa chọn ra một bounding box có xác suất chứa vật thể là lớn nhất. Sau đó tính toán chỉ số giao nhau IoU với các bounding box còn lại.



Anchor Boxes

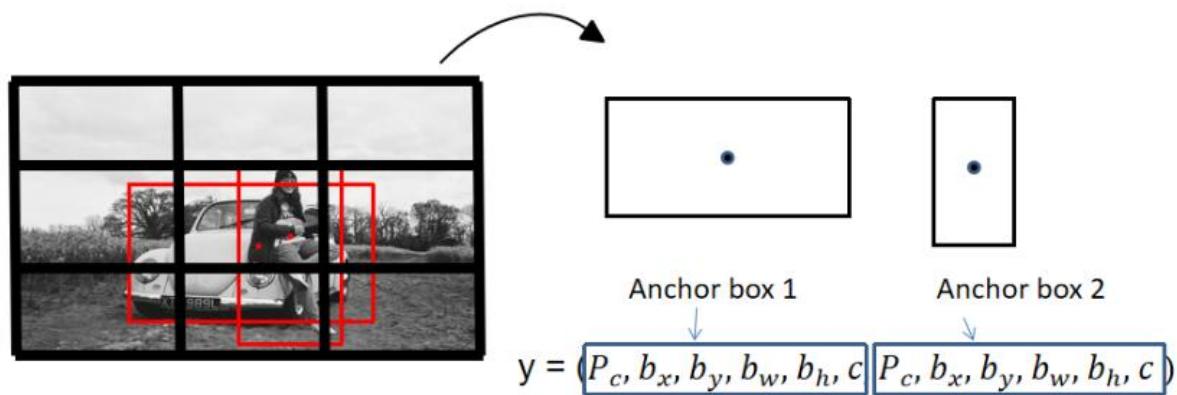
Điều gì sẽ xảy ra nếu nhiều đối tượng nằm trong cùng một grid cell? Các bounding box sẽ như thế nào? Thì đây là lúc có thể sử dụng ý tưởng về các anchor boxes.

Nhìn vào hình ảnh bên dưới, hãy để ý xem điểm giữa của người và xe nằm trong grid cell như thế nào. Hình ảnh được chia thành grid cell 3x3.



Hình ảnh mô tả grid cell chứa trung điểm của 2 đối tượng

Kết quả ô hiện tại $y = (P_c, b_x, b_y, b_w, b_h, c)$, ô chỉ có thể chọn một trong hai đối tượng được phát hiện. Nhưng với anchor box (thường được xác định trước bằng cách sử dụng phân tích k-mean trên tập dữ liệu train), kết quả y trở thành $(P_c, b_x, b_y, b_w, b_h, c, P_{c1}, b_{x1}, b_{y1}, b_{w1}, b_{h1}, c_1, \dots)$. Sẽ có 2 output, với output đầu tiên được mã hóa bằng anchor box 1 và output thứ hai được mã hóa bằng anchor box 2,.. Mỗi đơn vị output phát hiện cho một đối tượng, anchor box 1 tương tự như ô tô nên output c sẽ là ô tô, output tiếp theo c sẽ là người vì nó được mã hóa cho anchor box 2.



Mỗi đơn vị output c đại diện cho lớp đối tượng bị ảnh hưởng bởi IoU cao với ground truth bounding box của đối tượng.



Chương 3: GIẢI QUYẾT BÀI TOÁN

1. Hướng tiếp cận

Để nhằm mục đích thử đánh giá về kết quả test của một số model YOLO thì nhóm quyết định tiếp cận bài toán theo hai hướng tiếp cận là sử dụng hai thế hệ được ra đời gần nhất của YOLO đó là YOLOv4-Tiny và YOLOv5.

Khái niệm YOLOv4 và YOLOv5 đã được nhắc đến ở trên nhưng tại sao ở đây lại xuất hiện bản YOLOv4-Tiny? Vậy YOLOv4-Tiny là gì?

YOLOv4-Tiny

YOLOv4-Tiny là một phiên bản nén của YOLOv4. Nó được phát triển dựa trên YOLOv4 để làm cho cấu trúc mạng đơn giản hơn và giảm các thông số để nó trở nên khả thi cho việc phát triển trên các thiết bị di động và hệ thống nhúng.

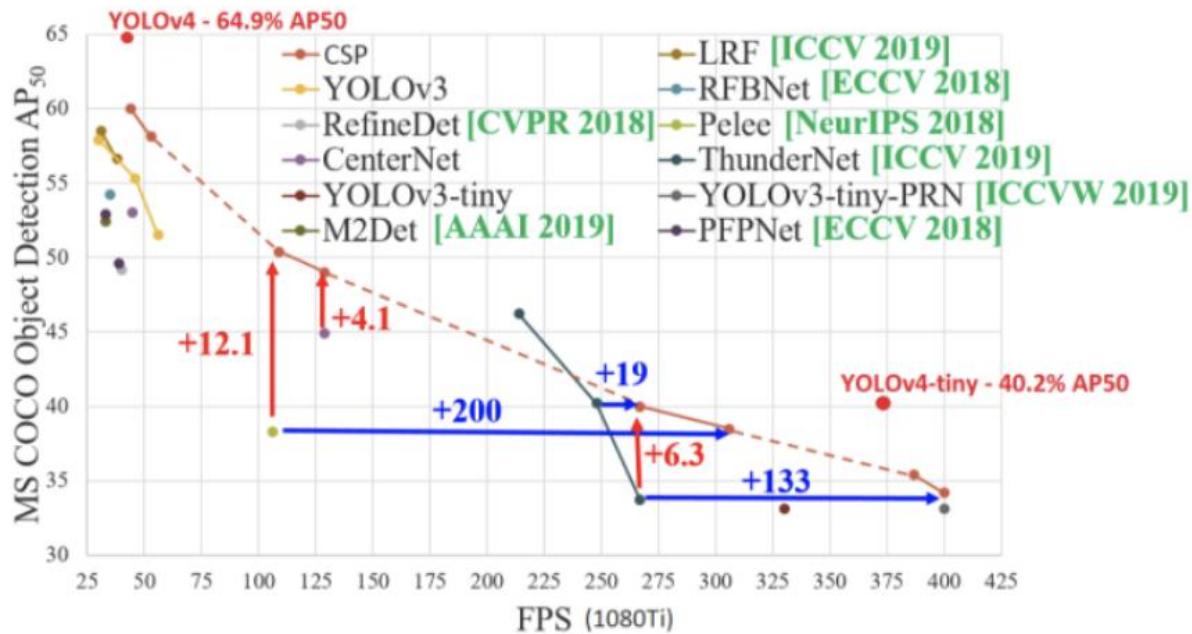
Chúng ta có thể sử dụng YOLOv4-Tiny để huấn luyện nhanh hơn và phát hiện vật thể nhanh hơn. Nó chỉ có hai đầu YOLO so với ba đầu trong YOLOv4 và nó đã được huấn luyện từ 29 lớp chập được đào tạo trước, trái ngược với YOLOv4 đã được huấn luyện từ 137 lớp chập được đào tạo từ trước.

FPS (Khung hình trên giấy) trong YOLOv4-Tiny xấp xỉ tầm lèn so với YOLOv4. Tuy nhiên, độ chính xác của YOLOv4-Tiny là 2/3 so với YOLOv4 khi thử nghiệm trên tập dữ liệu MS COCO.

Mô hình YOLOv4-Tiny đạt được 22% AP (42% AP50) ở tốc độ 443 FPS trên RTX 2080Ti, trong khi bằng cách sử dụng TensorRT, batch size = 4 và FP16-precision (độ chính xác FP16) YOLOv4-Tiny đạt được là 1774 FPS.

Để phát hiện đối tượng trong thời gian thực, YOLOv4-Tiny là lựa chọn tốt hơn khi so sánh với YOLOv4 vì thời gian suy luận nhanh hơn, quan trọng hơn là precision hoặc accuracy khi làm việc với môi trường phát hiện đối tượng thời gian thực.

Vấn đề về thời gian huấn luyện mô hình cũng là lý do để nhóm chọn phiên bản YOLOv4-Tiny thay vì chọn bản YOLOv4.



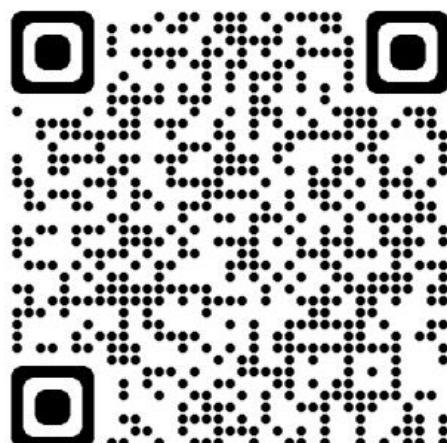
2. Dataset

Ở mỗi mô hình nhóm sẽ thực hiện train 3 lần. Vì vậy sẽ có 3 bộ dataset.

Ở lần train đầu tiên, nhóm sử dụng bộ dataset được tải về từ Kaggle theo format PASCAL VOC và sau đó được chuyển thành format YOLO bằng Roboflow.

Bộ dataset gồm có 848 ảnh và được chia thành 3 lớp:

- with_mask
- without_mask
- mask_weared_incorrect



đường dẫn truy cập vào bộ dataset



Gồm có 3962 khuôn mặt được gán nhãn:

- 3120 with_mask
- 703 without_mask
- 119 mask_weared_incorrect



đường dẫn truy cập vào bộ dataset

- mask_weared_incorrect
- with_mask
- without_mask

cách gán nhãn trong bộ dataset



hình ảnh được trích xuất từ bộ dataset

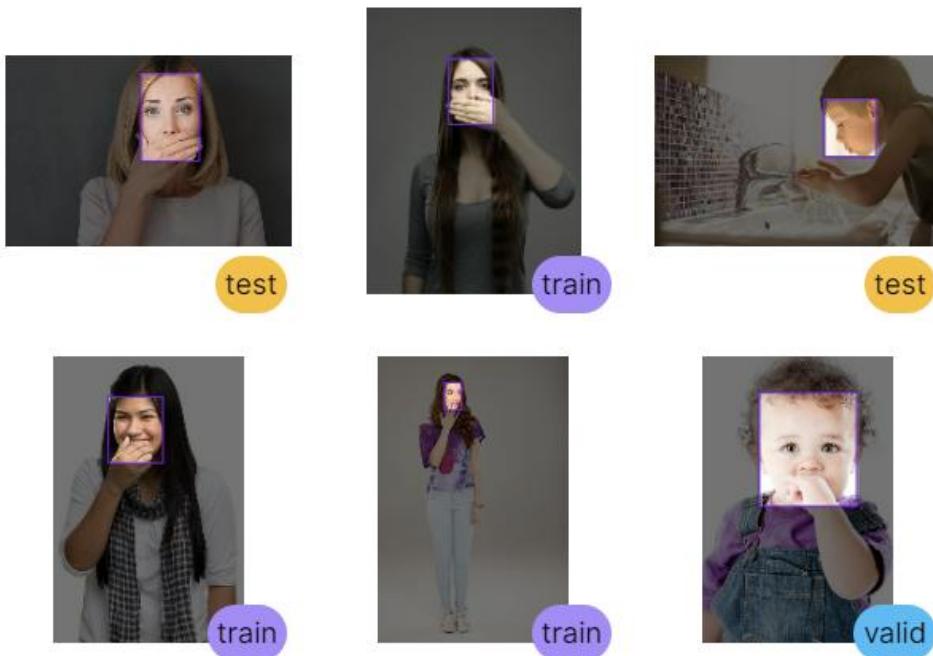


hình ảnh được trích xuất từ bộ dataset

Ở lần train thứ 2, nhóm lấy thêm bộ dataset lầy tay che miệng và trộn với dataset gốc để thực hiện quá trình train lần 2.

