

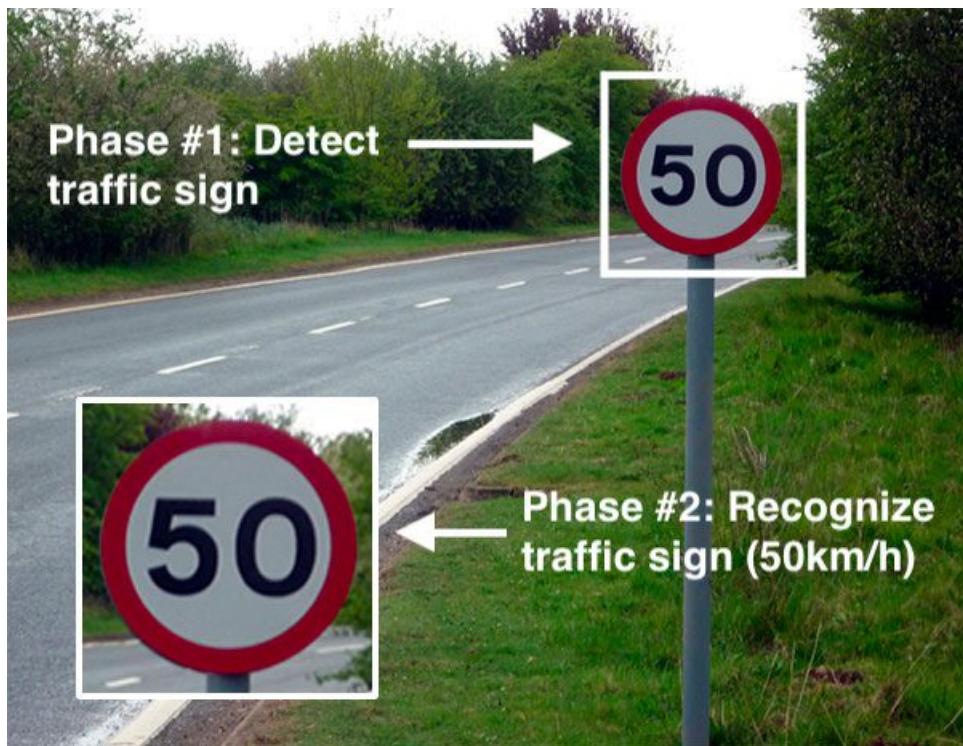
AI VIET NAM – COURSE 2023

# Traffic Sign Detection - Project

Ngày 26 tháng 9 năm 2023

## Phần I: Giới thiệu

**Traffic Sign Detection** là một bài toán ứng dụng các thuật toán liên quan đến Object Detection để phát hiện các biển báo giao thông trên đường. Các mô hình Traffic Sign Detection thường được sử dụng rất nhiều trong các bài toán lớn như Self-driving Cars, Advanced Driver Assistance Systems... Một chương trình Traffic Sign Detection thường bao gồm hai giai đoạn là xác định vị trí của biển báo và nhận diện tên biển báo. Vì vậy, một chương trình có độ chính xác cao cần xây dựng tốt cả hai thành phần này.



Trong project này, chúng ta sẽ xây dựng một chương trình Traffic Sign Detection sử dụng mô hình Support Vector Machine (SVM). Input và output của chương trình như sau:

- **Input:** Một bức ảnh có chứa biển báo giao thông.
- **Output:** Vị trí tọa độ và tên (class) của các biển báo có trong ảnh.

## Phần II: Cài đặt chương trình

Trong phần này, chúng ta tiến hành cài đặt chương trình phát hiện biển báo giao thông. Để thuận tiện trong việc đọc, phần này sẽ được chia làm 2 mục, tương ứng với hai module chính gồm phần Xây dựng mô hình phân loại biển báo dùng SVM và Xây dựng hàm phát hiện đối tượng sử dụng kỹ thuật sliding window.

### 1. Xây dựng mô hình phân loại biển báo giao thông

- (a) **Tải bộ dữ liệu:** Các bạn tải bộ dữ liệu về bài toán Phát hiện biển báo giao thông tại [đây](#). Bộ dữ liệu này gồm 877 ảnh với 4 class lần lượt là 'trafficlight', 'stop', 'speedlimit' và 'crosswalk'.



Hình 1: Một vài ảnh trong bộ dữ liệu

(b) **Import các thư viện cần thiết:**

```

1 import time
2 import os
3 import cv2
4 import numpy as np
5 import pandas as pd
6 import matplotlib.pyplot as plt
7 import xml.etree.ElementTree as ET
8
9 from skimage.transform import resize
10 from skimage import feature
11 from sklearn.svm import SVC
12 from sklearn.preprocessing import LabelEncoder
13 from sklearn.preprocessing import StandardScaler
14 from sklearn.model_selection import train_test_split
15 from sklearn.metrics import accuracy_score

```

- (c) **Đọc dữ liệu:** Ta tiến hành đọc các file ảnh và label đi kèm thành hai list khác nhau, tương ứng với cặp X và y trong bài toán phân loại biển báo của chúng ta. Sau khi giải nén file .zip, ta được hai folders có nội dung như sau:

- **images:** Thư mục chứa các file ảnh.
- **annotations:** Thư mục chứa các file .xml, là các file label, chứa thông tin tọa độ và class của các vật thể có trong ảnh ứng với tên file ở thư mục **images**.

