AI VIET NAM – COURSE 2024

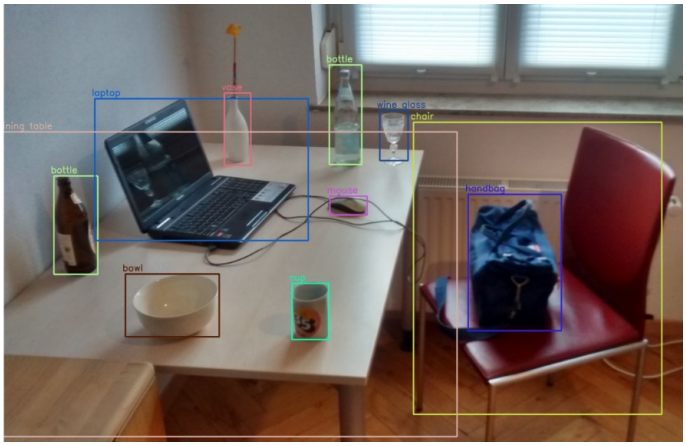
Image Project - Object Detection: Traffic Sign

Ngày 15 tháng 3 năm 2024

Ngày tóm tắt:	13/03/2024
Tác giả:	AIO
Nguồn:	Module 4 AIO 2023
Nguồn dữ liệu (nếu có):	
Từ khóa:	Object Detection, Classification, Localization
Người tóm tắt:	Nguyễn Mậu Ánh Ngân

1 Giới thiệu

- Trong bài hôm nay ta sẽ không sử dụng thư viện có sẵn mà xây dựng hoàn toàn theo từng bước.
- Object Detection (OD) là gì: ý tưởng chính là khi ta đưa vào một tấm ảnh (Input) thì ta sẽ xác định được những vật thể (Object) có trên ảnh kèm theo:
  - Hình chữ nhật (Bounding box) xác định vị trí giới hạn của Object.
  - Tên (Classname/ Label) của Object.



Hình 1: Ví dụ một Output của Object Detection.

- Phân biệt OD và Image Classification (IC)
  - Giống nhau: Input đều là ảnh.
  - Khác nhau:

  - OD: Output xác định được cả vị trí (qua Bounding Boxes) và classname/label của Object trong ảnh.
  - IC: Output chỉ xác định classname/Label chung của ảnh.

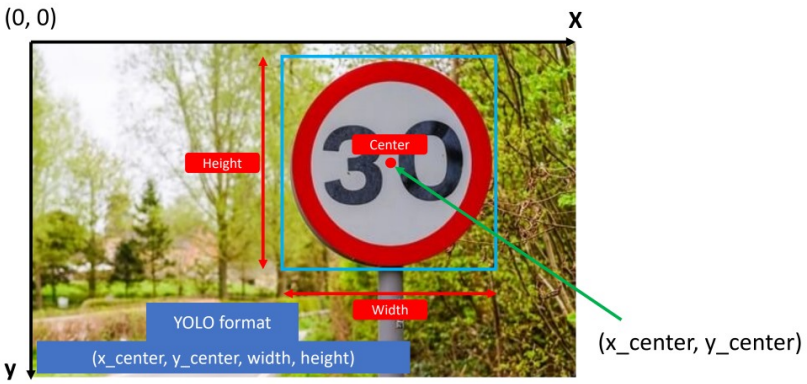
- Cách xây dựng một bải OD: bao gồm 2 model chính
  - Localization: xác định ví trị Object -> Output: bounding boxes.
  - Classification: Xác định classname cho các bounding box.

  Ta gọi là mô hình Two-stage Object Detection (hình ảnh được xử lý thường qua Localization trước)

1.1 Object Localization:

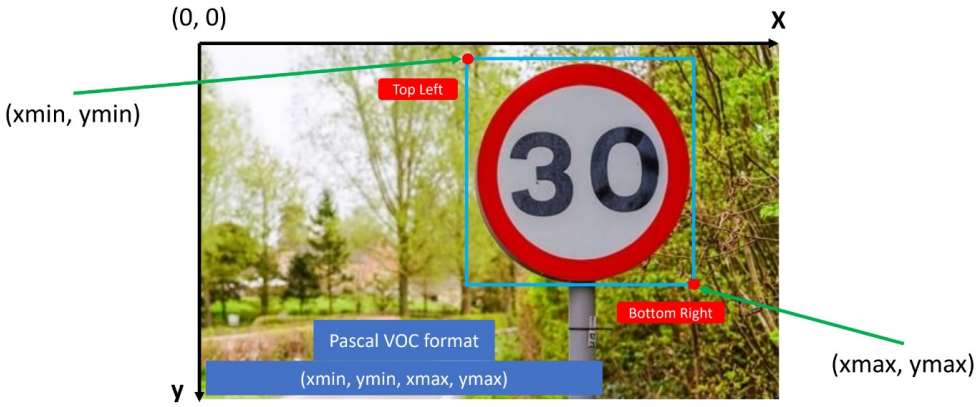
- Bounding box: là hình chữ nhật xác định vị trí của Object.
- Trong code: Output sẽ là một Tuple gồm tọa độ của bounding box. Thường có 2 dạng tuple:

1. YOLO format:



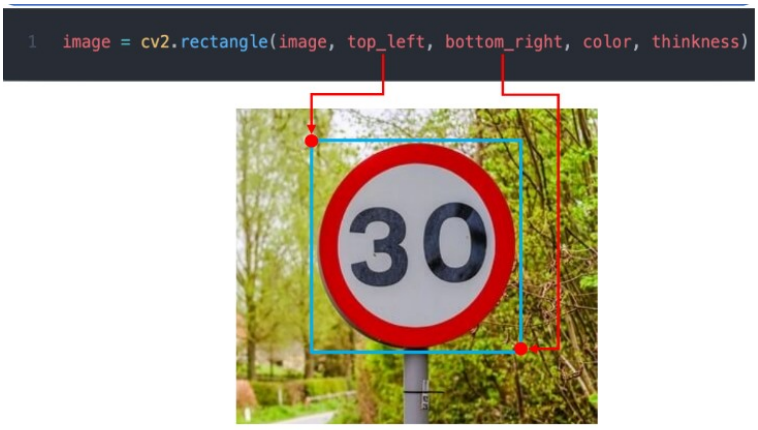
Hình 2: YOLO format.

2. Pascal VOC format:



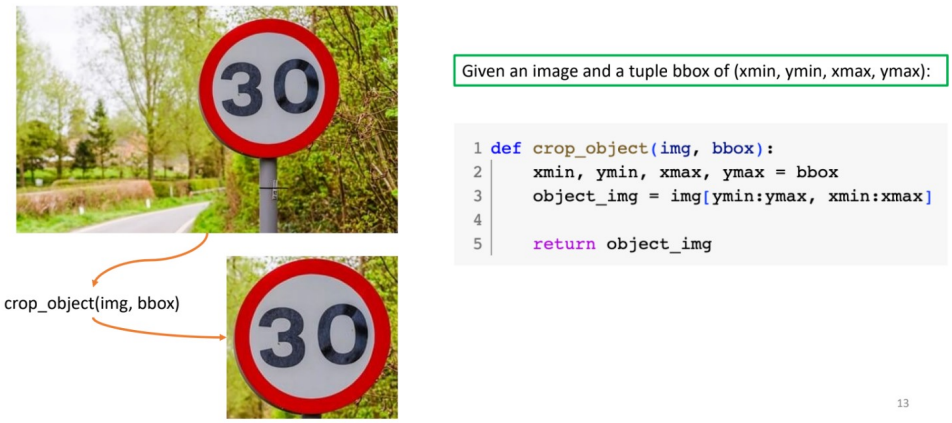
Hình 3: Pascal VOC format.