Bộ dataset gồm:

- 436 ảnh được tải từ istockphoto.com
- Được gán nhãn without_mask bằng Roboflow



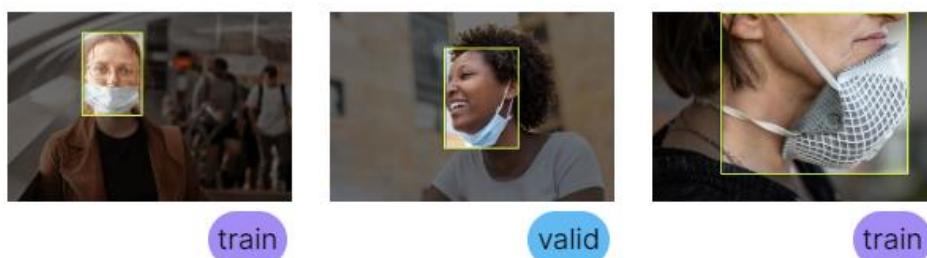
một số hình ảnh trích xuất từ bộ dataset



Ở lần train thứ 3, nhóm lấy thêm bộ dataset đeo khẩu trang sai cách trộn vào dữ liệu ở lần train thứ 2 để thực hiện quá trình train lần 3.

Bộ dataset đeo khẩu trang sai cách:

- 49 ảnh tải về từ istockphoto.com
- Thực hiện sinh data với roboflow thành 117 ảnh
- Được gán nhãn mask_weared_incorrect



một số hình ảnh được trích xuất từ bộ dataset

Lý do nhóm thực hiện 3 lần train với 3 bộ dữ liệu như trên sẽ được đề cập ở phần test và đánh giá mô hình.



3. Train Model

Quá trình train model sẽ được nhóm train trên Google Colab. Lý do nhóm chọn train trên Colab thay vì train trực tiếp trên máy PC là vì máy PC chậm nhung ngược lại Colab train khỏe, nhanh, có GPU khủng và lại miễn phí.

YOLOv4-Tiny

Chuẩn bị dữ liệu

Muốn train được model Deep Learning thì điều nhất thiết phải có đó là dữ liệu. Không có dữ liệu thì chả làm gì được, nên ở đề tài này ta cần train model YOLOv4-Tiny để phát hiện đám người có hoặc không đeo khẩu trai thì dữ liệu ta cần đó là hình ảnh nhìn thấy rõ mặt người để có thể nhìn vào và biết người đó có hay không đeo khẩu trang hay là có đeo nhưng lại đeo sai cách.

Bởi vì bộ dữ liệu được đề cập ở trên đang được tải xuống từ Roboflow theo chuẩn Pytorch nhưng khi sử dụng YOLOv4-Tiny lại là chuẩn Darknet nên toàn bộ dữ liệu ở file images và file labels sẽ được gộp chung vào nhau. Đồng thời khi train model bằng YOLOv4-Tiny thì nhóm sẽ gộp chung dữ liệu ở folder train và val vào lại với nhau tạo thành một folder mới có tên là data.

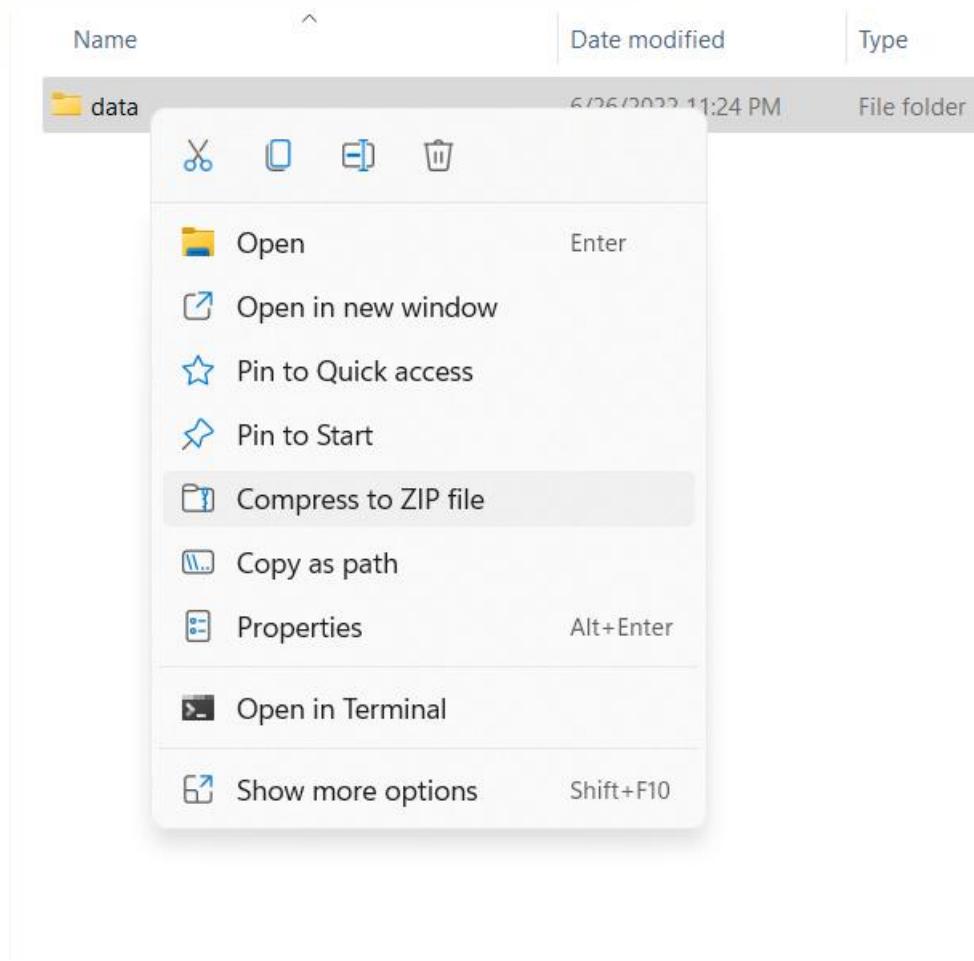
maksskskssss0_png.rf.7c062f62ff9267eb6...	5/25/2022 9:14 PM	JPG File	32 KB
maksskskssss0_png.rf.7c062f62ff9267eb6...	5/25/2022 9:14 PM	Text Document	1 KB
maksskskssss1_png.rf.092d1cfca31ba86e5...	5/25/2022 9:14 PM	JPG File	21 KB
maksskskssss1_png.rf.092d1cfca31ba86e5...	5/25/2022 9:14 PM	Text Document	1 KB
maksskskssss2_png.rf.58ba9815ef068b1b...	5/25/2022 9:14 PM	JPG File	26 KB
maksskskssss2_png.rf.58ba9815ef068b1b...	5/25/2022 9:14 PM	Text Document	1 KB
maksskskssss3_png.rf.27d78c72fabd4ef13...	5/25/2022 9:14 PM	JPG File	32 KB
maksskskssss3_png.rf.27d78c72fabd4ef13...	5/25/2022 9:14 PM	Text Document	1 KB

hình ảnh bộ dataset sau khi được gộp

Trong đó mỗi ảnh sẽ có hai file:

- File có đuôi .jpg: là file ảnh đầu vào
- File có đuôi .txt: là file nhãn của ảnh tương ứng
- Hai file này có tên y hệt nhau và chỉ khác nhau ở phần đuôi.

Sau khi chuẩn bị xong bộ dữ liệu (dataset) thì nén lại.

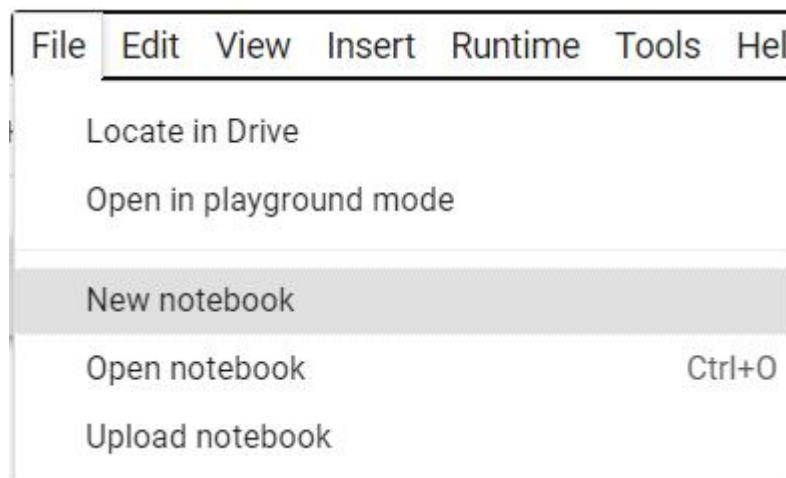


nén dữ liệu

Train model lần 1

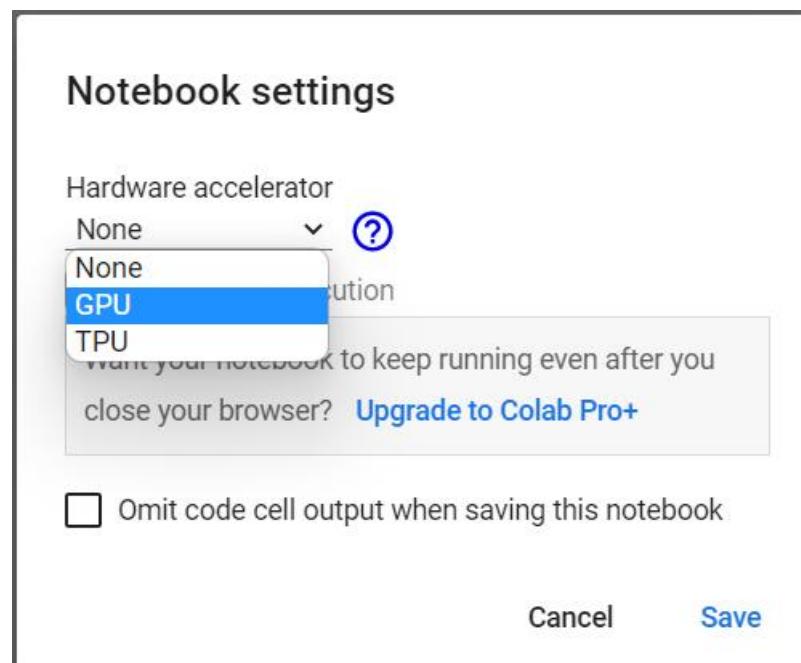
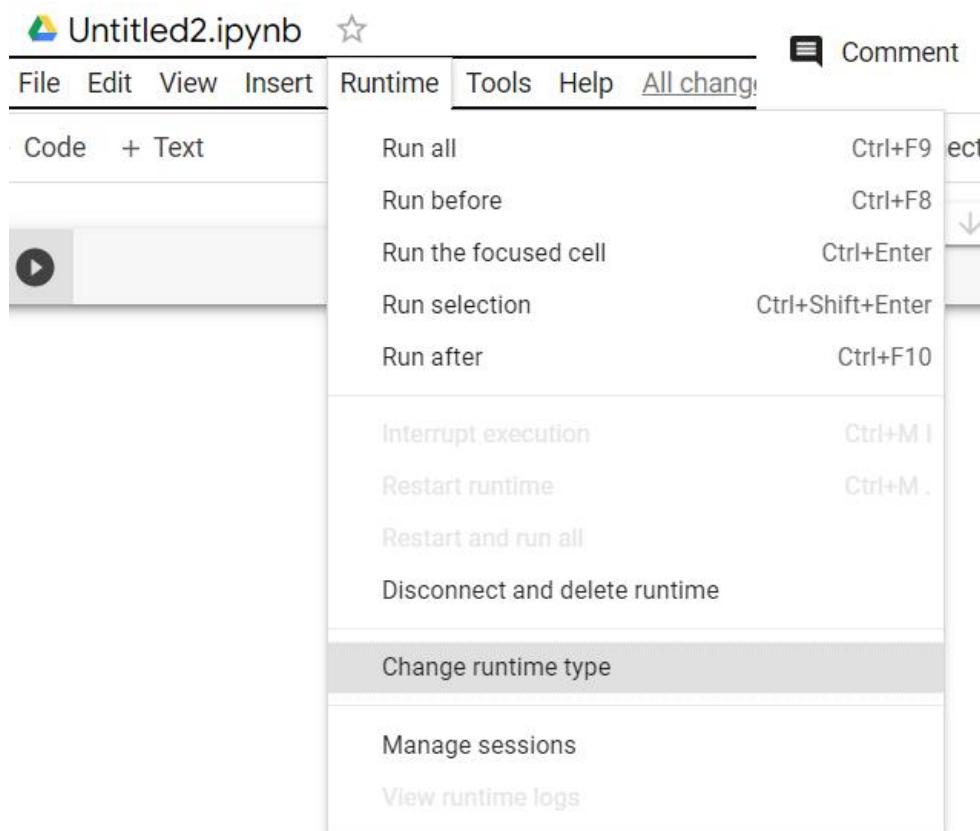
Bước 1: Mở và kết nối với Google Colab