Dầu tiên, chúng ta sẽ khai báo tên đường dẫn đến hai folders trên cũng như hai list rỗng dùng để chứa các ảnh và label đã đọc:

```

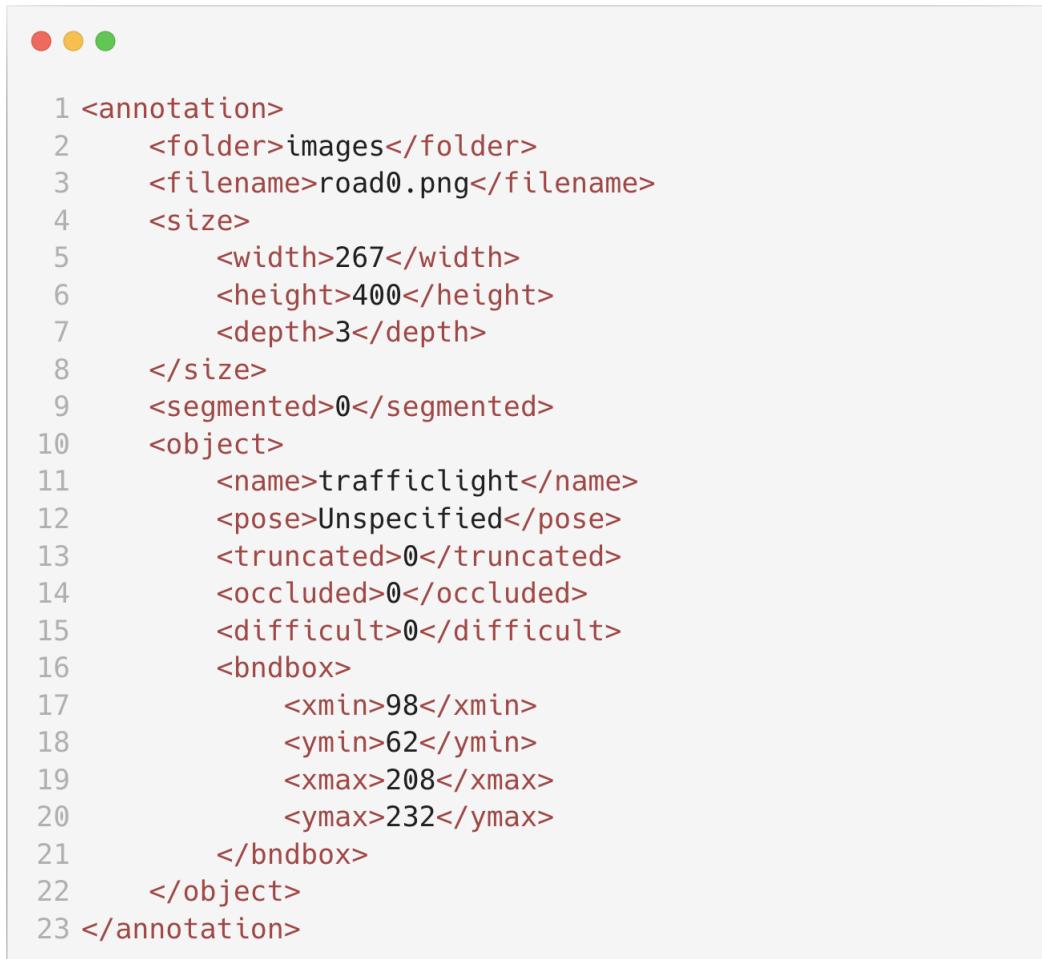
1 annotations_dir = 'annotations'
2 img_dir = 'images'
3
4 img_lst = []
5 label_lst = []

```

Tiếp đến, ta duyệt từng file .xml trong folder **annotations**. Để duyệt từng tên file có trong một folder, ta dùng hàm `os.listdir()`. Để tạo một đường dẫn đến file .xml hoàn chỉnh, ta dùng hàm `os.path.join()` để nối folder annotations và tên file lại với nhau:

```
1 for xml_file in os.listdir(annotations_dir):
2     xml_filepath = os.path.join(annotations_dir, xml_file)
```

Với đường dẫn file .xml có được, ta tiến hành đọc file và một số thông tin cần thiết. Trước hết, ta coi qua format của file .xml:



```
● ● ●

1 <annotation>
2   <folder>images</folder>
3   <filename>road0.png</filename>
4   <size>
5     <width>267</width>
6     <height>400</height>
7     <depth>3</depth>
8   </size>
9   <segmented>0</segmented>
10  <object>
11    <name>trafficlight</name>
12    <pose>Unspecified</pose>
13    <truncated>0</truncated>
14    <occluded>0</occluded>
15    <difficult>0</difficult>
16    <bndbox>
17      <xmin>98</xmin>
18      <ymin>62</ymin>
19      <xmax>208</xmax>
20      <ymax>232</ymax>
21    </bndbox>
22  </object>
23 </annotation>
```

Hình 2: Ví dụ về format file label cho bài toán phát hiện đối tượng (ví dụ trong ảnh được lấy từ file road0.png trong folder annotations)

Trong một file .xml sẽ cho biết các thông tin về ảnh, và quan trọng nhất là thông tin tọa độ, tên class của các object (ở đây là biển báo giao thông) có trong ảnh đó. Vì ta đang cần dữ liệu để huấn luyện cho mô hình phân loại ảnh, ta sẽ trích các object này ra thành 1 sample cho bộ dữ liệu của mình. Như vậy, chúng ta sẽ quan tâm đến thông tin của trường `<name>` và trường `<bndbox>`. Trong một trường `<object>`, `<name>` tương ứng với class của nó và `<bndbox>` cho biết tọa độ của nó ở trong ảnh. Để đọc nội dung của file .xml trong python, ta có thể dùng module `xml` như sau:

```
1 tree = ET.parse(xml_filepath)
2 root = tree.getroot()
```

Module này cho phép ta tương tác với các trường thuộc tính trong một file .xml, sau khi có được trường gốc (root), ta có thể tìm kiếm/trích xuất thông tin của các trường thuộc tính con trong root. Ví dụ, ta có thể lấy thông tin của trường <folder> để đọc file ảnh lên như sau:

```

1 folder = root.find('folder').text
2 img_filename = root.find('filename').text
3 img_filepath = os.path.join(img_dir, img_filename)
4 img = cv2.imread(img_filepath)

```

Với cách làm tương tự, ta sẽ lấy được các thông tin về <name> và <bndbox> của <object>. Vì một ảnh có thể sẽ chứa nhiều object, ta sẽ sử dụng vòng lặp để duyệt từng object nếu có:

```

1 for obj in root.findall('object'):
2     classname = obj.find('name').text
3     if classname == 'trafficlight':
4         continue

```

Ở đây, ta đã lấy thông tin <name>. Vì bài toán của chúng ta chỉ liên quan đến phát hiện biển báo giao thông, vì vậy ta sẽ loại bỏ đi class "trafficlight" trong bộ dữ liệu. Cuối cùng, ta trích thông tin tọa độ <bndbox> và cắt ảnh object như sau:

```

1 xmin = int(obj.find('bndbox/xmin').text)
2 ymin = int(obj.find('bndbox/ymin').text)
3 xmax = int(obj.find('bndbox/xmax').text)
4 ymax = int(obj.find('bndbox/ymax').text)
5
6 object_img = img[ymin:ymax, xmin:xmax]
7 img_lst.append(object_img)
8 label_lst.append(classname)

```

Với tọa độ xmin, ymin, xmax, ymax; ta dễ dàng lấy được ảnh object bằng kỹ thuật slicing trên ảnh gốc. Sau cùng, ta đưa ảnh cắt được vào list `img_lst` và tên class vào `label_lst`.

```

1 print('Number of objects: ', len(img_lst))
2 print('Class names: ', list(set(label_lst)))

