AI VIETNAM
All-in-One Course
(TA Session)

# Traffic Sign Detection

## Project


Dinh-Thang Duong – TA

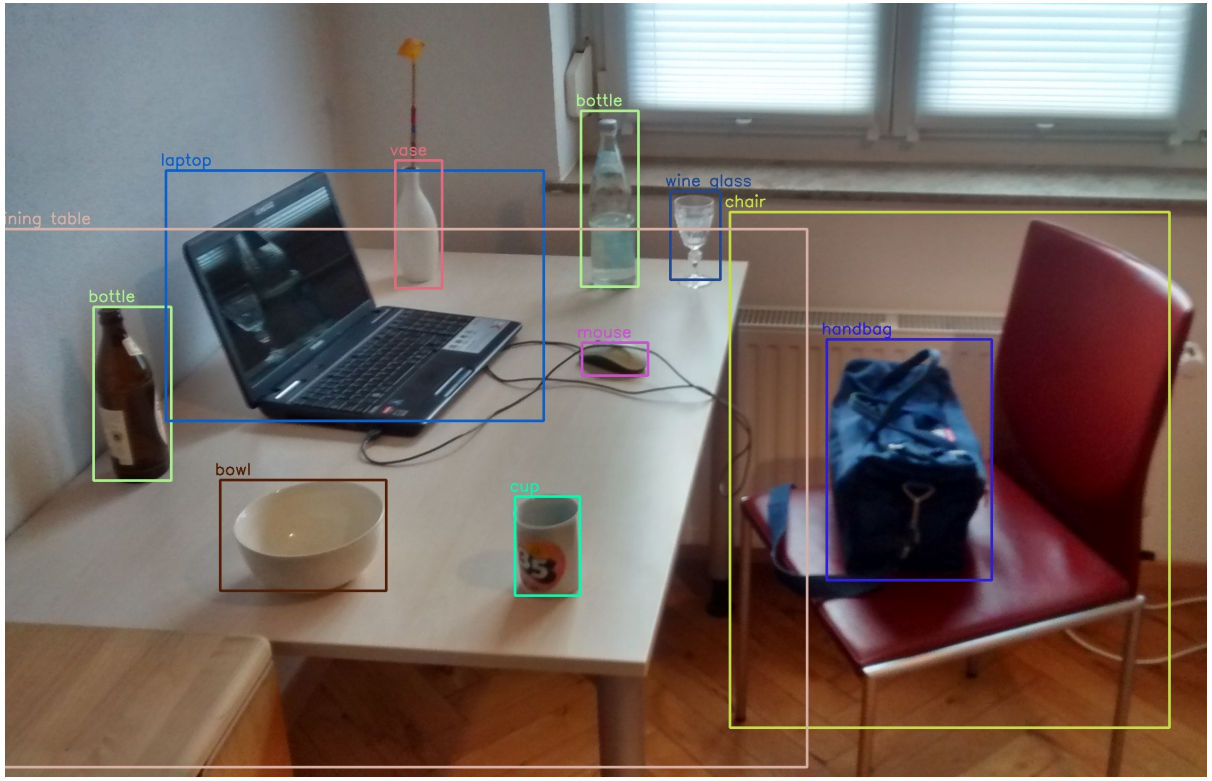
Year 2023

# Outline

- Introduction
- Project Pipeline
- Traffic Sign Classification
- Traffic Sign Localization
- Question

# Introduction

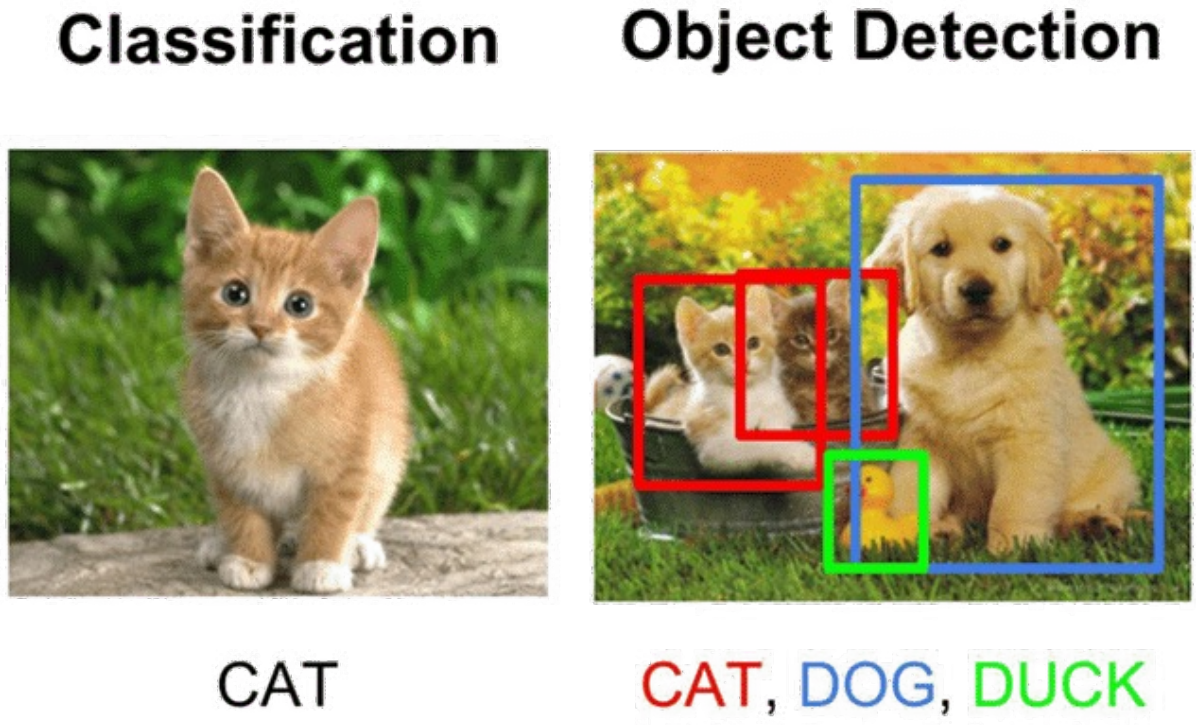
## ❖ Object Detection




**Object Detection:** A computer vision task that involves indentifying and locating objects of interest in an image or video.

# Introduction

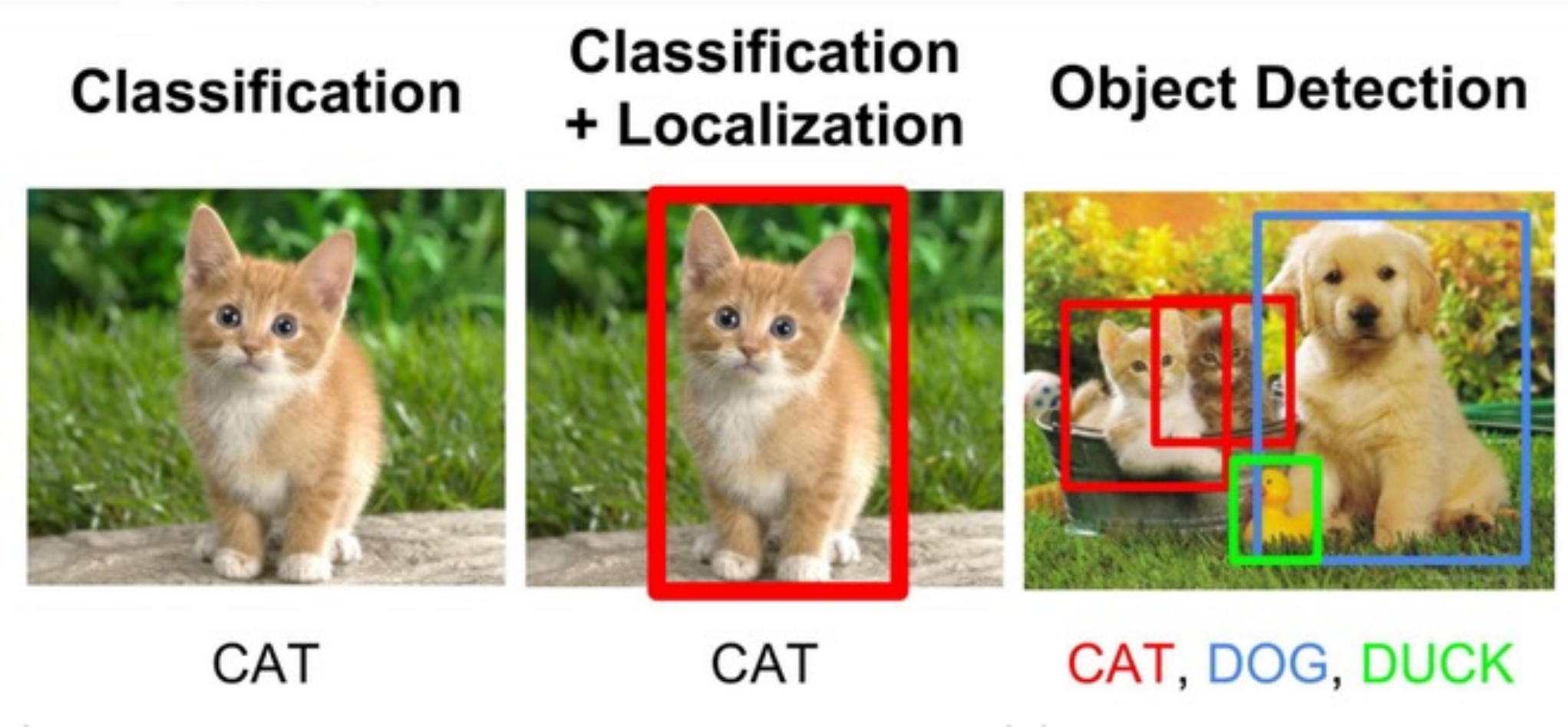
## ❖ Image Classification vs Object Detection




Unlike image classification, where the goal is to assign a label to an entire image, object detection aims to **identify multiple objects** and **provide bounding boxes** around them, specifying where each of object is in the image.