- Code vẽ Bounding Box:



Hình 4: Code vẽ Bounding box.

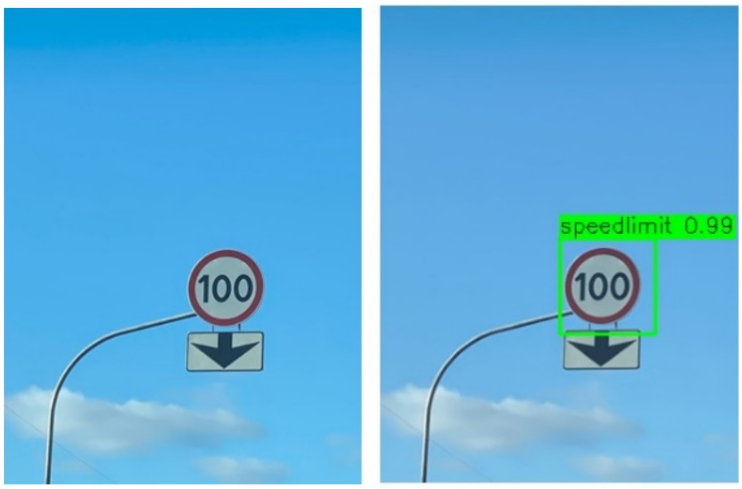
- Chuyển đổi giữa 2 format (xem thêm ở đoạn record: [00:23:07-00:24:18])
- Cắt Object ra từ hình (Format Pascal)



Hình 5: Code cắt Object từ ảnh.

1.2 Object Classification:

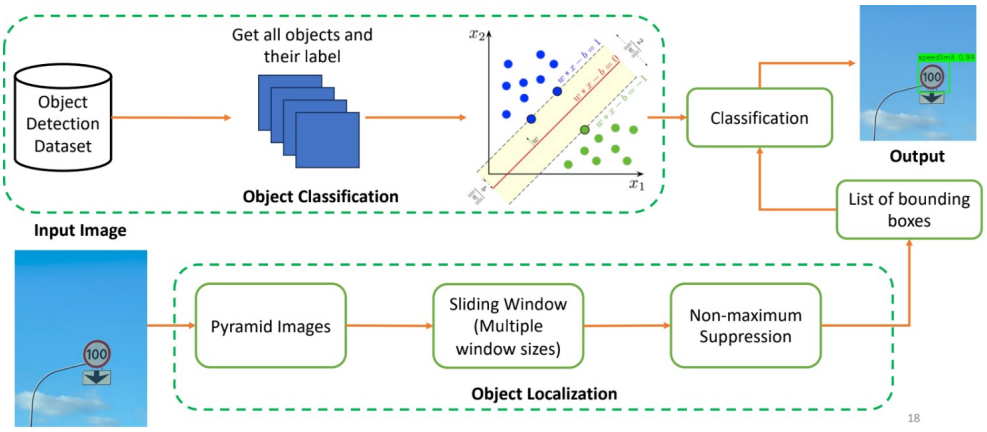
- Mục đích là phân loại, tạo classname cho bounding boxes
- Output: list tuple (tọa độ bounding box, confidence score) vì thường một hình sẽ có nhiều Object nên sẽ cho ra list.
- Confidence score: mức độ tự tin của mô hình về việc đoán đúng classname. Thường nằm trong khoảng từ 0 tới 1.



Hình 6: Hình sau khi đã qua cả Localization và Classification.

Hình 7: Confidence score.

1.3 Sơ đồ chung xây dựng bài toán:



Hình 8: Các bước xây dựng bài toán Object Detection.

1.4 Input các thư viện cần thiết:

```
import time
import os
import cv2
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import xml.etree.ElementTree as ET

from skimage.transform import resize
from skimage import feature
from sklearn.svm import SVC
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
```

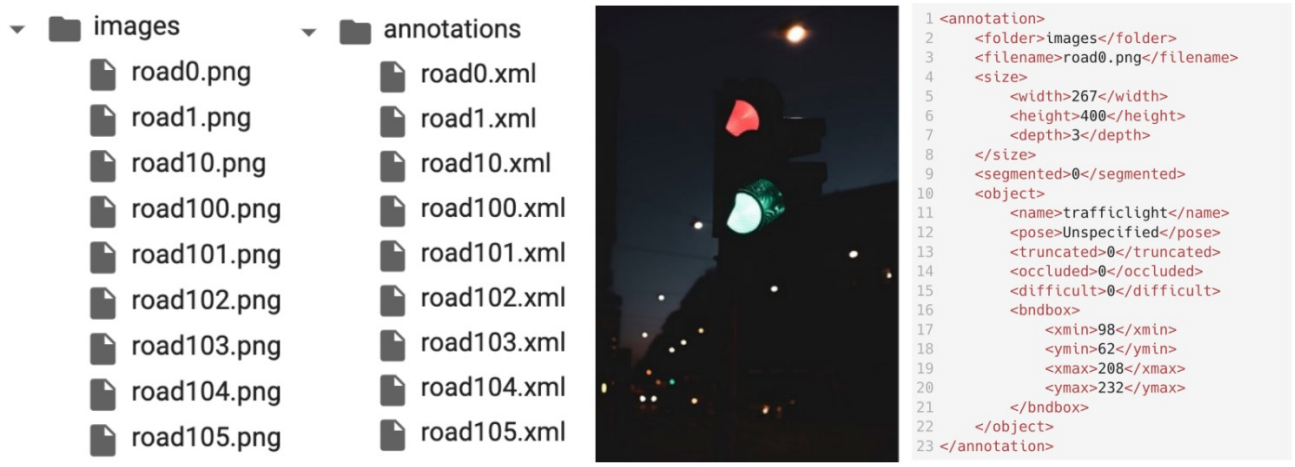
Hình 9: Các thư viện dùng trong code.

2 Classification:

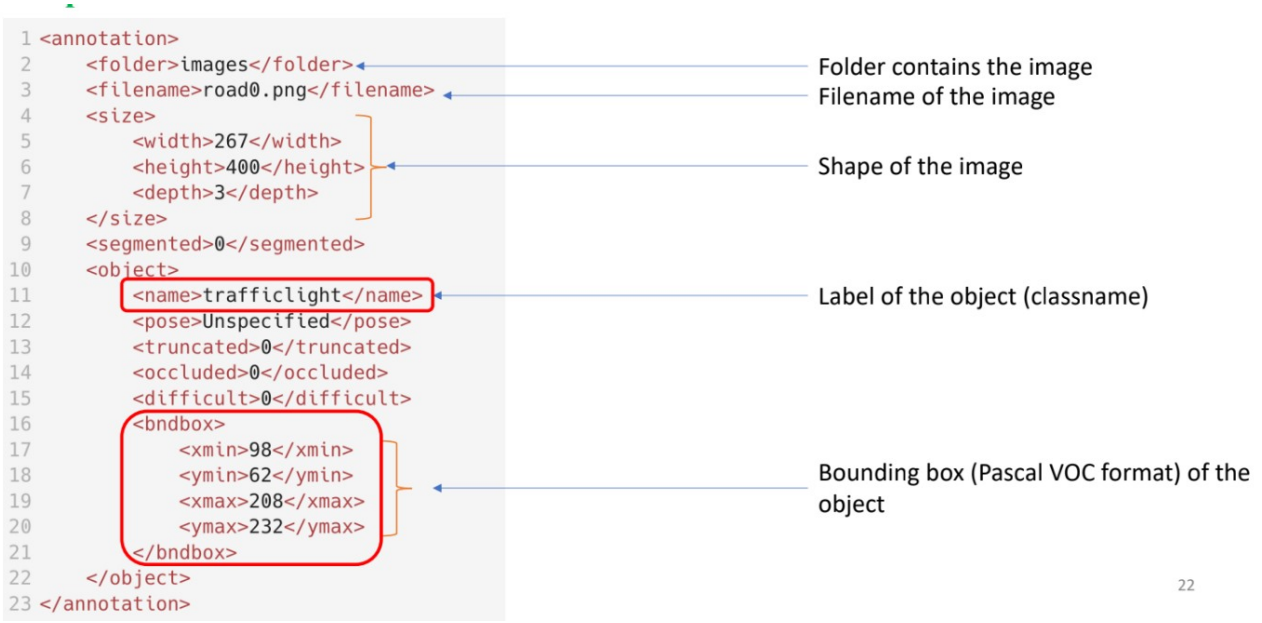
- Tải Dataset (tập dữ liệu dùng để train) từ link sau: Traffic Sign Detection.
- Model ta sẽ dùng là SVM.

