

**TRƯỜNG ĐẠI HỌC CÔNG NGHIỆP HÀ NỘI**

**KHOA CÔNG NGHỆ THÔNG TIN**



**BÀI TẬP LỚN MÔN**

**MỘT SỐ PHƯƠNG PHÁP TÍNH TOÁN MỀM**

**ĐỀ TÀI: NHẬN DIỆN BIỂN BÁO GIAO THÔNG SỬ DỤNG THUẬT TOÁN YOLO**

**GVHD: TS. Vũ Việt Thắng**

**Nhóm: 08**

**Sinh viên: Nguyễn Duy Quang**

**Trần Thọ Bằng**

**Khổng Quang Hà**

**Ngô Minh Khuê**

**Trần Cao Duy**

**Lớp: 202110503205001 Khóa 13**

**Hà Nội - 2021**

# **MỞ ĐẦU**

Ngày nay, với sự phát triển mạnh mẽ của công nghệ đã và đang thúc đẩy nhanh chóng sự phát triển của xã hội. Công nghệ đang trở thành một phần không thể thiếu trong cuộc sống hằng ngày, nó xuất hiện ở khắp mọi nơi từ trường học, công sở hay đến các công ty. Không những thế nó còn thực hiện nhiều nghiệp vụ đơn giản cũng như phức tạp để tạo ra những lợi ích tốt đẹp cho con người.

Nổi bật nhất đó chính là sự trỗi dậy của trí tuệ nhân tạo, nó có thể được định nghĩa như một ngành của khoa học máy tính liên quan đến việc tự động hóa các hành vi thông minh giống con người, được xây dựng trên nền tảng lý thuyết chắc chắn và có tính ứng dụng thực tiễn cao. Và những năm gần đây, trí tuệ nhân tạo đã được ứng dụng khá nhiều trong các lĩnh vực như: IoT(Internet of Things), đồ họa, sản xuất, y học và quốc phòng.

Có thể thấy rằng việc áp dụng trí tuệ nhân tạo vào đời sống đã và đang mang lại lợi ích vô cùng lớn. Và trong khuôn khổ nội dung của đề tài, nhóm chúng em xin đề cập đến một lĩnh vực đang được áp dụng trí tuệ nhân tạo trong thực tế đó là “Nhận diện biển báo giao thông”.

Thông thường, một hệ thống nhận diện tương đối phức tạp và được cài đặt trên các hệ thống có bộ xử lý có hiệu năng cao, chất lượng camera rất tốt. Hiểu được điều đó, nhóm chúng em quyết định thiết kế một chương trình để nhận diện biển báo giao thông sử dụng thuật toán YOLO áp dụng trên thiết bị máy tính.

Qua đây chúng em cũng xin gửi lời cảm ơn tới Thầy giáo, Tiến sĩ Vũ Việt Thắng. thầy đã giúp đỡ nhóm chúng em trong suốt quá trình học tập. Trong bài tập có thể có nhiều thiếu sót, mong thầy và các bạn đóng góp ý kiến để sản phẩm của nhóm được hoàn thiện hơn. Chúng em xin chân thành cảm ơn!

# **CHƯƠNG I - GIỚI THIỆU KHÁI QUÁT VỀ ĐỀ TÀI VÀ MỤC ĐÍCH CỦA ĐỀ TÀI.**

## 1.1 Giới Thiệu Đề Tài

Ngày nay, những tiến bộ mới trong khoa học kỹ thuật công nghệ đã giúp ích rất nhiều cho cuộc sống của con người. Mọi thứ hầu như đều được tự động và hiệu suất công việc được nâng cao hơn với sự trợ giúp của máy móc, thiết bị. Một trong những công nghệ tiên tiến đang được áp dụng rộng rãi trong đời sống chính là công nghệ nhận dạng vật thể.

Nhận dạng dữ liệu bao gồm có nhận dạng âm thanh và nhận dạng hình ảnh. Các đối tượng của bài toán nhận dạng thì rất phong phú, ví dụ như nhận dạng khuôn mặt, tiếng nói, nhận dạng chữ viết tay, nhận dạng mã vạch,… Nhận diện biển báo giao thông cũng là một trong số đó. Đây là kiểu đối tượng có tính chất hình học đặc trưng, thường bắt gặp trong đời sống hằng ngày, với công dụng là đưa ra những cảnh báo thông tin cho người tham gia giao thông. Tuy nhiên các biển báo giao thông thì rất đa dạng và có các ký hiệu với ý nghĩa quy ước kèm theo. Việc ghi nhớ hình dạng và ý nghĩa của tất cả các loại biển báo đối với chúng ta sẽ là một khó khăn lớn, do đó việc xây dựng một hệ thống nhận diện biển báo giao thông ngoài đời thực là một nhu cầu rất hữu ích và có tính thực tế cao.

Bài toán nhận diện biển báo giao thông hiện vẫn còn là một trong những chủ đề được các nhà khoa học nghiên cứu. Hiện tại đã có một số hệ thống tiên tiến của nước ngoài có khả năng nhận diện biển báo giao thông rất tốt và hầu hết các hệ thống này đều đòi hỏi một khả năng xử lý mạnh mẽ, đi kèm với nó là những camera có chất lượng cao. Do đó, việc ứng dụng công nghệ nhận diện biển báo giao thông trên máy tính, chính là ý tưởng mà nhóm hướng tới khi thực hiện đề tài này. Bài toán nhóm sẽ giải quyết là làm thế nào xây dựng một hệ thống tự động cho phép phát hiện và nhận dạng biển báo giao thông trên thiết bị máy tính.

## 1.2 Mục Tiêu Của Đề Tài

Mục tiêu của đề tài là nghiên cứu thuật toán YOLO trong việc nhận diện biển báo giao thông, từ đó việc xây dựng hệ thống nhận diện trên thiết bị máy tính. Chương trình sẽ được xây dựng theo quy trình dưới đây:



## 1.3 Tình hình nghiên cứu ngoài nước

**Bài báo 1:** **Real Time Detection And Classification Of Traffic Signs Based On YOLO Version 3 Algorithm**

Tác giả: V.N. Sichkar, S.A. Kolyubin

Phương pháp sử dụng: YOLO version 3 và CNN

Bộ dữ liệu sử dụng: GTSRB, GTSDB

Kết quả thử nghiệm trên tập dữ liệu test: độ chính xác 97.22 %.

**Bài báo 2: Traffic Sign Detection based on SSD**

Tác giả: Benhe Gao, Zhoungjun Jiang, Jiaman Zhang

Phương pháp sử dụng: SSD

Bộ dữ liệu sử dụng: GTSDB

Kết quả thử nghiệm trên tập dữ liệu test: độ chính xác 91.09%.

## 1.4 Tình hình nghiên cứu trong nước

**Bài báo 1: “Phát hiện và nhận dạng biển báo giao thông đường bộ sử dụng đặc trưng HOG và Mạng Nơron nhân tạo”**

Tác giả: Trương Quốc Bảo, Trương Hùng Chen, Trương Quốc Định

Phương pháp sử dụng: PCA và LDA

Bộ dữ liệu sử dụng: GTSDB

Kết quả thử nghiệm trên tập dữ liệu test: độ chính xác 94%

# **Chương II - Các phương pháp sử dụng giải quyết bài toán: kiến thức về xử lý ảnh, lý thuyết mạng CNN, thuật toán YOLO.**

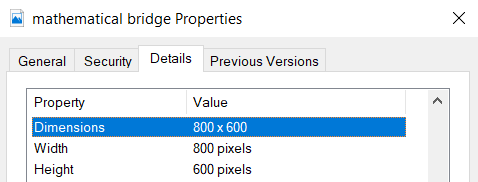
## 2.1 Các kiến thức cơ bản về xử lý ảnh

### 2.1.1 Ảnh màu

Ví dụ về ảnh màu

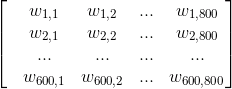
*Hình 1: Mathematical bridge*

Khi bạn kích chuột phải vào ảnh trong máy tính, bạn chọn properties (mục cuối cùng), rồi chọn tab detail



*Hình 2: Chi tiết bức ảnh*

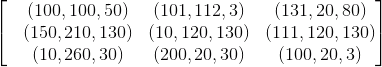
Ta thấy bức ảnh trên kích thước 800 pixel \* 600 pixel, có thể biểu diễn dưới dạng một ma trận kích thước 600 \* 800 (vì định nghĩa ma trận là số hàng nhân số cột).



Trong đó mỗi phần tử *wij*​ là một pixel.

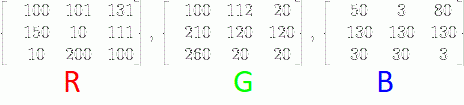
Như vậy có thể hiểu là mỗi pixel thì biểu diễn một màu và bức ảnh trên là sự kết hợp rất nhiều pixel. Hiểu đơn giản thì in bức ảnh ra, kẻ ô vuông như chơi cờ ca rô với 800 đường thẳng ở chiều dài, 600 đường ở chiều rộng, thì mỗi ô vuông là một pixel, biểu diễn một chấm màu.

Tuy nhiên để biểu diễn 1 màu ta cần 3 thông số (r,g,b) nên gọi *wij*​=(*rij*​,*gij*​,*bij*​) để biểu diễn dưới dạng ma trận thì sẽ như sau:



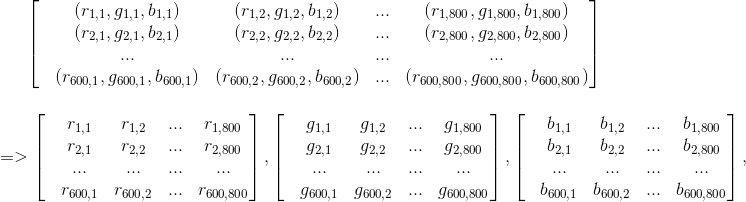
Ảnh màu kích thước 3\*3 biểu diễn dạng ma trận, mỗi pixel biểu diễn giá trị (r,g,b)

Để tiện lưu trữ và xử lý không thể lưu trong 1 ma trận như thế kia mà sẽ tách mỗi giá trị trong mỗi pixel ra một ma trận riêng.



Tách ma trận trên thành 3 ma trận cùng kích thước: mỗi ma trận lưu giá trị từng màu khác nhau red, green, blue

Tổng quát

Tách ma trận biểu diễn màu ra 3 ma trận, mỗi ma trận lưu giá trị 1 màu.

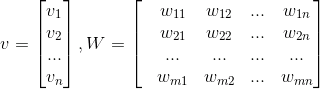
Mỗi ma trận được tách ra được gọi là 1 channel nên ảnh màu được gọi là 3 channel: channel red, channel green, channel blue.

