Week 2: Tuple, Set and Dictionary

Time-Series Team

Ngày 16 tháng 6 năm 2025

Hôm trước chúng ta đã học kiểu dữ liệu cấu trúc đầu tiên - List, trong bài này, chúng ta sẽ tìm hiểu các kiểu dữ liệu cấu trúc còn lại và ứng dụng của chúng trong AI. Nội dung bài học bao gồm:

- ***Phần I: Tuple trong Python***
- ***Phần II: Set trong Python***
- ***Phần III: Dictionary trong Python***
- ***Phần IV: Ứng dụng của Data Structure trong AI***
- ***Phần V: Mở rộng***

Trước khi vào tìm hiểu các loại kiểu cấu trúc dữ liệu chúng ta cần tìm hiểu tính chất quan trọng trong Data Structure:

- **Mutable (Khả biến)**: Những đối tượng này có thể thay đổi sau khi khởi tạo. Nhóm dữ liệu này bao gồm **list, set** và **dict** và các loại kiểu dữ liệu do người dùng định nghĩa (phụ thuộc vào các hành vi do người dùng định nghĩa)
- **Immutable (Bất biến)**: Những đối tượng này hoàn toàn không thể thay đổi sau khi khởi tạo. Nhóm dữ liệu này gồm **tuple,** một số kiểu dữ liệu built-in type như **number**, **string**,.. và các loại kiểu dữ liệu do người dùng định nghĩa (phụ thuộc vào các hành vi do người dùng định nghĩa).

Phần I: Tuple trong Python

1 Tuple là gì?

Tuple là một dạng dãy phần tử giống như list, nhưng không thể thay đổi sau khi đã khởi tạo. Điều này giúp tuple hoạt động hiệu quả hơn và an toàn hơn trong một số tình huống.

- Được khai báo bằng dấu ngoặc tròn ()
- Tuple là immutable, không thể thay đổi nội dung sau khi tạo.
- Có thể chứa nhiều kiểu dữ liệu và trùng lặp phần tử.
- Thường dùng cho dữ liệu cố định như toạ độ, cấu hình.

2 Thao tác với Tuple

Để kiểm tra các phương thức và hàm tích hợp có thể sử dụng trên tuple, hãy dùng hàm **dir(tuple())**.

2.1 Khai báo tuple

```
members = ("Dung","Van","Duc")
```

Nếu bạn khai báo tuple có 1 phần tử, hãy chú ý dấu phẩy, nếu không có dấu phẩy thì đây là định nghĩa một value của biến bình thường.

```
members = ("Dung",)
```

2.2 Một số phương thức và hàm built-in trên tuple

Tuple là một danh sách đặc biệt, không thể thay đổi khi đã tạo ra, do đó bạn có thể sử dụng tất cả các kỹ thuật, các hàm như với List nhưng loại trừ những hàm tác động thay đổi nội dung.

Ví dụ, bạn có thể sử dụng toán tử in, hàm len() với Tuple:

```
numbers = (1,2,3)
check = 3 in numbers
print(check)
```

=> Kết quả là True

```
numbers = (1,2,3)

print(f"Tuple has {len(numbers)} items")
```

=> Kết quả là 3

Tất cả các phương thức .append(), .extend(), .clear(), .copy(), .insert(), .pop(), .remove(), .reverse(), .sort() không sử dụng được với cấu trúc dữ liệu Tuple.

Chú ý: Không thể thay đổi Tuple nhưng có thể tạo ra một Tuple từ hai Tuple, ví dụ :

```
numbers = (1, 2, 3)
numbers = numbers + (9,)

print(numbers)
```

=> Kết quả là (1, 2, 3, 9)

Phần II: Set trong Python

3 Set là gì?

Set là một tập các phần tử không có thứ tự và không chứa phần tử trùng lặp. Đây là lựa chọn tốt khi muốn lưu trữ các giá trị duy nhất.

- Khai báo bằng dấu ngoặc nhọn hoặc hàm set().
- Tự động loại bỏ phần tử trùng.
- Không thể truy cập bằng chỉ số
- Hỗ trợ các phép toán tập hợp: hợp, giao, hiệu,...

4 Thao tác với Set

4.1 Khai báo Set

```
members = {"Dung","Van","Duc"}
```

4.2 Một số phép toán và method trên Set

Các phần tử trong tập hợp có thể thêm hoặc loại bỏ. Python hỗ trợ các phương thức để thực hiện thao tác thay đổi tập hợp.