# Introduction

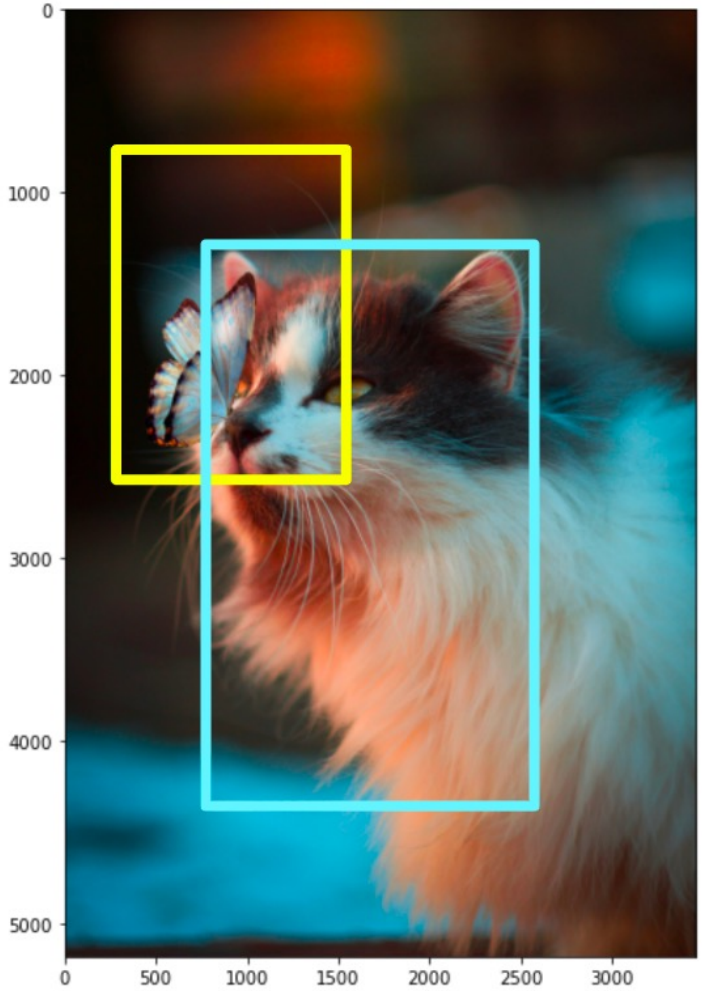
## ❖ Two-stage Object Detection




**Two-stage Object Detection:** A combination of Object Localization and Object Classification.

# Introduction

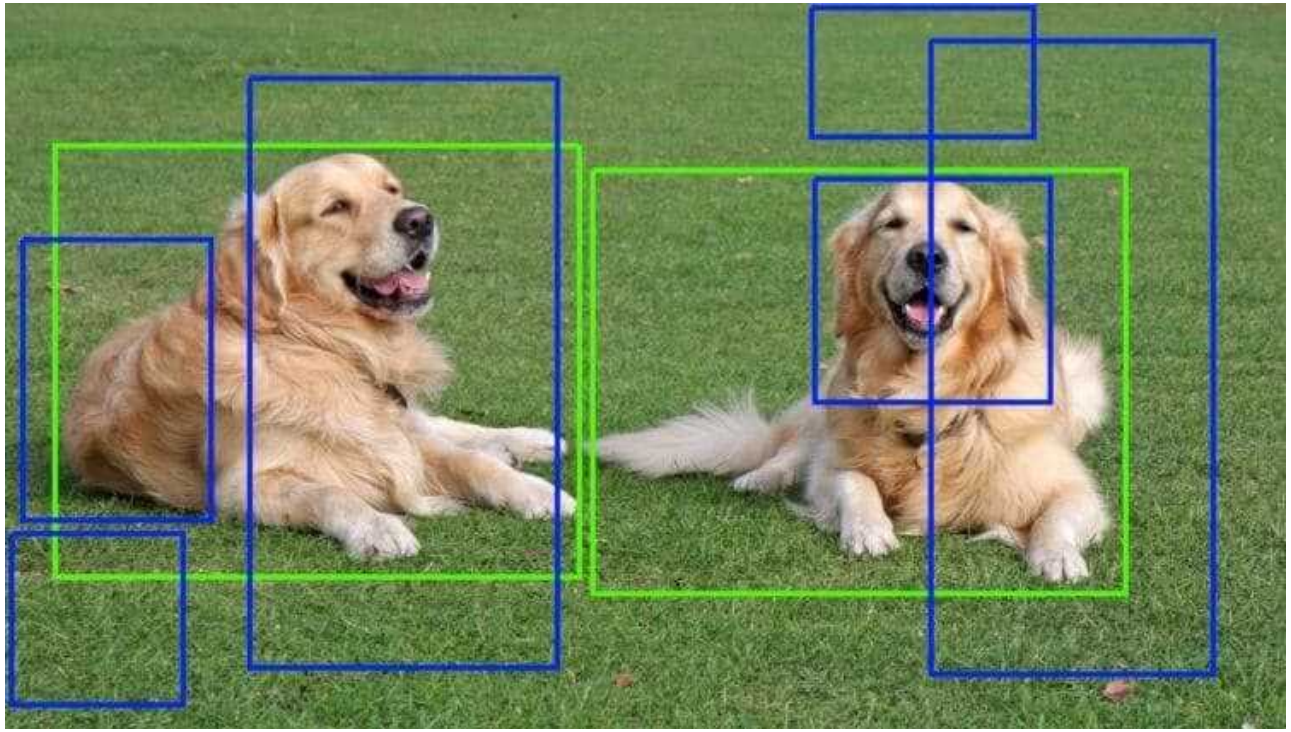
## ❖ Object Localization




**Object Localization:** A computer vision task that seeks to pinpoint the location of a specific object in an image. The primary output for object localization is usually a bounding box that encompasses the objects of interest.

# Introduction

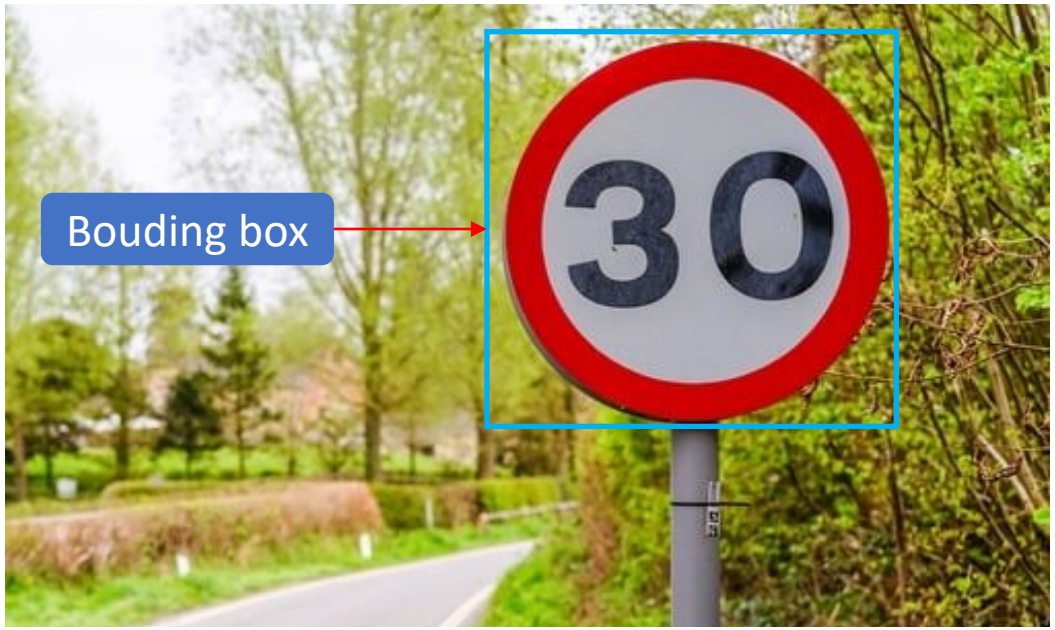
## ❖ Object Localization



**Example:** Multiple bounding boxes are presented to identify the position of two dogs. Green seems more accurate than Blue.

# Introduction

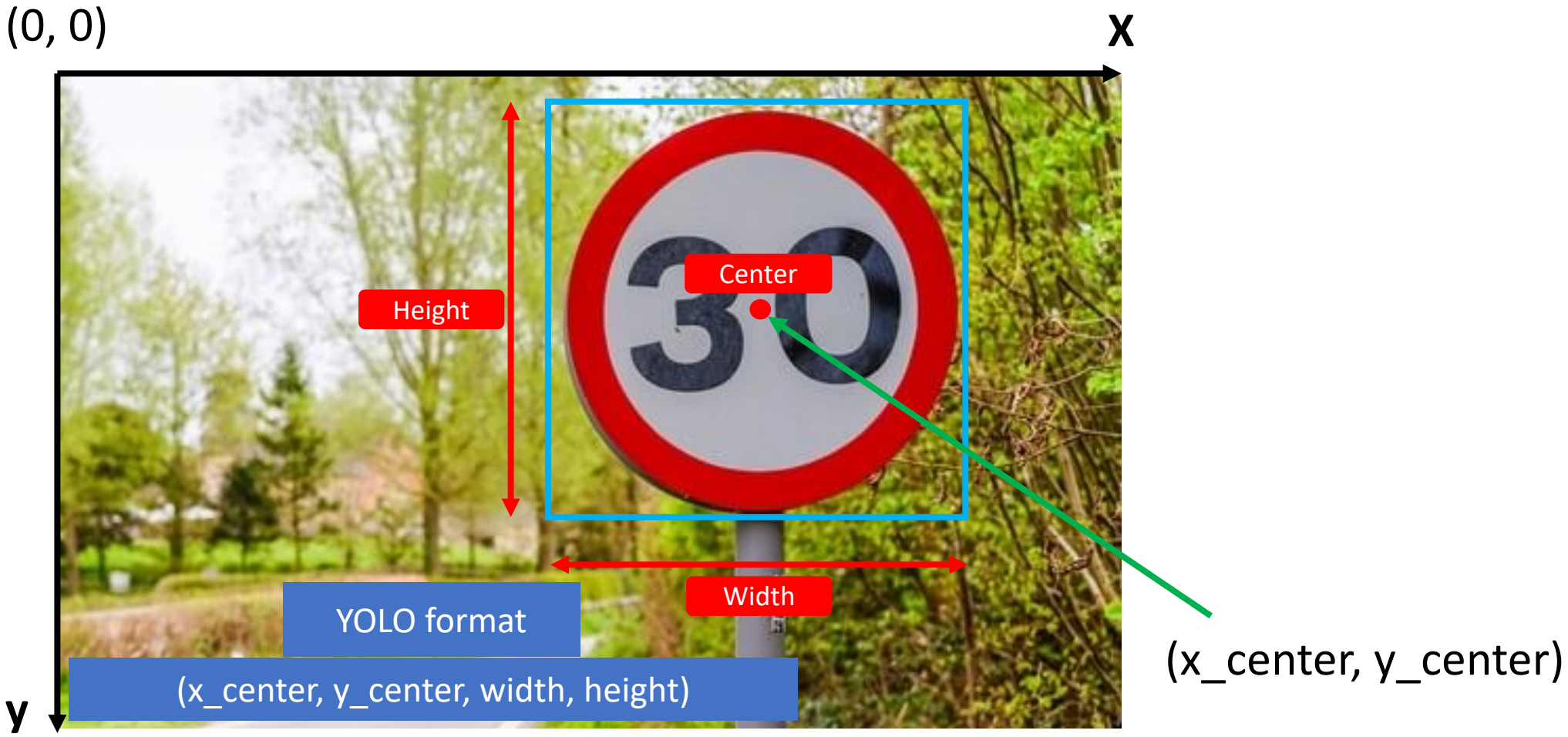
## ❖ Object Localization: Bounding box




**Bounding box:** A rectangular box used to describe the location of an object within an image. Bounding box is often represented as a tuple of (x_center, y_center, width, height) or (xmin, ymin, xmax, ymax).

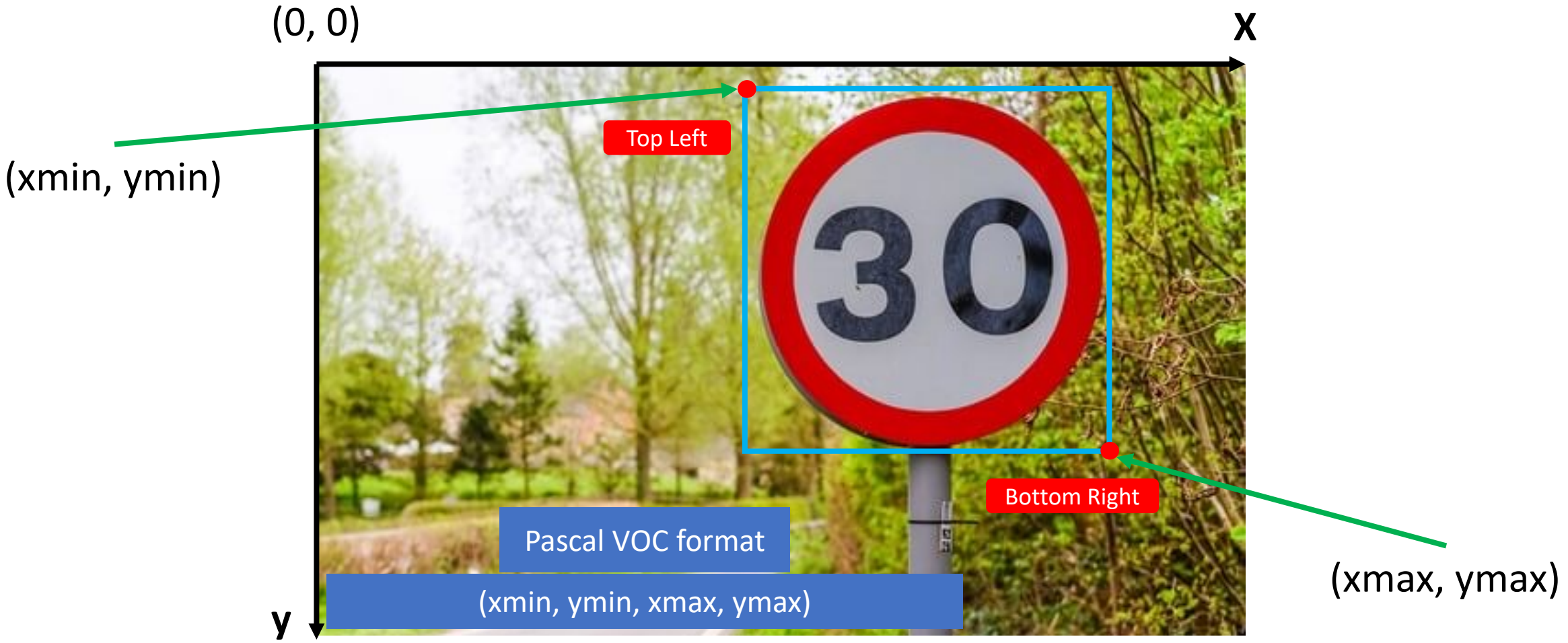
# Introduction

## ❖ Object Localization: Bounding box



# Introduction

## ❖ Object Localization: Bounding box



# Introduction

## ❖ Object Localization: Draw bounding box to an image

***Syntax:*** *cv2.rectangle(image, start_point, end_point, color, thickness)*

***Parameters:***

***image:*** *It is the image on which rectangle is to be drawn.*

***start_point:*** *It is the starting coordinates of rectangle. The coordinates are represented as tuples of two values i.e. (**X** coordinate value, **Y** coordinate value).*