**Tóm tắt**: Ảnh màu là một ma trận các pixel mà mỗi pixel biểu diễn một điểm màu. Mỗi điểm màu được biểu diễn bằng bộ 3 số (r,g,b). Để tiện cho việc xử lý ảnh thì sẽ tách ma trận pixel ra 3 channel red, green, blue.

### 2.1.2 Tensor

Khi dữ liệu biểu diễn dạng 1 chiều, người ta gọi là vector, mặc định khi viết vector sẽ viết dưới dạng cột.

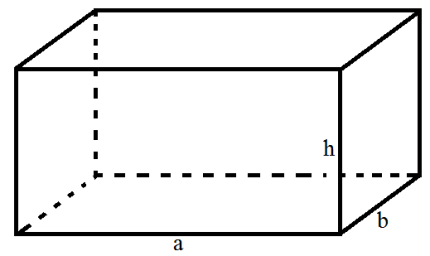
Khi dữ liệu dạng 2 chiều, người ta gọi là ma trận, kích thước là số hàng \* số cột.



Vector v kích thước n, ma trận W kích thước m\*n

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

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



*Hình 3: Hình hộp chữ nhật kích thước a\*b\*h*

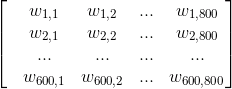
Tưởng tượng mặt đáy là một ma trận kích thước a \* b, được tạo bởi b vector kích thước a. Cả hình hộp là tensor 3 chiều kích thước a\*b\*h, được tạo bởi xếp h ma trận kích thước a\*b lên nhau.

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

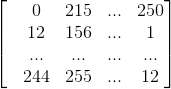
### 2.1.3 Ảnh xám

*Hình 4: Ảnh xám của cầu mathematical*

Tương tự ảnh màu, ảnh xám cũng có kích thước 800 pixel \* 600 pixel, có thể biểu diễn dưới dạng một ma trận kích thước 600 \* 800 (vì định nghĩa ma trận là số hàng nhân số cột).



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



Biểu diễn ảnh xám

Giá trị 0 là màu đen, 255 là màu trắng và giá trị pixel càng gần 0 thì càng tối và càng gần 255 thì càng sáng.

Chuyển hệ màu của ảnh

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

Khi chuyển từ ảnh màu sang ảnh xám ta có thể dùng công thức:

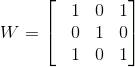
x = r \* 0.299 + g \* 0.587 + b \* 0.114.

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

### 2.1.4 Phép tính convolution

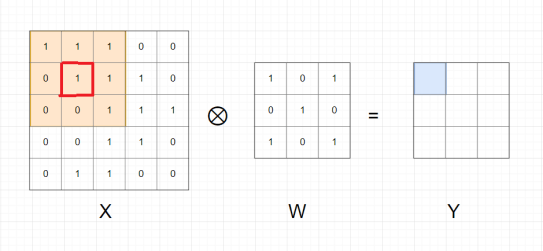
Để cho dễ hình dung mình sẽ lấy ví dụ trên ảnh xám, tức là ảnh được biểu diễn dưới dạng ma trận A kích thước m\*n.

Ta định nghĩa **kernel**là một ma trận vuông kích thước k\*k trong đó k là số lẻ. k có thể bằng 1, 3, 5, 7, 9,… Ví dụ kernel kích thước 3\*3



Kí hiệu phép tính convolution *Y*=*X*⊗*W*

Với mỗi phần tử *xij*​ trong ma trận X lấy ra một ma trận có kích thước bằng kích thước của kernel W có phần tử *xij*​ làm trung tâm (đây là vì sao kích thước của kernel thường lẻ) gọi là ma trận A. Sau đó tính tổng các phần tử của phép tính element-wise của ma trận A và ma trận W, rồi viết vào ma trận kết quả Y.



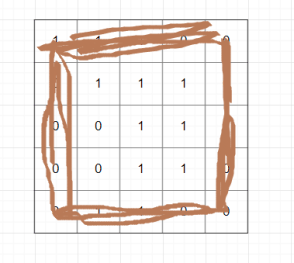
*Hình 5: Phép tính nhân chập*

Ví dụ khi tính tại *x*22​ (ô khoanh đỏ trong hình), ma trận A cùng kích thước với W, có *x*22​ làm trung tâm có màu nền da cam như trong hình. Sau đó tính

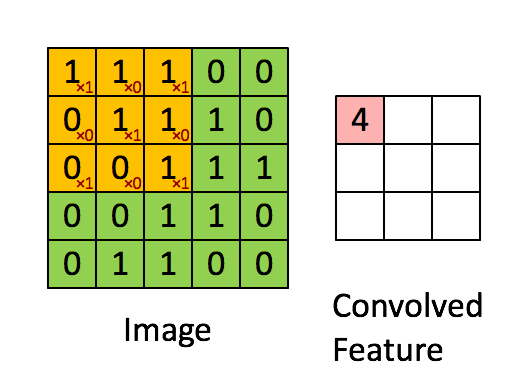
*y*11​=*sum*(*A*⊗*W*)=*x*11​∗*w*11​+*x*12​∗*w*12​+*x*13​∗*w*13​+*x*21​∗*w*21​+*x*22​∗*w*22​+*x*23​∗*w*23​+*x*31​∗*w*31​+*x*32​∗*w*32​+*x*33​∗*w*33​=4.

Và làm tương tự với các phần tử còn lại trong ma trận.

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



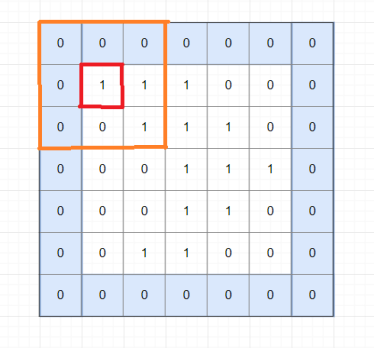
Nên bạn để ý thấy ma trận Y có kích thước nhỏ hơn ma trận X. Kích thước của ma trận Y là (m-k+1) \* (n-k+1).



*Hình 6: Các bước thực hiện phép tính convolution cho ma trận X với kernel K*

### 2.1.5 Padding

Như ở trên thì mỗi lần thực hiện phép tính convolution xong thì kích thước ma trận Y đều nhỏ hơn X. Tuy nhiên giờ ta muốn ma trận Y thu được có kích thước bằng ma trận X => Tìm cách giải quyết cho các phần tử ở viền => Thêm giá trị 0 ở viền ngoài ma trận X.



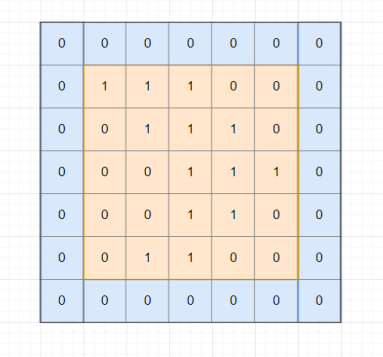
*Hinh 7: Ma trận X khi thêm viền 0 bên ngoài*

Rõ ràng là giờ đã giải quyết được vấn đề tìm A cho phần tử *x*11​ , và ma trận Y thu được sẽ bằng kích thước ma trận X ban đầu.

Phép tính này gọi là convolution với **padding=1**. Padding=k nghĩa là thêm k vector 0 vào mỗi phía của ma trận.

### 2.1.6 Stride

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



*Hình 8: stride=1, padding=1*

Tuy nhiên nếu **stride=k** (k > 1) thì ta chỉ thực hiện phép tính convolution trên các phần tử *x*1+*i*∗*k*,1+*j*∗*k*​. Ví dụ k = 2.

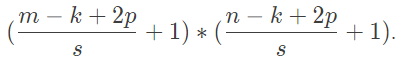


*Hình 9: padding=1, stride=2*

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

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

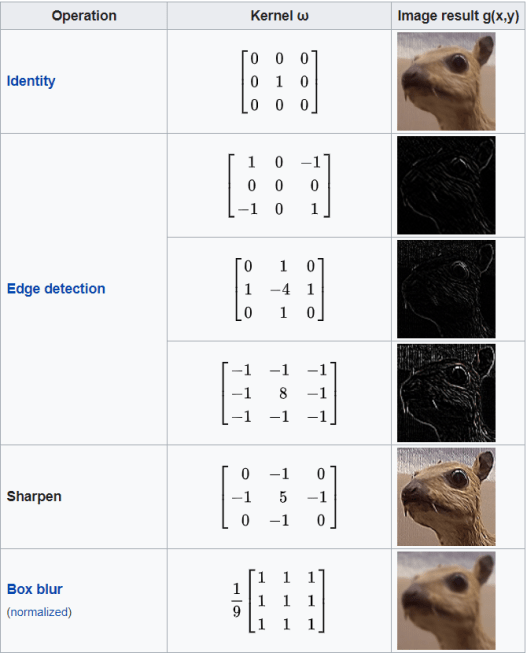
Công thức tổng quát cho phép tính convolution của ma trận X kích thước m\*n với kernel kích thước k\*k, stride = s, padding = p ra ma trận Y kích thước



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

**Ý nghĩa của phép tính convolution**

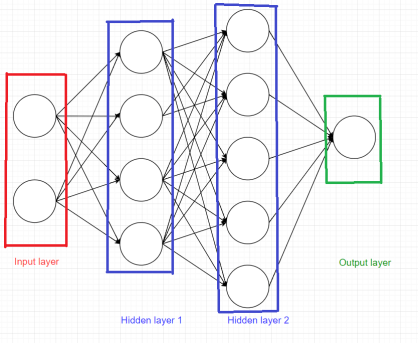
Mục đích của phép tính convolution trên ảnh là làm mở, làm nét ảnh; xác định các đường;… Mỗi kernel khác nhau thì sẽ phép tính convolution sẽ có ý nghĩa khác nhau. Ví dụ:



*Hình 10: Ý nghĩa của phép tính nhân chập*

## 2.2 Mạng CNN(Convolutional neural network)

### 2.2.1 Mô hình neural network

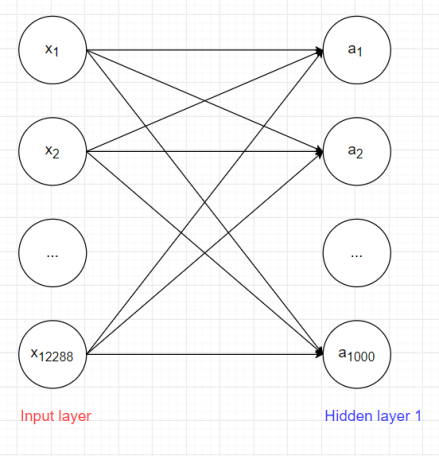


*Hình 11: Mô hình neural network.*

Mỗi hidden layer được gọi là **fully connected layer**, tên gọi theo đúng ý nghĩa, mỗi node trong hidden layer được kết nối với tất cả các node trong layer trước. Cả mô hình được gọi là **fully connected neural network (FCN)**.