Để bắt đầu train model, mở Colab lên bằng địa chỉ chính thức của Colab <https://colab.research.google.com/> và tạo một Notebook để làm việc bằng cách vào file -> chọn New notebook.



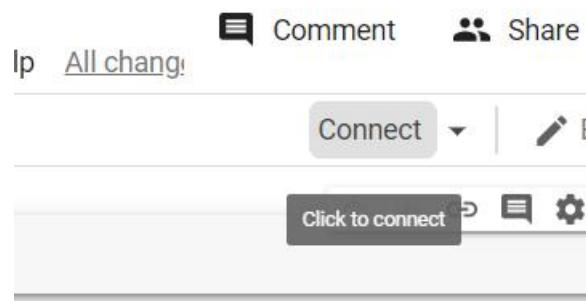


Sau đó vào Runtime -> Change runtime type để kết nối với GPU để quá trình train sẽ diễn ra nhanh hơn (nếu không kết nối với GPU thì khi chúng ta train sẽ train dựa trên CPU mà CPU nó chậm hơn GPU rất nhiều và train như vậy sẽ tốn rất nhiều thời gian của chúng ta).





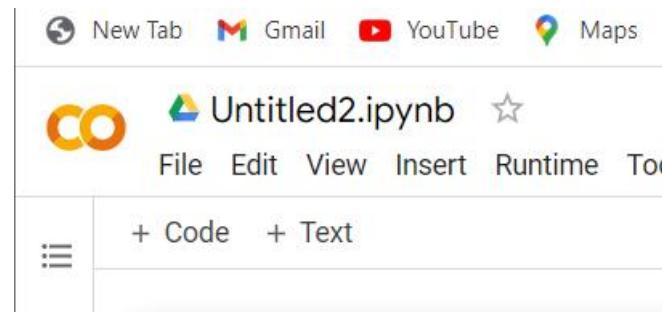
Sau khi chọn GPU thì ta bấm save để lưu lại và sau đó bấm kết nối để làm việc



Bước 2: Kết nối với Google Drive

Bởi vì sau 10 tiếng thì Colab sẽ kill, clear toàn bộ dữ liệu ở trên đó. Nên để đảm bảo an toàn thì cần chuẩn bị 1 tài khoản Google Drive để lưu lại.

Để tạo một Code Block mới bằng cách nhấn vào + Code



Thực hiện lệnh sau để kết nối với Google Drive

```
# Step 2. Mount drive
from google.colab import drive
drive.mount('/content/gdrive')
```

Chạy câu lệnh bằng cách nhấn mũi tên run

Hoặc có thể vào Runtime để chọn cách chạy trong Notebook.

Hoặc có thể nhấp vào biểu tượng Mount Drive để kết nối với Google Drive



Untitled2.ipynb

File Edit View Insert Runtime Tools Help All changes

Code + Text

Step 1. Mount drive

```
from google.colab import drive
drive.mount('/content/drive')
```

Run all Ctrl+F9

Run before Ctrl+F8

Run the focused cell Ctrl+Enter

Run selection Ctrl+Shift+Enter

Run after Ctrl+F10

Interrupt execution Ctrl+M I

Restart runtime Ctrl+M .

Restart and run all

Disconnect and delete runtime

Change runtime type

Manage sessions

View runtime logs

Bước 3: Tải mã nguồn YOLOv4 về Drive

```
# Step 3. Tai ma nguon YOLO ve drive
!rm -rf darknet
%cd /content/gdrive/MyDrive
!git clone https://github.com/AlexeyAB/darknet
%cd /content/gdrive/MyDrive/darknet
!rm -rf data
!mkdir data
```

Sau khi tải mã nguồn xong kiểm tra GPU bằng lệnh

```
!nvidia-smi
```

Fri May 27 14:45:26 2022

NVIDIA-SMI 460.32.03		Driver Version: 460.32.03		CUDA Version: 11.2	
GPU	Name	Persistence-M	Bus-Id	Disp.A	Volatile Uncorr. ECC
Fan	Temp	Perf	Pwr:Usage/Cap	Memory-Usage	GPU-Util Compute M.
					MIG M.
0	Tesla T4	Off	00000000:00:04.0	Off	0
N/A	42C	P8	9W / 70W	0MiB / 15109MiB	0% Default N/A

Processes:

GPU ID	GI ID	CI ID	PID	Type	Process name	GPU Memory Usage
No running processes found						



Nhập lệnh sau để thay đổi tham số

```
[ ] %env compute_capability=75
```

- Nếu Tesla K80 thì giá trị compute_capability=30
- Nếu Testla P100 thì giá trị compute_capability=60
- Nếu Tesla T4 thì giá trin compute_capability=75

Ở đây Tesla T4 nên nhóm sẽ dùng giá trị
compute_capability=75

Bước 4: Chuẩn bị file config

Truy cập vào darknet -> cfg -> yolov4-tiny-custom.cfg

- Xác định số class cần train (nhóm sử dụng 3 class)
- Ở dòng 6, sửa batch = 32
- Ở dòng 7, sửa subdivisions = 8
- Ở 21, sửa max_batches = max(<số class>*2000,6000). Nghĩa là nếu <số class>*2 mà nhỏ hơn 6000 thì lấy 6000 và ngược lại. Ở đây nhóm lấy 6000 vì 3*2000 = 6000 . Sửa max_batches=6000.
- Ở 22 sửa thành steps=80%, 90% của max_batches. Ở đây là steps=4800,5400.
- Replace toàn bộ các dòng có “classes=80” thành “classes=<số class>”. Ở đây nhóm sửa thành “classes=3”.
- Replace toàn bộ các dòng có “filters=255” thành “filters=<(số class+5)*3>”. Ở đây số class = 3 thì sẽ sửa thành (3+5)*3 = 24, có nghĩa là filters=24.

Bước 5: Chuẩn bị Makefile

```
[ ] # Step 5. Make file
%cd /content/gdrive/MyDrive/darknet
!sed -i 's/OPENCV=0/OPENCV=1/g' Makefile
!sed -i 's/GPU=0/GPU=1/g' Makefile
!sed -i 's/CUDNN=0/CUDNN=1/g' Makefile
!sed -i "s/ARCH= -gencode arch=compute_60,code=sm_60/ARCH= -gencode arch=compute_${compute_capability},code=sm_${compute_capability}/g" Makefile
!make
```

Bước 6: Tải Pretrain Weights

```
# Step 6. Download pretrain weight
%cd /content/gdrive/MyDrive/darknet
!wget https://github.com/AlexeyAB/darknet/releases/download/darknet_yolo_v4_pre/yolov4-tiny.weights
!wget https://github.com/AlexeyAB/darknet/releases/download/darknet_yolo_v4_pre/yolov4-tiny.conv.29
```



Bước 7: Upload bộ dataset từ máy lên mục data của darknet ở Google Drive và giải nén

```
▶ # Step 7. Giải nén file data  
%cd /content/gdrive/MyDrive/darknet/data  
!unzip data.zip
```

Bước 8: Tạo file yolo.names chứa tên các class

```
▶ # Step 8. Tạo file yolo.names  
%cd /content/gdrive/MyDrive/darknet  
  
!echo "mask_weared_incorrect" >> yolo.names  
!echo "with_mask" >> yolo.names  
!echo "without_mask" >> yolo.names
```

Bước 9: Tạo hai file train.txt và val.txt chứa danh sách các file ảnh

Ý nghĩa của 2 file này như sau:

- Chỉ cho YOLO biết ảnh nào dùng để train và ảnh nào dùng để val
- File train.txt chứa danh sách các file sẽ dùng để train
- File val.txt chứa danh sách các file sẽ dùng để val
- Danh sách cần chọn ngẫu nhiên đảm bảo tính phân phối dữ liệu.



```
# Step 9. Tạo file train.txt và val.txt
%cd /content/gdrive/MyDrive/darknet

import glob2
import math
import os
import numpy as np

files = []
for ext in ["*.png", "*.jpeg", "*.jpg"]:
    image_files = glob2.glob(os.path.join("data/data/", ext))
    files += image_files
print(files)
print(len(files))
nb_val = math.floor(len(files)*0.2)
rand_idx = np.random.randint(0, len(files), nb_val)
# print(len(nb_val))
print(len(rand_idx))
# Tạo file train.txt
with open("train.txt", "w") as f:
    for idx in np.arange(len(files)):
        # if (os.path.exists(files[idx][:-3] + "txt")):
        f.write(files[idx]+\n)

# Tạo file vali.txt
with open("val.txt", "w") as f:
    for idx in np.arange(len(files)):
        if (idx in rand_idx):
            f.write(files[idx]+\n)
```

Bước 10: Tạo file yolo.data chứa tham số train

```
# Step 10. Tạo file yolo.data
%cd /content/gdrive/MyDrive/darknet
!mkdir backup
!echo classes=3 > yolo.data
!echo train=train.txt >> yolo.data
!echo valid=val.txt >> yolo.data
!echo names=yolo.names >> yolo.data
!echo backup=backup >> yolo.data
```