2.1 Step 1: Đọc Dataset

- Trong Dataset gồm 2 folder:
  - Images: chứa các file ảnh.
  - Annotations: chứa các file label của ảnh.

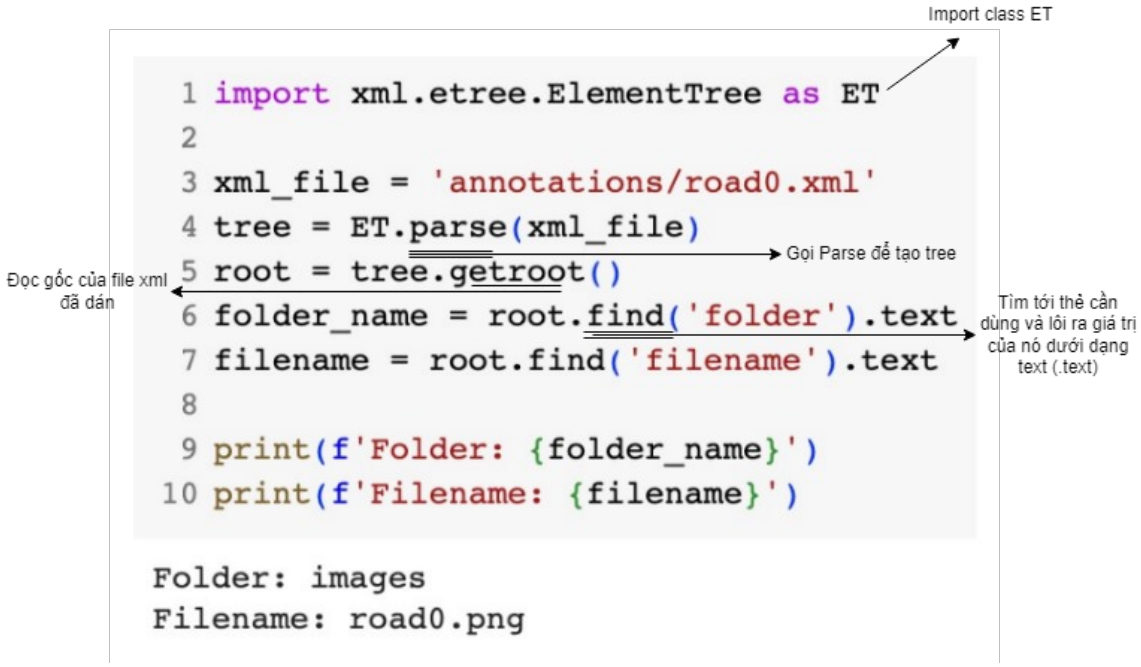


Hình 10: Các folder trong dataset: ảnh và label tương ứng.



Hình 11: Thông tin trong file label của ảnh.

- Dùng Xml để lấy thông tin từ file label




```python
import xml.etree.ElementTree as ET

xml_file = 'annotations/road0.xml'
tree = ET.parse(xml_file)
root = tree.getroot()
folder_name = root.find('folder').text
filename = root.find('filename').text

print(f'Folder: {folder_name}')
print(f'Filename: {filename}')
```

```
Folder: images
Filename: road0.png
```

Hình 12: Cách sử dụng xml cơ bản để lấy thông tin.

```xml
<annotation>
    <folder>images</folder>
    <filename>road0.png</filename>
    <size>
        <width>267</width>
        <height>400</height>
        <depth>3</depth>
    </size>
    <segmented>0</segmented>
    <object>
        <name>trafficlight</name>
        <pose>Unspecified</pose>
        <truncated>0</truncated>
        <occluded>0</occluded>
        <difficult>0</difficult>
        <bndbox>
            <xmin>98</xmin>
            <ymin>62</ymin>
            <xmax>208</xmax>
            <ymax>232</ymax>
        </bndbox>
    </object>
</annotation>
```

```python
import xml.etree.ElementTree as ET

xml_file = 'annotations/road0.xml'
tree = ET.parse(xml_file)
root = tree.getroot()

for obj in root.findall('object'):
    classname = obj.find('name').text

    xmin = int(obj.find('bndbox/xmin').text)
    ymin = int(obj.find('bndbox/ymin').text)
    xmax = int(obj.find('bndbox/xmax').text)
    ymax = int(obj.find('bndbox/ymax').text)

    print(f'Classname: {classname}')
    print(f'Bounding box: {(xmin, ymin, xmax, ymax)}')
```

```
Classname: trafficlight
Bounding box: (98, 62, 208, 232)
```

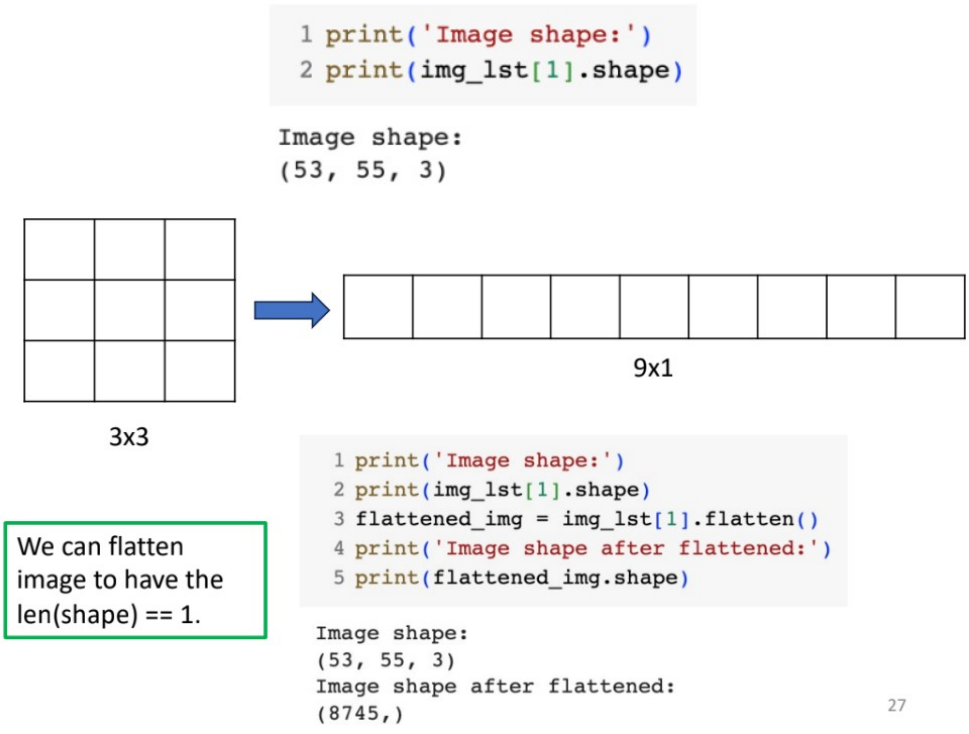
25

Hình 13: Lấy thông tin Classname và tọa độ.