Phương thức .add() : Thêm một phần tử vào tập hợp

```
members = {"Dung","Van","Duc"}
members.add("Huy")
print(friends)
```

=> Kết quả là {"Duc","Van","Huy","Dung"}

Chú ý: kết quả có thể khác đi do Set không sắp xếp các phần tử theo một trật tự nào cả.

Phương thức .remove():Loại bỏ một phần tử trong tập hợp.

```
members = {"Dung","Van","Duc"}
members.remove("Dung")
print(members)
```

=> Kết quả là {"Van","Duc"}

Lưu ý: Khi loại bỏ một phần tử, nếu phần tử đó không tồn tại trong tập hợp, chương trình sẽ dừng và một thông báo lỗi KeyError xuất hiện.

Phương thức .discard(): Giống như phương thức .remove() loại bỏ phần tử trong tập hợp, tuy nhiên nếu phần tử đó không tồn tại thì nó không báo lỗi gì cả.

Phương thức .pop(): Loại bỏ một phần tử ngẫu nhiên khỏi tập hợp.

Phương thức .clear(): Loại bỏ tất cả các phần tử trong tập hợp, khi đó tập hợp được gọi là tập rỗng.

Phương thức .update()

Phương thức .add() ở trên chỉ thêm được 1 phần tử vào tập hợp với 1 câu lệnh, để thêm nhiều phần tử, chúng ta sử dụng .update().

Chú ý, đầu vào của .update() có thể là một Set, một List hoặc một Tuple. Ngoài ra, không sử dụng chuỗi để cập nhập vào tập hợp mà các phần tử là chuỗi, bởi vì chuỗi sẽ được coi là một danh sách các ký tự, ví dụ:

```
members = {"Linh","Tung","Ha"}
members.update("Phuong")
print(members)
```

=> Kết quả là {'u', 'n', 'o', 'Ha', 'h', 'Linh', 'P', 'Tung', 'g'}

Phép hợp (Union): Hợp của hai tập hợp cho kết quả là tất cả các phần tử trong hai tập hợp, chú ý phần tử nào lặp lại sẽ chỉ xuất hiện 1 lần trong tập kết quả.

```
python_class = {"Dung", "Van", "Duc"}
php_class = {"Huy", "Dung"}
all_class = python_class.union(php_class)
print(all_class)
```

=> Kết quả là {'Duc', 'Van', 'Dung', 'Huy'}

Phép trừ (Difference): Hiệu của một tập A trừ đi một tập B cho kết quả là tất các phần tử thuộc A nhưng không thuộc B.

```
python_class = {"Dung", "Van", "Duc"}
c_class = {"Huy", "Dung"}
python_but_not_c = python_class.difference(php_class)
c_but_not_python = php_class.difference(python_class)

print(python_but_not_c)
print(c_but_not_python)
```

=> Kết quả là {'Duc', 'Van'} và {'Huy'}

Hiệu đối xứng của hai tập hợp (Symmetric difference): Hiệu đối xứng của hai tập A và B được kết quả là tập hợp các phần tử thuộc cả A và B nhưng không đồng thời thuộc cả tập A và B.

```
python_class = {"Dung", "Van", "Duc"}
c_class = {"Huy", "Dung"}

not_in_both_1 = python_class.symmetric_difference(c_class)
print(not_in_both_1)

not_in_both_2 = c_class.symmetric_difference(python_class)
print(not_in_both_2)
```

=> Kết quả là {'Huy', 'Duc', 'Van'}

Lưu ý: do tính chất đối xứng nên kết quả dùng set nào trước cũng sẽ ra kết quả như nhau.

Phép giao (Intersection): Phép giao hai tập hợp cho kết quả là các phần tử đồng thời thuộc cả hai tập hợp.

```
python_class = {"Dung", "Van", "Duc"}
c_class = {"Huy", "Dung"}

python_c = python_class.intersection(c_class)
print(python_c)
```

=> Kết quả là {'Dung'}

Thay đổi tập hợp dựa trên phép toán tập hợp:Python cho phép kết hợp phương thức update() và các phép toán tập hợp để thay đổi tập hợp, ở các phương thức này sẽ thực hiện phép tập hợp trước, rồi update vào tập hợp đích, ví du:

```
A = {1, 2, 3, 4}
B = {3, 4, 5, 6}

A.difference_update(B)
print(A)
```

=> Kết quả là {1, 2}

Ngoài ra cũng có một số phép toán hay ho khác với set như:

- .isdisjoint() : Trả về True nếu hai tập hợp không giao nhau (không có phần tử chung)
- .issubset(): Trả về True nếu tập này là tập con của tập đích (tập trong ngoặc).
- .issuperset(): Trả về True nếu tập này là tập cha của tập đích (tập trong ngoặc).