***end_point:*** *It is the ending coordinates of rectangle. The coordinates are represented as tuples of two values i.e. (**X** coordinate value, **Y** coordinate value).*

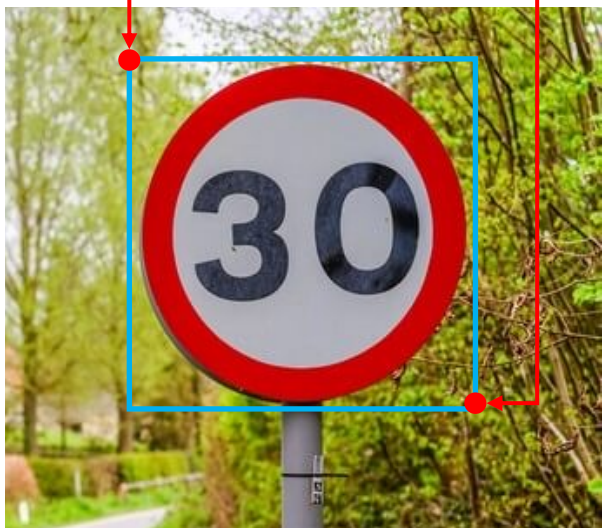
***color:*** *It is the color of border line of rectangle to be drawn. For **BGR**, we pass a tuple. eg: (255, 0, 0) for blue color.*

***thickness:*** *It is the thickness of the rectangle border line in **px**. Thickness of **-1 px** will fill the rectangle shape by the specified color.*

***Return Value:*** *It returns an image.*

Draw bounding box into image using cv2.rectangle

```
1  image = cv2.rectangle(image, top_left, bottom_right, color, thinkness)
```




We can use cv2.rectangle() to draw a bounding box

# Introduction

## ❖ Object Localization: Convert bounding box format


Given a tuple bbox of (x_center, y_center, width, height):

```python
def xywh2xyxy(bbox):
    """
    Input:
        - bbox: Tuple(x_center, y_center, width, height)
    Output:
        - Tuple(x1, y1, x2, y2) Where:
            - x1, y1: Top left point
            - x2, y2: Bottom right point
    """
    x1 = bbox[0] - bbox[2] / 2  # top left x
    y1 = bbox[1] - bbox[3] / 2  # top left y
    x2 = bbox[0] + bbox[2] / 2  # bottom right x
    y2 = bbox[1] + bbox[3] / 2  # bottom right y
    return (x1, y1, x2, y2)
```

# Introduction

## ❖ Object Localization: Convert bounding box format

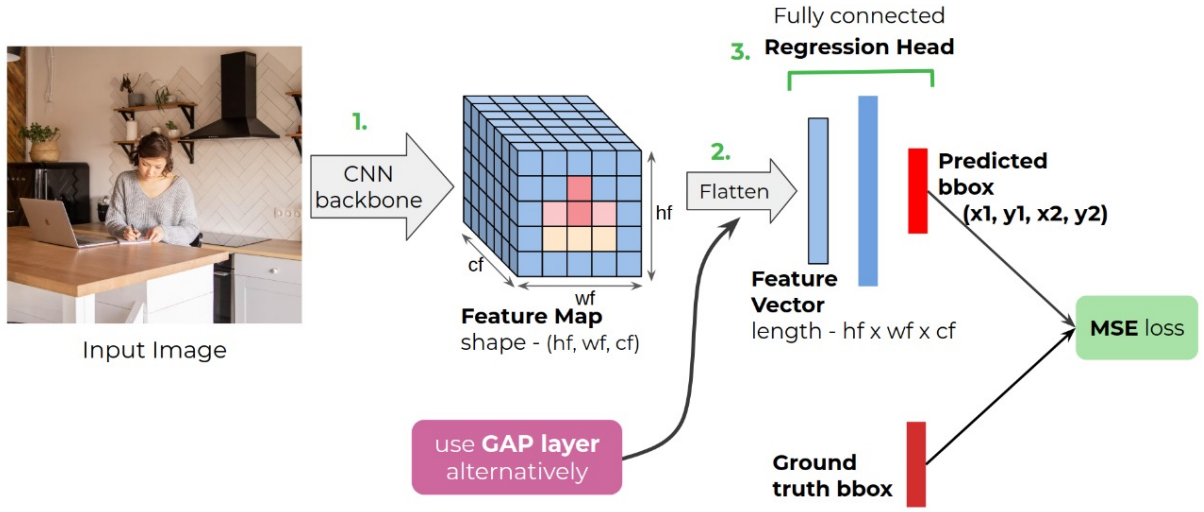

crop_object(img, bbox)


Given an image and a tuple bbox of (xmin, ymin, xmax, ymax):

```python
def crop_object(img, bbox):
    xmin, ymin, xmax, ymax = bbox
    object_img = img[ymin:ymax, xmin:xmax]

    return object_img
```

# Introduction

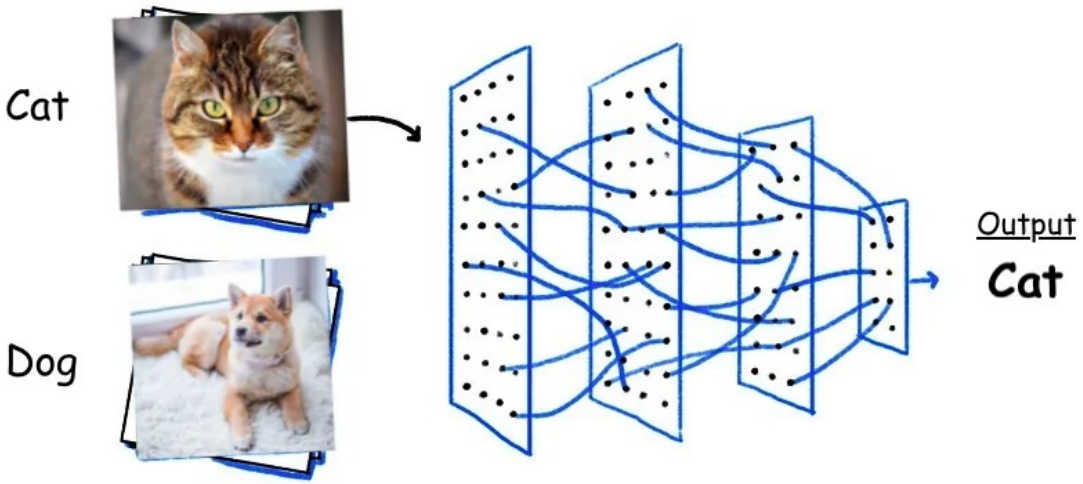
## ❖ Object Localization: Methods




There are several methods to solve the object localization task. In this project, we will use **Sliding Window** technique.

# Introduction

## ❖ Object Classification

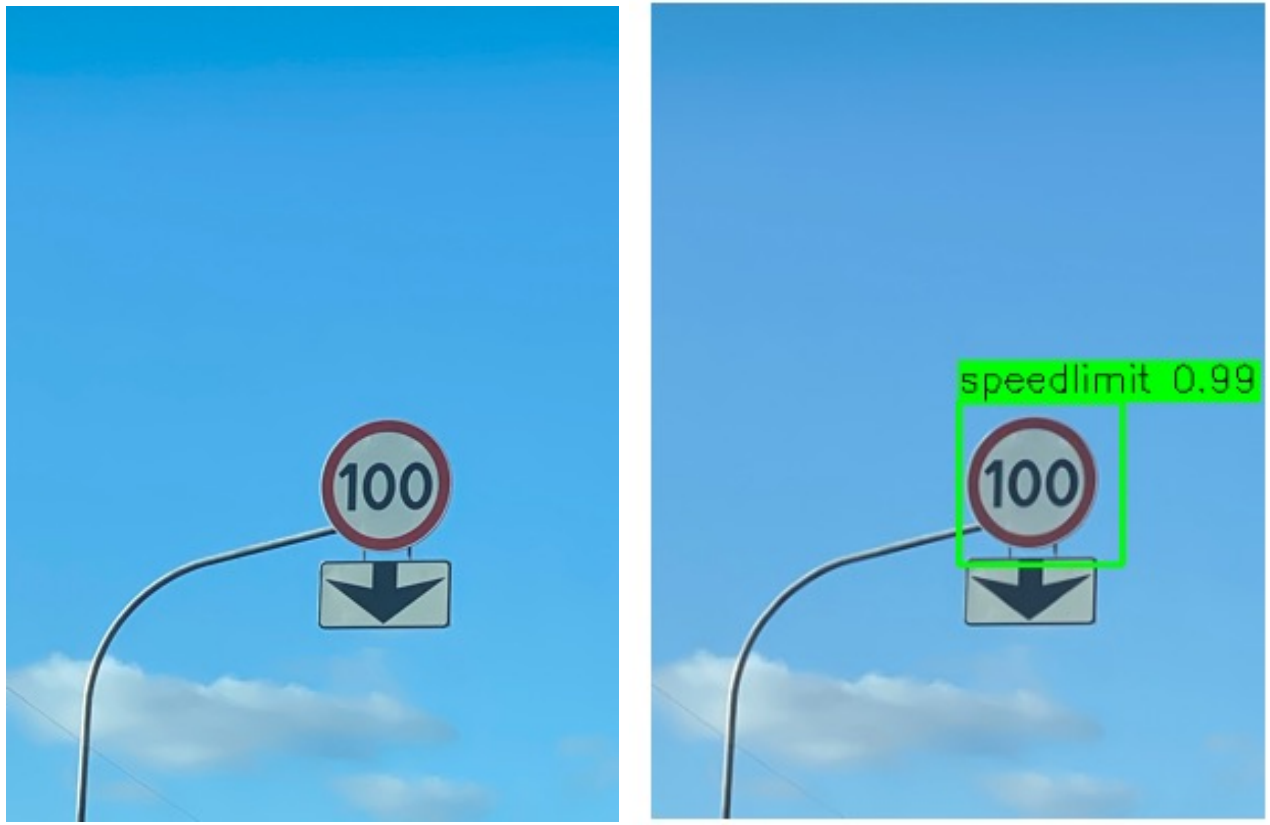



**Object Classification (Image Classification):** A computer vision task that involes assigning a predefined category or label to an image. In this project, we will use SVM to build an image classifier.

# Introduction

## ❖ Object Detection Output

**Output:** A list of predicted bounding boxes in the form of (xmin, ymin, xmax, ymax, confidence_score)




Output Visualization

# Introduction

## ❖ Object Detection Confidence Score

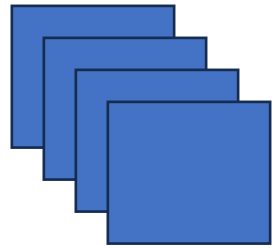

**Confidence Score:** A value represents how 'confident' the model is to assign a class to the corresponding object. Confidence Score is often **taken from the output of Object Classifier**.