Hình 14: Duyệt qua tất cả các file label, bỏ qua label Trafficlight.

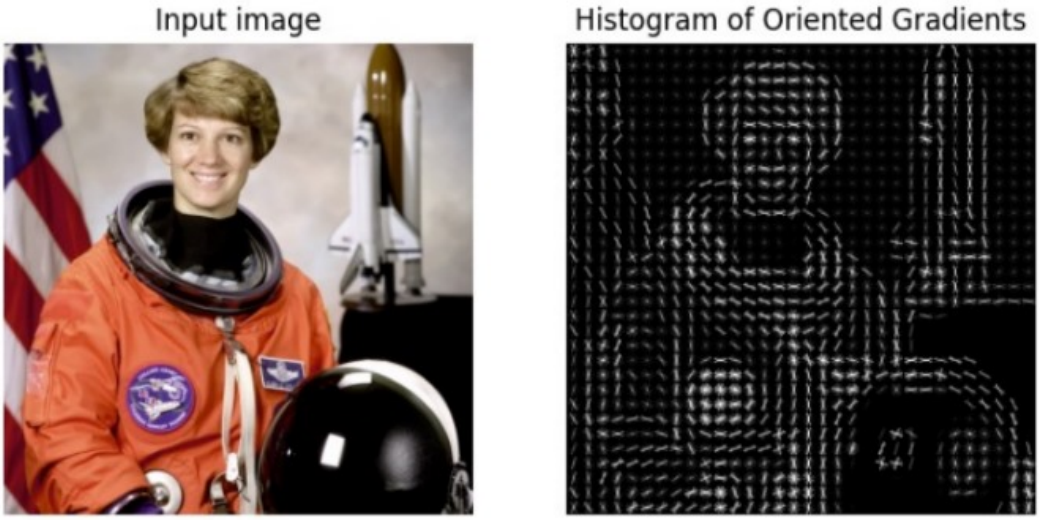
2.2 Step 2: Tiền xử lý ảnh

- Với shape như bên (cao,rộng,kênh màu) thì ta không thể đẩy vào mô hình SVM được -> Ta phải flatten về lại len(shape) == 1.



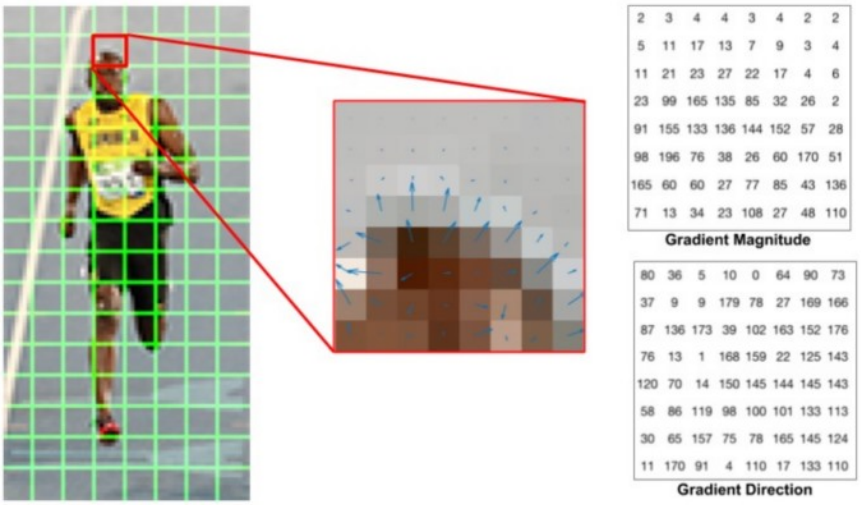
Hình 15: Flatten.

- Sau khi flatten, ta nhận được vector đại diện 1D có tới 8745 phần tử -> quá phức tạp -> nên xử lý ảnh gốc trước khi flatten. Ta dùng phương pháp HoG (Histogram of Oriented Gradiens).



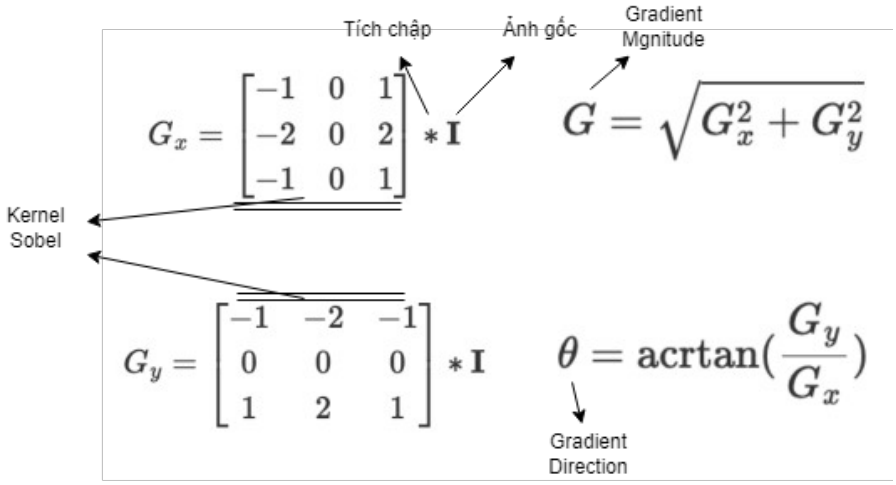
Hình 16: Phương pháp HoG.

- Cách HoG hoạt động:
  - Ảnh đầu vào sẽ được chia thành các cell, mỗi cell có kích thước 8x8.
  - Với mỗi cell ta sẽ tìm 2 ma trận:
    * Gradient Magnitude: Ma trận độ lớn Gradient.
    * Gradient Direction: Ma trận phương Gradient.



Hình 17: Cell và 2 ma trận cần tìm.

– Cách tính 2 ma trận:



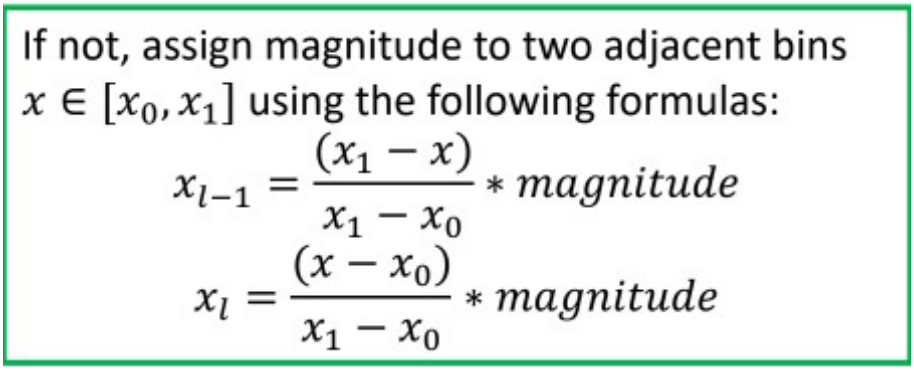
Hình 18: Công thức tính 2 ma trận.

– Tiếp theo, ta tạo ra vector 1D gồm có 9 phần tử đại diện cho miền giá trị của ma trận phương Gradient nằm trong khoảng [0,180]. Chia vector thành 9 khoảng như hình, mỗi khoảng được gọi là 1 bin. Vector này được gọi là vector Histogram of Gradient.

Hình 19: Vector đại diện gồm 9 bin.