**Vấn đề của fully connected neural network với xử lý ảnh**

Ví dụ ảnh màu 64\*64 được biểu diễn dưới dạng 1 tensor 64\*64\*3. Nên để biểu thị hết nội dung của bức ảnh thì cần truyền vào input layer tất cả các pixel (64\*64\*3 = 12288). Nghĩa là input layer giờ có 12288 nodes.



*Hình 12: Input layer và hidden layer 1*

Giả sử số lượng node trong hidden layer 1 là 1000. Số lượng weight W giữa input layer và hidden layer 1 là 12288\*1000 = 12288000, số lượng bias là 1000 => tổng số parameter là: 12289000. Đấy mới chỉ là số parameter giữa input layer và hidden layer 1, trong model còn nhiều layer nữa, và nếu kích thước ảnh tăng, ví dụ 512\*512 thì số lượng parameter tăng cực kì nhanh => Cần giải pháp tốt hơn !!!

Nhận xét:

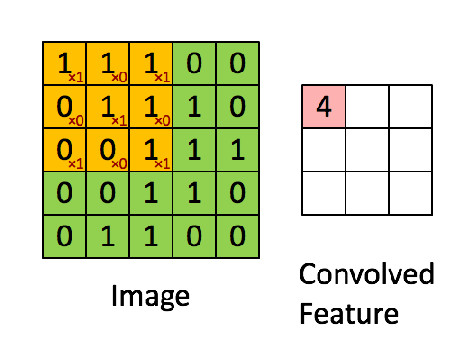
Trong ảnh các pixel ở cạnh nhau thường có **liên kết với nhau hơn** là những pixel ở xa. Ví dụ như phép tính convolution trên ảnh ở bài trước. Để tìm các đường trong ảnh, ta áp dụng sobel kernel trên mỗi vùng kích thước 3\*3. Hay làm nét ảnh ta áp dụng sharpen kernel cũng trên vùng có kích thước 3\*3.

Trong phép tính convolution trong ảnh, chỉ 1 kernel được dùng trên toàn bộ bức ảnh. Hay nói cách khác là các pixel ảnh **chia sẻ** hệ số với nhau.

=> Áp dụng phép tính convolution vào layer trong neural network ta có thể giải quyết được vấn đề lượng lớn parameter mà vẫn lấy ra được các đặc trưng của ảnh.

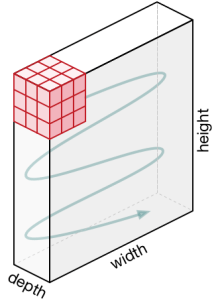
### 2.2.2 Convolutional layer đầu tiên

Bài trước phép tính convolution thực hiện trên ảnh xám với biểu diễn ảnh dạng ma trận



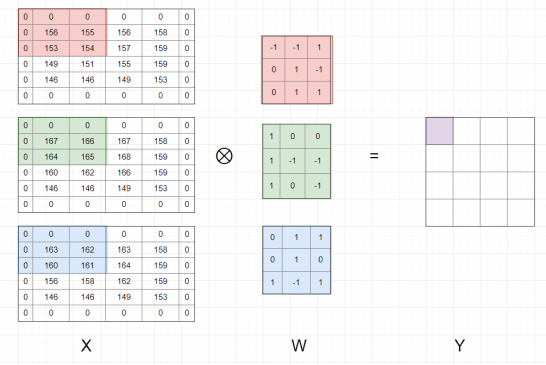
*Hình 13: Phép tính nhân chập*

Tuy nhiên ảnh màu có tới 3 channels red, green, blue nên khi biểu diễn ảnh dưới dạng tensor 3 chiều. Nên ta cũng sẽ định nghĩa kernel là 1 tensor 3 chiều kích thước k\*k\*3.



*Hình 14: Phép tính convolution trên ảnh màu với k=3.*

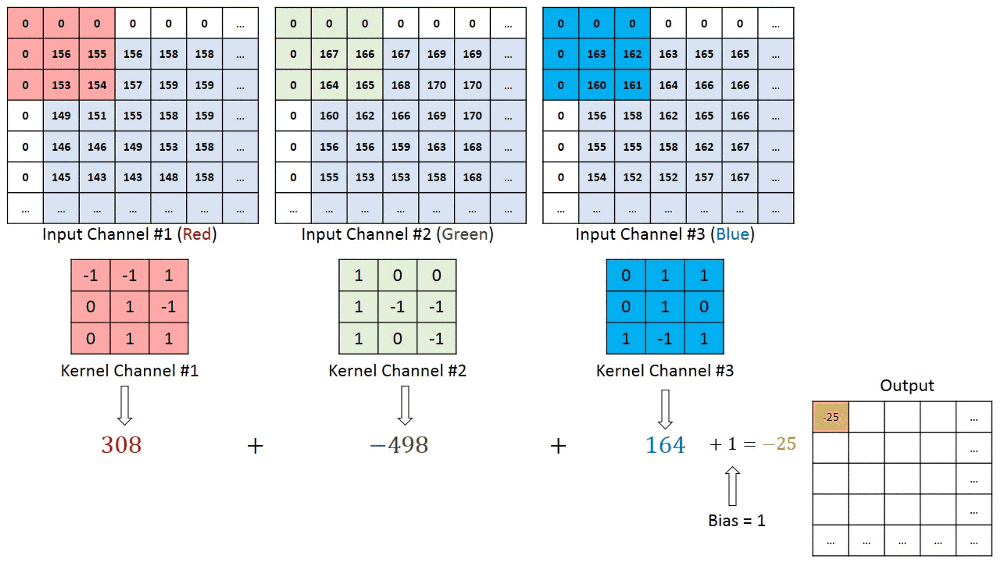
Ta định nghĩa kernel có cùng độ sâu (depth) với biểu diễn ảnh, rồi sau đó thực hiện di chuyển khối kernel tương tự như khi thực hiện trên ảnh xám.



*Hình 15: Tensor X, W 3 chiều được viết dưới dạng 3 matrix.*

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

*y*11​=*b*+(*x*111​∗*w*111​+*x*121​∗*w*121​+*x*131​∗*w*131​+*x*211​∗*w*211​+*x*221​∗*w*221​+*x*231​∗*w*231​+*x*311​∗*w*311​+*x*321​∗*w*321​+*x*331​∗*w*331​)+(*x*112​∗*w*112​+*x*122​∗*w*122​+*x*132​∗*w*132​+*x*212​∗*w*212​+*x*222​∗*w*222​+*x*232​∗*w*232​+*x*312​∗*w*312​+*x*322​∗*w*322​+*x*332​∗*w*332​)+(*x*113​∗*w*113​+*x*123​∗*w*123​+*x*133​∗*w*133​+*x*213​∗*w*213​+*x*223​∗*w*223​+*x*233​∗*w*233​+*x*313​∗*w*313​+*x*323​∗*w*323​+*x*333​∗*w*333​)=−25

*Hình 16: Thực hiện phép tính convolution trên ảnh màu*

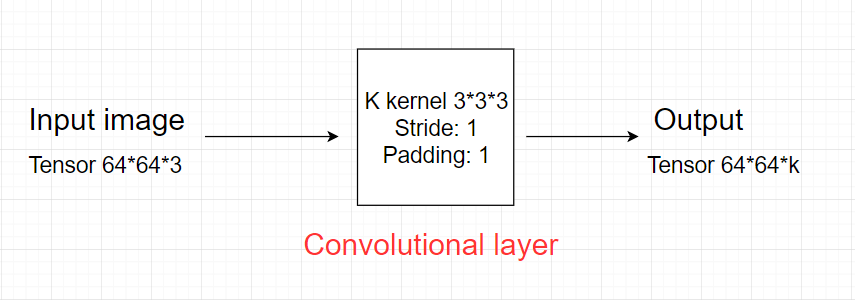
Nhận xét:

Output Y của phép tính convolution trên ảnh màu là 1 matrix.

Có 1 **hệ số bias** được cộng vào sau bước tính tổng các phần tử của phép tính element-wise

**Các quy tắc đối với padding và stride toàn hoàn tương tự như ở phần xử lý ảnh.**

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

*Hình 17: Convolutional layer đầu tiên*

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

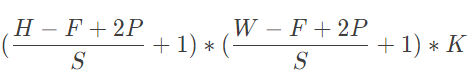
### 2.2.3 Convolutional layer tổng quát

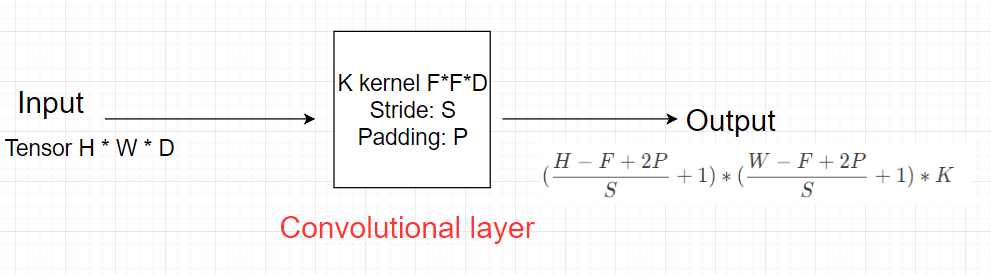
Giả sử input của 1 convolutional layer tổng quát là tensor kích thước H \* W \* D.

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

Convolutional layer áp dụng K kernel.

=> Output của layer là tensor 3 chiều có kích thước:





*Hình 18: Output của Convolutional layer áp dụng K kernel.*

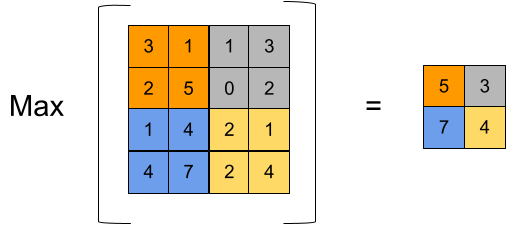
Lưu ý:

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

### 2.2.4 Pooling layer

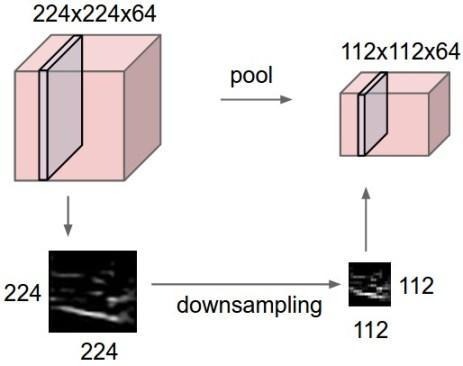
Pooling layer thường được dùng giữa các convolutional layer, để giảm kích thước dữ liệu nhưng vẫn giữ được các thuộc tính quan trọng. Kích thước dữ liệu giảm giúp giảm việc tính toán trong model.