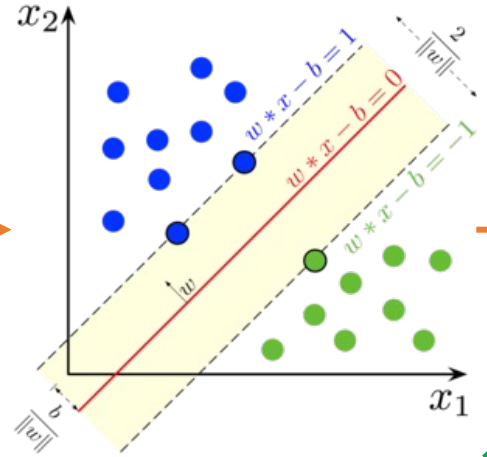
# Project Pipeline

## ❖ Introduction

Object Detection Dataset

Get all objects and their label





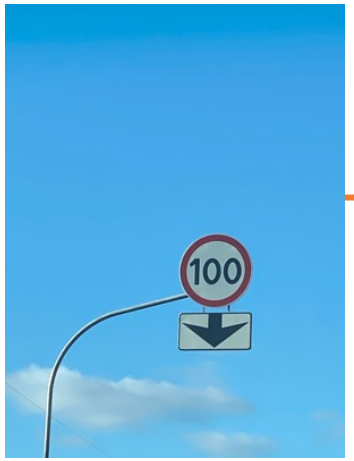

**Object Classification**

Classification


**Output**

List of bounding boxes

**Input Image**




Pyramid Images

Sliding Window (Multiple window sizes)

Non-maximum Suppression

**Object Localization**

# Project Pipeline

## ❖ Import essential libraries

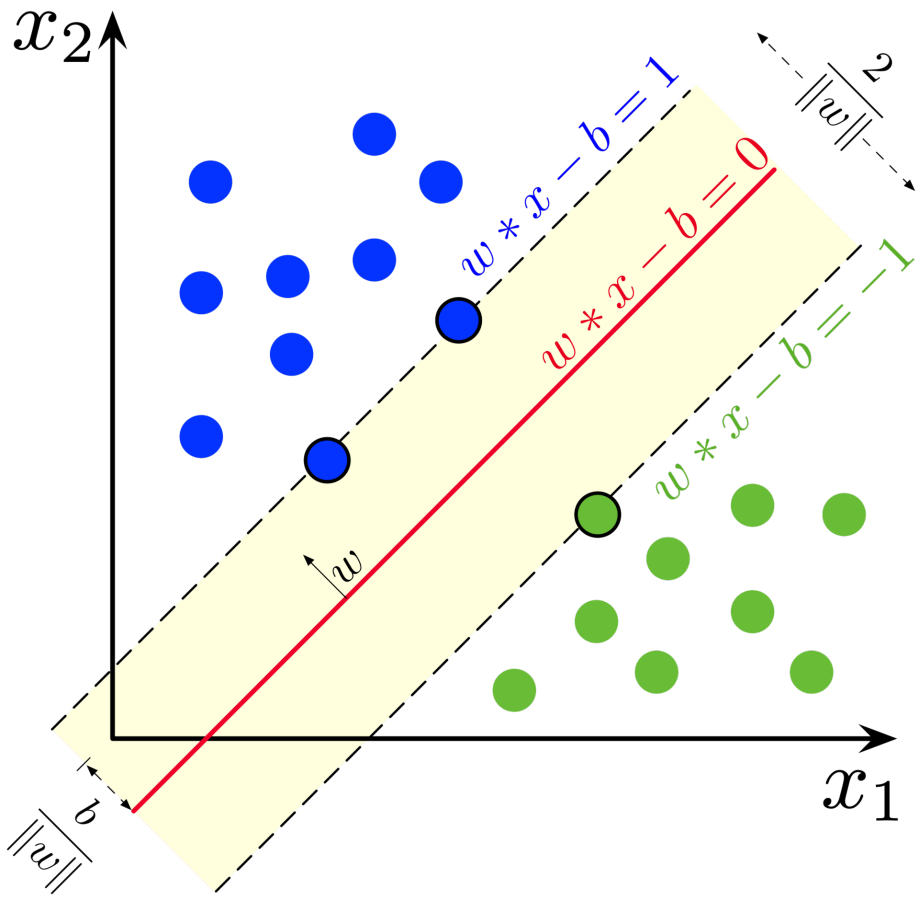
```
1 import time
2 import os
3 import cv2
4 import numpy as np
5 import pandas as pd
6 import matplotlib.pyplot as plt
7 import xml.etree.ElementTree as ET
8
9 from skimage.transform import resize
10 from skimage import feature
11 from sklearn.svm import SVC
12 from sklearn.preprocessing import LabelEncoder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

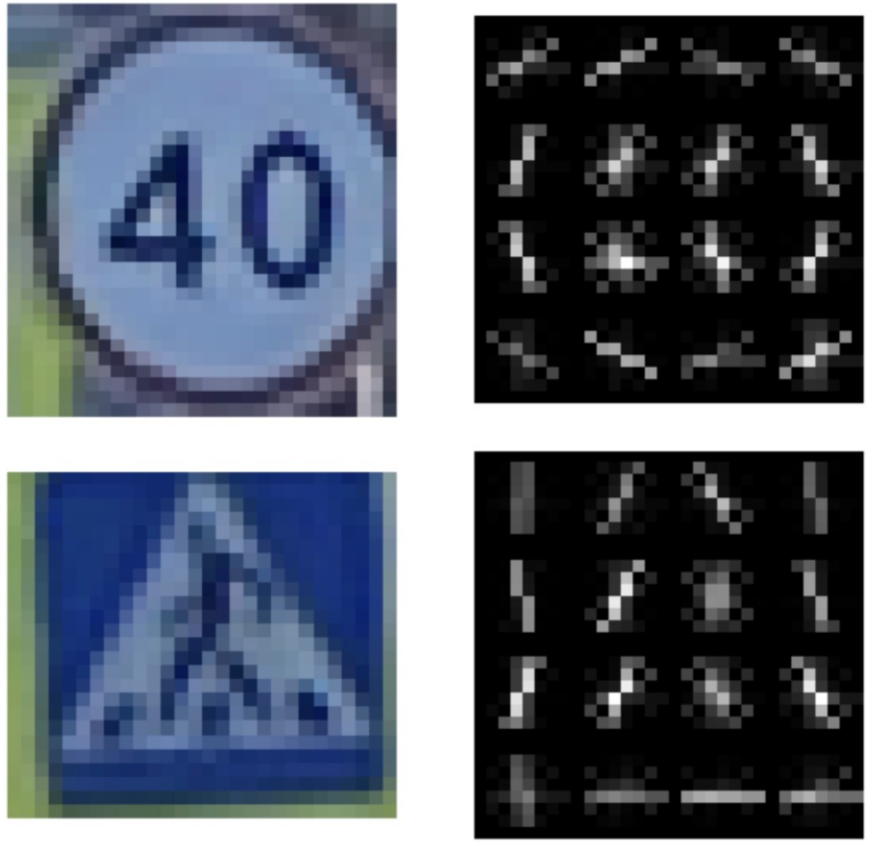

scikit-image
image processing in python

# Traffic Sign Classification

## ❖ Introduction

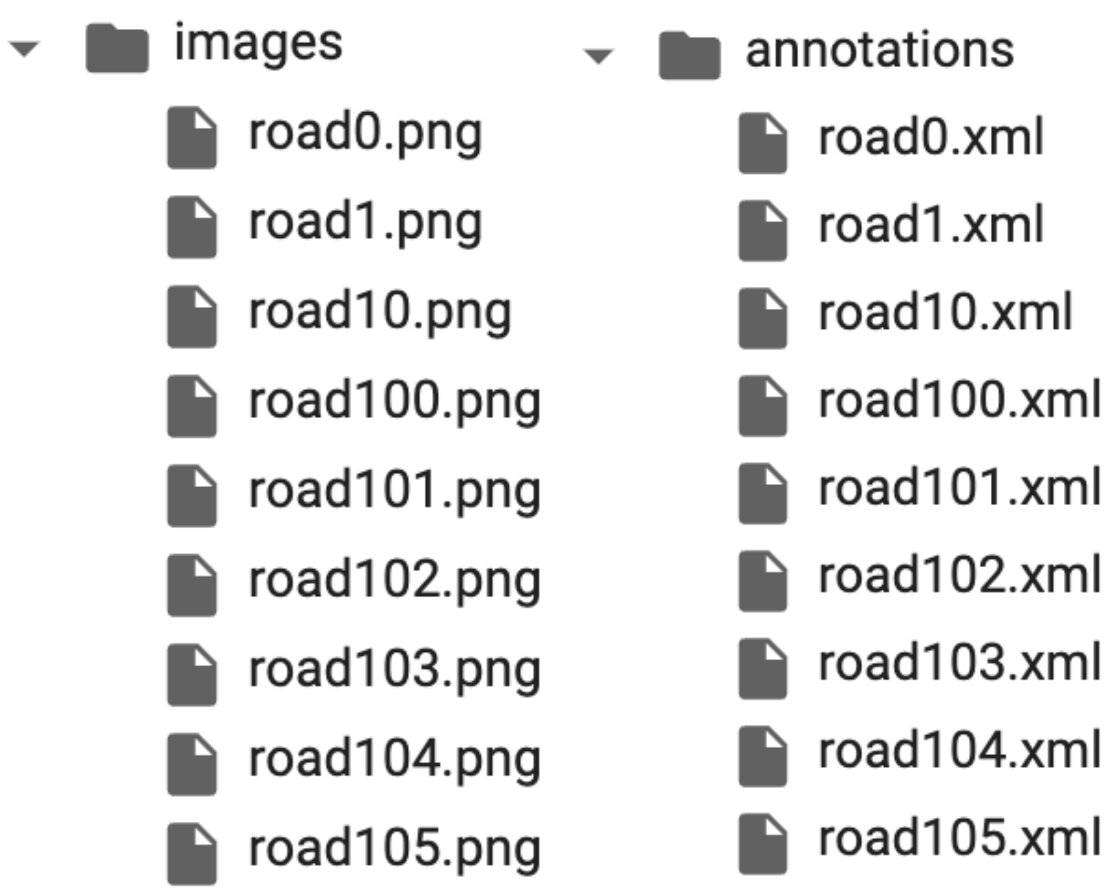
**Description:** Given a dataset of Traffic Sign Detection. Build a Traffic Sign Image Classifier using SVM model.



# Traffic Sign Classification

## ❖ Step 1: Read dataset




```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>
```

# Traffic Sign Classification

## ❖ Step 1: Read dataset

```
<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>
```

Folder contains the image

Filename of the image

Shape of the image

Label of the object (classname)

Bounding box (Pascal VOC format) of the object

# Traffic Sign Classification

## ❖ Step 1: Read dataset