```

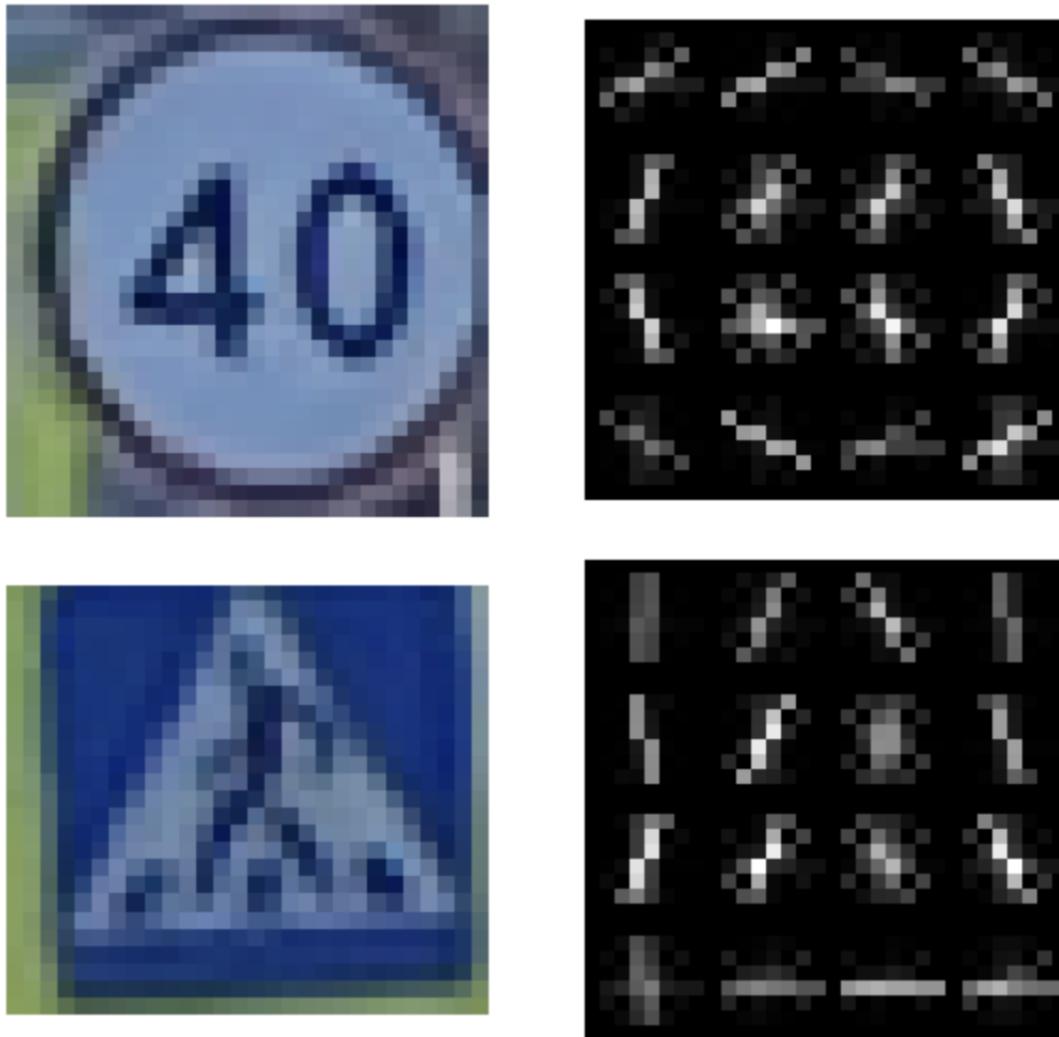
```

Number of objects: 1074
Class names:  ['stop', 'crosswalk', 'speedlimit']

```

Hình 3: Số lượng ảnh và tên các class có trong dataset dùng để huấn luyện mô hình phân loại biển báo giao thông. Bài toán gồm có ba class tương ứng với ba loại biển báo là 'stop', 'crosswalk' và 'speedlimit'

- (d) **Xây dựng hàm tiền xử lý ảnh:** Để mô hình SVM đạt được độ chính xác tốt hơn, ta sẽ tiến hành xây dựng hàm tiền xử lý trước ảnh đầu vào để tạo ra một dạng biểu diễn (đặc trưng) tốt hơn cho ảnh. Cụ thể, ta sẽ sử dụng đặc trưng **HOG (Histogram of Oriented Gradients)** trong bài này.



Hình 4: Đặc trưng HOG tương ứng (ảnh bên phải) cho các ảnh biển báo (ảnh bên trái)

Để tạo đặc trưng HOG, các bạn có thể sử dụng hàm `feature.hog()` trong thư viện `skimage` (các bạn có thể đọc thêm về hàm này tại [đây](#)). Như vậy, ta triển khai hàm tiền xử lý ảnh như sau:

```

1 def preprocess_img(img):
2     if len(img.shape) > 2:
3         img = cv2.cvtColor(
4             img,
5             cv2.COLOR_BGR2GRAY
6         )
7         img = img.astype(np.float32)
8
9     resized_img = resize(
10         img,
11         output_shape=(32, 32),
12         anti_aliasing=True
13     )
14
15     hog_feature = feature.hog(
16         resized_img,

```

```

17     orientations=9,
18     pixels_per_cell=(8, 8),
19     cells_per_block=(2, 2),
20     transform_sqrt=True,
21     block_norm="L2",
22     feature_vector=True
23 )
24
25     return hog_feature

```

Bên cạnh HOG, các bạn sẽ thấy ta cũng áp dụng chuyển đổi ảnh sang dạng ảnh mức xám (grayscale) cũng như thay đổi kích thước ảnh về (32, 32) trước khi tính HOG. Vì các object có các kích thước khác nhau nên việc thay đổi kích thước là cần thiết để vector đặc trưng HOG của mọi ảnh có shape là như nhau.

- (e) **Tiền xử lý ảnh:** Với hàm `preprocess_img()`, ta tiến hành tiền xử lý toàn bộ ảnh đầu vào như sau:

```

1 img_features_lst = []
2 for img in img_lst:
3     hog_feature = preprocess_img(img)
4     img_features_lst.append(hog_feature)
5
6 img_features = np.array(img_features_lst)

```

```

1 print('Shape of the first image before preprocessing: ', img_lst[0].shape)
2 print('Shape of the first image after preprocessing: ', img_features[0].shape)

```

```

Shape of the first image before preprocessing: (41, 41, 3)
Shape of the first image after preprocessing: (324,)

```

Hình 5: Shape của ảnh trước và sau khi tiền xử lý

- (f) **Encode label:** Hiện tại các label đang ở dạng string, ta cần chuyển đổi chúng thành dạng số để phù hợp với yêu cầu huấn luyện mô hình. Ở đây, ta dùng `LabelEncoder()` để đổi các tên class thành các số 0, 1, 2... tương ứng:

```

1 label_encoder = LabelEncoder()
2 encoded_labels = label_encoder.fit_transform(label_lst)

```

- (g) **Chia bộ dữ liệu train, val:** Với danh sách vector đặc trưng HOG đầu vào (X) và danh sách label (y) tương ứng, ta tiến hành chia bộ dữ liệu hiện tại thành hai tập train, val với tỉ lệ chia là 7:3 như sau:

```

1 random_state = 0
2 test_size = 0.3
3 is_shuffle = True
4
5 X_train, X_val, y_train, y_val = train_test_split(
6     img_features, encoded_labels,
7     test_size=test_size,
8     random_state=random_state,
9     shuffle=is_shuffle
10 )