– Tham chiếu 2 ma trận đã tìm được ở trên vào vector đại diện ta có 2 trường hợp gán giá trị:

* Gradient Direction có độ lớn trùng với bin (0,20,40...) => gán luôn giá trị tương ứng ở Gradient Magnitude vào bin đó. (Ví dụ vị trí màu xanh trong hình 21)
* Gradient Direction có độ lớn thuộc khoảng bin => dùng công thức nội suy sau để tìm giá trị gán:

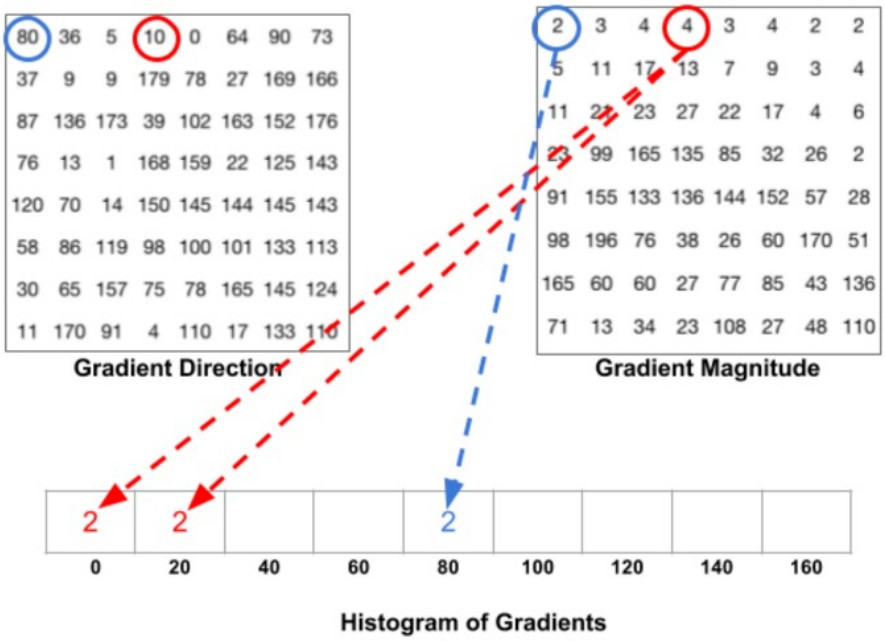


Hình 20: Công thức nội suy.

Ví dụ: Trong hình 21, vị trí màu đỏ ta chọn có giá trị Gradient Direction = 10 thuộc bin 0 tới 20 và Gradient Magnitude tương ứng = 4 ta áp dụng công thức:

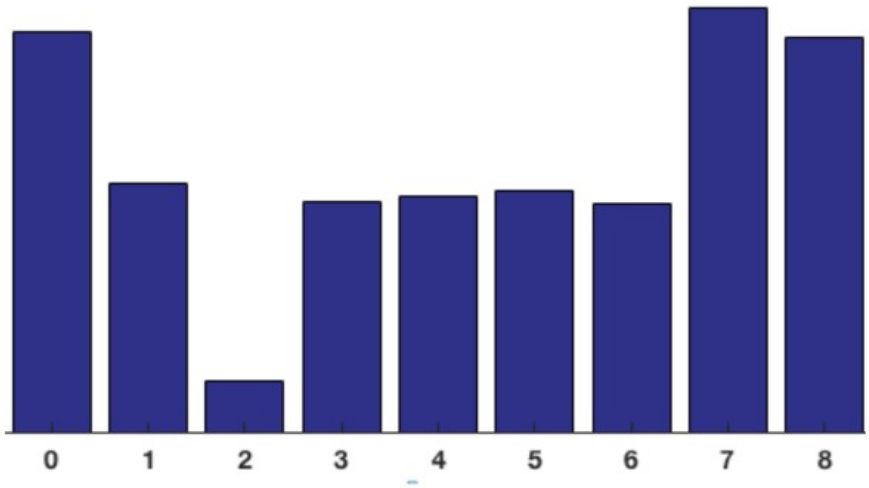
$$x_0 = \frac{20-10}{20-0} * 4 = 2 \quad , x_{10} = \frac{10-0}{20-0} * 4 = 2$$

Ta được hai giá trị gán vào bin 0 là 2 và bin 10 là 2.



Hình 21: Các trường hợp ví dụ gán giá trị cho vector HoG.

– Lặp lại quá trình tính toán và gán cho toàn bộ 2 ma trận.
– Tính tổng độ lớn Gradient của từng bin biểu diễn thành đồ thị ta có đồ thị Histogram gradient.

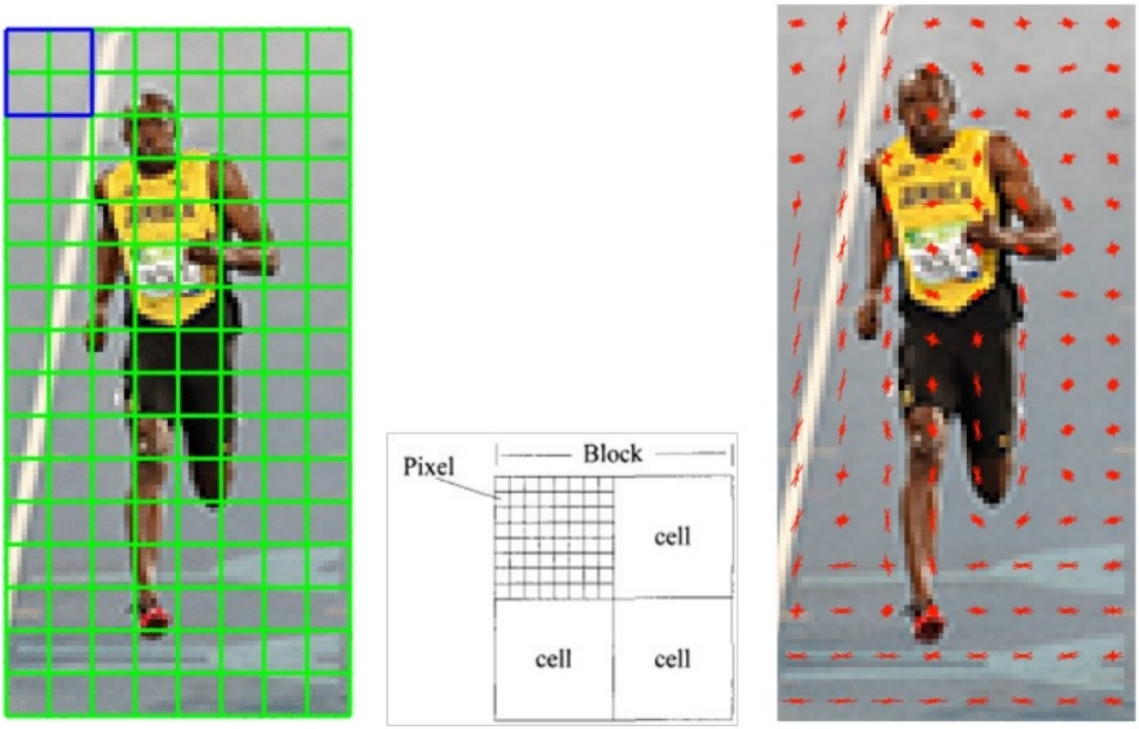


Hình 22: Đồ thị Histogram Gradient.

– Chuẩn hóa cụm cell theo block gồm 2x2 cell. Công thức chuẩn hóa:

$$normolize(h) = \frac{h}{|h|^2}$$

– Sau khi chuẩn hóa xong cho tát cả các block ta sẽ được hình biểu diễn đặc trưng của HoG.



Hình 23: Hình ảnh HoG thu được sau khi chuẩn hóa tất cả các block.