Bước 11: Biên dịch mã nguồn darknet

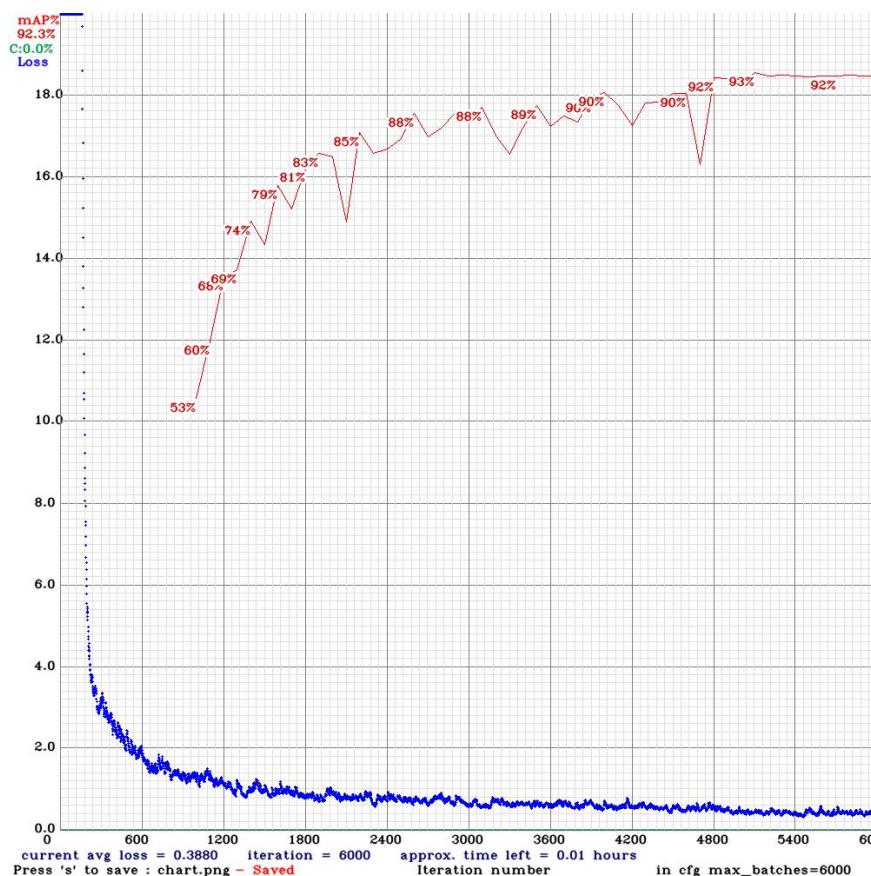
```
# Step 11. Make darknet
%cd /content/gdrive/MyDrive/darknet
!rm darknet
!make
```



Bước 12: Train

```
# Step 12. Train  
%cd /content/gdrive/MyDrive/darknet/  
!./darknet detector train yolo.data cfg/yolov4-tiny-custom.cfg yolov4-tiny.conv.29 -dont_show -map
```

Kết quả train



Nhìn chung mô hình khá tốt khi train 6000 vòng với bộ dataset đầu tiên với mAP = 93% và avg loss = 0.3880

Train model lần 2

Nhóm sẽ dùng model đã được train lần 1 để train model cho lần 2

Lần lượt thực hiện lại các bước 4,7,9,10,11,12 ở lần train thứ nhất nhưng có một chút thay đổi ở bước 4 và bước 12.

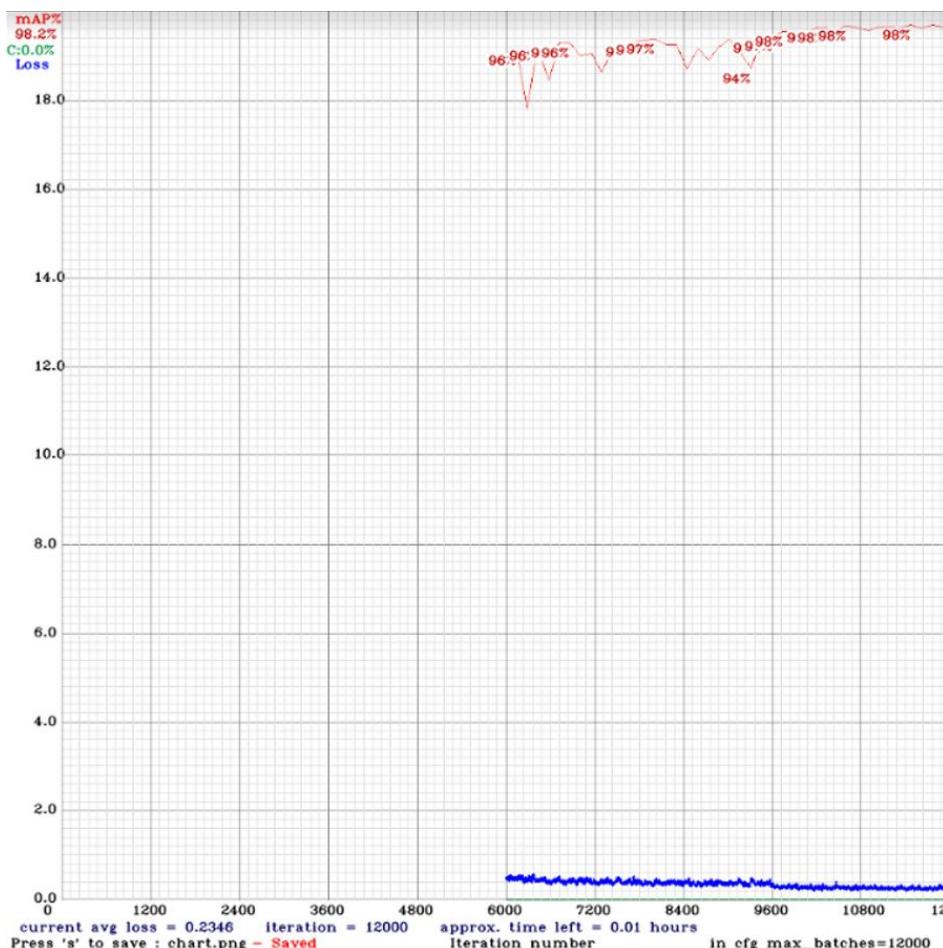
Bước 4: Sửa max_batches = 12000 sau đó cập nhật lại giá trị steps

Bước 12: Sử dụng file yolov4-tiny-custom_final.weights của lần train thứ nhất để train



```
▶ # Step 12. Train  
%cd /content/gdrive/MyDrive/darknet/  
!./darknet detector train yolo.data cfg/yolov4-tiny-custom.cfg backup/yolov4-tiny-custom_final.weights -dont_show -map
```

Kết quả train



Sau khi train tiếp 6000 vòng dựa trên mô hình được train lần đầu tiên với bộ dataset được thêm một số hình ảnh lấy tay che mặt, nhận thấy mô hình sau khi train rất tốt với mAP lên tới 98.2% và avg loss = 0.2346

Train model lần 3

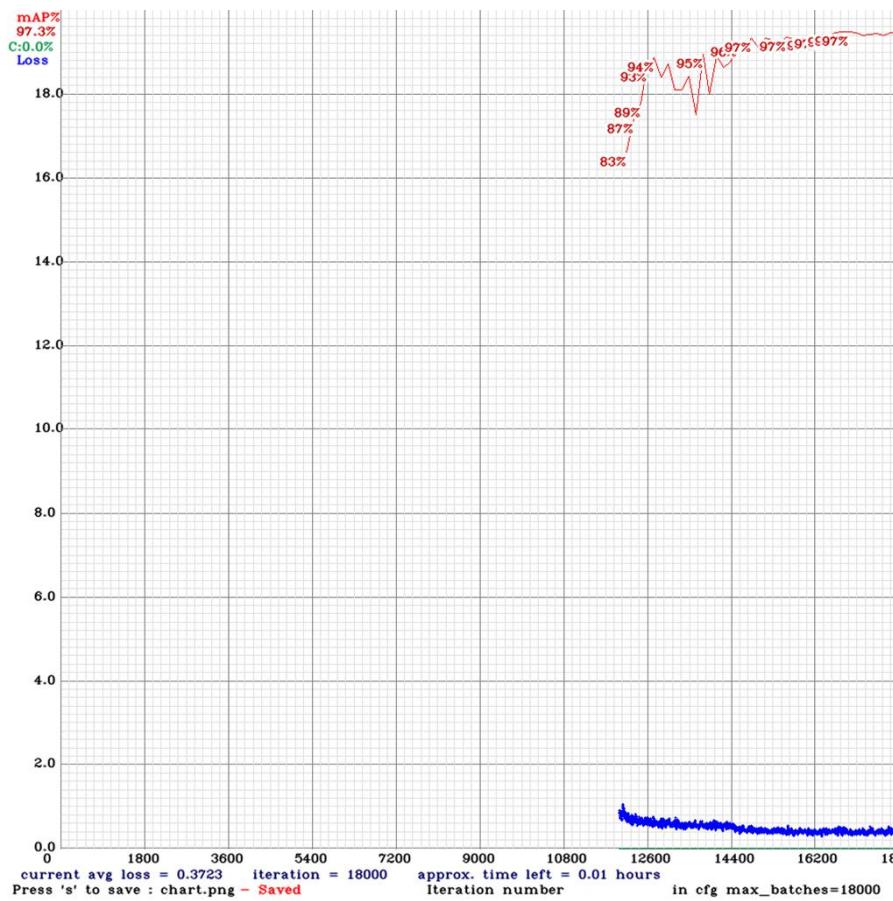
Nhóm sẽ dùng model đã được train lần 2 để train model cho lần 3

Lần lượt thực hiện các bước 4,7,9,10,11,12 như ở lần train thứ 2 và chỉ cần thay đổi một chút ở bước 4

Bước 4: Sửa max_batches = 18000 sau đó cập nhật lại giá trị steps



Kết quả train



Sau khi train thêm 6000 nghìn vòng với bộ dataset bổ sung thêm một số ảnh đeo khẩu trang sai dựa trên mô hình được train lần thứ 2. Chỉ số mAP giảm so với lần train thứ 2, tuy vậy nó cũng rất cao, cụ thể là 97.3%. Ngoài chỉ số mAP giảm so với lần train thứ 2 thì chỉ số avg loss = 0.3723 cũng tăng so với lần train thứ 2.

Tuy kết quả của mỗi lần train là như vậy nhưng khi cho thử nhận diện trên bộ dữ liệu test với những hình ảnh hoàn toàn mới và không có trong bộ dataset ở lần train còn là một ẩn số. Có thể mô hình với chỉ số mAP cao hơn và avg loss thấp hơn nhưng khi nhận diện trên một ảnh hoàn toàn mới thì chưa chắc nhận diện đúng hơn so với mô hình có chỉ số mAP thấp hơn và avg loss cao hơn.



YOLOv5

Chuẩn bị dữ liệu

Cũng như ở YOLOv4-Tiny, để train được một mô hình bằng để phát hiện đám người có hoặc không đeo khẩu trai thì dữ liệu ta cần đó là hình ảnh nhìn thấy rõ mặt người để có thể nhìn vào và biết người đó có hay không đeo khẩu trang hay là có đeo nhưng lại đeo sai cách.

Tải dữ liệu đã được chuyển sang format YOLO về máy.

Train

Bước 1,2 thực hiện tương tự ở quá trình train mô hình YOLOv4-Tiny.