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

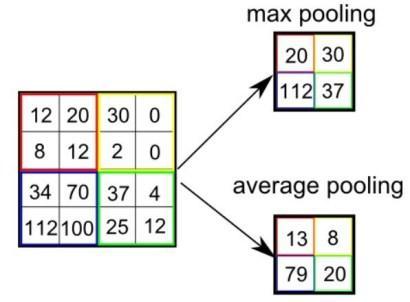


*Hình 19: Max pooling layer với size=(2,2), stride=2, padding=0*

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

  
*Hình 20: Sau khi pooling layer (2\*2).*

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

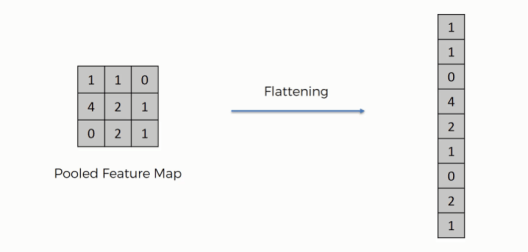


*Hình 21: Ví dụ về pooling layer*

Trong một số model người ta dùng convolutional layer với stride > 1 để giảm kích thước dữ liệu thay cho pooling layer.

### 2.2.5 Fully connected layer

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



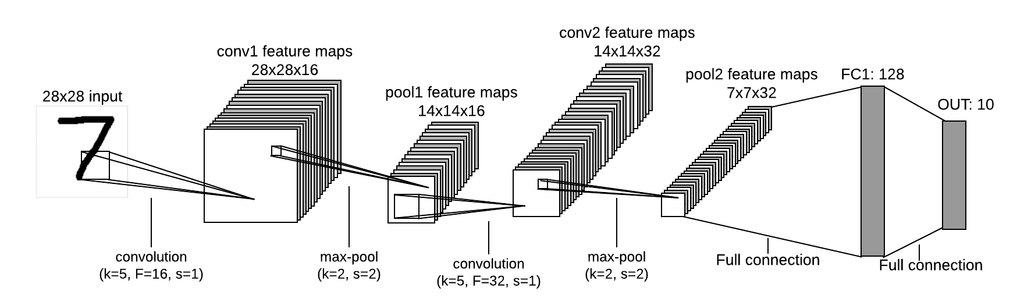
*Hình 22: Làm phẳng ma trận*

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

**Visualise convolutional neural network**

Mô hình convolutional neural network:

Input image -> Convolutional layer (Conv) + Pooling layer (Pool) -> Fully connected layer (FC) -> Output.



*Hình 23: Mô hình mạng CNN*

## 2.3 Thuật toán YOLO

### 2.3.1 Thuật toán YOLO

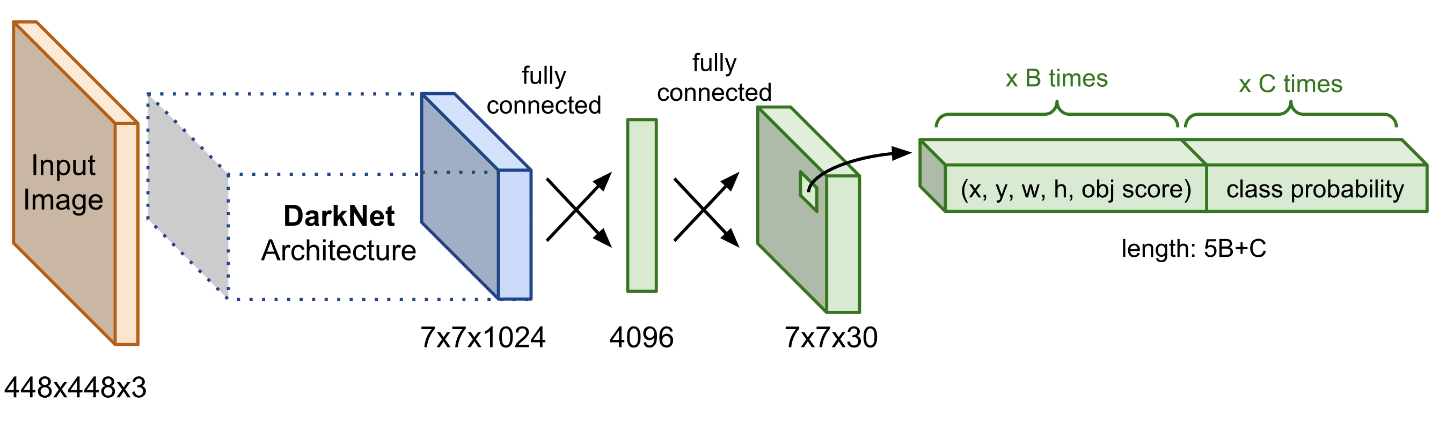
YOLO là thuật toán object detection nên mục tiêu của mô hình không chỉ là dự báo nhãn cho vật thể như các bài toán classification mà nó còn xác định location của vật thể. Do đó YOLO có thể phát hiện được nhiều vật thể có nhãn khác nhau trong một bức ảnh thay vì chỉ phân loại duy nhất một nhãn cho một bức ảnh.

You only look once (YOLO) là một mô hình CNN để detect object mà một ưu điểm nổi trội là nhanh hơn nhiều so với những mô hình cũ. Thậm chí có thể chạy tốt trên những IOT device như raspberry pi.

### 2.3.2 Kiến trúc mạng YOLO

Kiến trúc mạng YOLO bao gồm: base network là các mạng convolution làm nhiệm vụ trích xuất đặc trưng. Phần phía sau là những Extra Layers được áp dụng để phát hiện vật thể trên feature map của base network.

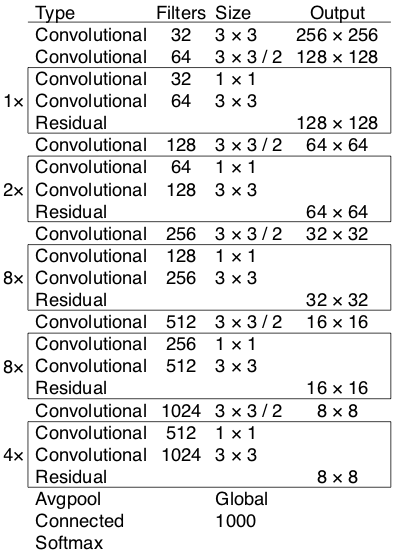
Base network của YOLO sử dụng chủ yếu là các convolutional layer và các fully conntected layer. Các kiến trúc YOLO cũng khá đa dạng và có thể tùy biến thành các version cho nhiều input shape khác nhau.



*Hình 24:* Kiến trúc mạng YOLO version 1.

Thành phần Darknet Architechture được gọi là base network có tác dụng trích suất đặc trưng. Output của base network là một feature map có kích thước 7x7x1024 sẽ được sử dụng làm input cho các Extra layers có tác dụng dự đoán nhãn và tọa độ bounding box của vật thể.

Trong YOLO version 3 tác giả áp dụng một mạng feature extractor là darknet-53. Mạng này gồm 53 convolutional layers kết nối liên tiếp, mỗi layer được theo sau bởi một batch normalization và một activation Leaky Relu. Để giảm kích thước của output sau mỗi convolution layer, tác giả down sample bằng các filter với kích thước là 2. Do đó có tác dụng giảm thiểu số lượng tham số cho mô hình.



*Hình 25: Các layer trong mạng darknet-53.*

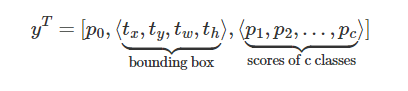
Các bức ảnh khi được đưa vào mô hình sẽ được scale để về chung một kích thước phù hợp với input shape của mô hình và sau đó được gom lại thành batch đưa vào huấn luyện.

Hiện tại YOLO version 3 đang hỗ trợ 2 đầu vào chính là 416x416 và 608x608. Mỗi một đầu vào sẽ có một thiết kế các layers riêng phù hợp với shape của input. Sau khi đi qua các layer convolutional thì shape giảm dần theo cấp số nhân là 2. Cuối cùng ta thu được một feature map có kích thước tương đối nhỏ để dự báo vật thể trên từng ô của feature map.

Kích thước của feature map sẽ phụ thuộc vào đầu vào. Đối với input 416x416 thì feature map có các kích thước là 13x13, 26x26 và 52x52. Và khi input là 608x608 sẽ tạo ra feature map 19x19, 38x38, 72x72.

### 2.3.3 Đầu ra của mạng YOLO

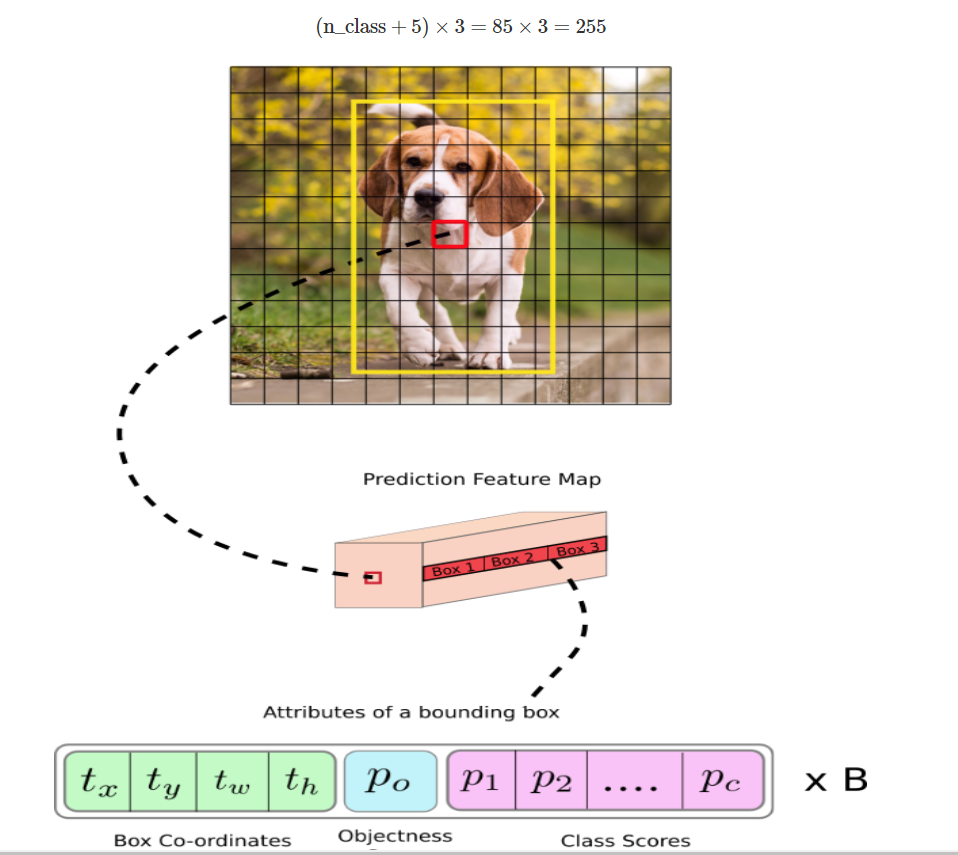
Output của mô hình YOLO là một véc tơ sẽ bao gồm các thành phần:



Trong đó:

* là xác suất dự báo vật thể xuất hiện trong bounding box.
* giúp xác định bounding box. Trong đó  là tọa độ tâm và là kích thước rộng, dài của bounding box.
* là véc tơ phân phối xác suất dự báo của các classes.