Phần III: Dictionary trong Python

5 Dictionary là gì?

Dictionary là một cấu trúc ánh xạ giữa các key và value. Đây là kiểu dữ liệu rất mạnh trong Python khi cần truy cập nhanh theo khoá định danh.

- Khai báo bằng dấu ngoặc nhọn với các cặp key: value.
- Key là duy nhất và không thể thay đổi (immutable).
- Value có thể là bất kỳ kiểu dữ liệu nào.
- Hỗ trợ thêm, cập nhật, xoá, truy cập theo khoá.

6 Thao tác với Dict

6.1 Khai báo Dict

Chú ý: Trong Dict key phải đảm bảo tính duy nhất.

$$\texttt{dictionary\_name} = \{\texttt{key}_1 \ : \ value_1, ..., key_n \ : \ value_n\}$$

Dictionary có thể truy xuất giá trị từng phần tử thông qua các khóa theo cú pháp:

```
dictionary_name[key_x]
```

Dictionary trong Python có thể thay đổi thêm bớt các phần tử, thay đổi giá trị của các phần tử hiện có.

Xóa phần tử khỏi từ điển: Cũng giống như List, Dictionary có rất nhiều các phương thức có sẵn để có thể xóa một phần tử khỏi từ điển như .clear(), .pop(), .popitem() hoặc sử dụng từ khóa del.

Thêm và cập nhật phần tử: Dictionary truy xuất thông qua khóa, do vậy khi gán giá trị cho một phần tử: Khóa tồn tại thì giá trị phần tử được cập nhật. Khóa không tồn tại thì Dictionary được thêm một phần tử có khóa và giá trị trong câu lệnh.

6.2 Một số Method với Dict

Phương thức .copy(): Phương thức này trả về một bản copy riêng biệt.

Phương thức .get(): Trả về giá trị một phần tử trong Dictionary với một khóa cho trước, nó giống với truy xuất giá trị nhưng có một khác biệt nhỏ, nếu khóa không tồn tại, chương trình sẽ không báo lỗi mà trả về giá trị None.

```
friend_ages = {"Rolf": 24, "Adam": 30, "Anne": 27}

print(friend_ages.get("FirebirD"))
print(friend_ages.get("Rolf"))
```

=> Kết quả là: None 24

Phương thức .keys() và .values(): Hai phương thức này trả về danh sách khóa và danh sách các giá trị của từ điển.

```
friend_ages = {"Rolf": 24, "Adam": 30, "Anne": 27}

print(friend_ages.keys())
print(friend_ages.values())
```

=>Kết quả là:
dict_keys(['Rolf', 'Adam', 'Anne'])
dict_values([24, 30, 27])

Phần IV: Ứng dụng của Data Structure trong AI

7 Set in Text classification

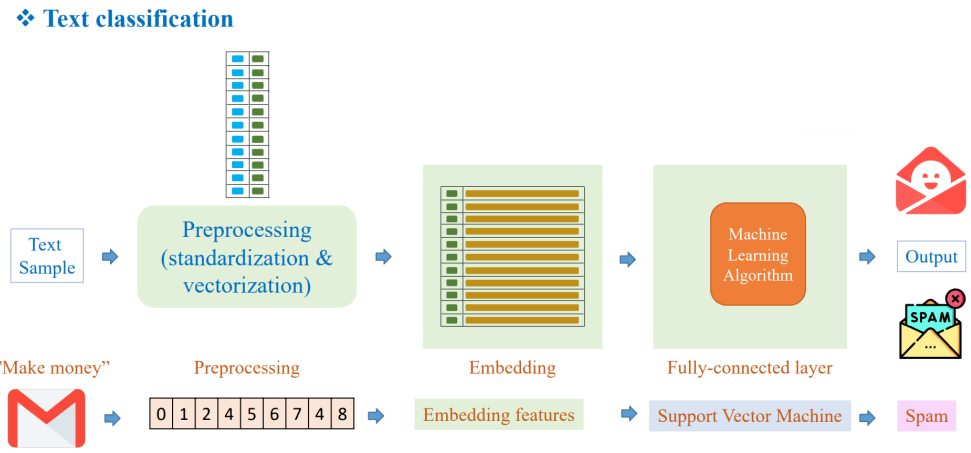
7.1 Tổng quan bài toán

Trong thực tế, có rất nhiều ứng dụng liên quan đến Text Classification như : phân tích bình luận người dùng, tự động phát hiện thư rác (spam email)...