Bước 3: Tạo folder có tên Detect Mask YoloV5

```
[ ] %cd /content/drive/MyDrive/Detect Mask YoloV5
```

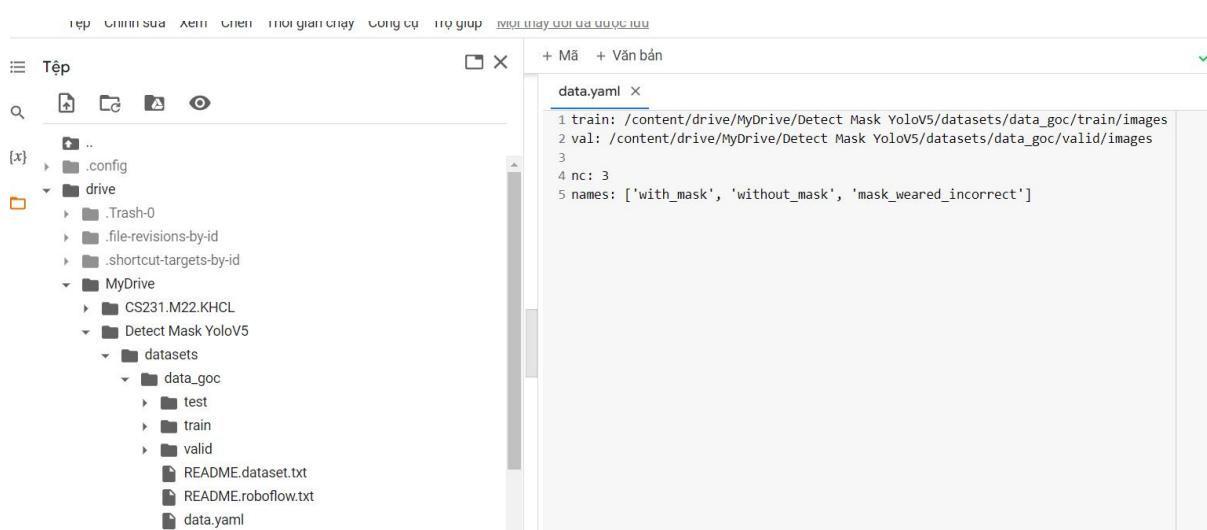
```
/content/drive/MyDrive/Detect Mask YoloV5
```

Bước 4: Tải repo yolov5 về và đưa vào Google Drive sau đó cài đặt các thư viện trong requirements

```
[ ] !pip install -r yolov5/requirements.txt
```

Train lần 1:

Bước 5: Upload dataset và chuẩn bị file data.yaml chứa tên các class cần train

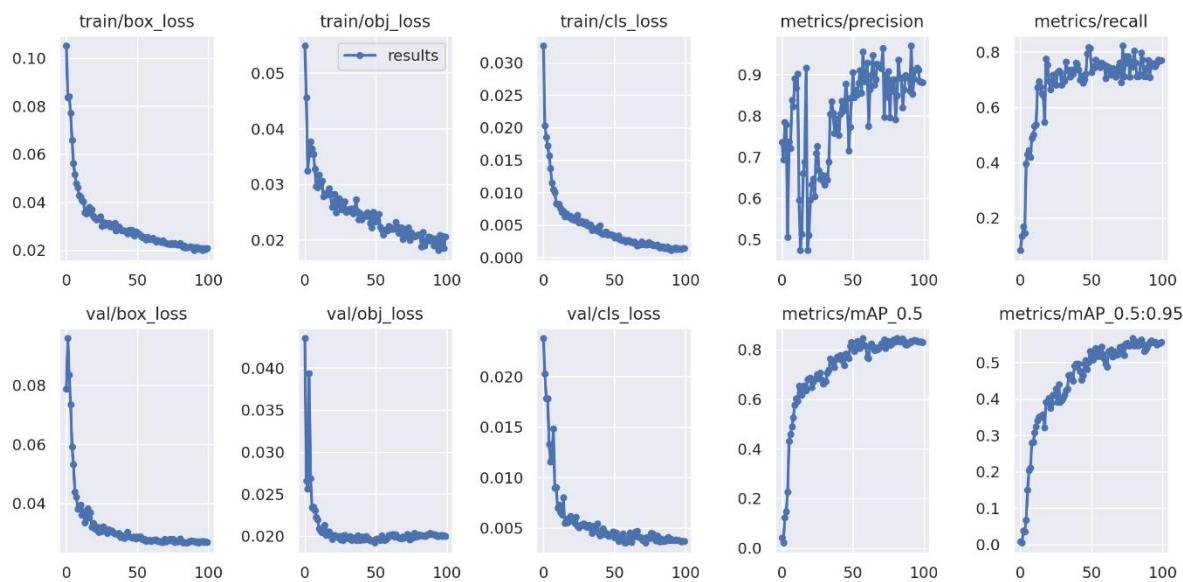


Bước 6: Train



```
[ ] !python train.py --img 640 --batch 16 --epochs 100 --data ./datasets/data_goc/data.yaml --weights yolov5s.pt  
--workers 0 --device 0
```

Kết quả:



Sau khi train với mô hình YOLOv5s với 100 epochs thì nhóm có sơ đồ như sau:

- train/box_loss và val/box_loss: biểu hiện thuật toán có thể xác định tâm của vật thể cũng như dự đoán bounding box bao phủ vật thể (train thì của bộ train, val thì của bộ valid tại vì lúc train model mình sử dụng 2 bộ đó)
- train/obj_loss và val/obj_loss: cơ bản là thước đo xác suất sự tồn tại của một vật thể trong một khu vực quan tâm được đề xuất
- train/cls_loss và val/cls_loss: độ lỗi của việc dự đoán loại nhãn của object, hàm lỗi này chỉ tính trên những ô vuông có xuất hiện object, còn những ô vuông khác ta không quan tâm, hàm loss khi dự đoán class
- Precision: Dự đoán đo lường mức độ chính xác là dự đoán của mô hình tức là tỷ lệ phần trăm dự đoán của mô hình là chính xác.
- “Recall” đo lường như thế nào tốt mô hình tìm thấy tất cả các mẫu tích cực. Ví dụ: chúng ta có thể tìm thấy 80% các trường hợp tích cực có thể có trong các dự đoán K hàng đầu của mô hình.
- mAP_0.5 (mean Average Precision): là độ chính xác trung bình tại ngưỡng 0.5 (ví dụ mAP_0.5 = 0.7 thì tại ngưỡng IoU = 0.5 có AP bằng 0.7 (Average Precision))



- mAP_0.5:0.95: nó cũng là độ chính xác trung bình nhưng nó ở ngưỡng từ 0.5 -> 0.95 chứ không ở một ngưỡng như mAP_0.5

Theo nhóm được biết, đồ thị càng mượt thì càng tốt cho nên bằng mắt thường ta có thể thấy loss, mAP và recall tốt, nhưng precision thì đồ thị gãy khúc quá nhiều cho nên, theo phỏng đoán của nhóm thì precision sẽ không được tốt cho lắm (nhóm chỉ dự đoán mô hình tốt hay xấu dựa trên sự biến động của những biểu đồ của các giá trị, đồng thời nhóm chỉ tập trung vào đồ thị của hai giá trị precision và recall bởi vì nhóm đánh giá mô hình dựa trên kết quả của hai giá trị đó ở lần test).

Train lần 2:

Bước 5: Upload dataset và chuẩn bị file data.yaml chứa tên các class cần train

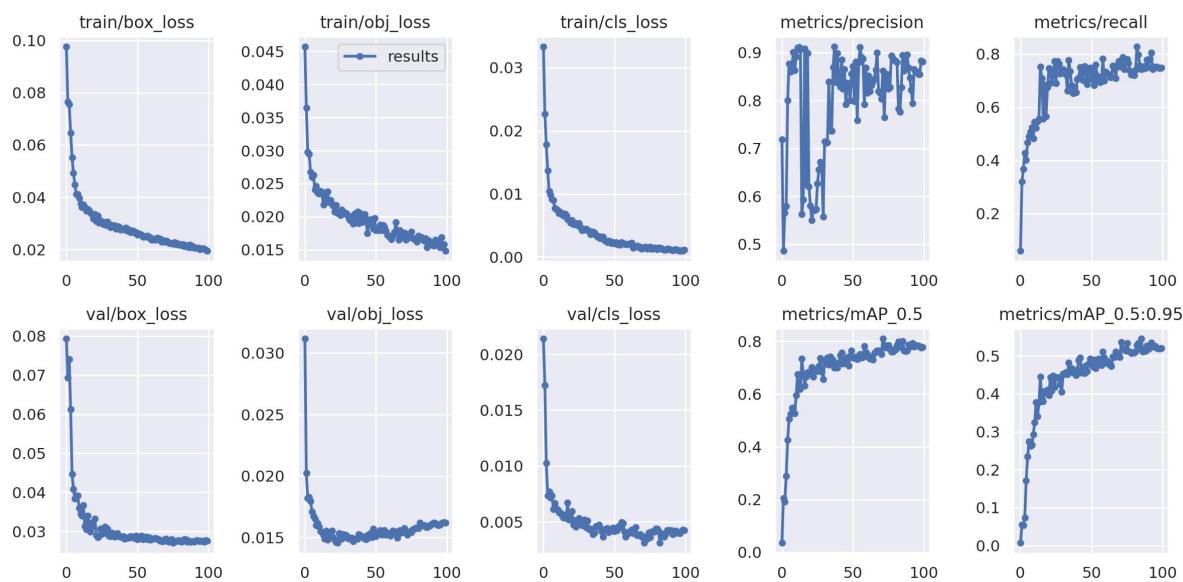
A screenshot of a file explorer window. The left sidebar shows a tree view of files and folders. In the center, there is a text editor window titled 'data.yaml'. The code in the editor is as follows:

```
1 train: /content/drive/MyDrive/Detect Mask Yolov5/datasets/data_new/train/images
2 val: /content/drive/MyDrive/Detect Mask Yolov5/datasets/data_new/valid/images
3
4 nc: 3
5 names: ['mask_weared_incorrect', 'with_mask', 'without_mask']
```

Bước 6: Train

```
[ ] !python train.py --img 640 --batch 16 --epochs 100 --data ../datasets/data_new/data.yaml --weights yolov5s.pt --workers 0 --device 0
```

Kết quả:



Sau khi train ở lần thứ hai, nhìn một cách tổng quan, đồ thị của precision và recall tương đương và không chênh lệch nhau mấy, đồ thị của những giá trị còn lại cũng mượt và sự biến động trong biểu đồ ở mỗi giá trị là tương đương với lần thứ nhất.

Train lần 3:

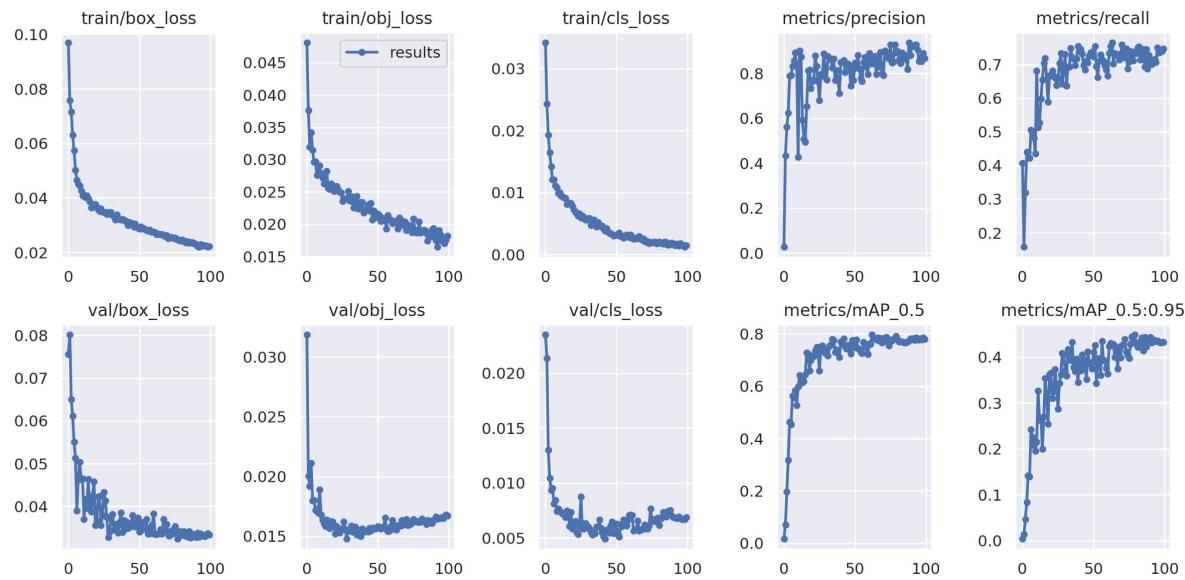
Bước 5: Upload dataset và chuẩn bị file data.yaml chứa tên các class cần train

```
data.yaml
1 names:
2 - mask_weared_incorrect
3 - with_mask
4 - without_mask
5 nc: 3
6 train: /content/drive/MyDrive/Detect Mask YoloV5/datasets/data_new_v2-1/train/images
7 val: /content/drive/MyDrive/Detect Mask YoloV5/datasets/data_new_v2-1/valid/images
8
```

Bước 6: Train

```
[ ] !python train.py --img 640 --batch 16 --epochs 100 --data ./datasets/data_new_v2-1/data.yaml --weights yolov5s.pt --workers 0 --device 0
```

Kết quả:



Sau khi train lần thứ ba, những biểu đồ của các giá trị ngoài precision có một số biểu đồ biến động nhiều hơn so với hai lần trước và có thể là ở epochs thứ 100 giảm ít hơn hoặc là tăng ít hơn. Nhưng ở biểu đồ của precision, ta có thể rõ ràng nhìn thấy tốt hơn so với hai lần trước (ít biến động hơn đồng thời kết quả qua các epochs cũng cao hơn).