Việc hiểu output khá là quan trọng để chúng ta cấu hình tham số chuẩn xác khi huấn luyện model qua các open source như darknet. Như vậy output sẽ được xác định theo số lượng classes theo công thức(n\_class+5) . Nếu huấn luyện 80 classes thì bạn sẽ có output là 85. Trường hợp bạn áp dụng 3 anchors/cell thì số lượng tham số output sẽ là:

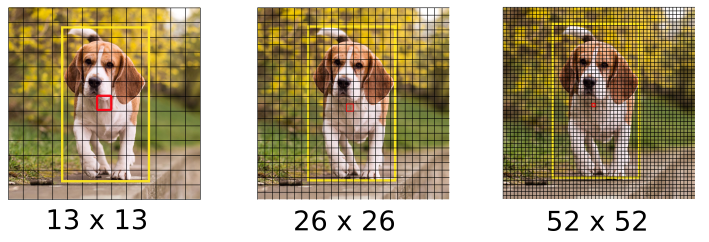


*Hình 26: Kiến trúc một output của model YOLO.*

Hình ảnh gốc là một feature map kích thước 13x13. Trên mỗi một cell của feature map chúng ta lựa chọn ra 3 anchor boxes với kích thước khác nhau lần lượt là Box 1, Box 2, Box 3 sao cho tâm của các anchor boxes trùng với cell. Khi đó output của YOLO là một véc tơ concatenate của 3 bounding boxes. Các attributes của một bounding box được mô tả như dòng cuối cùng trong hình.

### 2.3.4 Dự báo trên nhiều feature map

Cũng tương tự như SSD, YOLOv3 dự báo trên nhiều feature map. Những feature map ban đầu có kích thước nhỏ giúp dự báo được các object kích thước lớn. Những feature map sau có kích thước lớn hơn trong khi anchor box được giữ cố định kích thước nên sẽ giúp dự báo các vật thể kích thước nhỏ.



*Hình 27****:*** *Các feature maps của mạng YOLOv3 với input shape là 416x416, output là 3 feature maps có kích thước lần lượt là 13x13, 26x26 và 52x52*

Trên mỗi một cell của các feature map chúng ta sẽ áp dụng 3 anchor box để dự đoán vật thể. Như vậy số lượng các anchor box khác nhau trong một mô hình YOLO sẽ là 9 (3 featue map x 3 anchor box).

Đồng thời trên một feature map hình vuông S x S, mô hình YOLOv3 sinh ra một số lượng anchor box là: S x S x 3. Như vậy số lượng anchor boxes trên một bức ảnh sẽ là:

Đây là một số lượng rất lớn và là nguyên nhân khiến quá trình huấn luyện mô hình YOLO vô cùng chậm bởi chúng ta cần dự báo đồng thời nhãn và bounding box trên đồng thời 10647 bounding boxes.

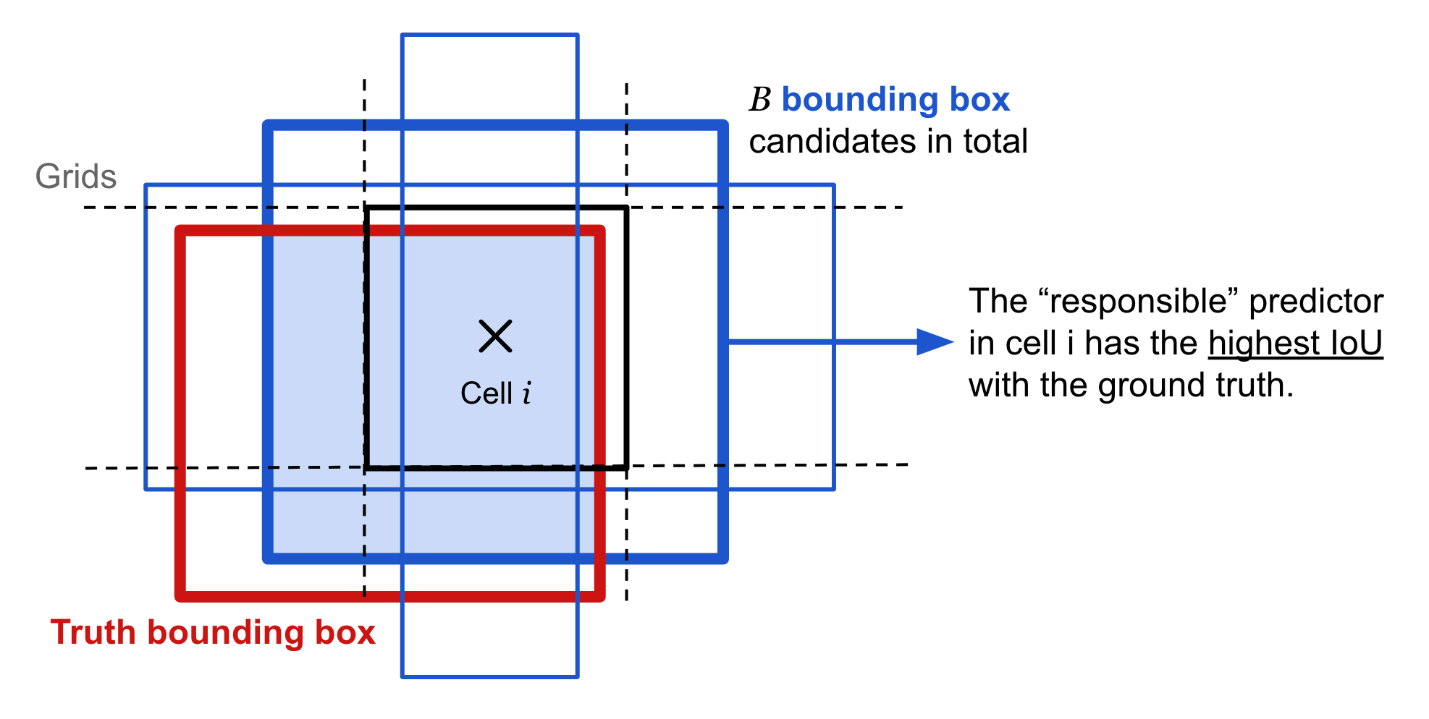
Một số lưu ý khi huấn luyện YOLO:

* Khi huấn luyện YOLO sẽ cần phải có RAM dung lượng lớn hơn để save được 10647 bounding boxes như trong kiến trúc này.
* Không thể thiết lập các batch\_size quá lớn như trong các mô hình classification vì rất dễ Out of memory. Package darknet của YOLO đã chia nhỏ một batch thành các subdivisions cho vừa với RAM.
* Thời gian xử lý của một step trên YOLO lâu hơn rất rất nhiều lần so với các mô hình classification. Do đó nên thiết lập steps giới hạn huấn luyện cho YOLO nhỏ. Đối với các tác vụ nhận diện dưới 5 classes, dưới 5000 steps là có thể thu được nghiệm tạm chấp nhận được. Các mô hình có nhiều classes hơn có thể tăng số lượng steps theo cấp số nhân tùy bạn.

### 2.3.5 Anchor box

Để tìm được bounding box cho vật thể, YOLO sẽ cần các anchor box làm cơ sở ước lượng. Những anchor box này sẽ được xác định trước và sẽ bao quanh vật thể một cách tương đối chính xác. Sau này thuật toán regression bounding box sẽ tinh chỉnh lại anchor box để tạo ra bounding box dự đoán cho vật thể. Trong một mô hình YOLO:

* Mỗi một vật thể trong hình ảnh huấn luyện được phân bố về một anchor box. Trong trường hợp có từ 2 anchor boxes trở lên cùng bao quanh vật thể thì ta sẽ xác định anchor box mà có IoU với ground truth bounding box là cao nhất.



*Hình 28: Xác định anchor box cho một vật thể.*

* Từ Cell i ta xác định được 3 anchor boxes viền xanh như trong hình. Cả 3 anchor boxes này đều giao nhau với bounding box của vật thể. Tuy nhiên chỉ anchor box có đường viền dày nhất màu xanh được lựa chọn làm anchor box cho vật thể bởi nó có IoU so với ground truth bounding box là cao nhất.
* Mỗi một vật thể trong hình ảnh huấn luyện được phân bố về một cell trên feature map mà chứa điểm mid point của vật thể. Chẳng hạn như hình chú chó trong hình 3 sẽ được phân về cho cell màu đỏ vì điểm mid point của ảnh chú chó rơi vào đúng cell này. Từ cell ta sẽ xác định các anchor boxes bao quanh hình ảnh chú chó.

Như vậy khi xác định một vật thể ta sẽ cần xác định 2 thành phần gắn liền với nó là (cell, anchor box). Không chỉ riêng mình cell hoặc chỉ mình anchor box.

Một số trường hợp 2 vật thể bị trùng mid point, mặc dù rất hiếm khi xảy ra, thuật toán sẽ rất khó xác định được class cho chúng.

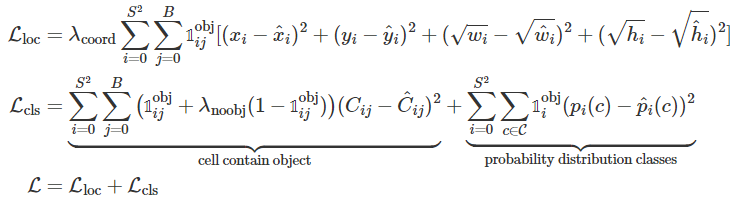


*Hình 29****:*** *Khi 2 vật thể người và xe trùng mid point và cùng thuộc một cell. Thuật toán sẽ cần thêm những lượt tiebreak để quyết định đâu là class cho cell.*

### 2.3.6 Hàm loss function

Hàm loss function của YOLO chia thành 2 phần:

(localization loss) đo lường sai số của bounding box và (confidence loss) đo lường sai số của phân phối xác suất các classes.