```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>
```

### `xml.etree.ElementTree` — The ElementTree XML API

**Source code:** Lib/xml/etree/ElementTree.py

The `xml.etree.ElementTree` module implements a simple and efficient API for parsing and creating XML data.

*Changed in version 3.3:* This module will use a fast implementation whenever available.

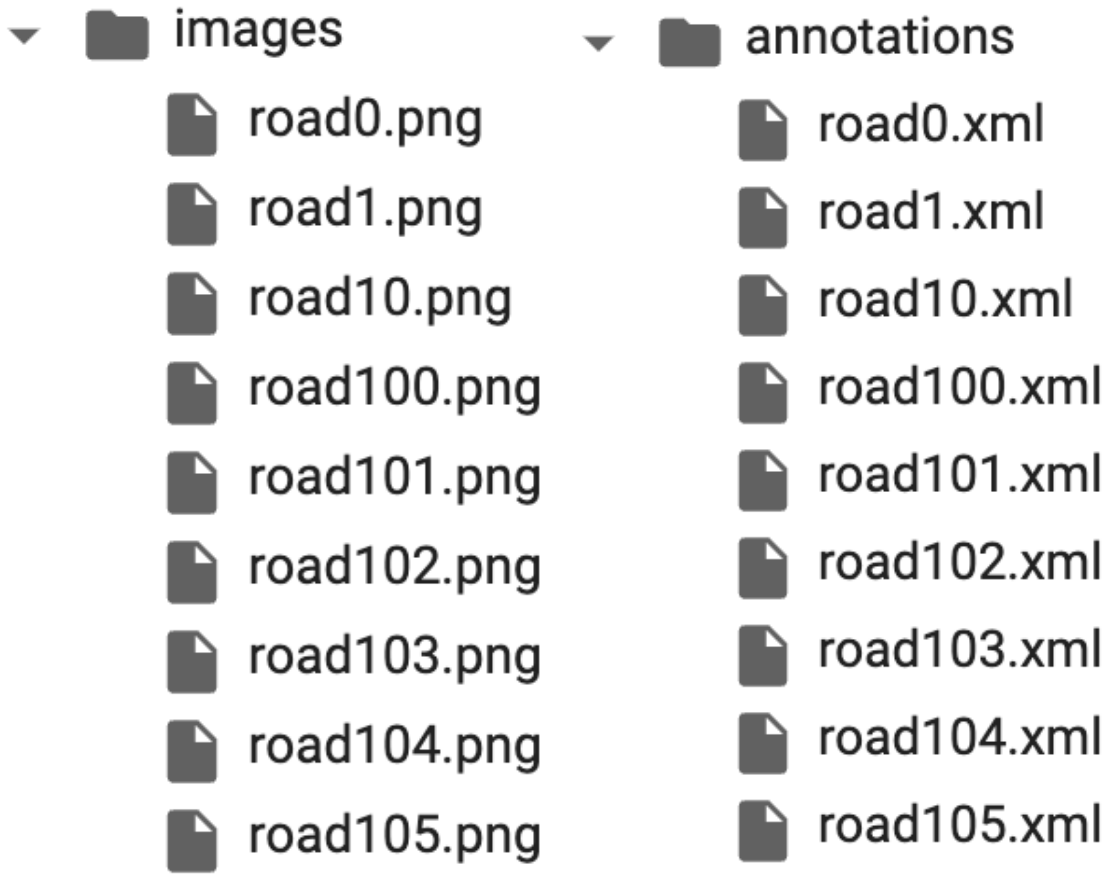
*Deprecated since version 3.3:* The `xml.etree.cElementTree` module is deprecated.

```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
```

# Traffic Sign Classification

## ❖ Step 1: Read dataset




We want to extract all objects as images using bounding box and their corresponding label as string.

# Traffic Sign Classification

## ❖ Step 1: Read dataset

```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)
```

# Traffic Sign Classification

## ❖ Step 1: Read dataset


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

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

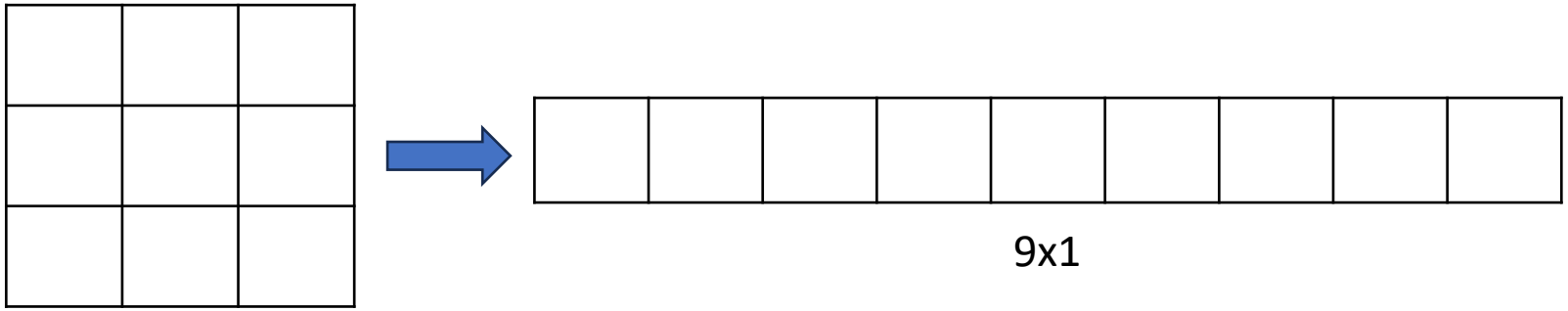
```
annotations_dir = 'annotations'
img_dir = 'images'

img_lst = []
label_lst = []
for xml_file in os.listdir(annotations_dir):
    xml_filepath = os.path.join(annotations_dir, xml_file)
    tree = ET.parse(xml_filepath)
    root = tree.getroot()

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

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

        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)

        object_img = img[ymin:ymax, xmin:xmax]
        img_lst.append(object_img)
        label_lst.append(classname)
```

We ignore 'trafficlight' class since we do Traffic Sign Classification

# Traffic Sign Classification

## ❖ Step 2: Preprocess image


Currently, we cannot feed the entire image (len(shape) == 3) as X to train an SVM model.




```
1 print('Image shape:')
2 print(img_lst[1].shape)
```

```
Image shape:
(53, 55, 3)
```

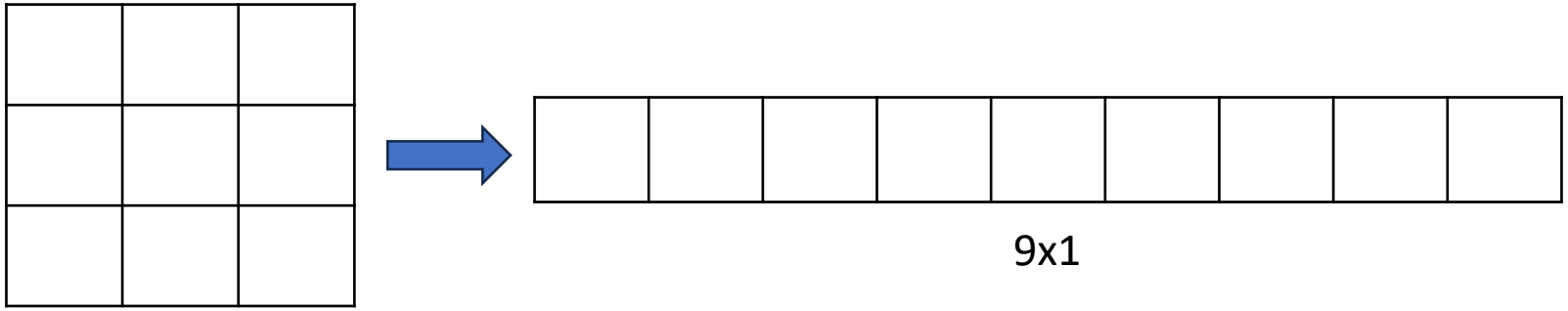
We can flatten image to have the len(shape) == 1.