Điểm chung của các bài toán này chính là ngôn ngữ tự nhiên(text) là input. Và như chúng ta đã biết, máy học làm việc với dữ liệu số chứ không phải ngôn ngữ tự nhiên như con người. Vậy vấn đề là làm sao từ tập dữ liệu text chúng ta có thể chuyển thành số để đưa vào học máy huấn luyện đây?

Ý tưởng ở đây là ứng dụng các kiểu dữ liệu cấu trúc để thực hiện chuyển bộ dữ liệu từ text -> number, quá trình này được gọi là Tiền xử lý dữ liệu.

Quy trình giải quyết bài toán được minh họa ở hình dưới đây



7.2 Tiền xử lý dữ liệu: chuyển text -> number

với mục tiêu chuyển text -> number, chúng ta hãy nhớ lại những kiến thức về các kiểu cấu trúc dữ liệu đã học nhé. Để chuyển text -> number đưa vào học máy thì ta cần đảm bảo rằng :

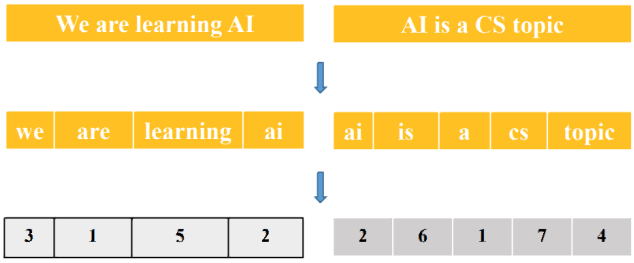
- ứng với mỗi text ta có 1 index/key duy nhất
- các items text là không được trùng nhau

Từ 2 điều kiệu tiên quyết trên, ta nghĩ ngay đến set và dictionary đúng không nào? Set có thể tự động xóa thành phần trùng lặp, còn Dict thì lại ánh xạ key:value duy nhất. Từ điều này, chúng ta sẽ thực hiện việc chuyển text thành number với 2 bước chính sau:

- Bước 1: Xây dựng bộ từ điển từ tập dữ liệu gốc, ví dụ:

index	0	1	2	3	4	5	6	7
word	pad	are/a	ai	we	topic	learning	is	cs

- Bước 2: Xử lý dữ liệu gồm: tách từ đơn, ánh xạ text -> number tương ứng.



7.3 Thực chiến

Sau khi hiểu được ý tưởng và mường tượng cách thực hiện, hãy bắt tay cùng chúng mình code thử nghiệm nhé. Liệu rằng ý tưởng trên có thực hiện được không?

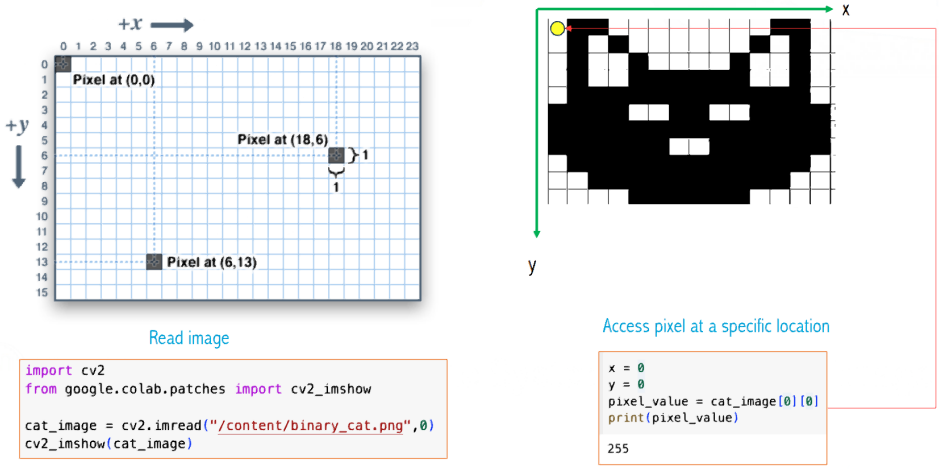
Các bạn có thể xem code phần này tại: folder [Code_sample]

8 Image in Computer Vision

8.1 Tổng quan bài toán

Trong lĩnh vực thị giác máy tính, chúng ta thường thấy đầu vào của bài toán là những bức ảnh(xám hoặc màu). Mỗi bức ảnh xám được cấu tạo từ 1 ma trận 2 chiều với giá trị pixel chạy từ 0-255, đối với ảnh màu thì sẽ được cấu tạo từ 3 ma trận 2 chiều ứng với 3 màu R-G-B.