Tuy kết quả của mỗi lần train là như vậy nhưng khi cho thử nhận diện trên bộ dữ liệu test với những hình ảnh hoàn toàn mới và không có trong bộ dataset ở lần train còn là một ẩn số. Có thể mô hình nhìn có vẻ tốt hơn nhưng khi nhận diện trên một ảnh hoàn toàn mới thì chưa chắc nhận diện đúng hơn so với mô hình bằng mắt thường chúng ta kết luận nó xấu hơn.



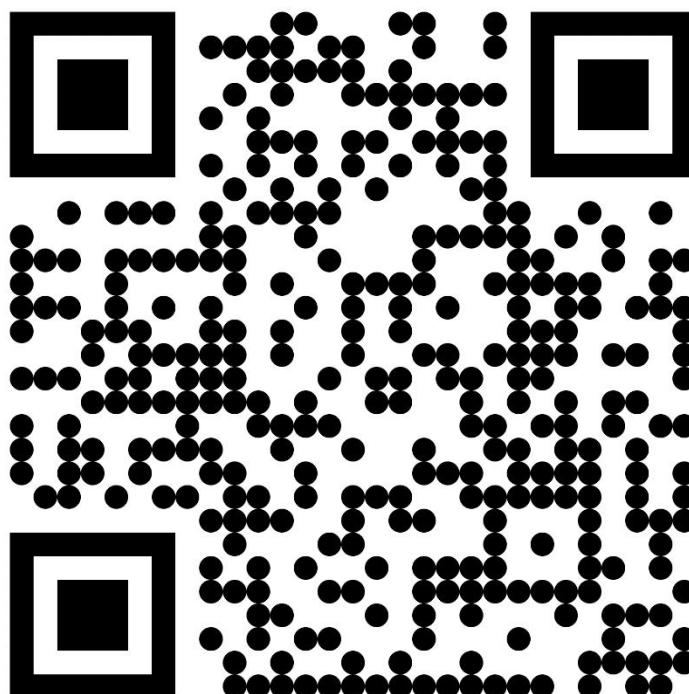
4. Test và đánh giá mô hình

Bộ dữ liệu test

Bộ dữ liệu test (data test) gồm 120 ảnh được nhóm trích từ file test trong bộ dataset đồng thời được tải từ những nguồn khác như google. Những hình ảnh được trích ngoài bộ dataset được gán nhãn bằng Roboflow theo định dạng ở bộ dataset.

Gồm có 179 khuôn mặt được gán nhãn:

- 94 with_mask
- 48 without_mask
- 37 mask_weared_incorrect



Mã QR truy cập vào những hình ảnh trong bộ data test được trích xuất ngoài bộ test của dataset được gán nhãn bằng Roboflow

Cách đánh giá mô hình

Mô hình được đánh giá bằng 3 thông số đó là precision, recall và F1-score. Đồng thời nhóm chỉ đánh giá dựa trên loại nhãn dự đoán được trên mỗi khuôn mặt chứ không phụ thuộc vào chỉ số score hay còn gọi là độ tin cậy của nhãn được dự đoán.



$$Precision = \frac{TP}{TP + FP} = \frac{\text{Số dự đoán chính xác}}{\text{Tổng số lần dự đoán}}$$
$$Recall = \frac{TP}{TP + FN} = \frac{\text{Số lần dự đoán chính xác}}{\text{Số lần nhận dạng đúng có thể có}}$$

$$F1 = 2 \frac{Precision * Recall}{Precision + Recall}$$

Công thức tính

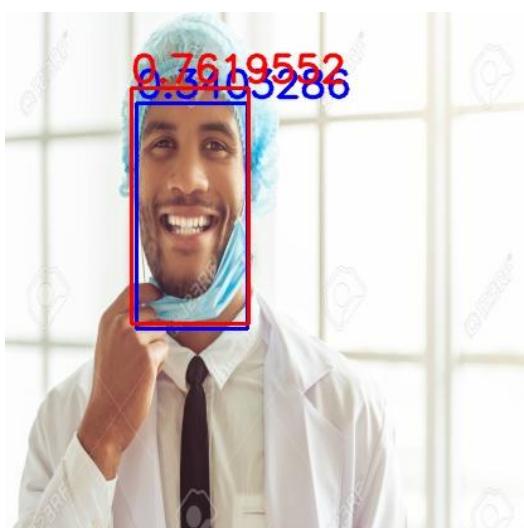
Cách xác định TP, FP, FN

- TP: Tổng số khuôn mặt được dự đoán đúng nhãn
- FP: Tổng số khuôn mặt được dự đoán sai nhãn
- FN: Tổng số khuôn mặt không được dự đoán được nhãn

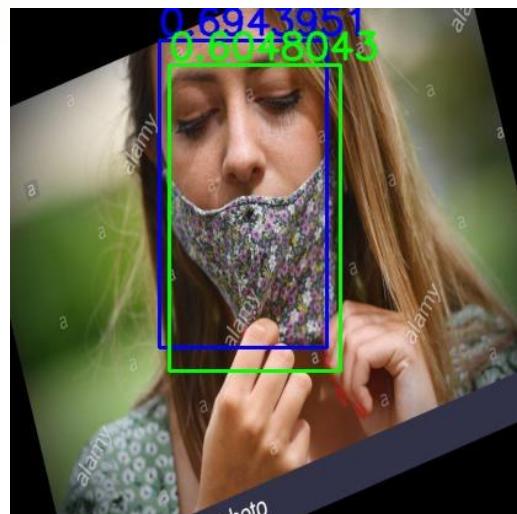
Vấn đề gặp phải

Vì là một mô hình máy học nên không thể tránh khỏi những sai sót trong quá trình dự đoán. Điểm hình ở bài này, ở một số khuôn mặt mô hình vẽ nhiều hơn một bounding box ở một khuôn mặt dẫn đến có thể xảy ra trường hợp có nhiều hơn một class được dự đoán ở khuôn mặt đó. Dưới đây là một số hình ảnh được trích xuất từ kết quả đánh giá của mô hình YOLOv4-Tiny (sử dụng Open CV để dự đoán).

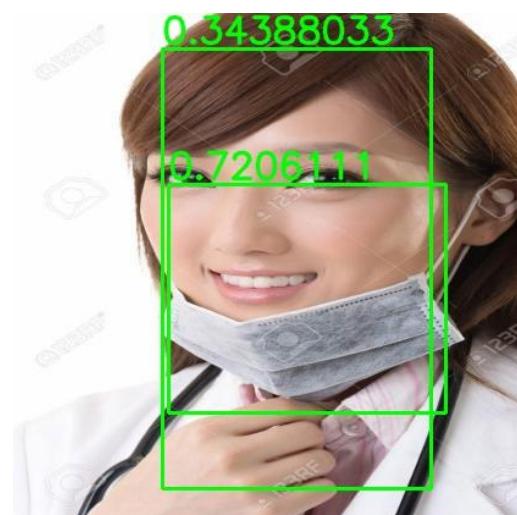
- **mask_weared_incorrect**
- **with_mask**
- **without_mask**



Hình 1



Hình 2



Hình 3

Ở hình 1 mô hình vẽ ra 2 bounding box và được dự đoán thành 2 class đó là mask_weared_incorrect và without_mask.

Ở hình 2 mô hình vẽ ra 2 bounding box và được dự đoán thành 2 class đó là with_mask và without_mask.

Ở hình 3 mô hình vẽ ra 2 bounding box nhưng cả 2 bounding box đều được dự đoán là class with_mask.

Các trường hợp lúc thực nghiệm, mô hình vẽ được nhiều hơn 1 bounding box đồng thời dự đoán nhiều hơn 1 class (các trường hợp như hình 1 và hình 2) thì nhóm sẽ cho những trường hợp đó là FN.

Các trường hợp mà lúc thực nghiệm, mô hình vẽ được nhiều hơn 1 bounding box nhưng được dự đoán chỉ 1 class (các trường hợp như ở hình 2) thì nhóm sẽ cho những trường hợp đó là TP.



YOLOv4-Tiny

Kết quả bên dưới được nhóm thực nghiệm dự đoán trên từng hình ảnh và sau đó đếm số lượng khuôn mặt ở các class và tính các giá trị precision, recall, F1-score bằng một cách thủ công chứ không dùng bất kỳ thuật toán nào.

Kết quả đánh giá mô hình với lần train thứ nhất

	mask_weared_incorrect	with_mask	without_mask	Tổng	Không dự đoán được
Đúng	18	87	20	125	35
Sai	3	13	3	19	
Không có trong data	0	8	4	12	
Tổng	21	108	27	156	

Hình 1

TP	125
FP	19
FN	35

Hình 2

Precision	0.86806
Recall	0.78125
F1-score	0.82237

Hình 3

Ta có thể dễ dàng nhận thấy mặc dù có tới 35 khuôn mặt không dự đoán được người đó có hay không đeo khẩu trang hay là có đeo nhưng mà đeo sai cách nhưng ở class with_mask mô hình dự đoán tới 100 khuôn mặt (đúng 87 và sai 13, số lượng khuôn mặt dự đoán sai ở class này chiếm đa số trong tổng) trong lúc chỉ có 94 khuôn mặt được gán nhãn with_mask ở bộ data test. Rõ ràng đó là một con số thể hiện sự chênh lệch rất lớn khi dự đoán. Từ đó dẫn đến 2 class còn lại số lượng khuôn mặt được dự đoán trên mỗi class thấp hơn nhiều so với số khuôn mặt được gán nhãn ở mỗi class trong bộ data test.

Lần train này giá trị precision cũng khá cao khi xấp xỉ 87%, nhưng recall lại khá thấp cụ thể là chỉ 78.1%. Như vậy nó cũng đã bù trừ cho nhau nên kết quả của F1-score ở đây đạt được là 82.2%.

Ở đây theo nhóm nhận thấy thì số khuôn mặt được dự đoán ở class without_mask chênh lệch với số khuôn mặt có trong bộ data test là nhiều nhất nên nhóm sẽ tiến hành kiểm tra ở file được tổng hợp sau



quá trình dự đoán những hình ảnh ở bộ data test nhằm mục đích xem trường hợp nào của class without_mask là bị dự đoán sai nhiều nhất. Sau quá trình kiểm tra, nhóm nhận thấy trường hợp lấy tay che mặt được dự đoán sai nhiều nhất, tất cả các class with_mask, without_mask và mask_weared_incorrect đều có các trường hợp lấy tay che mặt ở trong đó.