```
1 print('Image shape:')
2 print(img_lst[1].shape)
3 flattened_img = img_lst[1].flatten()
4 print('Image shape after flattened:')
5 print(flattened_img.shape)
```

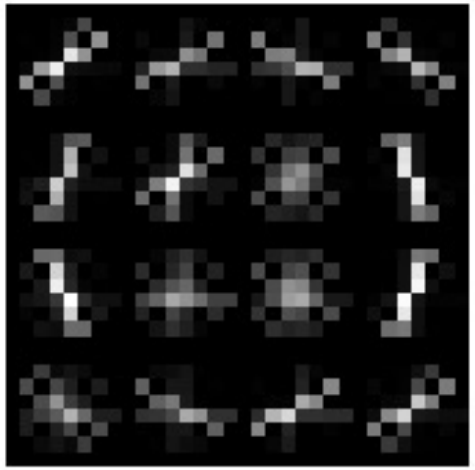
```
Image shape:
(53, 55, 3)
Image shape after flattened:
(8745,)
```

# Traffic Sign Classification

## ❖ Step 2: Preprocess image





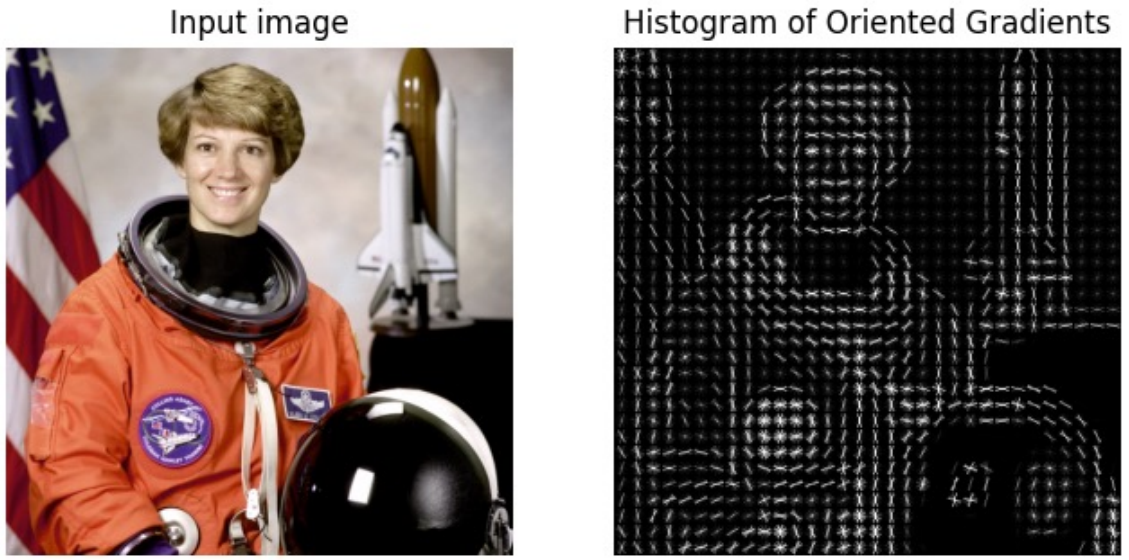


We need to use a better representation

**Flattening on raw image** does not seem good enough since the vector does not represent the image well.

# Traffic Sign Classification

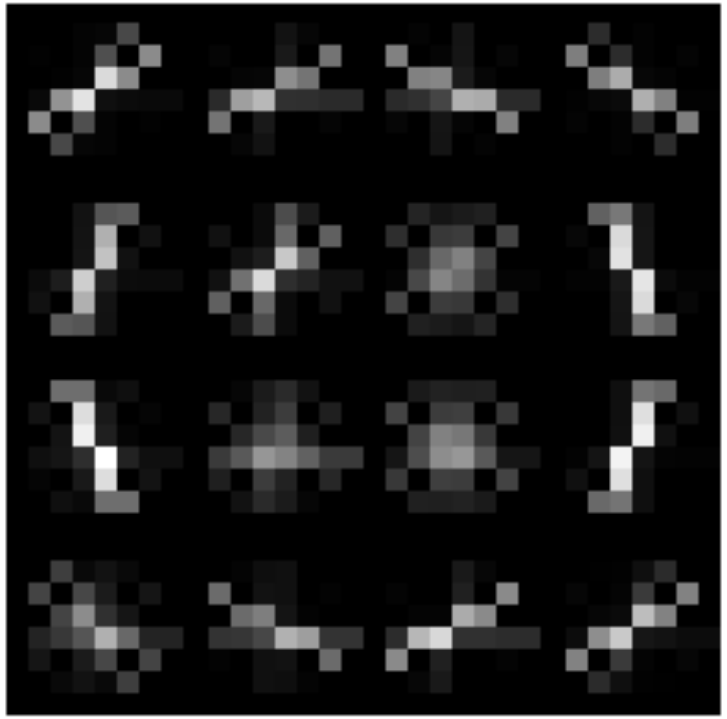
## ❖ HoG




**Histogram of Oriented Gradients (HoG):** A feature descriptor used primarily in object detection. It was introduced in a 2005 paper by Dalal and Triggs, where it was applied to pedestrian detection and achieved significant improvements over then-existing algorithms.

# Traffic Sign Classification

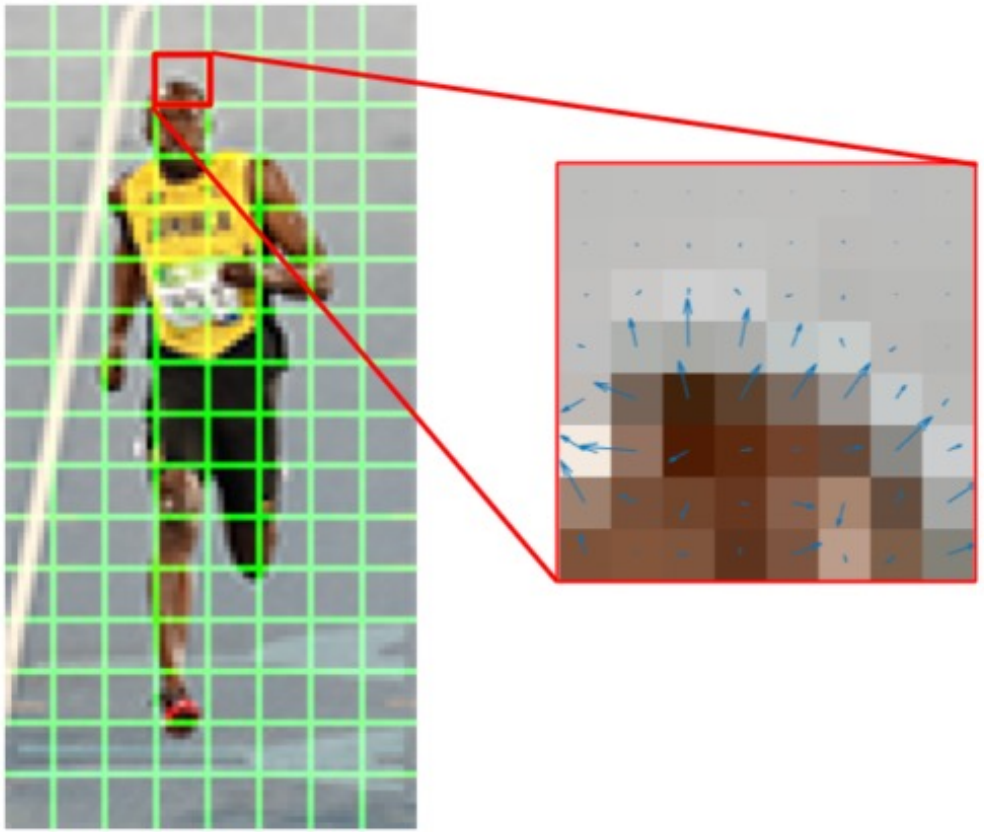
## ❖ HoG

**Objective:** Convert raw image to HoG representation, then do flattening on HoG.

# Traffic Sign Classification

## ❖ HoG



| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 2 | 3 | 4 | 4 | 3 | 4 | 2 | 2 |
| 5 | 11 | 17 | 13 | 7 | 9 | 3 | 4 |
| 11 | 21 | 23 | 27 | 22 | 17 | 4 | 6 |
| 23 | 99 | 165 | 135 | 85 | 32 | 26 | 2 |
| 91 | 155 | 133 | 136 | 144 | 152 | 57 | 28 |
| 98 | 196 | 76 | 38 | 26 | 60 | 170 | 51 |
| 165 | 60 | 60 | 27 | 77 | 85 | 43 | 136 |
| 71 | 13 | 34 | 23 | 108 | 27 | 48 | 110 |

Gradient Magnitude

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 80 | 36 | 5 | 10 | 0 | 64 | 90 | 73 |
| 37 | 9 | 9 | 179 | 78 | 27 | 169 | 166 |
| 87 | 136 | 173 | 39 | 102 | 163 | 152 | 176 |
| 76 | 13 | 1 | 168 | 159 | 22 | 125 | 143 |
| 120 | 70 | 14 | 150 | 145 | 144 | 145 | 143 |