Xét một bức ảnh nhị phân(đen và trắng), ta sẽ có ma trận vị trí pixel như hình:

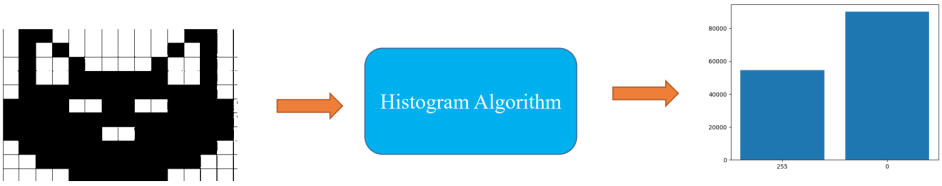


Như các bạn thấy, mỗi pixel có vị trí tọa độ duy nhất và giá trị value tương ứng. Để truy xuất giá trị value tại 1 pixel ta sử dụng index trong 2D list.

Nhưng bài toán ở đây không chỉ là truy xuất giá trị value của pixel mà là tính toán histogram của bức ảnh. Vậy Histogram là gì? Và tại sao cần tính toán histogram?

Histogram (biểu đồ tần suất): là dạng biểu đồ dùng để hiênt thị phân bố của dữ liệu. ví dụ, với bức ảnh nhị phân cat bên trên nó sẽ đếm có bao nhiêu pixel có giá trị 0(đen), có bao nhiêu pixel có giá trị 255(trắng) và hiển thị lên biểu đồ.

Tại sao lại tính histogram?



Việc tính Histogram có thể cho ta biết tình trạng của bức ảnh. Ví dụ:

- Nếu histogram thể hiển giá trị 0 nhiều hơn giá trị 255 -> bức ảnh ảnh rất tối
- Nếu histogram thể hiện giá trị 255 nhiều hơn giá trị 0 -> bức ảnh đang rất sáng

=> Điều này có ích để chúng ta biết rằng mình nên làm gì tiếp theo để tiền xử lý bức ảnh trước khi đưa nó vào mô hình máy học.

8.2 Thực hành

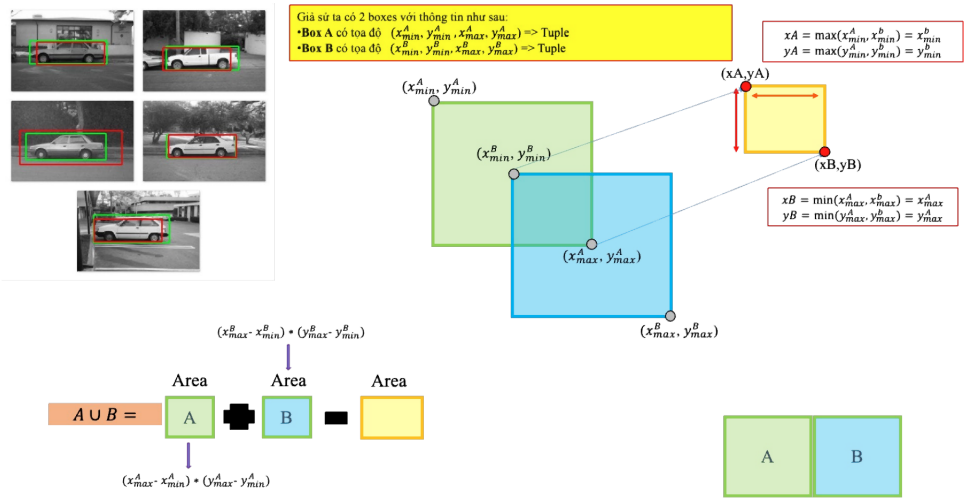
Các bạn có thể tìm thấy code cho phần này tại mục : [./Code_sample/Image Histogram.ipynb]

9 IOU Calculation Using Tuple

9.1 Tổng quan bài toán

Trước khi đi vào cụ thể bài toán tính toán IoU, chúng ta hãy cùng tìm hiểu, IoU là gì trước nhé!