* : Hàm indicator có giá trị nhằm xác định xem cell có chứa vật thể hay không. Bằng 1 nếu chứa vật thể và 0 nếu không chứa.
* : Cho biết bounding box thứ j của cell i có phải là bounding box của vật thể được dự đoán hay không.
* : Điểm tin cậy của ô i, P(contain object) \* IoU (predict bbox, ground truth bbox)
* : Điểm tự tin dự đoán.
* : Tập hợp tất cả các lớp.
* :  Xác suất có điều kiện, có hay không ô i có chứa một đối tượng của lớp
* : Xác suất có điều kiện dự đoán.

Có thể ban đầu công thức trên khá khó hiểu với người bắt đầu. Chúng ta hãy hiểu đơn giản hóa mục đích của chúng:

* : là hàm mất mát của bounding box dự báo so với thực tế.
* : là hàm mất mát của phân phối xác suất. Trong đó tổng đầu tiên là mất mát của dự đoán có vật thể trong cell hay không. Và tổng thứ 2 là mất mát của phân phối xác suất nếu có vật thể trong cell.

Ngoài ra để điều chỉnh phạt loss function trong trường hợp dự đoán sai bounding box ta thông qua hệ số điều chỉnh và ta muốn giảm nhẹ hàm loss function trong trường hợp cell không chứa vật thể bằng hệ số điều chỉnh .

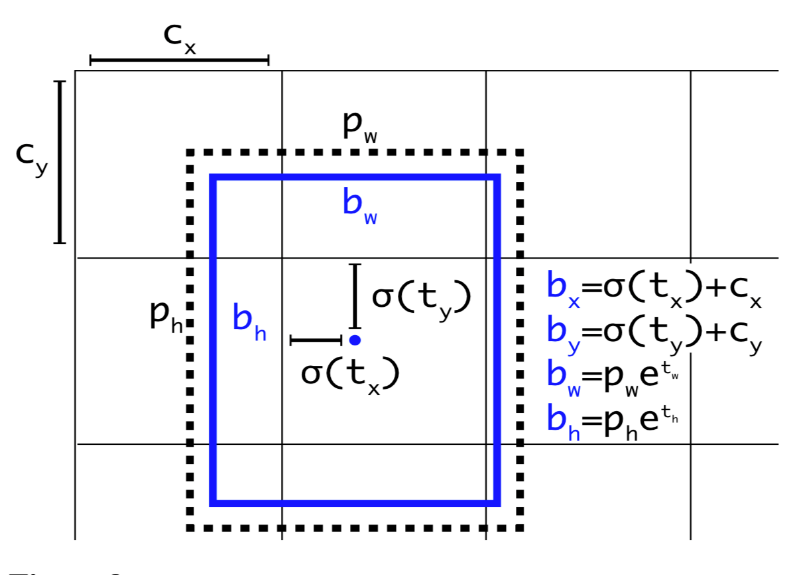
### 2.3.7 Dự báo bounding box

Để dự báo bounding box cho một vật thể chúng ta dựa trên một phép biến đổi từ anchor box và cell.

YOLO version 2 vả YOLO version 3 dự đoán bounding box sao cho nó sẽ không lệch khỏi vị trí trung tâm quá nhiều. Nếu bounding box dự đoán có thể đặt vào bất kỳ phần nào của hình ảnh, như trong mạng regional proposal network, việc huấn luyện mô hình có thể trở nên không ổn định.

Cho một anchor box có kích thước  tại cell nằm trên feature map với góc trên cùng bên trái của nó là , mô hình dự đoán 4 tham số  trong đó 2 tham số đầu là độ lệch (offset) so với góc trên cùng bên trái của cell và 2 tham số sau là tỷ lệ so với anchor box. Và các tham số này sẽ giúp xác định bounding box dự đoán b có tâm  và kích thước  thông qua hàm sigmoid và hàm exponential như các công thức bên dưới:

Ngoài ra do các tọa độ đã được hiệu chỉnh theo width và height của bức ảnh nên luôn có giá trị nằm trong ngưỡng [0, 1]. Do đó khi áp dụng hàm sigmoid giúp ta giới hạn được tọa độ không vượt quá xa các ngưỡng này.

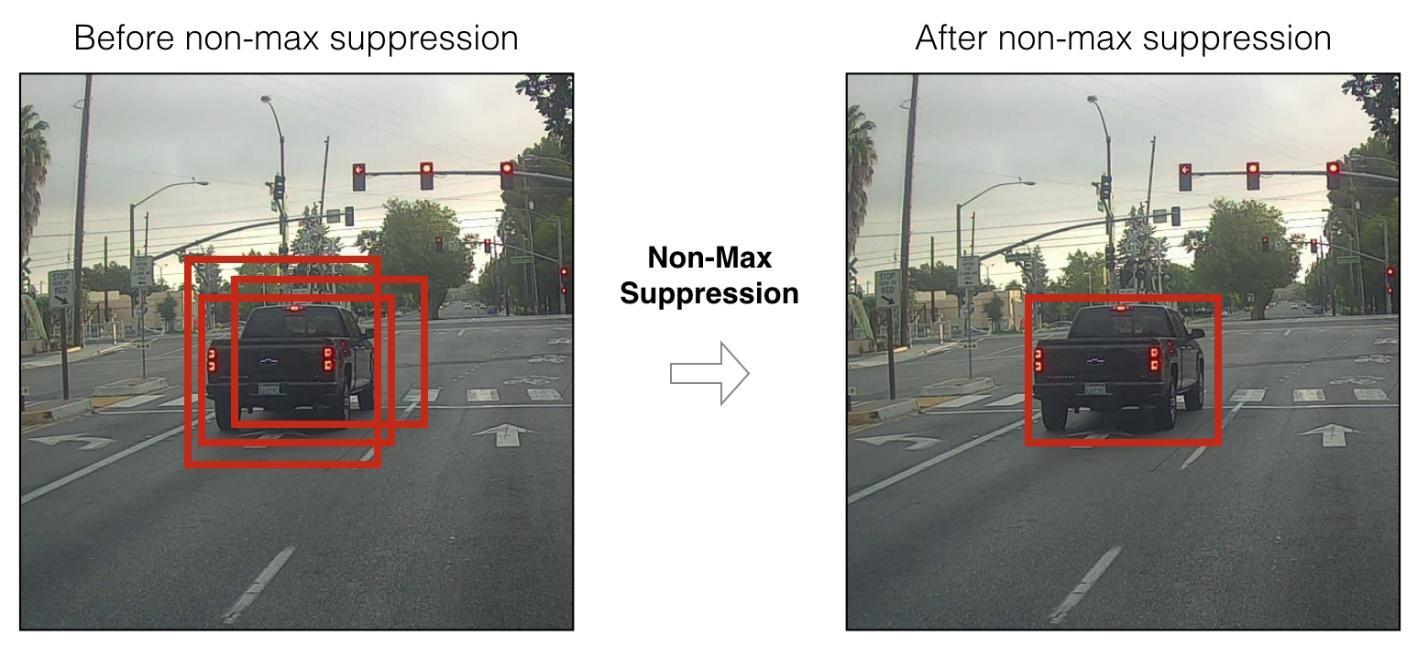


*Hình 30****:****Công thức ước lượng bounding box từ anchor box.*

Hình chữ nhật nét đứt bên ngoài là anchor box có kích thước là . Tọa độ của một bounding box sẽ được xác định dựa trên đồng thời cả anchor box và cell mà nó thuộc về. Điều này giúp kiểm soát vị trí của bounding box dự đoán đâu đó quanh vị trí của cell và bounding box mà không vượt quá xa ra bên ngoài giới hạn này. Do đó quá trình huấn luyện sẽ ổn định hơn rất nhiều so với YOLO version 1.

### 2.3.8 Non-max suppression

Do thuật toán YOLO dự báo ra rất nhiều bounding box trên một bức ảnh nên đối với những cell có vị trí gần nhau, khả năng các khung hình bị overlap là rất cao. Trong trường hợp đó YOLO sẽ cần đến non-max suppression để giảm bớt số lượng các khung hình được sinh ra một cách đáng kể



*Hình 31: Non-max suppression.*

Từ 3 bounding box ban đầu cùng bao quanh chiếc xe đã giảm xuống còn một bounding box cuối cùng.

Các bước của non-max suppression:

* Step 1: Đầu tiên chúng ta 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 xuất chứa vật thể là lớn nhất. Sau đó tính toán chỉ số giao thoa IoU với các bounding box còn lại.

Nếu chỉ số này lớn hơn ngưỡng threshold thì điều đó chứng tỏ 2 bounding boxes đang overlap nhau rất cao. Ta sẽ xóa các bounding có có xác xuất thấp hơn và giữ lại bounding box có xác suất cao nhất. Cuối cùng, ta thu được một bounding box duy nhất cho một vật thể.

# **Chương III - Kết quả thực nghiệm**

## 3.1 Mô tả về dữ liệu

### 3.1.1 Bộ dữ liệu GTSDB

Định dạng file hình ảnh:

* Hình ảnh được chụp từ môi trường thực tế ở Đức, gồm nhiều vật thể khác trong bức hình bao gồm người, xe, cây cối và các biển báo giao thông.
* 900 hình ảnh (chia thành 600 hình ảnh đào tạo và 300 hình ảnh đánh giá), kích thước ảnh của các bức ảnh là như nhau 1360 \* 880 (pixel).