| 58 | 86 | 119 | 98 | 100 | 101 | 133 | 113 |
| 30 | 65 | 157 | 75 | 78 | 165 | 145 | 124 |
| 11 | 170 | 91 | 4 | 110 | 17 | 133 | 110 |

Gradient Direction

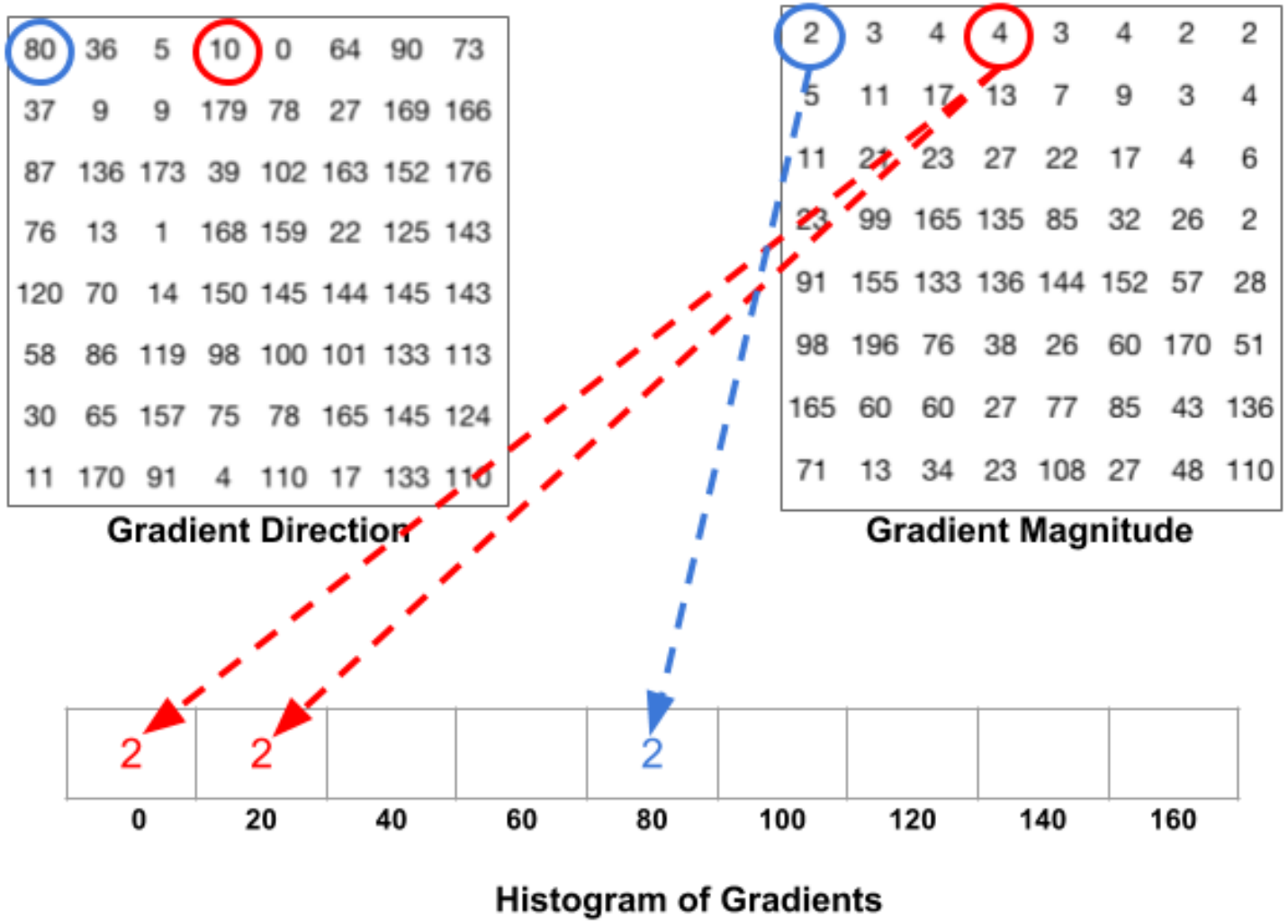
1. Split the images into cells. Each cell contains 8x8 pixels.
2. For each cell, compute the Gradient Direction and Gradient Magnitude.

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} * \mathbf{I} \qquad G = \sqrt{G_x^2 + G_y^2}$$

$$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} * \mathbf{I} \qquad \theta = \arctan\left(\frac{G_y}{G_x}\right)$$

# Traffic Sign Classification

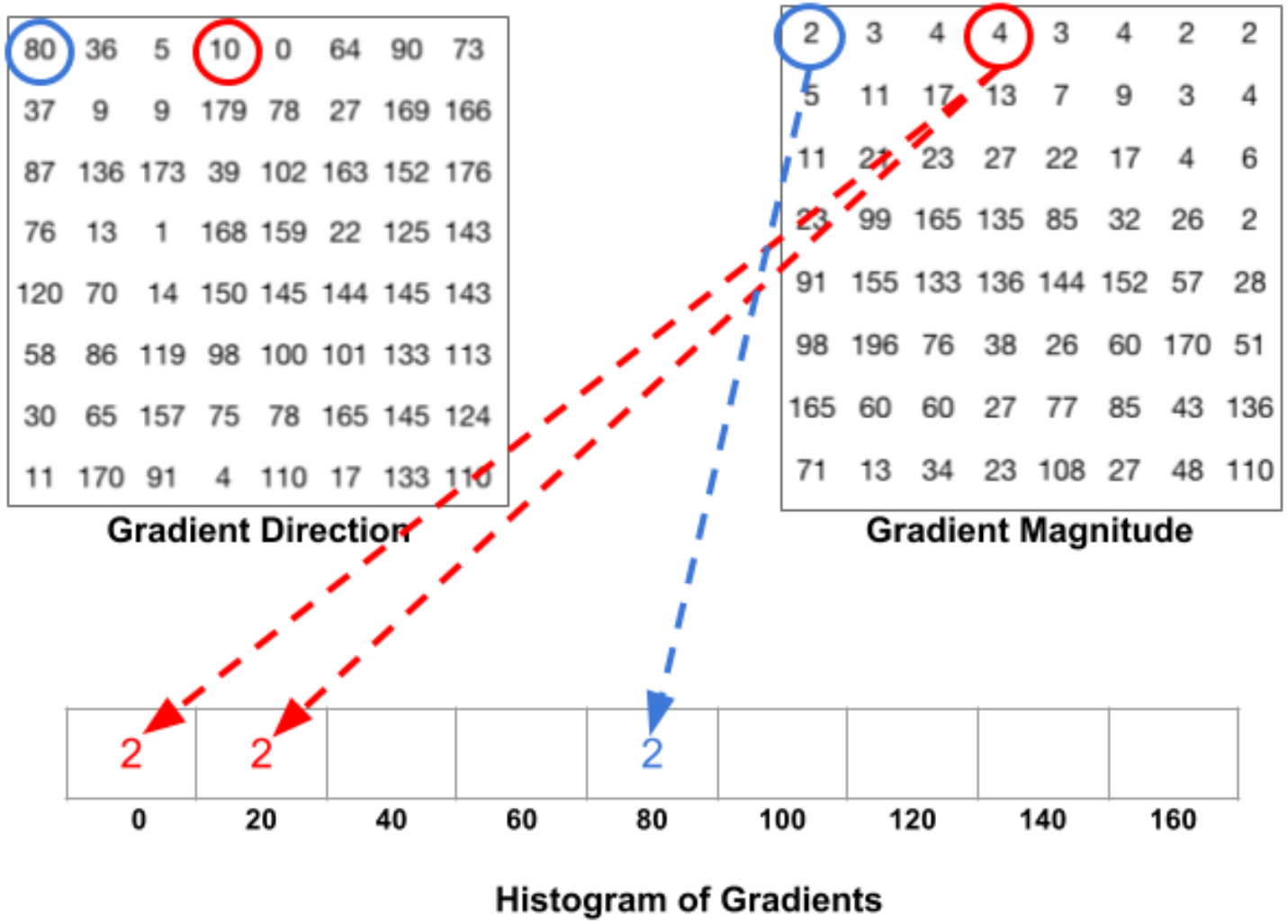
## ❖ HoG: Compute Histogram of Gradients




We create a vector of 9 elements (9 bins), representing the value range of Gradient Direction [0, 180]. If direction is in a bin, assign magnitude to the bin.

# Traffic Sign Classification

## ❖ HoG: Compute Histogram of Gradients



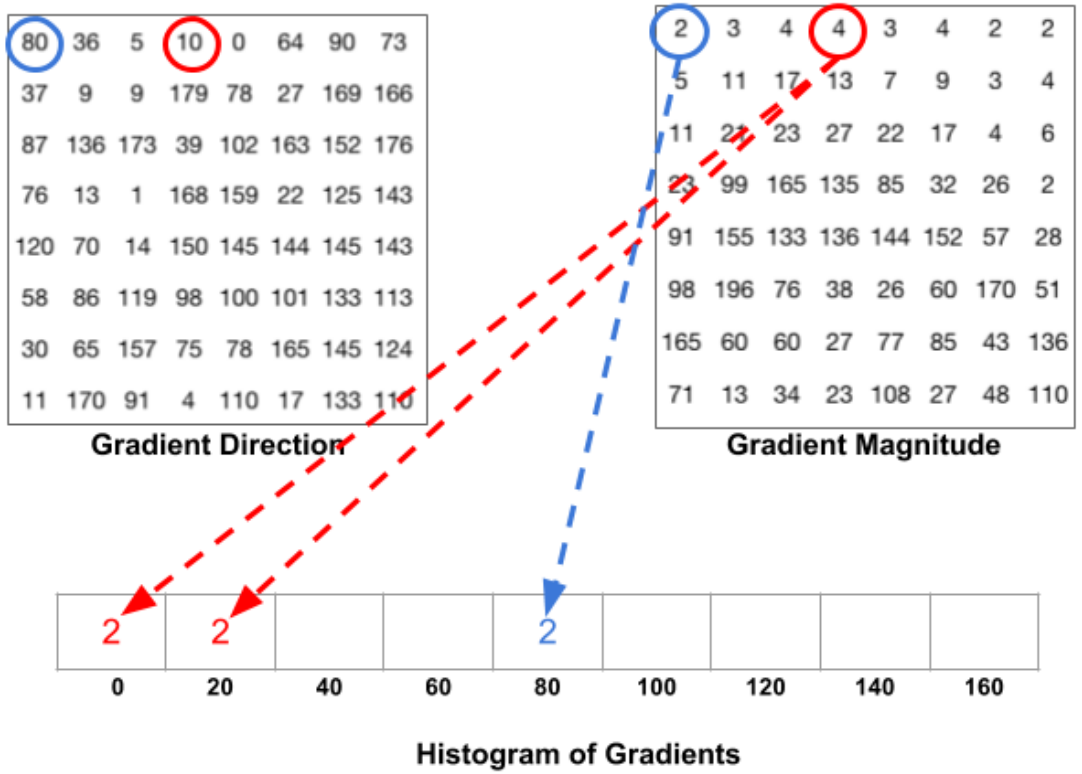

If not, assign magnitude to two adjacent bins $x \in [x_0, x_1]$ using the following formulas:

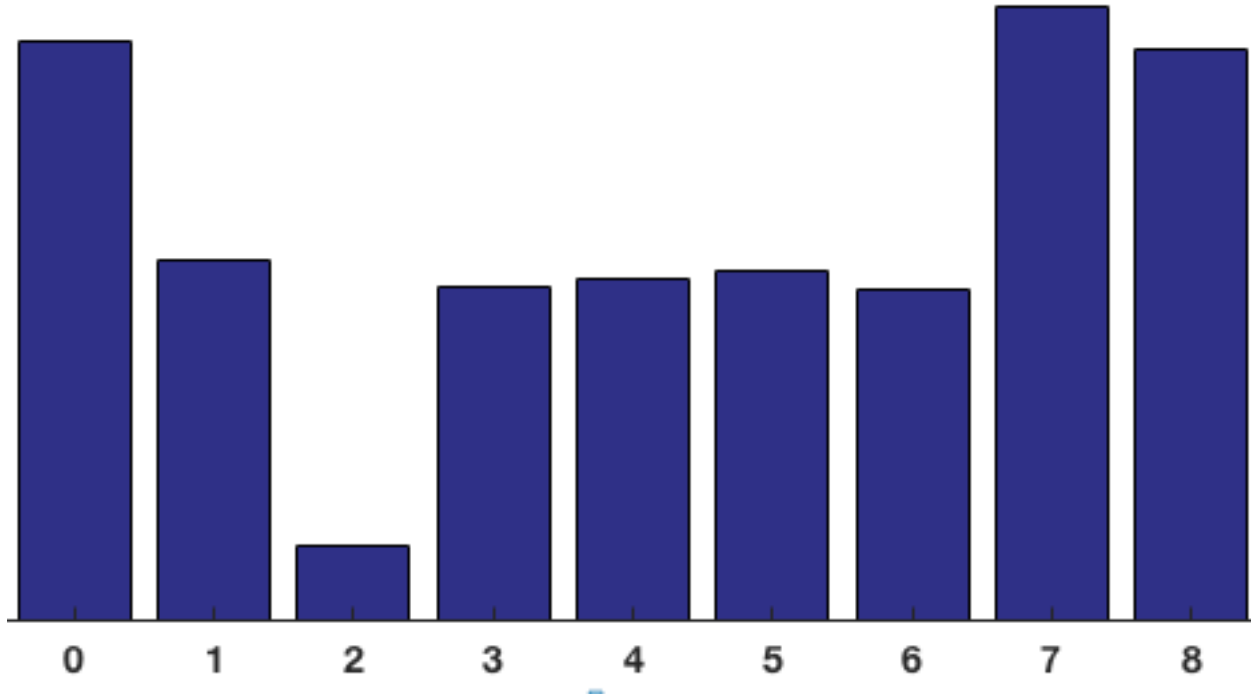
$$x_{l-1} = \frac{(x_1 - x)}{x_1 - x_0} * magnitude$$

$$x_l = \frac{(x - x_0)}{x_1 - x_0} * magnitude$$

# Traffic Sign Classification

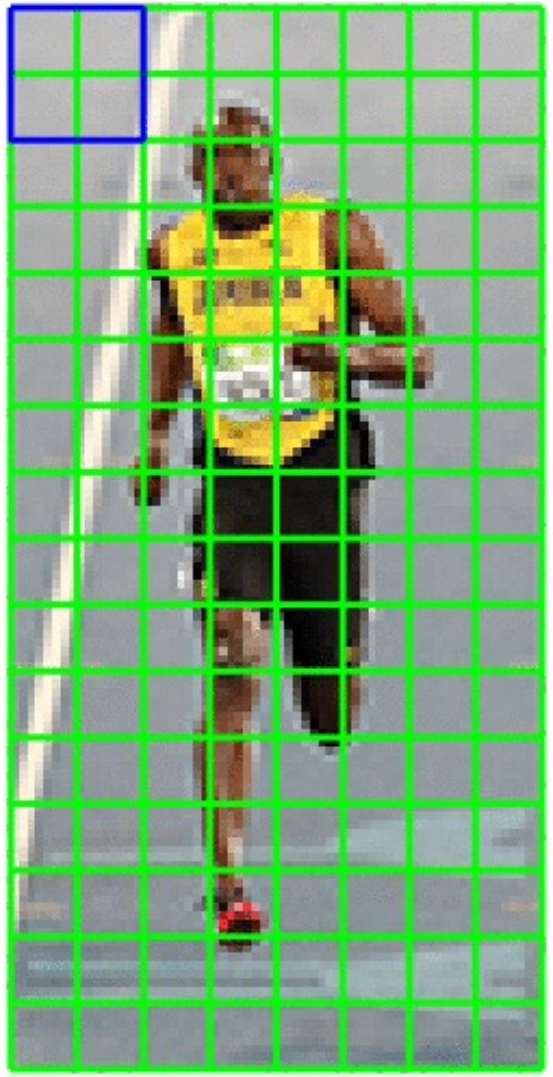
## ❖ HoG: Compute Histogram of Gradients



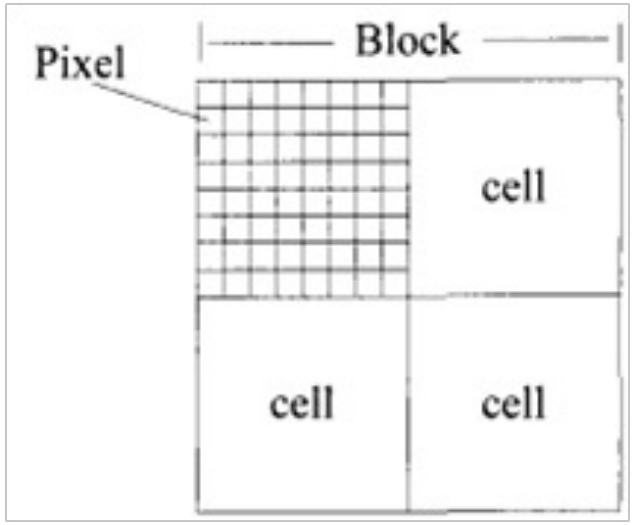



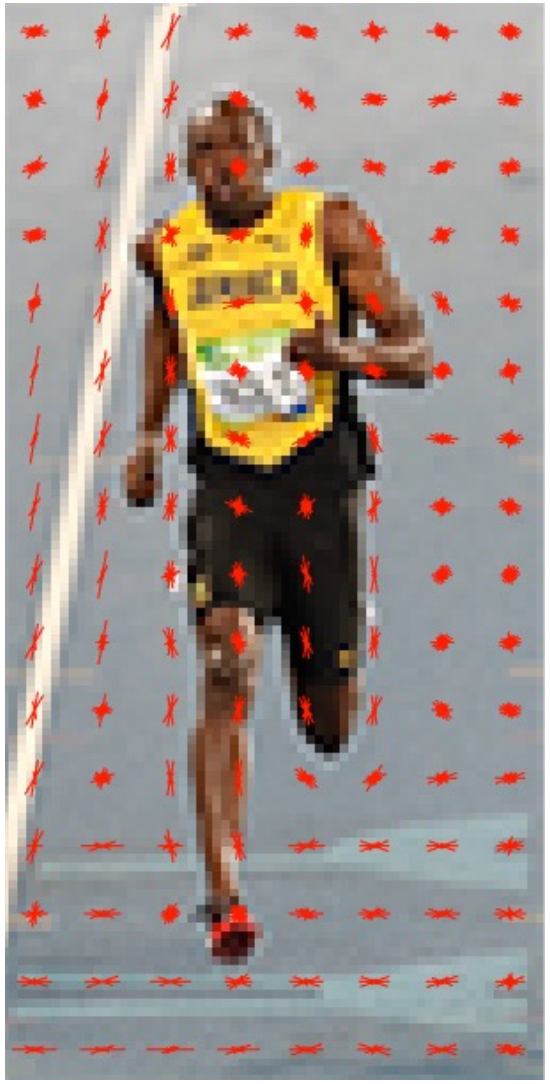
# Traffic Sign Classification

## ❖ HoG: Normalize block of cells



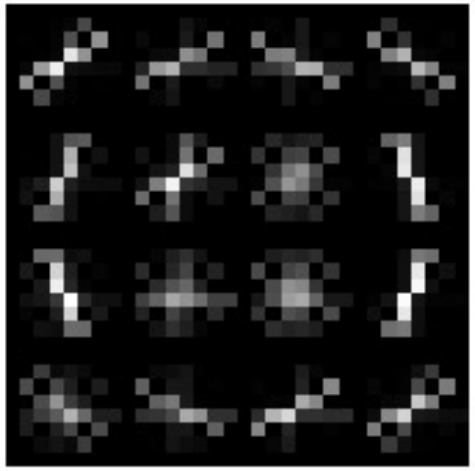
Do normalization on each 2x2 cells block = normalization on 1x36 vector.



# Traffic Sign Classification

## ❖ Step 2: Preprocess image

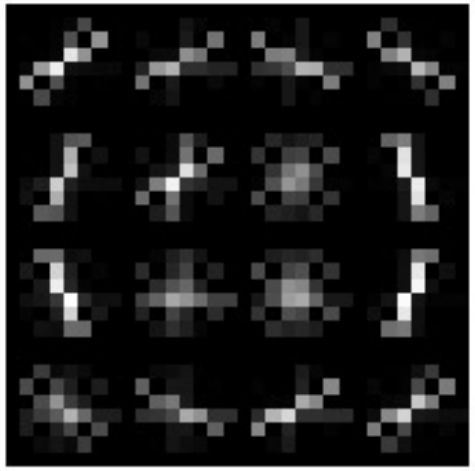
Convert all raw images to HOG representation

