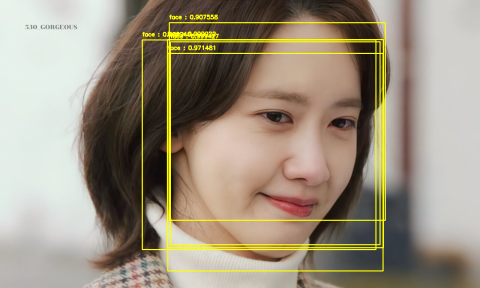
**1. Giới thiệu**

Thông thường output của các mô hình Object detection sẽ có rất nhiều các bounding boxes. Với ouput như vậy thì sẽ có hiện tượng có rất nhiều các bounding box cho cùng một object duy nhất, điều đó gây nên sự dư thừa thông tin khi mục đích của ta chỉ cần duy nhất một bounding box cho một đối tượng.

Chính vì vấn đề đó nên thuật toán Non-Maximum Suppression sinh ra để loại bỏ đi các bounding box dư thừa của cùng một đối tượng tượng trong ảnh.

Dưới đây là một ví dụ output của một mô hình face detection khi chưa được xử lý bằng NMS:

Như các bạn có thể thấy khi chưa áp dụng thuật toán NMS thì có rất nhiều bounding box cho cùng 1 khuôn mặt trong ảnh. Và khi áp dụng NMS thì ta sẽ thu được kết quả như hình dưới đây:

A picture containing person, hairpiece

Description automatically generated

Trong bài viết này tôi sẽ cùng các bạn đi tìm hiểu thuật toán, và triển khai thuật toán này với python nhé 

# 2. Intersection Over Union (IoU)

Trước khi vào tìm hiểu nội dung thuật toán NMS, tôi sẽ cùng các bạn tìm hiểu về IoU trước, lý do là trong thuật toán NMS có sử dụng đến chỉ số IoU này.

IoU là một thông số được sử dụng để đánh giá độ che lấp lên nhau giữa 2 bounding boxes.

Giả sử ta có 2 boxes với thông tin như sau:

* **Box 1** có tọa độ top-left là *x*1​,*y*1​), tọa độ bottom-right là *x*2​,*y*2​).
* **Box 2** có tọa độ top-left là  (*x*3​,*y*3​), tọa độ bottom-right là (*x*4​,*y*4​).

Chart, box and whisker chart

Description automatically generated

Khi đó IoU được tính theo công thứcShape

Description automatically generated with medium confidence

Chart, box and whisker chart

Description automatically generated

***Triển khai với Python***

# Tìm diện tích của 2 box

area1 = (x2-x1)\*(y2-y1);

area2 = (x4-x3)\*(y4-y3);

# Tìm tọa đọ của vùng giao nhau (Intersection)

xx = max(x1, x3)

yy = max(y1, y3)

aa = min(x2, x4)

bb = min(y2, y4)

# Tính diện tích vùng giao nhau

w = max(0, aa - xx)

h = max(0, bb - yy)

intersection\_area = w\*h

# Tính diện tích phần hợp nhau

union\_area = area1 + area2 - intersection\_area

# Dựa trên phần giao và phần hợp để tính IoU

IoU = intersection\_area / union\_area

# 3. Thuật toán Non-Maximum Suppression (NMS)

## 3.1 Nội dung thuật toán

***Input:***

Một mảng các bounding box, mỗi box sẽ có dạng (�1,�1,�2,�2,�)(*x*1​,*y*1​,*x*2​,*y*2​,*c*) trong đó:

* (�1,�1)(*x*1​,*y*1​) và (�2,�2)(*x*2​,*y*2​) lần lượt là tọa độ điểm top-left và bottom-right của bounding box
* �*c* là confidence score tương ứng với box đó, được trả về từ mô hình object detection.

Giá trị ngưỡng IOU.

***Output:***

Một mảng các bounding box sau khi đã được loại bỏ đi các bounding box dư thừa.