- Code mẫu: (xem thêm giải thích từ record [01:09:16-01:20:30])

```python
def preprocess_img(img):
    if len(img.shape) > 2:
        img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    img = img.astype(np.float32)

    resized_img = resize(
        img,
        output_shape=(32, 32),
        anti_aliasing=True
    )

    hog_feature = feature.hog(
        resized_img,
        orientations=9,
        pixels_per_cell=(8, 8),
        cells_per_block=(2, 2),
        transform_sqrt=True,
        block_norm="L2",
        feature_vector=True
    )

    return hog_feature
```

We resize all images to the same size to ensure HOG vectors have the same shape.

```python
img_features_lst = []
for img in img_lst:
    hog_feature = preprocess_img(img)
    img_features_lst.append(hog_feature)

img_features = np.array(img_features_lst)
print('X shape:')
print(img_features.shape)
```

```
X shape:
(1074, 324)
```

Hình 24: Code mẫu step 2.

2.3 Step 3: Đánh số cho label

Label Encoder	
'crosswalk'	0
'speedlimit'	1
'stop'	2

```
label_encoder = LabelEncoder()
encoded_labels = label_encoder.fit_transform(label_lst)
```

```
label_encoder.classes_
```

```
array(['crosswalk', 'speedlimit', 'stop'], dtype='<U10')
```

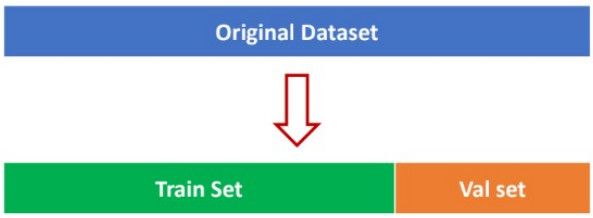
```
encoded_labels
```

```
array([1, 1, 2, ..., 0, 1, 0])
```

Hình 25: Endcode label.

2.4 Step 4: Tạo tập train và tập val

```
random_state = 0
test_size = 0.3
is_shuffle = True

X_train, X_val, y_train, y_val = train_test_split(
    img_features, encoded_labels,
    test_size=test_size,
    random_state=random_state,
    shuffle=is_shuffle
)
```



Hình 26: Tạo tập train (70) và tập val (30)

2.5 Step 5: Chuẩn hóa data

```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_val = scaler.transform(X_val)
```

$$z = \frac{x_i - \mu}{\sigma}$$

Hình 27: Chuẩn hóa cho tập train và val.

2.6 Step 6,7: Đưa vào hàm SVM và cho ra kết quả (xem giải thích code tại đoạn record [01:23:46-01:31:15])

```
clf = SVC(
    kernel='rbf',
    random_state=random_state,
    probability=True,
    C=0.5
)
clf.fit(X_train, y_train)
```

```
SVC(C=0.5, probability=True, random_state=0)
```

```
y_pred = clf.predict(X_val)
score = accuracy_score(y_pred, y_val)

print('Evaluation results on val set')
print('Accuracy: ', score)
```

```
Evaluation results on val set
Accuracy:  0.9659442724458205
```


```
input_img = img_lst[1]
normalized_img = preprocess_img(input_img)
y_pred = clf.predict([normalized_img])[0]
print(f'Normal prediction: {y_pred}')
y_pred_prob = clf.predict_proba([normalized_img])
prediction = np.argmax(y_pred_prob)
y_pred_prob = [f'{p:.10f}' for p in y_pred_prob[0]]
print(f'Probability of each class: {y_pred_prob}')
print(f'Class with highest probability: {prediction}')
```

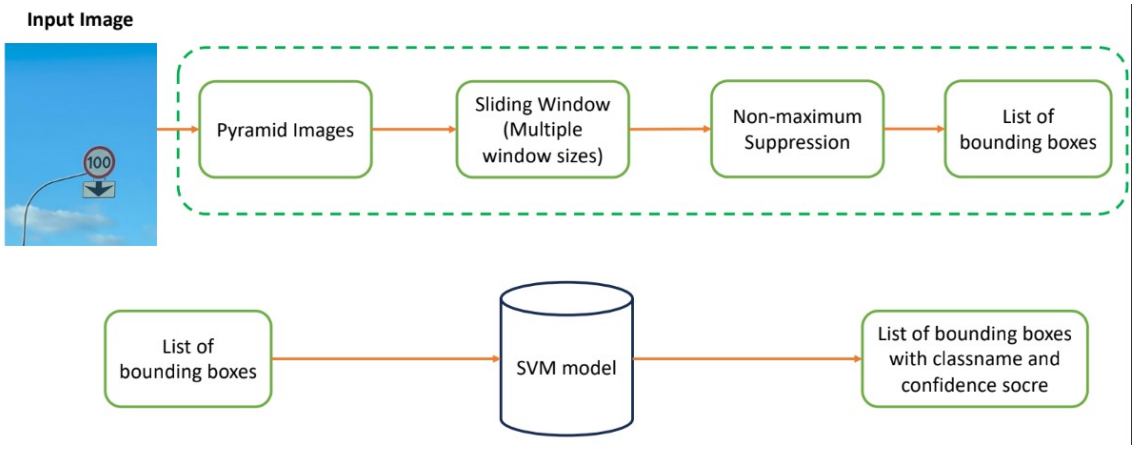
```
Normal prediction: 1
Probability of each class: ['0.0000000040', '0.9999306581', '0.0000693379']
Class with highest probability: 1
```

42

Hình 28: Code đưa vào mô hình SVM và kết quả.

3 Localization:

- Ý tưởng chung:



Hình 29: Ý tưởng xây dựng mô hình Localization và kết quả sau khi xây dựng thành công.

3.1 Sliding window:

- Là phần cốt lõi trong xây dựng mô hình Localization ta đang xét.
- Thường được sử dụng trong thị giác máy tính và âm thanh.
- Ý tưởng chung: Tạo ra một hình chữ nhật có kích thước cố định -> áp vào ảnh -> trượt trên ảnh từ trái sang phải từ trên xuống dưới -> nơi nào thấy được object thì làm bounding box luôn. (Giải thuật Gredy - tham lam)

Hình 30: Ý tưởng của Sliding window.

- Có 2 vấn đề cần quan tâm:
  - Kích thước object có thể khác nhau -> không thể dùng hình chữ nhật có kích thước cố định được.
  - Khoảng cách trượt theo chiều ngang (Stride) gây ra sự tỉ lệ nghịch cho thời gian và độ chính xác:
    * Stride càng nhỏ -> độ chính xác lớn -> tăng thời gian.
    * Stride càng lớn -> độ chính xác bé -> giảm thời gian.

Hình 31: 2 vấn đề chính của Sliding window.

- Code tạo window

```python
def sliding_window(img, window_sizes, stride, scale_factor):
    img_height, img_width = img.shape[:2]
    windows = []
    for window_size in window_sizes:
        window_width, window_height = window_size
        for ymin in range(0, img_height - window_height + 1, stride):
            for xmin in range(0, img_width - window_width + 1, stride):
                xmax = xmin + window_width
                ymax = ymin + window_height

                windows.append([xmin, ymin, xmax, ymax])

    return windows
```

Hình 32: Code mẫu phương pháp Sliding window.

3.2 Pyramid Images:

- Vấn đề đặt ra: Khi Sliding với window có kích thước cố định nhưng những object ở xa thì sẽ nhỏ hơn window đó rất nhiều -> phương pháp Pyramid Image trước khi đưa vào Sliding.

Hình 33: Khắc phục vấn đề object nhỏ hơn so mới Window bằng phương pháp Pyramid Images.

- Ý tưởng chính: Ta scale thu nhỏ dần tấm hình gốc lại để object bé được phóng lớn hơn -> window sẽ vừa và dễ dàng tìm thấy object hơn.

Hình 34: Scale ảnh theo các tỉ lệ khác nhau.