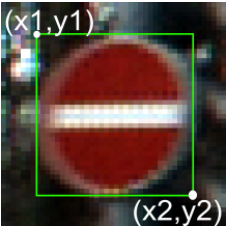
*Hình 32: Hình ảnh biển báo của bộ GTSDB*

* Mỗi hình ảnh chứa không đến sáu biển báo giao thông
* Hình ảnh được lưu trữ ở định dạng PPM
* Kích thước của các biển báo giao thông trong hình ảnh thay đổi từ 16x16 đến 128x128 (pixel)
* Biển báo giao thông có thể xuất hiện ở mọi góc độ và mọi điều kiện ánh sáng

Định dạng flie chú thích:

Chú thích được cung cấp trong tệp gt.txt cho 900 ảnh. Các trường được phân tách bằng dấu ";" (dấu chấm phẩy). Chú thích chứa các thông tin sau:

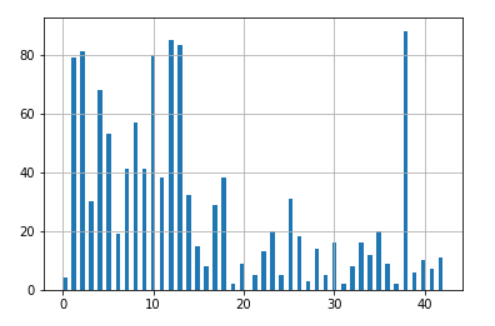
* Tên tệp: Tên tệp của hình ảnh tương ứng
* Width: Chiều rộng của hình ảnh
* Height: Chiều cao của hình ảnh
* ROI.x1: Tọa độ X của góc trên bên trái của hộp giới hạn biển báo giao thông
* ROI.y1: Tọa độ Y của góc trên bên trái của hộp giới hạn biển báo giao thông
* ROI.x2: Tọa độ X của góc dưới bên phải của hộp giới hạn biển báo giao thông
* ROI.y2: Tọa độ Y của góc dưới bên phải của hộp giới hạn biển báo giao thông
* ClassID: Nhãn lớp được chỉ định



*Hình 33: Tọa độ ROI của biển báo.*



*Hình 34: Hình ảnh của 43 loại biển báo trong tập dữ liệu.*

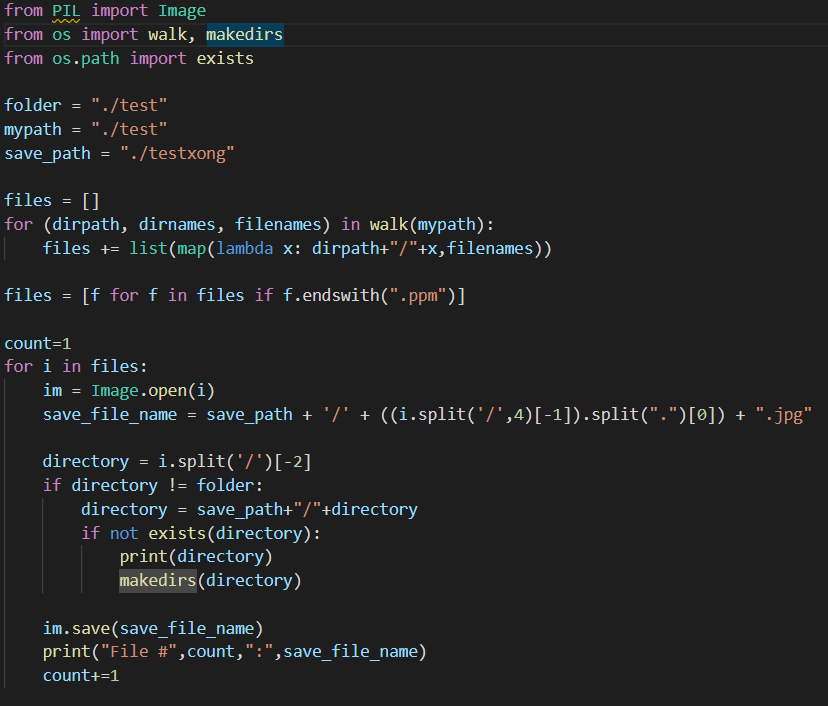


*Hình 35: Sự phân bố dữ liệu của 43 loại biển báo trong tập dữ liệu*

Qua hình trên ta thấy sự phân bố dữ liệu của 43 loại biển báo là không đồng đều, có 1 số loại biển báo có tần suất xuất hiện rất lớn, 1 số loại thì xuất hiện rất ít nên sẽ ảnh hưởng tới kết quả của quá trình huấn luyện.

## 3.2 Mô tả thực nghiệm

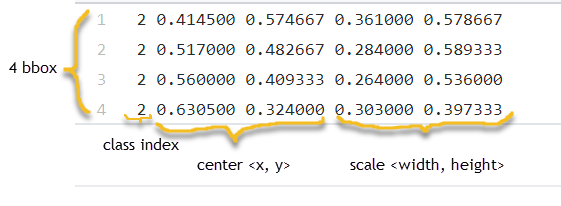
### 3.2.1 Các thức thực hiện thực nghiệm

**Bước 1**: Ta tiến hành chuyển đổi lần lượt các bức ảnh từ định dạng PPM sang định dạng file JPG

*Hình 36: Chuyển ảnh từ định dạng PPM sang định dạng file JPG*

**Bước 2**: Xây dựng lại các file chú thích phù hợp với các hình ảnh đã được chuyển đổi từ bước 1 để có thể huấn luyện được model YOLO.

Khi huấn luyện model YOLO trên darknet chúng ta sẽ cần sử dụng đầu vào là các bức ảnh (có thể là một trong các định dạng png, jpg, jpeg) và file chú thích của chúng (định dạng txt). Bên dưới là nội dung của một file chú thích



*Hình 37: Nội dung file annotation*

Nội dung của file annotation sẽ bao gồm:

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

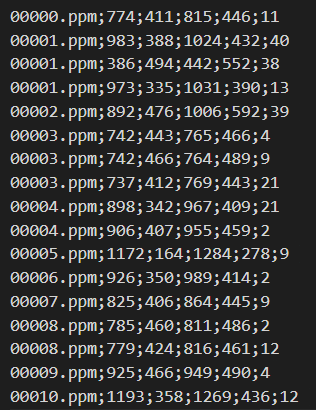
Trong đó: các giá trị <center-x> <center-y> <bbox-width> <bbox-height> là tâm và kích thước width, height của bounding box đã được chuẩn hóa bằng cách chia cho width và height của ảnh, do đó các giá trị ngày luôn nằm trong khoảng [0, 1].

<id-class> là giá trị index đánh dấu các classes.

Trong trường hợp một ảnh có nhiều bounding box thì file annotation sẽ gồm nhiều dòng, mỗi một bounding box là một dòng.

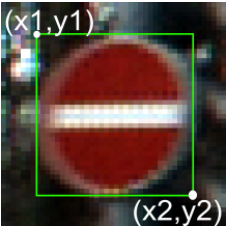
Cảc ảnh và annotation phải được để chung trong cùng 1 folder.

Trong khi đó file gt.txt của chúng ta thì chưa được định dạng đúng và chứa các thông tin sau:



*Hình 38: Nội dung file gt.txt*

* Tên tệp: Tên tệp của hình ảnh tương ứng
* Width: Chiều rộng của hình ảnh
* Height: Chiều cao của hình ảnh
* ROI.x1: Tọa độ X của góc trên bên trái của hộp giới hạn biển báo giao thông
* ROI.y1: Tọa độ Y của góc trên bên trái của hộp giới hạn biển báo giao thông
* ROI.x2: Tọa độ X của góc dưới bên phải của hộp giới hạn biển báo giao thông
* ROI.y2: Tọa độ Y của góc dưới bên phải của hộp giới hạn biển báo giao thông
* ClassID: Nhãn lớp được chỉ định



*Hình 39: Tọa độ ROI của biển báo.*

Từ toạ độ (x1,y1) và (x2,y2) ở trên chúng ta cần phải chuyển đổi về tọa độ tâm, tỉ lệ chiều dài, chiều rộng tương ứng của bounding box so với ảnh gốc có chứa biển báo:

Công thức chuyển đổi

x\_tâm = ((x2 – x1) / 2 + x1) / 1360

y\_tâm = ((y2 – y1) / 2 + y1) / 800

width = (x2 – x1) / 1360

height = (y2 - y1) / 800

trong đó bức ảnh có kích thước 1360\*800

**Bước 3**: Cấu hình các tham số trong mạng YOLO cho phù hợp với số lượng các loại biển báo, kích thước đầu vào của hình ảnh và số bước huấn luyện.

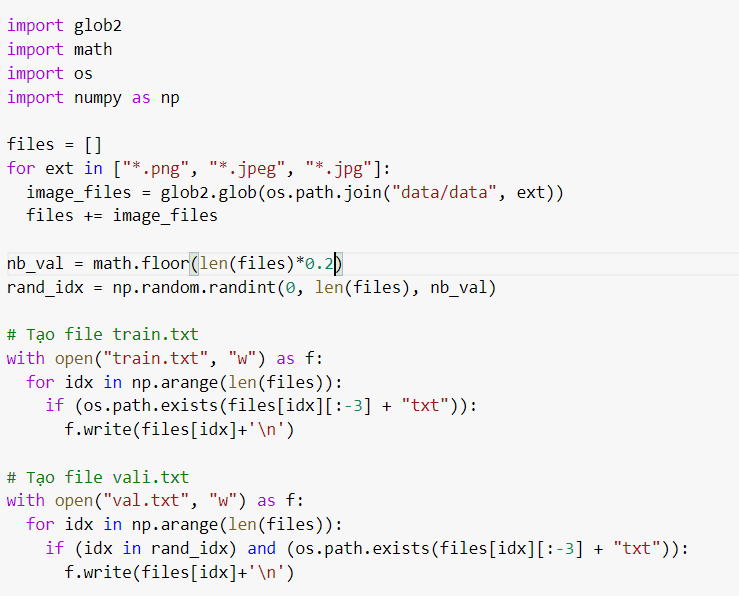
Chúng ta sẽ huấn luyện mô hình trên mạng YOLO version 3, đầu tiên chúng ta cần tải về bộ mã nguồn Darknet về và chỉnh sửa file cấu hình cho phù hợp với mạng YOLO version 3.Đây là bước quan trọng nhất khi huấn luyện model YOLO. Chúng ta sẽ sử dụng file [YOLOv3.cfg](https://github.com/pjreddie/darknet/blob/master/cfg/yolov3.cfg) để cấu hình mô hình huấn luyện. Các bạn download file trên về máy và điều chỉnh các dòng:

* Tại các dòng 610, 696, 783: Thay classes=80 thành classes=43 là số lượng classes chúng ta huấn luyện.
* Tại các dòng 603, 689, 776: Thay số lượng filters=255 về filter=144. Đây chính là layer cuối cùng của base network. Do đó chúng có output shape thay đổi theo số lượng classes theo đúng công thức của bài trước đó là: (n\_classes + 5)x3 = (43+5)x3 = 144.
* max\_batches: tại dòng 20 là số lượng steps tối đa để huấn luyện models YOLO. Đối với dữ liệu 43 classes thì cần điều chỉnh max\_batches=2000 \* n\_classes=43\*2000=86000
* burn\_in: Tại dòng 19 là số lượng steps ban đầu được giữ sao cho learning\_rate rất bé. Giá trị này sẽ tăng dần từ 0 đến learning\_rate. Sau đó learning\_rate sẽ được giữ ổn định. Thực nghiệm cho thấy thiết lập learning\_rate bé ở những steps đầu sẽ giúp cho thuật toán hội tụ nhanh hơn. Do số lượng max\_batches là 86000 nên cần điều chỉnh giảm burn\_in = 1000.
* steps: Tại dòng 22. Điều chỉnh về steps=68800,77400. Đây là các vị trí step mà chúng ta sẽ bắt đầu giảm dần learning\_rate vì thuật toán đã đạt tới điểm hội tụ nên không cần thiết lập learning\_rate quá cao.
* Width, height, subdivisions, batch: Tại dòng 6, 7, 8, 9. Điều chỉnh về batch=64, subdivisions=32, width=416, height=416. Đây là các vị trí mà chúng ta giảm đầu vào bức ảnh, số lượng hình ảnh được đưa vào mỗi bước huấn luyện sao cho không bị tràn RAM trong quá trình huấn luyện trên Google Colab.