***Chi tiết thuật toán***

Các ký hiệu:

* **S**: bounding box đang xét
* **P**: Tập các box đầu vào của thuật toán
* **thresh\_iou**: Ngưỡng IoU để loại bỏ các box thừa
* **keep**: Tập các box sau khi đã loại bỏ các box thừa

Thuật toán bao gồm 3 bước:

* **Bước 1**: Chọn box **S** có confidence score cao nhất trong tập **P**, loại bỏ box đó ra khỏi tập **P** và thêm box đó vào tập **keep**.
* **Bước 2** Thực hiện tính toán IOU giữa box **S** vừa lấy ra ở bước 1 với toàn bộ các box còn lại trong tập **P**. Nếu có box nào trong **P** có IOU với box **S** đang xét mà lớn hơn ngưỡng **thresh\_iou** thì loại bỏ box đó ra khỏi **P**
* **Bước 3** Lặp lại bước 1 cho đến khi **P** không còn box nào.

Sau khi kết thúc thuật toán thì **keep** chứa toàn bộ những box sau khi đã loại bớt các box thừa.

Nhìn vào các bước của thuật toán trên thì các bạn cũng có thể nhìn thấy việc loại bỏ đi các box chỉ phụ thuộc vào giá trị ngưỡng IoU **thresh\_iou**. Do đó việc lựa chọn giá trị ngưỡng này sẽ ảnh hưởng đến hiệu suất của mô hình object detection.

## 3.2 Triển khai NMS với Python

***Khởi tạo hàm***

def nms(P: torch.tensor, thresh\_iou: float):

x1 = P[:, 0]

y1 = P[:, 1]

x2 = P[:, 2]

y2 = P[:, 3]

# confidence score

scores = P[:, 4]

Tương tự với thuật toán đã nêu ra ở trên, input parameters của hàm gồm:

* **P** list các box trả về từ mô hình object detection
* **thresh\_iou**: Ngưỡng IoU để loại bỏ các box dư thừa.

Các biến ***x1 y1 x2 y2*** là tọa độ của tất cả các box trong **P**

Biến **scores**: confidence của tất cả các box trong **P**

***Tính diện tích các boxes trong P và sắp xếp thứ tự các box theo confidence score:***

# Tính diện tích các boxes

areas = (x2 - x1) \* (y2 - y1)

# sắp xếp lại vị trí các box theo confidence score

order = scores.argsort()

keep = []

***Tìm box có confidence score cao nhất trong tập P***

while len(order) > 0:

idx = order[-1] # lấy index của box có confidence cao nhất

keep.append(P[idx]) # đưa box đó vào tập keep

order = order[:-1] # Xóa box đó ra khỏi tập P

***Lấy tọa độ của các box còn lại trong P:***

xx1 = torch.index\_select(x1, dim=0, index=order)

xx2 = torch.index\_select(x2, dim=0, index=order)

yy1 = torch.index\_select(y1, dim=0, index=order)

yy2 = torch.index\_select(y2, dim=0, index=order)

***Tìm diện tích vùng intersection giữa box S và các box còn lại trong P:***

# Tìm tọa độ của vùng intersection

xx1 = torch.max(xx1, x1[idx])

yy1 = torch.max(yy1, y1[idx])

xx2 = torch.min(xx2, x2[idx])

yy2 = torch.min(yy2, y2[idx])

# Tính chiều cao và chiều rộng của vùng intersection

w = xx2 - xx1

h = yy2 - yy1

# Loại bỏ đi các vùng có chiều cao và chiều rộng âm (giá trị âm do 2 box không có phần giao nhau)

w = torch.clamp(w, min=0.0)

h = torch.clamp(h, min=0.0)

***Tính toán Iou giữa box S và các box còn lại trong P***

rem\_areas = torch.index\_select(areas, dim = 0, index = order)

union = (rem\_areas - inter) + areas[idx]

IoU = inter / union

***Loại bỏ các box có IoU lớn hơn ngưỡng thresh\_iou***

Sau khi đã tính toán được giá trị IoU giữa box S và các box ta sẽ loại đi các box có IoU lớn hơn ngưỡng **thresh\_iou**

mask = IoU < thresh\_iou

order = order[mask]