```

(h) **Chuẩn hóa dữ liệu:** Để việc tính toán cũng như quá trình huấn luyện mô hình diễn ra thuận lợi, ta chuẩn hóa giá trị trong các vector đặc trưng HOG sử dụng **StandardScaler()**:

```
1 scaler = StandardScaler()
2 X_train = scaler.fit_transform(X_train)
3 X_val = scaler.transform(X_val)
```

(i) **Huấn luyện mô hình:** Sau khi hoàn tất các bước cần thiết, ta tiến hành huấn luyện mô hình SVM trên bộ dữ liệu train:

```
1 clf = SVC(
2     kernel='rbf',
3     random_state=random_state,
4     probability=True,
5     C=0.5
6 )
7 clf.fit(X_train, y_train)
```

(j) **Đánh giá mô hình:** Ta đánh giá mô hình đã huấn luyện được trên tập val:

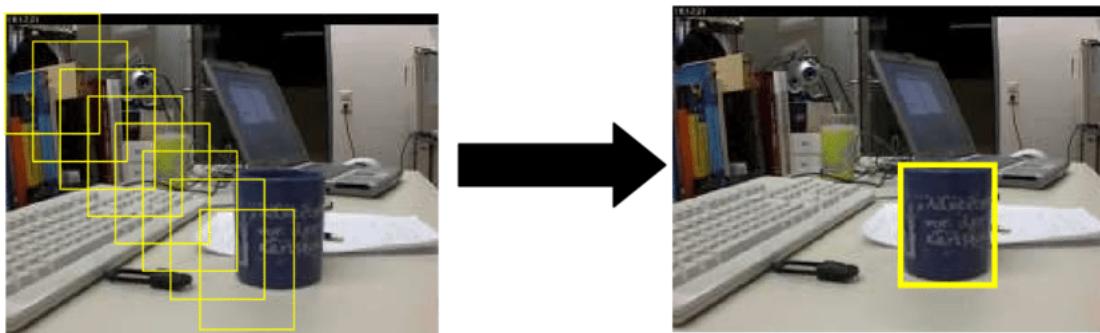
```
1 y_pred = clf.predict(X_val)
2 score = accuracy_score(y_pred, y_val)
3
4 print('Evaluation results on val set')
5 print('Accuracy:', score)
```

Như vậy, ta đã hoàn tất xây dựng một mô hình phân loại biển báo giao thông. Giờ đây, ta tiếp tục xây dựng hàm xác định vị trí biển báo có trong ảnh để hoàn tất chương trình nhận diện biển báo.

## 2. Xây dựng hàm hàm phát hiện đối tượng

Đối với bài toán Object Detection, việc khó khăn nhất là phải xác định vị trí tọa độ của vật thể cần tìm một cách chính xác. Trong project này, chúng ta sẽ áp dụng một kỹ thuật tìm kiếm đối tượng cơ bản nhất đó là kỹ thuật Sliding Window. Ý tưởng chính của thuật toán này như sau:

- **Bước 1:** Định nghĩa một khung cửa sổ có kích thước (w, h). Kích thước cửa sổ phải được lựa chọn cho phù hợp với kích thước object.
- **Bước 2:** Cho cửa sổ này trượt trên từng pixel ảnh đầu vào, lần lượt từ trái qua phải, trên xuống dưới.
- **Bước 3:** Với mỗi lần trượt, ta đưa khung hình ghi nhận được tại khung cửa sổ vào mô hình phân loại biển báo để xác định xem khung cửa sổ này có chứa biển báo giao thông hay không.



Hình 6: Minh họa về sliding window

Ta sẽ triển khai ý tưởng trên thành code như sau:

```

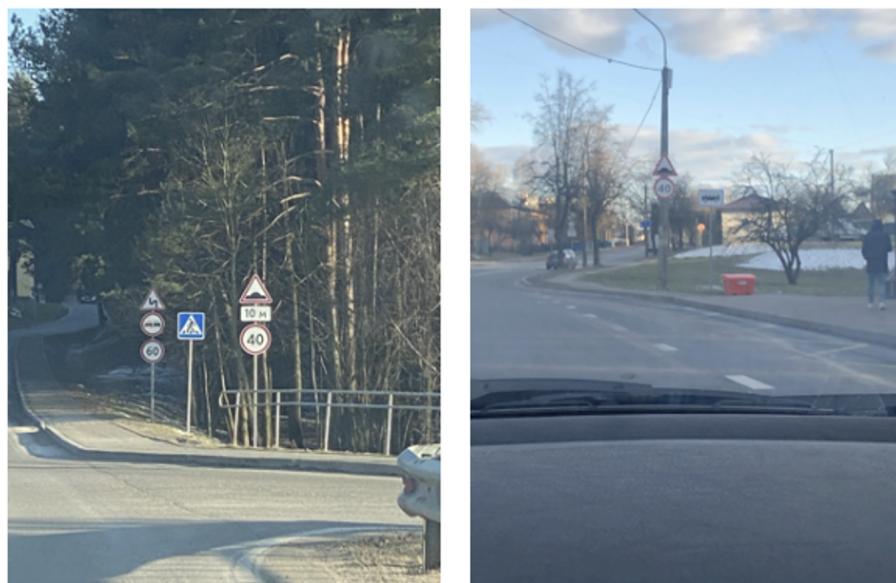
1 def sliding_window(img, window_sizes, stride):
2     img_height, img_width = img.shape[:2]
3     windows = []
4     for window_size in window_sizes:
5         window_width, window_height = window_size
6         for ymin in range(0, img_height - window_height + 1, stride):
7             for xmin in range(0, img_width - window_width + 1, stride):
8                 xmax = xmin + window_width
9                 ymax = ymin + window_height
10
11             windows.append([xmin, ymin, xmax, ymax])
12
13
14 return windows

```

Trong đó:

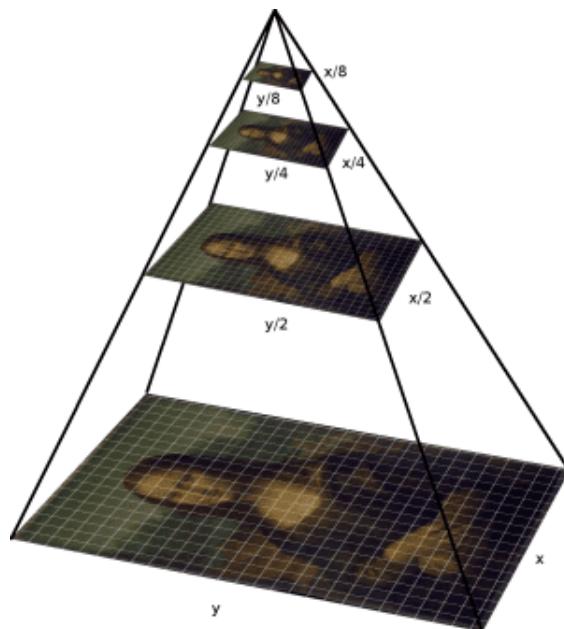
- **Dòng 1:** Khai báo hàm `sliding_window()`, với tham số đầu vào gồm ảnh đầu vào (`img`), danh sách kích thước cửa sổ (`window_sizes`), khoảng cách trượt (`stride`).
- **Dòng 2:** Lấy thông tin chiều rộng và chiều cao của ảnh đầu vào.
- **Dòng 3:** Khai báo danh sách rỗng dùng để chứa các cửa sổ.
- **Dòng 4, 5:** Duyệt qua từng kích thước cửa sổ và lấy thông tin chiều cao và chiều rộng của cửa sổ đang xét.
- **Dòng 6, 7:** Duyệt qua ảnh từ trái qua phải, trên xuống dưới.
- **Dòng 8, 9, 10, 11:** Tính tọa độ `xmax`, `ymax` trên ảnh của cửa sổ để tạo thành một list gồm `[xmin, ymin, xmax, ymax]`. Đây chính là cửa sổ được đại diện bằng thông tin tọa độ.
- **Dòng 13:** Trả về danh sách các cửa sổ.

Tuy vậy, kỹ thuật sliding window có rất nhiều mặt hạn chế. Một trong số đó là việc tìm kiếm các object có kích thước nhỏ.



Hình 7: Một vài trường hợp ảnh có chứa biển báo với kích thước rất nhỏ

Để phần nào khắc phục được tình trạng này, ta có thể áp dụng kỹ thuật **Pyramid Image**. Về cơ bản, kỹ thuật này sẽ tạo ra một chuỗi các ảnh với kích thước nhỏ dần. Khi đó, các object nhỏ trong ảnh gốc sẽ to hơn khi ở trong ảnh nhỏ. Từ đó, ta áp dụng sliding window trên chuỗi ảnh với hy vọng có thể tìm được object nhỏ có trong ảnh gốc.



Hình 8: Mô phỏng chuỗi ảnh với các mức scale khác nhau, được sinh ra từ kỹ thuật Pyramid Image

Lưu ý rằng việc áp dụng kỹ thuật này sẽ làm chậm thời gian dự đoán tổng thể của chương trình khi phải chạy hàm sliding window nhiều lần. Để triển khai hàm pyramid, ta làm như sau:

```

1 def pyramid(img, scale=0.8, min_size=(30, 30)):
2     acc_scale = 1.0
3     pyramid_imgs = [(img, acc_scale)]
4
5     while True:
6         acc_scale = acc_scale * scale
7         h = int(img.shape[0] * acc_scale)
8         w = int(img.shape[1] * acc_scale)
9         if h < min_size[1] or w < min_size[0]:
10             break
11         img = cv2.resize(img, (w, h))
12         pyramid_imgs.append((img, acc_scale))
13
14 return pyramid_imgs

```

Trong đó:

- **Dòng 1:** Khai báo hàm `pyramid()` với tham số đầu vào gồm ảnh (`img`), tỉ lệ scale (`scale`) và kích thước ảnh tối thiểu (`min_size`).
- **Dòng 2:** Khai báo biến dùng để chứa tỉ lệ scale tích lũy.
- **Dòng 3:** Khai báo list chứa tuple là ảnh ở mức scale 1.0, tức là ảnh gốc.
- **Dòng 5, 6:** Lặp qua từng mốc scale.
- **Dòng 7, 8:** Tính kích thước ảnh ở mốc scale mới.

- **Dòng 9, 10:** Kiểm tra nếu kích thước
- **Dòng 11, 12:** Resize lại ảnh với kích thước đã tính được và đưa vào danh sách ảnh pyramid.
- **Dòng 14:** Trả về danh sách ảnh pyramid.

Để vẽ các bounding box tìm được lên ảnh, ta sẽ xây dựng hàm visualize các bounding box lên như sau:

```

1 def visualize_bbox(img, bboxes, label_encoder):
2     img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
3
4     for box in bboxes:
5         xmin, ymin, xmax, ymax, predict_id, conf_score = box
6
7         cv2.rectangle(img, (xmin, ymin), (xmax, ymax), (0, 255, 0), 2)
8
9         classname = label_encoder.inverse_transform([predict_id])[0]
10        label = f'{classname} {conf_score:.2f}'
11
12        (w, h), _ = cv2.getTextSize(label, cv2.FONT_HERSHEY_SIMPLEX, 0.6, 1)
13
14        cv2.rectangle(img, (xmin, ymin - 20), (xmin + w, ymin), (0, 255, 0), -1)
15
16        cv2.putText(img, label, (xmin, ymin - 5), cv2.FONT_HERSHEY_SIMPLEX, 0.6,
17        (0, 0, 0), 1)
18
19    plt.imshow(img)
20    plt.axis('off')
21    plt.show()

```

Trong đó:

- **Dòng 1:** Khai báo hàm `visualize_bbox()` với tham số đầu vào gồm ảnh (`img`), danh sách các bounding box (`bboxes`) và label encoder dùng để convert ID class thành tên class.
- **Dòng 2:** Chuyển đổi kênh màu từ BGR sang RGB.
- **Dòng 4, 5:** Duyệt qua danh sách bounding box và lấy các thông tin của bounding box.
- **Dòng 7:** Với tọa độ `xmin` `ymin` `xmax` `ymax`, ta vẽ một hình chữ nhật màu xanh lá (0, 255, 0) lên ảnh.
- **Dòng 9, 10:** Đổi ID class thành tên class và tạo một string ghi tên class kèm độ tự tin của mô hình.
- **Dòng 12, 13, 14, 15, 16:** Vẽ một hình chữ nhật màu xanh lá bên trên góc trái bounding box với thông tin của biến `label`.
- **Dòng 18, 19, 20:** Hiển thị ảnh đã vẽ bounding box lên màn hình.