Intersection over Union(IoU) là chỉ số đánh giá được sử dụng để đo độ chính xác của Object detector trên tập dữ liệu cụ thể. Nghe có vẻ hơi khó hiểu nhỉ? IoU đơn giản chỉ là một chỉ số đánh giá. Mọi thuật toán có khả năng predict ra các bounding box làm output đều có thể được đánh giá thông qua IoU. IoU được tính như sau:



Dựa vào hình trên, bạn hãy tưởng tượng boxA là predict (do mô hình dự đoán), boxB là ground truth box (do con người gán nhãn).

Tại sao lại phải tính IoU? Nó nói lên điều gì?

Đơn giản là IoU sẽ cho biết mức độ trùng khớp giữa box 1 và box 2. Mà trong bài toán phát hiện vật thể thì mình cần biết: box dự đoán có khớp với sự thật không? Đó chính là tác dụng của IoU.

- Nếu IoU 1 → dự đoán gần như chính xác với ground truth

- Nếu IoU 0 → box dự đoán sai hoàn toàn

9.2 Thực hành

Như đã nói ở phần trên, tính IoU chính chính là tính tỉ lệ phần giao / phần hợp giữa 2 hộp. Mà trong python, loại dữ liệu có thể tính toán dễ dàng với phép toán tập hợp chỉ có tuple thôi! Vì tuple:

- Tuple: Kiểu dữ liệu bất biến (immutable) trong Python, dùng để đại diện cho toà độ của hộp: (x_min, y_min, x_max, y_max)
- Tuple dễ dàng thao tác với tập hợp

```
def compute_iou(box1, box2):
    xA = max(box1[0], box2[0])
    yA = max(box1[1], box2[1])
    xB = min(box1[2], box2[2])
    yB = min(box1[3], box2[3])

    inter_width = max(0, xB - xA)
    inter_height = max(0, yB - yA)
    inter_area = inter_width * inter_height

    area1 = (box1[2] - box1[0]) * (box1[3] - box1[1])
    area2 = (box2[2] - box2[0]) * (box2[3] - box2[1])
    union_area = area1 + area2 - inter_area

    iou = inter_area / union_area if union_area > 0 else 0
    return iou

# VD
boxA = (10, 10, 50, 50)
boxB = (30, 30, 70, 70)
print(f"IoU = {compute_iou(boxA, boxB)}")
```

=> Kết quả là IoU = 0.14285714285714285

10 Non-Maxima Suppression: List & Tuple

10.1 Tổng quan bài toán

Cũng trong bài toán Object Detection, việc có nhiều hộp predict được tạo ra cho một vật thể khiến mô hình bối rối không biết box nào mới thực sự chính xác. Lúc này, chúng ta cần biết đến Non-Maxima. Non-Maxima giúp loại bỏ những box bị chồng lấp và giữ lại những dự đoán có độ chắc cao nhất. Điều này được thực hiện bằng cách so sánh IoU của các box với nhau và với 1 ngưỡng nhất định(do chúng ta định sẵn), Nếu IoU > threshold → bỏ box đó.

NMS hoạt động dựa trên IoU như sau:

- Nếu IoU giữa box đang xét và box đã giữ > threshold → loại bỏ box đó (vì nó quá giống)
- Nếu IoU < threshold → giữ lại box

10.2 Thực hành

```
def non_max_suppression(boxes, iou_thresh=0.5):
    boxes = sorted(boxes, key=lambda x: x[1], reverse=True)
    keep = []

    while boxes:
        current = boxes.pop(0)
        keep.append(current)
        boxes = [box for box in boxes if compute_iou(current[2:], box[2:]) <= iou_thresh]

    return keep

# VD
boxes = [
    ("A", 0.9, 10, 10, 50, 50),
    ("B", 0.8, 30, 30, 60, 60),
    ("C", 0.7, 55, 55, 90, 90),
    ("D", 0.6, 15, 15, 40, 40)
]

result = non_max_suppression(boxes, iou_thresh=0.3)
for box in result:
    print(box)
```

=> Kết quả là
('A', 0.9, 10, 10, 50, 50)
('B', 0.8, 30, 30, 60, 60)
('C', 0.7, 55, 55, 90, 90)

Phần V: Mở rộng

Trong buổi học ngày 13/6 - Advanced Data Structure (IoU, NMS, and Histogram) đa phần mọi người thắc mắc hàm swap trong slide xử lý con trỏ như thế nào. Team có phầ giải thích về cách hoạt động của hàm và cách python "Pass-by-Assignment phần này được trình bày bởi bạn Phạm Đình Huân trong file: Đắng sau Swap