Trường hợp lấy tay che mặt được dự đoán sai

Mong muốn của nhóm là trường hợp lấy tay che mặt sẽ được mô hình dự đoán thành class without_mask. Vì vậy nhóm đã đề xuất lấy thêm một số hình ảnh lấy tay che mặt, được gán nhãn without_mask và trộn vào với file dataset ở lần train thứ nhất để thực hiện quá trình train lần thứ hai.

Kết quả đánh giá mô hình với lần train thứ hai

Tiến hành kiểm tra lại các trường hợp lấy tay che mặt ở trên.





Sau khi train lần thứ hai thì kết quả dự đoán các trường hợp lấy tay che mặt đã đúng với mong muốn của nhóm.

Dưới đây là kết quả đánh giá khả năng dự đoán của mô hình sau lần train thứ hai

	mask_weared_incorrect	with_mask	without_mask	Tổng	Không dự đoán được
Đúng	11	86	34	131	25
Sai	0	4	19	23	
Không có trong data	0	9	4	13	
Tổng	11	99	57	167	

Hình 1

TP	131
FP	23
FN	25

Hình 2

Precision	0.85065
Recall	0.83974
F1-score	0.84516

Hình 3

Sau khi đã thêm vào bộ dataset những hình ảnh lấy tay che mặt với class được gán nhãn là without_mask. Lần này số lượng khuôn mặt ở class without_mask đã tăng lên rất nhiều, cụ thể là có 53 khuôn mặt được dự đoán là nhãn without_mask, chỉ số đó vẫn khá cao so với 48 khuôn mặt với class without_mask ở bộ data test trong lúc có tới 25 khuôn mặt không dự đoán được, với sự tăng số lượng khuôn mặt dự đoán được thì đồng thời số lượng khuôn mặt dự đoán cũng sẽ tăng theo (từ sai 3 trở thành sai 19). Còn với class with_mask ở lần này đã giảm xuống so với lần thứ nhất, số khuôn mặt dự đoán đúng chỉ giảm xuống 1, đồng thời số lượng khuôn mặt dự đoán sai đã giảm xuống chỉ còn 4. Nhưng ở lần này số lượng khuôn mặt được dự đoán là nhãn mask_weared_incorrect cũng đã bị giảm sút dẫn đến chênh lệch rất nhiều so với con số 37 ở bộ data test.

Lần này thì giá trị precision thấp hơn so với lần đầu tiên (85.1% so với 87%). Tuy nhiên giá trị recall lại cao hơn so với lần thứ nhất không ít (84% so với 78.1%). Vì vậy giá trị F1-score của lần này đã tăng lên so với kết quả dự đoán của mô hình được train lần đầu tiên.



Như đã được nhận xét ở trên, số lượng khuôn mặt được dự đoán với class là mask_weared_incorrect là chênh lệch rất lớn so với số lượng khuôn mặt của cùng class đó ở trong bộ data test. Vì vậy nhóm đã tự đề xuất trộn thêm một số hình ảnh được gán nhãn mask_weared_incorrect sau đó thực hiện quá trình train mô hình lần thứ ba.

Kết quả đánh giá mô hình với lần train thứ ba

	mask_weared_incorrect	with_mask	without_mask	Tổng	Không dự đoán được
Đúng	18	86	36	140	23
Sai	1	1	14	16	
Không có trong data	0	12	4	16	
Tổng	19	99	54	172	

Hình 1

TP	140
FP	16
FN	23

Hình 2

Precision	0.89744
Recall	0.85890
F1-score	0.87774

Hình 3

Sau khi train lần thứ ba, kết quả dự đoán ở class mask_weared_incorrect mặc dù là còn thấp hơn so với cùng class đó ở trong bộ data test nhưng cũng đã có một kết quả khả quan hơn so với lần train thứ hai. Ở class with_mask số lượng khuôn mặt dự đoán đúng vẫn giữ nguyên nhưng số lượng khuôn mặt dự đoán sai lại giảm nên dẫn đến việc số lượng khuôn mặt được dự đoán thành class giảm, nhưng điều đó là không quan trọng khi số lượng khuôn mặt dự đoán đúng không đổi. Còn về class without_mask số lượng khuôn mặt dự đoán đúng tăng và số lượng khuôn mặt dự đoán sai giảm.

Kết quả các giá trị của precision, recall và F1-score đều tăng so với những lần trước.



So sánh kết quả đánh giá của mô hình qua 3 lần train

	mask_weared_incorrect	with_mask	without_mask	Tổng	Không dự đoán được
Đúng	18	87	20	125	35
Sai	3	13	3	19	
Không có trong data	0	8	4	12	
Tổng	21	108	27	156	
	mask_weared_incorrect	with_mask	without_mask	Tổng	Không dự đoán được
Đúng	11	86	34	131	25
Sai	0	4	19	23	
Không có trong data	0	9	4	13	
Tổng	11	99	57	167	
	mask_weared_incorrect	with_mask	without_mask	Tổng	Không dự đoán được
Đúng	18	86	36	140	23
Sai	1	1	14	16	
Không có trong data	0	12	4	16	
Tổng	19	99	54	172	

Precision	0.86806
Recall	0.78125
F1-score	0.82237

Precision	0.85065
Recall	0.83974
F1-score	0.84516

Precision	0.89744
Recall	0.85890
F1-score	0.87774

Thông qua kết quả đánh giá ở trên nhóm kết luận được yếu tố đầy đủ và cân bằng của bộ dataset dùng để train mô hình là rất quan trọng. Bộ dataset càng đầy đủ và cân bằng thì sau khi train mô hình dự đoán càng tốt, bằng chứng cụ thể là giá trị precision của lần thứ hai giảm so với lần thứ nhất nhưng ở lần thứ ba thì giá trị precision cải thiện hơn nhiều so với hai lần train đầu tiên (nguyên nhân do mất cân bằng dữ liệu), F1-score tăng dần từ mô hình thứ nhất đến mô hình thứ ba. Khi dùng mô hình đã được train để train cho lần tiếp theo thì số lượng khuôn mặt được dự đoán có xu hướng tăng ($156 \rightarrow 167 \rightarrow 172$), đi đôi với sự tăng của số lượng khuôn mặt được dự đoán thì số lượng khuôn mặt mà không dự đoán được giảm và số lượng khuôn mặt được dự đoán nhưng lại không có trong bộ data test tăng và số lượng khuôn mặt không dự đoán được lại giảm (số lượng khuôn mặt không dự đoán được cũng là một trong những nguyên nhân làm tăng giá trị recall sau 3 lần đánh giá). Điều này chứng tỏ một điều rằng khi dựa trên mô hình đã được train để train cho lần tiếp theo với một bộ dataset chi tiết hơn thì mô hình dự đoán càng chi tiết (số lượng khuôn mặt được dự đoán tăng).



YOLOv5

Cũng như ở YOLOv4-Tiny, ngoài việc dự đoán thì tất cả các phần còn lại đều được thực hiện thủ công và không áp dụng bất cứ thuật toán nào.

Kết quả đánh giá mô hình với lần train thứ nhất

	mask_weared_incorrect	with_mask	without_mask	Tổng	Không dự đoán được
Đúng	16	88	26	130	26
Sai	2	17	4	23	
Không có trong data	1	8	2	11	
Tổng	19	113	32	164	

Hình 1

TP	130
FP	23
FN	26

Hình 2

Precision	0.84967
Recall	0.83333
F1-score	0.84142

Hình 3

Cũng giống như YOLOv4-Tiny, số lượng khuôn mặt được dự đoán ở class with_mask rất nhiều (105 khuôn mặt được dự đoán, số lượng khuôn mặt được dự đoán sai ở class này chiếm đa số trong tổng), vượt quá số lượng khuôn mặt của class đó có trong bộ data test trong lúc còn có tới 26 khuôn mặt không dự đoán được. Điều đó dẫn đến số lượng khuôn mặt được dự đoán ở 2 class còn lại chênh lệch khá lớn so với số lượng khuôn mặt có trong bộ data test.

Giá trị precision và recall sau khi dự đoán của mô hình lần này cũng không chênh lệch nhau mấy (precision 85% và recall 83.3%). Cả 2 giá trị trên đều tốt và không chênh lệch nhau nhiều nên giá trị F1-score cũng tốt.



Nhóm tiến hành kiểm tra kết quả dự đoán được ở class without_mask thì điều tương tự như ở YOLOv4-Tiny đã xảy ra. Đó là các trường hợp lấy tay che mặt được dự đoán sai class. Dưới đây là một số hình ảnh chứng minh cho điều đó.



Với trường hợp lấy tay che mặt được dự đoán thành 3 class. Nhưng mong muốn của nhóm ở mô hình YOLOv5 cũng như ở YOLOv4-Tiny đó là các trường hợp lấy tay che mặt này sẽ được dự đoán thành nhãn without_mask. Vì vậy nhóm đã đề xuất lấy bộ dataset được dùng để train lần hai ở mô hình YOLOv4-Tiny để train lần hai cho mô hình YOLOv5.

Kết quả đánh giá mô hình với lần train thứ hai

Tiến hành kiểm tra lại các trường hợp lấy tay che mặt như trên.



Ở lần này mô hình đã dự đoán đúng class cho trường hợp lấy tay che mặt là without_mask, kết quả đó cũng đã đúng với mong muốn của nhóm đối với lần train này của mô hình.

Dưới đây là kết quả đánh giá khả năng dự đoán của mô hình sau lần train thứ hai

	mask_weared_incorrect	with_mask	without_mask	Tổng	Không dự đoán được
Đúng	6	87	36	129	24
Sai	0	2	24	26	
Không có trong data	0	5	4	9	
Tổng	6	94	64	164	

Hình 1



TP	129
FP	26
FN	24

Hình 2

Precision	0.83226
Recall	0.84314
F1-score	0.83766

Hình 3

Số lượng khuôn mặt được dự đoán ở clas with _mask đã giảm, đặc biệt mặc dù giảm như vậy nhưng số lượng khuôn mặt được dự đoán đúng gần như là không giảm mà thay vào đó số lượng khuôn mặt dự đoán sai giảm đi rất nhiều (từ con số 17 ở lần thứ nhất và lần này chỉ còn có 2). Bên cạnh đó số lượng khuôn mặt được dự đoán ở class without _mask cũng là class được thêm hình ảnh vào ở bộ dataset để train mô hình lần này tăng một cách chống mặt, tăng gấp đôi so với lần dự đoán trước ($30 \rightarrow 60$). Đương nhiên kết quả dự đoán tăng nhiều như vậy thì số lượng khuôn mặt được dự đoán sai cũng sẽ tăng theo và nó cũng chiếm số lượng áp đảo trong tổng số lượng khuôn mặt được dự đoán sai ($24/26$ khuôn mặt được dự đoán sai). Do số lượng khuôn mặt được dự đoán ở class đã tăng lên gấp đôi so với lần trước, chênh lệch lớn so với số lượng khuôn mặt của class đó có trong bộ data test, cộng thêm việc có 24 khuôn mặt không dự đoán được đã đến số lượng khuôn mặt được dự đoán ở class còn lại (mask_weared_incorrect) chỉ có 6 khuôn mặt ($6/37$ khuôn mặt có trong bộ data test). Điều này bắt buộc nhóm phải có phương pháp để cải thiện mô hình và thực hiện train mô hình lần thứ ba là cần thiết.