Với tất cả các thành phần trên, ta đã có thể kết hợp chúng để tạo nên một chương trình phát hiện biển báo giao thông. Triển khai như sau:

- Khai báo đường dẫn ảnh mong muốn thực hiện detection và các tham số dùng để tinh chỉnh kết quả của chương trình (gồm `conf_threshold`, `stride`, `window_sizes`):

```

1 img_dir = 'images'
2 img_filename_lst = os.listdir(img_dir)[:20]
3 conf_threshold = 0.8
4 stride = 12
5 window_sizes = [
6     (32, 32),

```

```

7     (64, 64),
8     (128, 128)
9 ]

```

- Duyệt qua từng ảnh trong img\_filename\_lst, đọc ảnh lên và tạo pyramid image:

```

1 for img_filename in img_filename_lst:
2     start_time = time.time()
3     img_filepath = os.path.join(img_dir, img_filename)
4     bboxes = []
5     img = cv2.imread(img_filepath)
6     pyramid_imgs = pyramid(img)

```

- Duyệt qua từng ảnh trong pyramid và chạy sliding window:

```

1 for pyramid_img_info in pyramid_imgs:
2     pyramid_img, scale_factor = pyramid_img_info
3     window_lst = sliding_window(
4         pyramid_img,
5         window_sizes=window_sizes,
6         stride=stride,
7         scale_factor=scale_factor
8     )

```

- Duyệt qua từng danh sách cửa sổ tìm được, thực hiện tiền xử lý ảnh và phân loại chúng:

```

1 for window in window_lst:
2     xmin, ymin, xmax, ymax = window
3     object_img = pyramid_img[ymin:ymax, xmin:xmax]
4     preprocessed_img = preprocess_img(object_img)
5     normalized_img = scaler.transform([preprocessed_img])[0]
6     decision = clf.predict_proba([normalized_img])[0]

```

Lưu ý rằng ở phần huấn luyện mô hình SVM, chúng ta đã cài đặt cho kết quả mô hình trả về là xác suất dự đoán của từng class. Vì vậy, thay vì dùng hàm predict(), ta dùng hàm predict\_proba(). Đây chính là confidence score của bounding box.

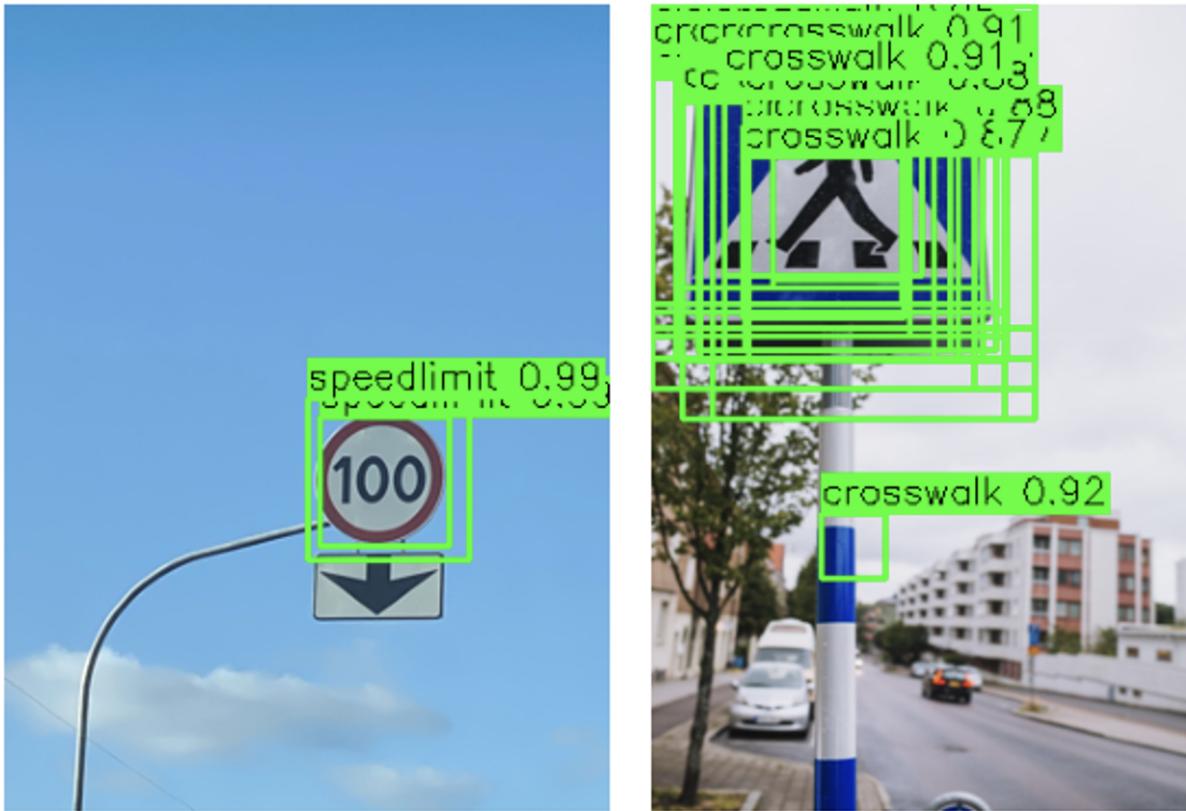
- Với kết quả dự đoán là xác suất, ta sẽ thực hiện sàng lọc kết quả dự đoán để tránh các dự đoán không tốt. Với tham số conf\_threshold, ta sẽ chỉ xét các dự đoán mà có ít nhất một xác suất dự đoán lớn hơn conf\_threshold. Nếu thỏa mãn thì ta chọn class có xác suất cao nhất. Nếu không có bất kì class nào thỏa mãn, ta bỏ qua cửa sổ đang xét.

```

1 if np.all(decision < conf_threshold):
2     continue
3 else:
4     predict_id = np.argmax(decision)
5     conf_score = decision[predict_id]
6     xmin = int(xmin / scale_factor)
7     ymin = int(ymin / scale_factor)
8     xmax = int(xmax / scale_factor)
9     ymax = int(ymax / scale_factor)
10    bboxes.append(
11        [xmin, ymin, xmax, ymax, predict_id, conf_score]
12    )

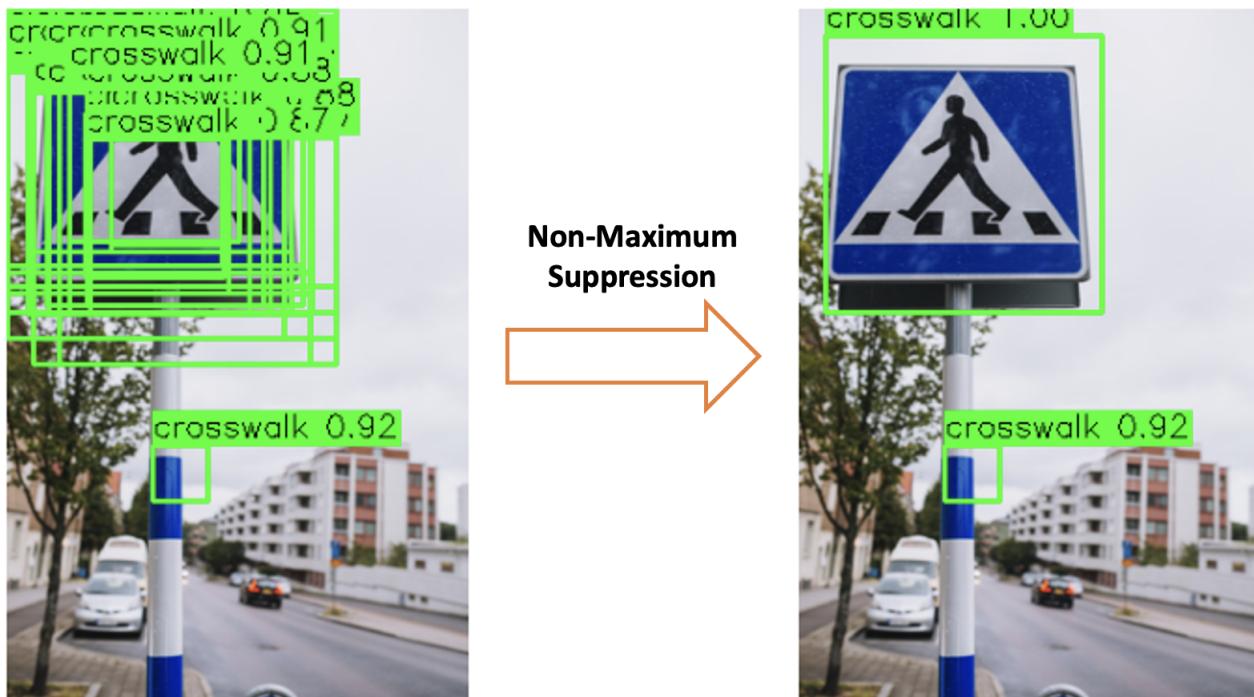
```