- Code:

```python
def pyramid(img, scale=0.8, min_size=(30, 30)):
    acc_scale = 1.0
    pyramid_imgs = [(img, acc_scale)]

    i = 0
    while True:
        acc_scale = acc_scale * scale
        h = int(img.shape[0] * acc_scale)
        w = int(img.shape[1] * acc_scale)
        if h < min_size[1] or w < min_size[0]:
            break
        img = cv2.resize(img, (w, h))
        pyramid_imgs.append((img, acc_scale * (scale ** i)))
        i += 1

    return pyramid_imgs
```

Hình 35: Code mẫu phương pháp Pyramid Images.

- Ảnh động minh họa hiệu quả hình ảnh sau khi được Paramid Images và chạy Sliding Window Minh họa.

3.3 Thể hiện bounding box và classname:

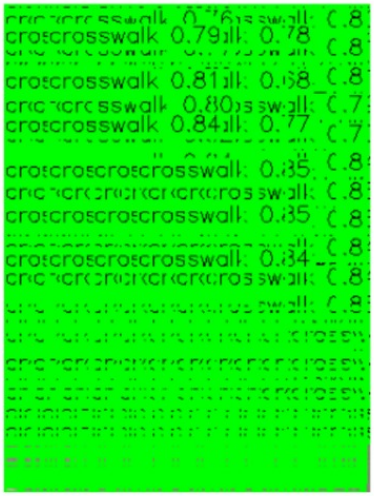
- Xem thêm giải thích từ record [01:53:54-01:57:40]

```python
def visualize_bbox(img, bboxes, label_encoder):
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

    for box in bboxes:
        xmin, ymin, xmax, ymax, predict_id, conf_score = box

        cv2.rectangle(img, (xmin, ymin), (xmax, ymax), (0, 255, 0), 2)

        classname = label_encoder.inverse_transform([predict_id])[0]
        label = f"{classname} {conf_score:.2f}"

        (w, h), _ = cv2.getTextSize(label, cv2.FONT_HERSHEY_SIMPLEX, 0.6, 1)

        cv2.rectangle(img, (xmin, ymin - 20), (xmin + w, ymin), (0, 255, 0), -1)

        cv2.putText(img, label, (xmin, ymin - 5), cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 0, 0), 1)

    plt.imshow(img)
    plt.axis('off')
    plt.show()
```

3.4 Dự đoán:

- Đưa hình chương trình để cho ra bounding box cùng với classname kèm theo. Ta thấy hình ảnh cho ra bị vấn đề là chồng chéo tất cả các dự đoán.

```python
for pyramid_img_info in pyramid_imgs:
    pyramid_img, scale_factor = pyramid_img_info
    window_lst = sliding_window(
        pyramid_img,
        window_sizes=window_sizes,
        stride=stride,
        scale_factor=scale_factor
    )
    for window in window_lst:
        xmin, ymin, xmax, ymax = window
        object_img = pyramid_img[ymin:ymax, xmin:xmax]
        preprocessed_img = preprocess_img(object_img)
        normalized_img = scaler.transform([preprocessed_img])[0]
        decision = clf.predict_proba([normalized_img])[0]
        predict_id = np.argmax(decision)
        conf_score = decision[predict_id]
        xmin = int(xmin / scale_factor)
        ymin = int(ymin / scale_factor)
        xmax = int(xmax / scale_factor)
        ymax = int(ymax / scale_factor)
        bboxes.append(
            [xmin, ymin, xmax, ymax, predict_id, conf_score]
        )
```



Hình 36: Lỗi khi dự đoán.

- Để khắc phục lỗi này, ta đưa vào giá trị confidence threshold để lọc: hình nào có confidence score lớn hơn thì thể hiện.

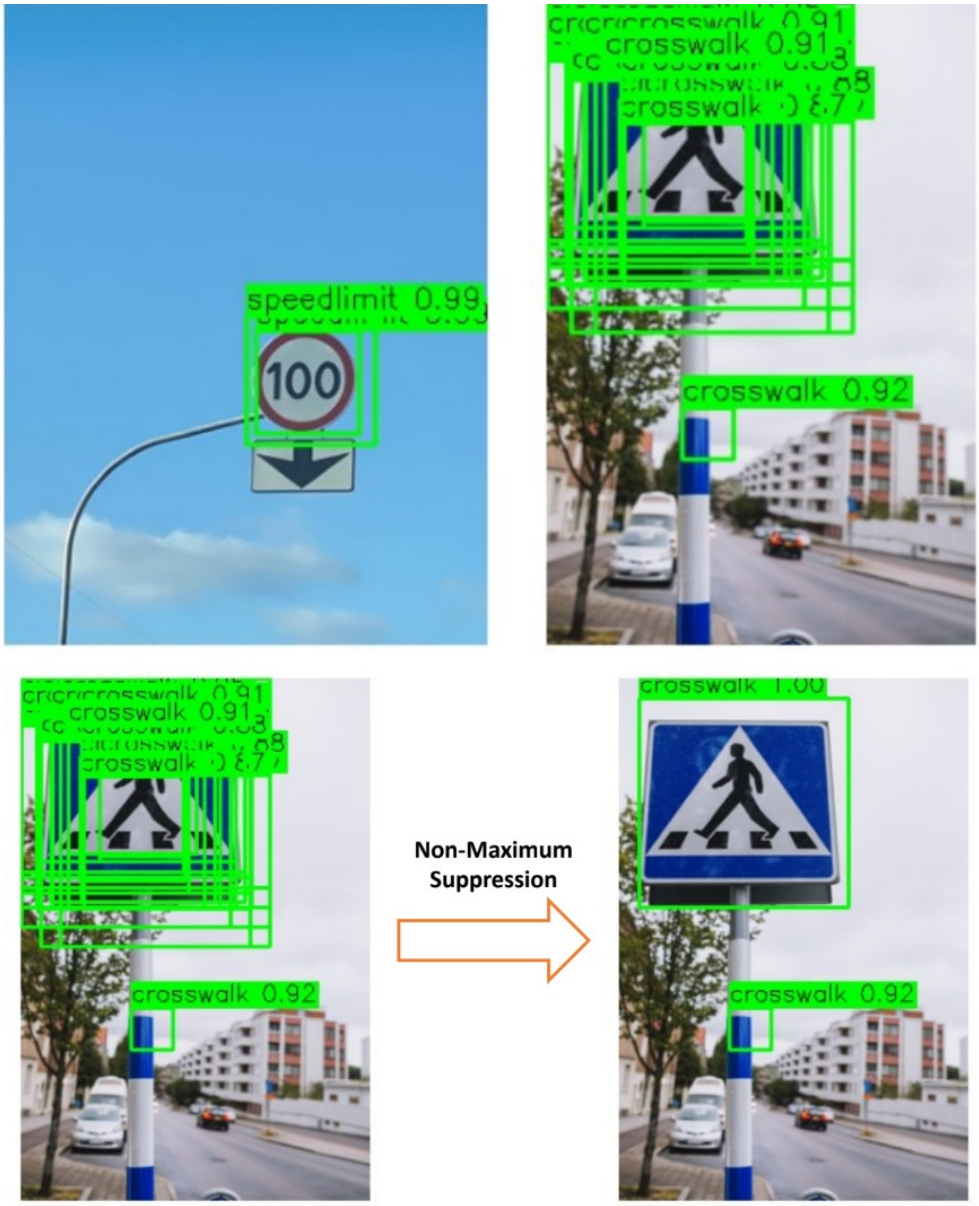
```
for window in window_lst:
    xmin, ymin, xmax, ymax = window
    object_img = pyramid_img[ymin:ymax, xmin:xmax]
    preprocessed_img = preprocess_img(object_img)
    normalized_img = scaler.transform([preprocessed_img])[0]
    decision = clf.predict_proba([normalized_img])[0]
    if np.all(decision < conf_threshold):
        continue
    else:
        predict_id = np.argmax(decision)
        conf_score = decision[predict_id]
        xmin = int(xmin / scale_factor)
        ymin = int(ymin / scale_factor)
        xmax = int(xmax / scale_factor)
        ymax = int(ymax / scale_factor)
        bboxes.append(
            [xmin, ymin, xmax, ymax, predict_id, conf_score]
        )
```

Hình 37: Thêm giá trị lọc Confidence Threshold.