Giá trị precision và recall đều trên 80% cũng là một điều tốt nhưng precision lại giảm nhiều hơn so với tăng của recall khi so sánh với kết quả ở lần trước (precision: $85\% \rightarrow 83.2\%$; recall: $83.3\% \rightarrow 84\%$). Điều đó dẫn đến kết quả là giá trị F1-score giảm so với lần train đầu tiên.

Sử dụng bộ dataset được dùng để train lần thứ ba ở mô hình YOLOv4-Tiny để train lần thứ ba cho mô hình YOLOv5.



Kết quả đánh giá mô hình với lần train thứ ba

	mask_weared_incorrect	with_mask	without_mask	Tổng	Không dự đoán được
Đúng	28	82	31	141	26
Sai	0	4	8	12	
Không có trong data	0	4	2	6	
Tổng	28	90	41	159	

Hình 1

TP	141
FP	12
FN	26

Hình 2

Precision	0.92157
Recall	0.84431
F1-score	0.88125

Hình 3

Sau lần train thứ ba, kết quả dự đoán của mô hình đã đạt được theo mong muốn của nhóm. Cụ thể số lượng khuôn mặt được dự đoán ở class without_mask đã giảm đi rất nhiều (số lượng khuôn mặt được dự đoán sai cũng giảm đáng kể) đồng thời số lượng khuôn mặt được dự đoán ở class mask_weared_incorrect tăng lên cũng rất nhiều, nhưng không vì 2 điều đó mà ảnh hưởng nhiều đến số kết quả được dự đoán ở class with_mask (tuy lần này kết quả không được tốt như ở kết quả dự đoán của lần train thứ hai, nhưng nó cũng không chênh lệch nhau nhiều (giảm 5 khuôn mặt được dự đoán đúng và tăng 2 khuôn mặt được dự đoán sai)).

Giá trị precision lần này thực sự rất ấn tượng khi lớn hơn 90% (92,1%). Về giá trị recall của lần này gần như không tăng so với giá trị của lần train thứ hai (84.4% so với 84.3% như ở lần train thứ hai). Giá trị precision và recall đều là những giá trị rất khả quan nên giá trị F1-score lần này cũng rất khả quan và nó tốt hơn so với 2 lần train trước.



So sánh kết quả đánh giá của mô hình qua 3 lần train

	mask_weared_incorrect	with_mask	without_mask	Tổng	Không dự đoán được
Đúng	16	88	26	130	26
Sai	2	17	4	23	
Không có trong data	1	8	2	11	
Tổng	19	113	32	164	

Precision	0.84967
Recall	0.83333
F1-score	0.84142

	mask_weared_incorrect	with_mask	without_mask	Tổng	Không dự đoán được
Đúng	6	87	36	129	24
Sai	0	2	24	26	
Không có trong data	0	5	4	9	
Tổng	6	94	64	164	

Precision	0.83226
Recall	0.84314
F1-score	0.83766

	mask_weared_incorrect	with_mask	without_mask	Tổng	Không dự đoán được
Đúng	28	82	31	141	26
Sai	0	4	8	12	
Không có trong data	0	4	2	6	
Tổng	28	90	41	159	

Precision	0.92157
Recall	0.84431
F1-score	0.88125

Từ kết quả ở bảng trên một lần nữa nhóm kết luận được yếu tố đầy đủ và cân bằng của bộ dataset dùng để train mô hình là rất quan trọng. Bộ dataset càng đầy đủ và cân bằng thì sau khi train mô hình dự đoán càng tốt, cụ thể giá trị precision của lần thứ hai giảm so với lần thứ nhất nhưng qua lần thứ ba thì giá trị precision lại vượt trội hơn khá nhiều (nguyên nhân của sự mất cân bằng dữ liệu), F1-score của 2 lần train đầu tiên gần như là không thay đổi cho tới lần train thứ ba lúc đó mới nhận thấy sự thay đổi rõ ràng của nó. Nhận thấy có sự giảm trong sự dự đoán các khuôn mặt mà không được gán nhãn trong bộ dataset và bên cạnh đó số khuôn mặt không dự đoán được gần như là không thay đổi (điều này cũng là một trong những lý do khiến giá trị recall gần như không thay đổi (chỉ tăng nhẹ) sau 3 lần đánh giá).



So sánh kết quả đánh giá của hai mô hình

YOLOv4-tiny

Precision	0.86806
Recall	0.78125
F1-score	0.82237

Precision	0.85065
Recall	0.83974
F1-score	0.84516

Precision	0.89744
Recall	0.85890
F1-score	0.87774

YOLOv5

Precision	0.84967
Recall	0.83333
F1-score	0.84142

Precision	0.83226
Recall	0.84314
F1-score	0.83766

Precision	0.92157
Recall	0.84431
F1-score	0.88125

Ảnh hưởng của sự mất cân bằng dữ liệu ở YOLOv5 là nhiều hơn so với YOLOv4-Tiny. Về precision ở hai lần train đầu tiên của YOLOv4-Tiny là có nhỉnh hơn chút xíu so với ở YOLOv5 nhưng đến lần thứ ba thì lại thấp hơn ở YOLOv5, bên cạnh đó ở 2 mô hình này có một điểm chung đó là giá trị precision lần 2 < lần 1 < lần 3. Về recall ở cả 2 mô hình đều có sự biến thiên tăng dần từ lần đầu tiên đến lần thứ ba, nhưng ở mô hình YOLOv4-Tiny khoảng cách giữa các lần là lớn hơn so với ở YOLOv5. Còn về F1-score ở YOLOv4-Tiny có sự biến thiên tăng dần từ lần đầu tiên đến lần thứ ba, nhưng ở YOLOv5, mặc dù ở 2 lần đầu tiên gần như là không có sự thay đổi nhưng dù gì thì ở lần thứ hai vẫn thấp hơn so với lần thứ nhất và lần có giá trị F1-score cao nhất là lần thứ ba, cũng giống ở YOLOv4-Tiny với lần thứ ba có giá trị F1-score cao nhất.



Chương 4: KẾT LUẬN

1. Nhận xét về mô hình

Mô hình YOLOv4-Tiny thì độ chính xác phụ thuộc rất nhiều vào bộ dataset. Có nghĩa là bộ dataset khi train càng đầy đủ các trường hợp thì mô hình dự đoán càng đúng, ngoài ra thì tỉ lệ các loại nhãn có trong bộ dataset chiếm tỉ lệ phần trăm chênh lệch càng ít thì một phần nào đó sẽ làm giúp tăng được độ chính xác của mô hình khi dự đoán đối tượng. Tuy vậy quá trình train của nó diễn ra cũng khá nhanh.

Mô hình YOLOv5 độ chính xác phụ thuộc rất nhiều vào độ phức tạp của mô hình. Khi chúng ta chọn mô hình có độ phức tạp cao thì độ chính xác của mô hình càng cao, nhưng ngoài sự phụ thuộc vào độ phức tạp của mô hình thì nó còn phụ thuộc vào 2 yếu tố đó là số epochs khi train mô hình và bộ dataset. Khi chúng ta train với epochs càng cao thì chắc chắn độ chính xác của mô hình khi dự đoán càng cao, nhưng đổi lại epochs càng cao thì thời gian train càng lâu. Còn xét về bộ dataset thì cũng giống như YOLOv4-Tiny bộ dataset khi train càng đầy đủ càng trường hợp thì mô hình dự đoán càng tốt, và tỉ lệ các loại nhãn có trong bộ dataset chiếm tỉ lệ phần trăm chênh lệch nhau càng ít thì một phần nào đó cũng sẽ giúp cải thiện được độ chính xác khi dự đoán đối tượng.

2. Bài học kinh nghiệm

Sau khi hoàn thành xong đề tài này, các thành viên trong nhóm đã đều có khả năng trả lời các câu hỏi liên quan tới mô hình nhận diện người có hay không đeo khẩu trang sử dụng YOLO. Bài báo cáo đi sát với toàn bộ nội dung thành viên đã thực hiện cũng như suru tầm được những kiến thức cần phải phân tích rõ ràng. Trong quá trình nghiên cứu và tìm hiểu, nhóm đã thu được những kết quả thực nghiệm hữu ích thông qua việc cải thiện mô hình bằng bộ dataset.

Qua việc đánh giá các lần thực nghiệm, nhóm cũng đã thu được một số kinh nghiệm quý báu từ mô hình. Mục đích của nhóm khi thực hiện việc cải thiện mô hình thông qua bộ dataset là để tìm ra cách thức tạo bộ dataset làm sao để mô hình tối ưu nhất với độ chính xác cao nhất có thể đáp ứng được các yêu cầu đề ra.



TÀI LIỆU THAM KHẢO

TIẾNG VIỆT

1. <https://bkaii.com.vn/tin-tuc/795-thi-giac-may-tinh-la-gi-su-khac-nhau-giu-a-thi-giac-may-va-thi-giac-may-tinh>
 2. <https://bkaii.com.vn/tin-tuc/798-nhung-ung-dung-thuc-tien-cua-thi-giac-may-tinh>
 3. <https://thigiacmay.com/nhung-ung-dung-hieu-qua-cua-thi-giac-may-tinh-computer-vision-vao-thuc-tien/>
 4. <https://www.thegioimaychu.vn/blog/ai-hpc/computer-vision-thi-giac-may-tinh-la-gi-p5951/#:~:text=Nh%E1%BB%AFng%20h%E1%BA%A1n%20ch%E1%BA%BF%20c%E1%BB%A7a%20Th%E1%BB%8B%20gi%C3%A1c%20M%C3%A1y%20t%C3%ADADnh&text=Ch%C3%BAng%20kh%C3%B3ng%20hi%E1%BB%83u%20nh%E1%BB%AFng%20g%C3%ACAC,%E2%80%8Bth%E1%BB%A9c%20c%C6%A1%20b%E1%BA%A3n%20chung.>
 5. <https://phamdinhkhanh.github.io/2019/09/29/OverviewObjectDetection.html#:~:text=bounding%20box%3A%20L%C3%A0%20h%C3%ACACnh%20ch%E1%BB%AF,v%C3%A0%20chi%E1%BB%81u%20d%C3%A0i%2C%20chi%E1%BB%81u%20r%E1%BB%99ng%20.>
 6. <https://viblo.asia/p/tong-hop-kien-thuc-tu-yolov1-den-yolov5-phan-2-V3m5WRDblO7>
 7. <https://viblo.asia/p/tong-hop-kien-thuc-tu-yolov1-den-yolov5-phan-1-naQZRRj0Zvx>
 8. <https://devai.info/2021/02/24/series-yolo-4-tim-hieu-cau-truc-yolov1v2v3-va-v4-phan-2/>
 9. <https://devai.info/2021/01/21/series-yolo-4-tim-hieu-cau-truc-yolov1v2v3-va-v4/>
 10. <https://miai.vn/2020/05/25/yolo-series-train-yolo-v4-train-tren-colab-chi-tiet-va-day-du-a-z/>
 11. <https://devai.info/2020/12/17/tim-hieu-mapmean-average-precision-danh-gia-mo-hinh-object-detection-su-dung-yolov4/>

TIẾNG ANH

1. <https://towardsdatascience.com/mask-detection-using-yolov5-ae40979227a6>
 2. <https://towardsdatascience.com/yolo-v4-or-yolo-v5-or-pp-yolo-dad8e40f7109>
 3. <https://blog.roboflow.com/yolov5-improvements-and-evaluation/>
 4. <https://medium.com/analytics-vidhya/yolov4-vs-yolov4-tiny-97932b6ec8ec>



5. https://www.researchgate.net/publication/350540629_Short_Communication_Detecting_Heavy_Goods_Vehicles_in_Rest_Areas_in_Winter_Conditions_Using_YOLOv5
6. <https://github.com/ultralytics/yolov5>