Cuối cùng, visualize toàn bộ các bounding box tìm được sử dụng hàm visualize. Ta sẽ được một số kết quả như sau:



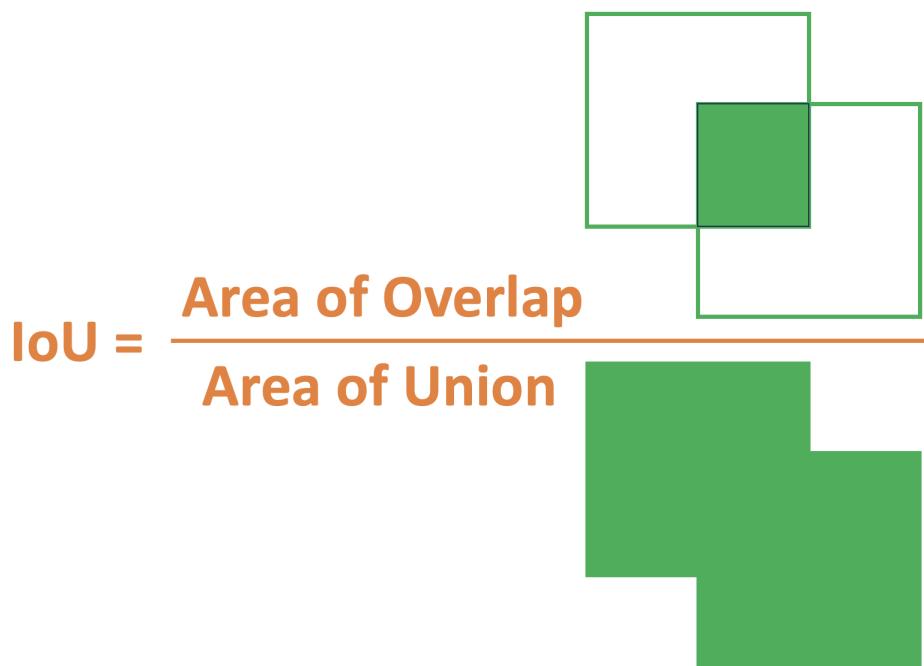
Hình 9: Kết quả Detection với cài đặt  $\text{conf\_threshold}=0.85$ ,  $\text{window\_size}$  lần lượt là  $(32, 32)$ ;  $(64, 64)$ ;  $(128, 128)$  và  $\text{stride}=12$

Quan sát cho thấy, chương trình của chúng ta có khả năng phát hiện được biển báo giao thông tương đối chính xác. Tuy nhiên, kết quả hiện tại lại sinh ra rất nhiều bounding box nằm đè lên nhau với sự chênh lệch không quá lớn. Để giải quyết tình trạng này, chúng ta sẽ cài đặt một bước hậu xử lý kết quả dự đoán bằng thuật toán **Non-Maximum Suppression (NMS)**.



Hình 10: Kết quả trước và sau khi áp dụng Non-Maximum Suppression

Kỹ thuật NMS sẽ loại bỏ các bounding box nằm đè nhau và chỉ giữ lại bounding box có confidence score là cao nhất. Để xây dựng hàm NMS, đầu tiên, ta triển khai hàm tính **IoU (Intersection over Union)**. IoU là tỉ lệ giao nhau giữa hai bounding box, có công thức tính như sau:



Hình 11: Công thức tính IoU là tỉ lệ giữa diện tích phần giao nhau và diện tích tổng thể của hai bounding box

```

1 def compute_iou(bbox, bboxes, bbox_area, bboxes_area):
2     xxmin = np.maximum(bbox[0], bboxes[:, 0])
3     yymin = np.maximum(bbox[1], bboxes[:, 1])
4     xxmax = np.minimum(bbox[2], bboxes[:, 2])
5     yymax = np.minimum(bbox[3], bboxes[:, 3])
6
7     w = np.maximum(0, xxmax - xxmin + 1)
8     h = np.maximum(0, yymax - yymin + 1)
9
10    intersection = w * h
11    iou = intersection / (bbox_area + bboxes_area - intersection)
12
13    return iou

```

Sau đó, xây dựng hàm tính NMS với ý tưởng như sau:

- **Bước 1:** Sắp xếp danh sách các bounding box theo thứ tự giảm dần về confidence score.
- **Bước 2:** Với bounding box có confidence score cao nhất, ta đưa vào danh sách giữ lại (keep), thực hiện tính IoU giữa bounding box này với từng bounding box còn lại.
- **Bước 3:** Sau đó, loại bỏ toàn bộ các bounding box có IoU lớn hơn ngưỡng IoU cho trước (iou\_threshold).
- **Bước 4:** Với danh sách các bounding box không thỏa mãn điều kiện ở bước 3, quay lại thực hiện bước 2 cho tới khi duyệt qua toàn bộ bounding box.

Ta triển khai code như sau:

```

1 def nms(bboxes, iou_threshold):
2     if not bboxes:
3         return []
4
5     scores = np.array([bbox[5] for bbox in bboxes])
6     sorted_indices = np.argsort(scores)[::-1]
7
8     xmin = np.array([bbox[0] for bbox in bboxes])
9     ymin = np.array([bbox[1] for bbox in bboxes])
10    xmax = np.array([bbox[2] for bbox in bboxes])
11    ymax = np.array([bbox[3] for bbox in bboxes])
12
13    areas = (xmax - xmin + 1) * (ymax - ymin + 1)
14
15    keep = []
16
17    while sorted_indices.size > 0:
18        i = sorted_indices[0]
19        keep.append(i)
20
21        iou = compute_iou(
22            [xmin[i], ymin[i], xmax[i], ymax[i]],
23            np.array([
24                [
25                    xmin[sorted_indices[1:]],
26                    ymin[sorted_indices[1:]],
27                    xmax[sorted_indices[1:]],
28                    ymax[sorted_indices[1:]]
29                ].T,
30                areas[i],
31                areas[sorted_indices[1:]]
32            ])