Toàn bộ code của hàm này như sau:

import torch

def nms(P: torch.tensor, thresh\_iou: float):

x1 = P[:, 0]

y1 = P[:, 1]

x2 = P[:, 2]

y2 = P[:, 3]

scores = P[:, 4]

areas = (x2 - x1) \* (y2 - y1)

order = scores.argsort()

keep = []

while len(order) > 0:

idx = order[-1]

keep.append(P[idx])

order = order[:-1]

xx1 = torch.index\_select(x1, dim=0, index=order)

xx2 = torch.index\_select(x2, dim=0, index=order)

yy1 = torch.index\_select(y1, dim=0, index=order)

yy2 = torch.index\_select(y2, dim=0, index=order)

xx1 = torch.max(xx1, x1[idx])

yy1 = torch.max(yy1, y1[idx])

xx2 = torch.min(xx2, x2[idx])

yy2 = torch.min(yy2, y2[idx])

w = xx2 - xx1

h = yy2 - yy1

w = torch.clamp(w, min=0.0)

h = torch.clamp(h, min=0.0)

inter = w\*h

rem\_areas = torch.index\_select(areas, dim = 0, index = order)

union = (rem\_areas - inter) + areas[idx]

IoU = inter / union

mask = IoU < thresh\_iou

order = order[mask]

return keep

## 3.3 Thử nghiệm

Bên trên tôi đã cùng các bạn xây dựng hàm cho thuật toán NMS. Giờ chúng ta cùng thử nghiệm xem nó có hoạt động đúng không nhé.

P = torch.tensor([

[1,8,7,2,0.7],

[2,6,7,1,0.75],

[3,7,6,2,0.91],

[8,5,11,2,0.76],

[9,7,12,3,0.93],

[10,8,13,4,0.77]

])

Biến P bên trên là đầu vào của thuật toán NMS, ở đây tôi lấy ví dụ với 6 boxes khác nhau. Tọa độ của 6 box này tôi thể hiện như ở hình dưới đây.

Chart, box and whisker chart

Description automatically generated

Lưu ý: Phần code phía trên tôi sử dụng là để dành cho hệ tọa độ của ảnh, tức là gốc tọa độ nằm ở điểm **top-left** của ảnh. Tuy nhiên trong ví dụ này tôi lại lấy gốc tọa độ ở điểm **bottom-left**. Vậy nên nếu các bạn muốn thử nghiệm với phần dữ liệu tôi tạo ở phía trên thì sửa đoạn code ở chỗ này nhé:

x1 = P[:, 0]

y2 = P[:, 1]

x2 = P[:, 2]

y1 = P[:, 3]

Và sau khi chạy thuật toán trên thì kết quả chỉ còn lại 2 boxes như hình vẽ dưới đây:Chart, box and whisker chart

Description automatically generated

# 4. Kết luận

Như vậy là trong bài viết ngăn ngắn này tôi đã cùng các bạn tìm hiểu về các khái niệm như IoU và thuật toán NMS, cùng với đó là việc triển khai lại chúng với Python. Thuật toán này khá là đơn giản tuy nhiên nó được ứng dụng rất nhiều trong các phần hậu xử lý kết quả của mô hình Object Detection. Hầu như các bạn sẽ gặp thuật toán này trong các source code implement các mô hình như Yolo, SSD...

Nếu các bạn thấy bài viết này giúp cho các bạn một chút gì đó cho bạn thì đừng quên cho tôi xin 1 upvote nhé 😍. Cảm ơn các bạn 🤗🤗

# Tham khảo

* [Non Maximum Suppression: Theory and Implementation in PyTorch](https://learnopencv.com/non-maximum-suppression-theory-and-implementation-in-pytorch/)
* [IOU (Intersection over Union)](https://medium.com/analytics-vidhya/iou-intersection-over-union-705a39e7acef)