Xem thêm giải thích code từ record [01:58:09-02:05:00]

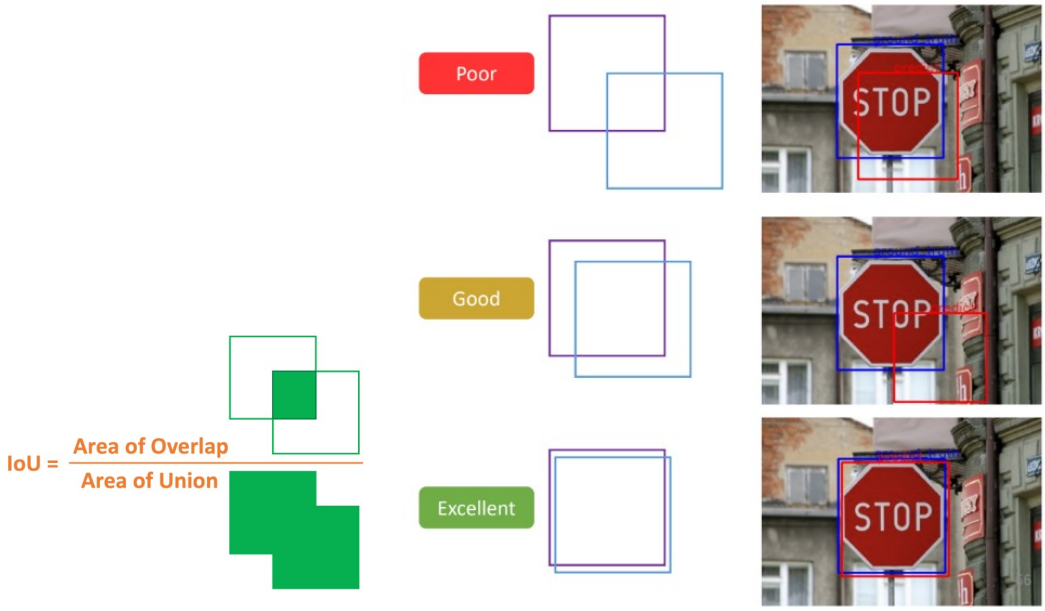
3.5 Non-maximum suppression:

- Trong một số trường hợp nhiều bounding box sẽ dự đoán cùng 1 object gây ra hiện tượng đè nhau (Overlapped). -> dùng phương pháp Non-Maxinum Suppression để loại bỏ những bounding box đè nhau.
- Chú ý: Non-maximum Suppression chỉ lọc đè nhau không lọc đúng sai của label.

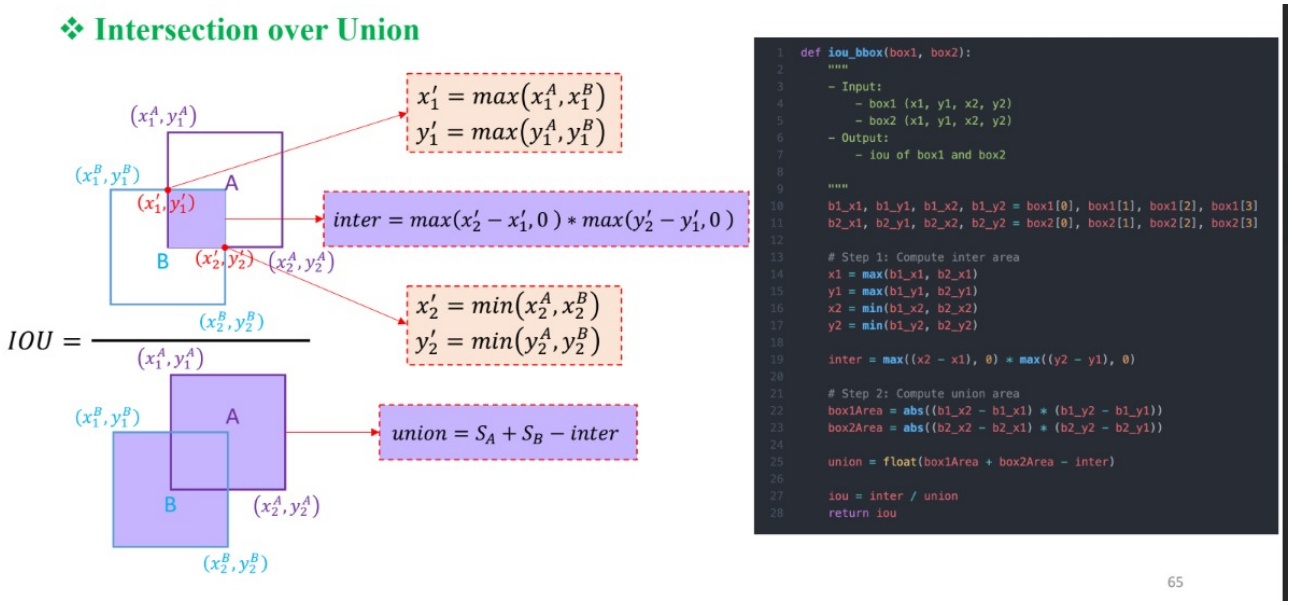


Hình 38: Overlapped bounding box problem.

- Ý tưởng của Non-Maximum Suppression:
  - Sắp xếp các bounding box giảm dần theo giá trị confidence score.
  - Tính IoU giữ giá trị bounding box có confidence score lớn nhất và tất cả những cái còn lại.
  - Giữ lại những bounding box có IoU $<$ IoU threshold.
- IoU: Intersection over Union là thông số đo độ đè nhau giữa 2 bounding box. Cách tính IoU được đề cập ở hình 40 cùng với code kèm theo.



Hình 39: IoU và các trạng thái.



Hình 40: Công thức tính IoU.

- Code của Non-maximum suppression:

```python
def nms(bboxes, iou_threshold):
    if not bboxes:
        return []
    scores = np.array([bbox[5] for bbox in bboxes])
    sorted_indices = np.argsort(scores)[::-1]

    xmin = np.array([bbox[0] for bbox in bboxes])
    ymin = np.array([bbox[1] for bbox in bboxes])
    xmax = np.array([bbox[2] for bbox in bboxes])
    ymax = np.array([bbox[3] for bbox in bboxes])

    areas = (xmax - xmin + 1) * (ymax - ymin + 1)

    keep = []
    while sorted_indices.size > 0:
        i = sorted_indices[0]
        keep.append(i)

        iou = compute_iou(
            [xmin[i], ymin[i], xmax[i], ymax[i]],
            np.array(
                [
                    xmin[sorted_indices[1:]],
                    ymin[sorted_indices[1:]],
                    xmax[sorted_indices[1:]],
                    ymax[sorted_indices[1:]]]
            ).T,
            areas[i],
            areas[sorted_indices[1:]]
        )

        idx_to_keep = np.where(iou <= iou_threshold)[0]
        sorted_indices = sorted_indices[idx_to_keep + 1]

    return [bboxes[i] for i in keep]
```


Hình 41: Code mẫu hàm Non-maximum suppression.

Xem thêm giải thích code từ record [02:19:31-02:26:12]. Từ sau trở về cuối record là phần hỏi đáp lại thắc mắc của toàn bài.

Kết quả cho ra:

Hình 42: Kết quả cuối cùng.