```

```

33
34     idx_to_keep = np.where(iou <= iou_threshold)[0]
35     sorted_indices = sorted_indices[idx_to_keep + 1]
36
37     return [bboxes[i] for i in keep]

```

Trong đó:

- **Dòng 1:** Khai báo hàm `nms()` với tham số đầu vào là danh sách các bounding box (`bboxes`) và ngưỡng IoU (`iou_threshold`).
- **Dòng 2, 3:** Nếu `bboxes` là rỗng, ta trả về danh sách rỗng.
- **Dòng 5, 6:** Lấy danh sách confidence score và sắp xếp chúng theo thứ tự giảm dần (kết quả trả về là chỉ mục của confidence score).
- **Dòng 8, 9, 10, 11:** Lấy danh sách `xmin`, `ymin`, `xmax`, `ymax`.
- **Dòng 13:** Tính diện tích của toàn bộ bounding box.
- **Dòng 15:** Khai báo danh sách rỗng dùng để chứa các bounding box được giữ lại sau khi lọc.
- **Dòng 17:** Tạo một vòng lặp chỉ kết thúc khi danh sách điểm confidence score không còn phần tử nào.
- **Dòng 18, 19:** Lấy bounding box có điểm confidence score cao nhất và đưa vào danh sách giữ lại.
- **Từ dòng 21 đến dòng 32:** Tính IoU của bounding box có confidence score cao nhất với toàn bộ các bounding box còn lại.
- **Dòng 34, 35:** Giữ lại các bounding box có IoU nhỏ hơn `iou_threshold` để tính toán cho vòng lặp tiếp theo.
- **Dòng 37:** Trả về danh sách bounding box được giữ lại.

Khi áp dụng hàm NMS trên với `iou_threshold=0.1` kèm với `conf_threshold=0.95`, ta sẽ có được kết quả như sau:



Hình 12: Kết quả của chương trình Phát hiện biển báo giao thông

## Phần III: Câu hỏi trắc nghiệm

1. Bài toán Object Detection thường giải quyết hai bài toán con chính nào sau đây?
  - (a) Object Localization và Object Classification.
  - (b) Object Localization và Object Segmentation.
  - (c) Object Classification và Object Tracking.
  - (d) Object Classification và Object Counting.
2. Mục tiêu của Object Localization trong bài Traffic Sign Detection là gì?
  - (a) Để xác định màu của biển báo
  - (b) Để xác định vị trí và kích thước của biển báo trong ảnh
  - (c) Để đếm số lượng biển báo có trong ảnh
  - (d) Để phân loại biển báo giao thông thuộc class nào
3. Mục tiêu của Object Classification trong bài Traffic Sign Detection là gì?
  - (a) Để xác định màu của biển báo
  - (b) Để xác định vị trí và kích thước của biển báo trong ảnh
  - (c) Để đếm số lượng biển báo có trong ảnh
  - (d) Để phân loại biển báo giao thông thuộc class nào
4. Module nào sau đây được sử dụng để đọc file có đuôi .xml?

(a) json	(c) xml.etree.ElementTree
(b) xmlparse	(d) yaml
5. Với tọa độ [xmin, ymin, xmax, ymax], đoạn code nào sau đây là đúng để dùng trong việc tách object từ ảnh img đọc từ thư viện opencv?

(a) img[ymin:ymax, xmin:xmax]	(c) img[ymax:ymin] [xmin:xmax]
(b) img[xmin:xmax, ymin:ymax]	(d) img[xmax:xmin, ymax:ymin]
6. Dòng code nào sau đây dùng để thay đổi kênh màu từ BGR sang grayscale sử dụng thư viện opencv?
  - (a) cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)
  - (b) cv2.convertToGray(image, cv2.COLOR\_BGR2GRAY)
  - (c) cv2.toGray(image, cv2.COLOR\_BGR2GRAY)
  - (d) cv2.bgr2gray(image)
7. Lý do chính trong việc sử dụng đặc trưng HOG cho Traffic Sign Classification là?
  - (a) Giảm độ phân giải ảnh
  - (b) Tăng độ sáng và độ tương phản ảnh
  - (c) Lọc nhiễu trong ảnh
  - (d) Biểu diễn hình dạng và kết cấu của các vật thể cục bộ trong ảnh

8. Lý do chính trong việc resize lại ảnh trước khi tạo đặc trưng HOG?
- (a) Tăng độ phân giải ảnh
  - (b) Cho output shape của hog features là như nhau
  - (c) Yêu cầu bắt buộc trong thuật toán HOG
  - (d) Tăng độ chính xác cho mô hình phân loại
9. Module nào sau đây dùng để mã hóa label trong thư viện scikit-learn?
- (a) `sklearn.encoder.LabelBinarizer`
  - (b) `sklearn.preprocessing.LabelEncoder`
  - (c) `sklearn.transform.LabelEncoder`
  - (d) `sklearn.feature_extraction.LabelEncoder`
10. Khi cài đặt tham số `probability=True` trong module `sklearn.svm.SVC()`, sự thay đổi nào của SVM sau đây là đúng?
- (a) SVM sử dụng kernel khác
  - (b) Giảm số lượng support vectors được sử dụng
  - (c) Kích hoạt ước lượng xác suất cho từng class khi dự đoán
  - (d) Tăng độ chính xác của mô hình SVM
11. Kỹ thuật sliding window được dùng để làm gì?
- (a) Tìm kiếm vị trí của vật thể trong ảnh
  - (b) Khử nhiễu trong ảnh
  - (c) Thay đổi kích thước ảnh
  - (d) Trích xuất đặc trưng của ảnh
12. Trong Object Detection sử dụng kỹ thuật sliding window, với mỗi cửa sổ thu được, ta thực hiện điều gì tiếp theo?
- (a) Tiếp tục tìm kiếm các cửa sổ khác
  - (b) Tìm các cửa sổ tương đồng
  - (c) Phân loại ảnh thu được trong cửa sổ
  - (d) Loại bỏ các cửa sổ có độ phân giải thấp
13. Kết quả phân loại của mô hình SVM ở dạng xác suất trong bài toán Object Detection còn được gọi là gì?
- (a) Accuracy
  - (b) Margin
  - (c) Kernel
  - (d) Confidence Score
14. Để lấy kết quả dự đoán của mô hình SVM dưới dạng xác suất, ta dùng phương thức nào sau đây trong scikit-learn?
- (a) `predict()`
  - (b) `decision_function()`
  - (c) `predict_proba()`
  - (d) `score()`
15. Mệnh đề nào sau đây là đúng khi nói về lý do sử dụng kỹ thuật Pyramid Image trong Object Detection?
- (a) Tăng cơ hội phát hiện các object với nhiều kích cỡ khác nhau
  - (b) Làm cho vật thể trong ảnh to và rõ hơn

- Hét -