Sau khi thực hiện các thay đổi xong, các bạn lưu file lại và hãy đổi tên lại thành YOLOv3-5c-5000-maxsteps.cfg để đánh dấu đây là cấu hình cho YOLO version 3 với 43 classes và 86000 bước huấn luyện.

**Bước 4**: Phân chia dữ liệu train/validation

Ở bước này ta sẽ tạo ra 2 file train.txt và valid.txt chứa dữ liệu đường dẫn tới các file hình ảnh nằm trong tập train và validation. Chúng ta sẽ sử dụng đoạn code bên dưới để lựa chọn ra ngẫu nhiên 20% files làm dữ liệu validation và các files còn lại làm dữ liệu train.



*Hình 40: Chia bộ dữ liệu thành tập train và tập valid*

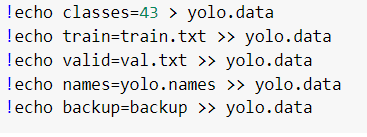
**Bước 5**: Tạo file YOLO.name gồm tên gọi tiếng anh của 43 loại biển báo

**Bước 6**: Tạo file YOLO.data

File YOLO.data sẽ khai báo một số thông tin như:

* Số lượng classes
* Đường dẫn tới các file train.txt, valid.txt
* Đường dẫn tới file YOLO.names
* Thư mục backup mô hình huấn luyện.

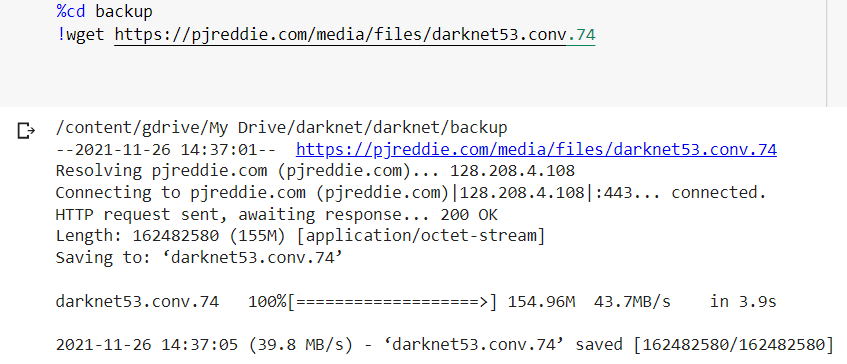
Chạy lệnh bên dưới để tạo file này.



*Hình 41: Tạo file yolo.data*

**Bước 7**: Tải file pretrain model:

Ở bước này, chúng ta sẽ sử dụng pretrain model [darknet53.conv.74](https://pjreddie.com/media/files/darknet53.conv.74) được huấn luyện từ bộ dữ liệu ImageNet. Trước tiên hãy clone file weight về google drive.



*Hình 42: Tải file pretrain model darknet53.conv.74*

**Bước 8**: Backup model:

Tạo một folder backup để lưu kết quả huấn luyện. Folder backup này phải trùng với tên với link folder backup đã được khai báo ở bước tạo file YOLO.data.

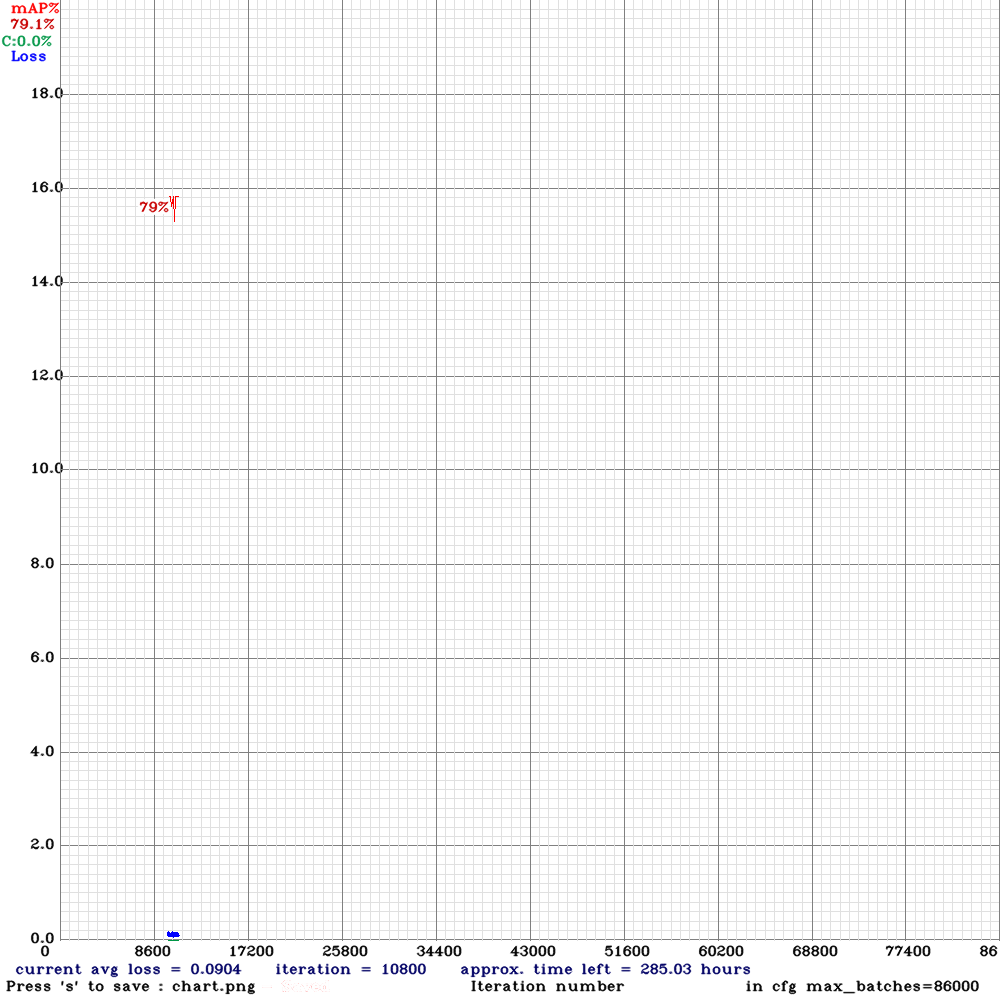
**Bước 9:** Huấn luyện bộ dữ liệu trên mạng YOLO version 3:

Chúng ta chạy lệnh dưới đây để huấn luyện mô hình:

!./darknet detector train YOLO.data cfg/YOLOv3-43c-86000-max-steps.cfg darknet53.conv.74 -dont\_show -map > YOLOv3-43c.log

## 3.3 Kết quả thực nghiệm

Đánh giá mô hình:



*Hình 43: Bảng kết quả đánh giá độ chính xác mô hình.*

Đồ thị loss function cho thấy thuật toán đã hội tụ sau khoảng 10000 batches. Loss function ở giai đoạn sau có xu hướng tiệm cận 0. Điều này chứng tỏ chiến lược lựa chọn learning\_rate nhỏ ở 1000 steps đầu tiên đã phát huy hiệu quả giúp thuật toán hội tụ nhanh hơn và độ chính xác trung bình của mô hình là 79.1%.

Thử nghiệm trên ảnh test:

****

*Hình 44: Ảnh dùng để kiểm tra*

Ta thấy model YOLO của chúng ta đã nhận diện được vật thể rất chính xác.

## 3.4 Nhận xét

Quá trình huấn luyện mô hình YOLO mất rất nhiều thời gian, trong khi đó Google colab sẽ chỉ cho phép bạn huấn luyện trong 6 giờ liên tục.

Hãy lưu lại file log trong quá trình huấn luyện và tải về biểu đồ loss function. Biểu đồ loss function cho ta biết quá trình huấn luyện đã đi tới trạng thái hội tụ hay chưa. Có thể dừng sớm quá trình huấn luyện nếu bạn quan sát thấy loss function dường như đã hội tụ

# **Kết Luận**

Qua đề tài nhận diện biển báo giao thông bằng thuật toán YOLO, nhóm của chúng em đã học được rất nhiều bài học có ứng dụng thực tiễn cao như tìm hiểu về các kiến thức tiền xử lý, thuật toán YOLO và đặc biệt là cách huấn luyện mô hình YOLO trên Google Colab. Quá trình huấn luyện mạng YOLO đòi hỏi công sức và thời gian bỏ ra rất lớn mà giai đoạn quan trọng nhất đó chính là việc tiền xử lý dữ liệu. Trong quá trình làm bài, nhóm đã biết cách tiền xử lý dữ liệu đầu vào, chuyển đổi định dạng file hình ảnh và các file cấu hình để huấn luyện được mạng YOLO trên Google Colab.

* Về thuật toán YOLO

Ưu điểm: YOLO là một mô hình để nhận diện vật thể với ưu điểm nổi trội là nhanh hơn nhiều so với những mô hình cũ. Thậm chí có thể chạy tốt trên những thiết bị IOT device như raspberry pi.

Nhược điểm:

* Do mô hình YOLO có nhiều tầng học sâu nên tốn nhiều thời gian để training nên chưa tối ưu

Ứng dụng thực tế: Hiện tại ứng dụng nhận diện biển báo giao thông đang góp một phần không nhỏ trong cuộc sống hằng ngày.Mục đích chính là nhằm tăng năng suất, sức lực cho con người, giúp con người có thể di chuyển một cách nhanh chóng và an toàn hơn.

Đã làm được:

* Xây dựng được một chương trình đơn giản để nhận dạng biển báo giao thông
* Biết được quy trình và cách một chương trình nhận dạng làm việc

Chưa làm được:

* Chương trình còn nhiều hạn chế về phần cứng và bộ dữ liệu

# **Tài liệu tham khảo**

[1] <https://phamdinhkhanh.github.io/2020/03/09/DarknetAlgorithm.html>

[2] <https://phamdinhkhanh.github.io/2020/03/10/DarknetGoogleColab.html>

[3] <https://nttuan8.com/bai-5-gioi-thieu-ve-xu-ly-anh/>

[4] <https://nttuan8.com/bai-6-convolutional-neural-network/>