```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.

# Traffic Sign Classification

## ❖ Step 2: Preprocess image

Convert all raw images to HOG representation

```
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)
7 print('X shape:')
8 print(img_features.shape)
```

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

# Traffic Sign Classification

## ❖ Step 3: Encode label

| Label Encoder | |
|---|---|
| ‘crosswalk’ | 0 |
| ‘speedlimit’ | 1 |
| ‘stop’ | 2 |

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

```
1 label_encoder.classes_
```

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

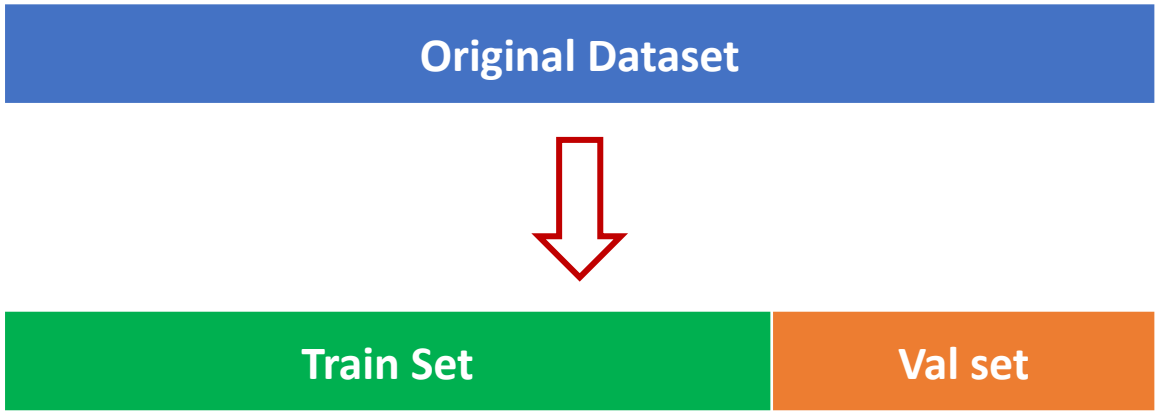
```
1 encoded_labels
```

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

# Traffic Sign Classification

## ❖ Step 4: Create train, val set

```
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
)
```



# Traffic Sign Classification

## ❖ Step 5: Normalization

Using sklearn.preprocessing.StandardScaler() to scale all values in dataset.

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

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

```
1 X_train
```

```
array([[ 0.5161459 ,  1.4854372 , -0.5399236 , ...,  0.03029941,
        -0.29923776, -0.12044104],
       [ 0.86671036,  0.45618102, -0.6950155 , ..., -0.2180403 ,
        -0.1757149 ,  0.2960105 ],
       [-0.78160155, -0.0928254 ,  0.02434517, ..., -0.6275371 ,
        -0.52018636, -0.6298612 ],
       ...,
       [ 0.24302678, -0.7746629 , -1.5431894 , ..., -0.6275371 ,
        -0.4982059 ,  0.39656585],
       [-0.8555847 , -0.21795394, -0.52630377, ..., -0.6275371 ,
        -0.4973315 , -0.5843945 ],
       [-0.74369645, -0.38316953,  0.10891949, ..., -0.503356  ,
        -0.5255429 , -0.49774203]], dtype=float32)
```

# Traffic Sign Classification

## ❖ Step 6, 7: Train and evaluate SVM model

```
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
```

# Traffic Sign Classification

## ❖ Step 6, 7: Train and evaluate SVM model

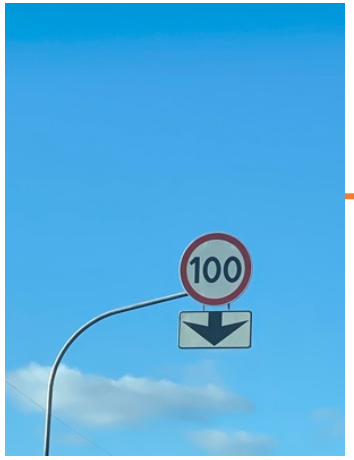

```
1 input_img = img_lst[1]
2 normalized_img = preprocess_img(input_img)
3 y_pred = clf.predict([normalized_img])[0]
4 print(f'Normal prediction: {y_pred}')
5 y_pred_prob = clf.predict_proba([normalized_img])
6 prediction = np.argmax(y_pred_prob)
7 y_pred_prob = [f'{p:.10f}' for p in y_pred_prob[0]]
8 print(f'Probability of each class: {y_pred_prob}')
9 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
```

# Traffic Sign Localization

## ❖ Introduction

**Input Image**




Pyramid Images → Sliding Window (Multiple window sizes) → Non-maximum Suppression → List of bounding boxes

List of bounding boxes → SVM model → List of bounding boxes with classname and confidence socre

# Traffic Sign Localization

## ❖ Sliding Window

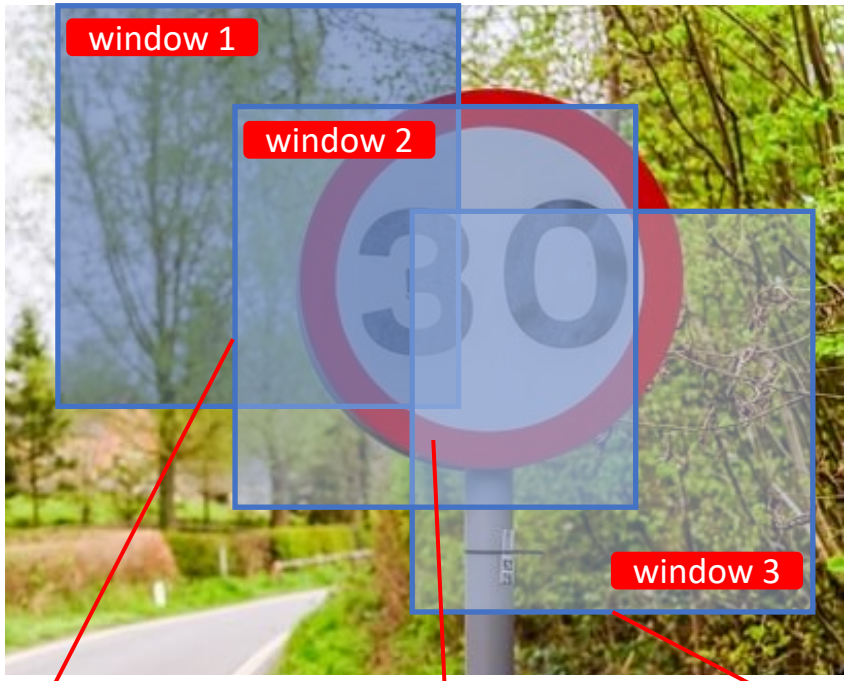
**Example:** Using sliding window to find a cup within an image.

**Sliding Window:** A technique used in computer vision and signal processing where a fixed-size "window" (or a sub-rectangle) is moved across an image (or a signal) to analyze a local region at a time.

# Traffic Sign Localization

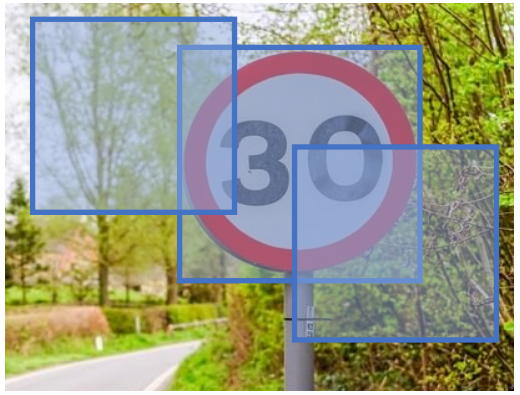
## ❖ Sliding Window

Target of finding

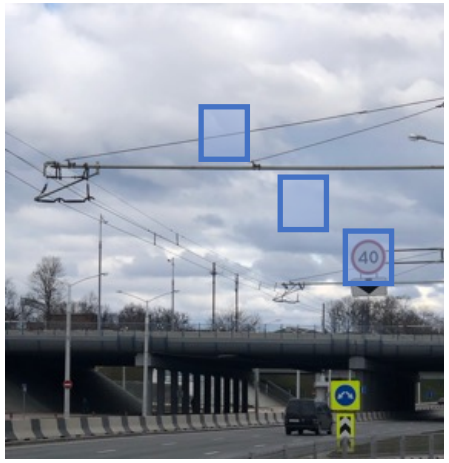



# Traffic Sign Localization

## ❖ Sliding Window



Window size 1



Window size 2

Difference window size

Detect objects of different sizes

Stride = 10

Stride = 20

Small stride archive with high accuracy

A bigger stride archive with high speed

# Traffic Sign Localization

## ❖ Sliding 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
```

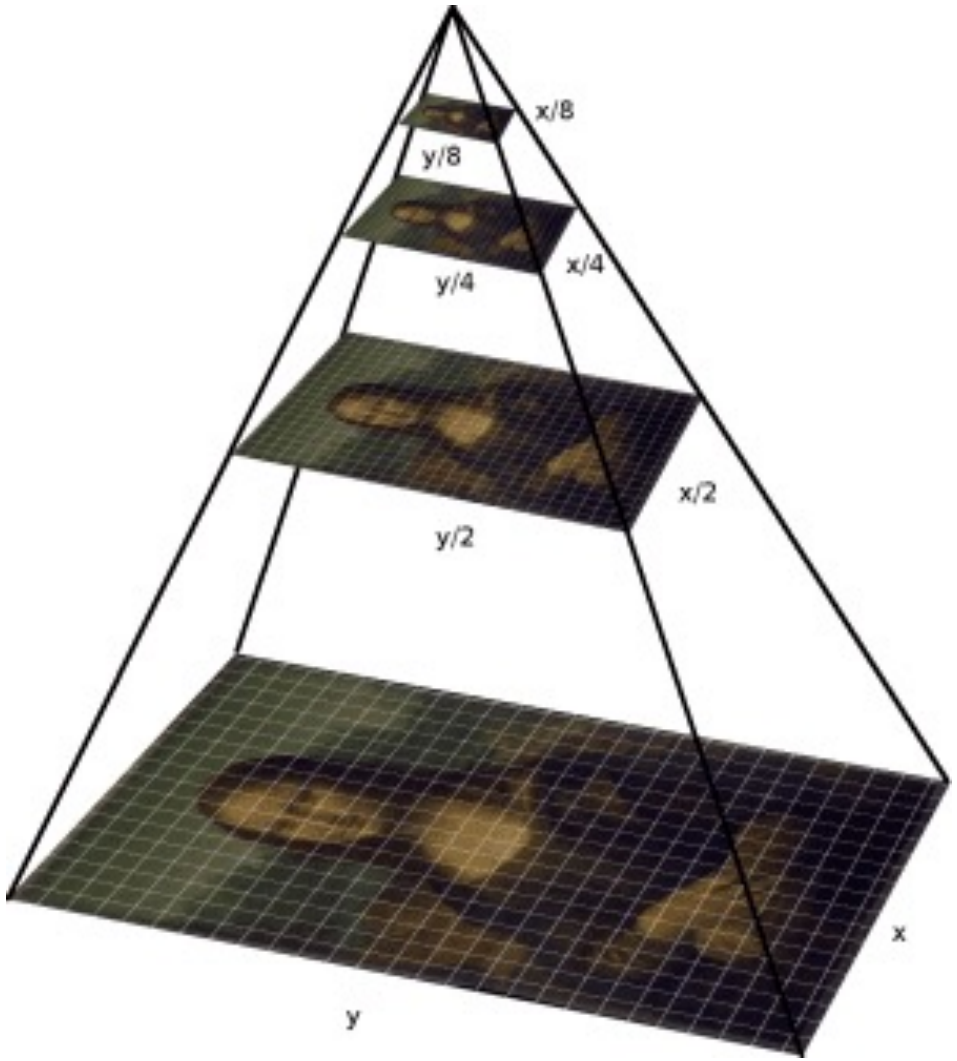
# Traffic Sign Localization

## ❖ Small objects problem


In many situations, **the objects of interest are often presented very small in the image**. Directly apply a fixed-size window might not be effectively handle this.

# Traffic Sign Localization

## ❖ Pyramid Images




**Pyramid Image:** A multi-scale representation of an input image where the image is processed at various scales (resolutions) to detect objects at different sizes.

# Traffic Sign Localization

## ❖ Pyramid Images

Scaled at: 1.0


Scaled at: 0.5120000000000001


Scaled at: 0.32768000000000014

# Traffic Sign Localization

## ❖ Pyramid Images

Scaled at: 1.0


Scaled at: 0.5120000000000001


Apply sliding window on each scaled image to find the objects of interest.

# Traffic Sign Localization

## ❖ Pyramid Images

```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
```

# Traffic Sign Localization

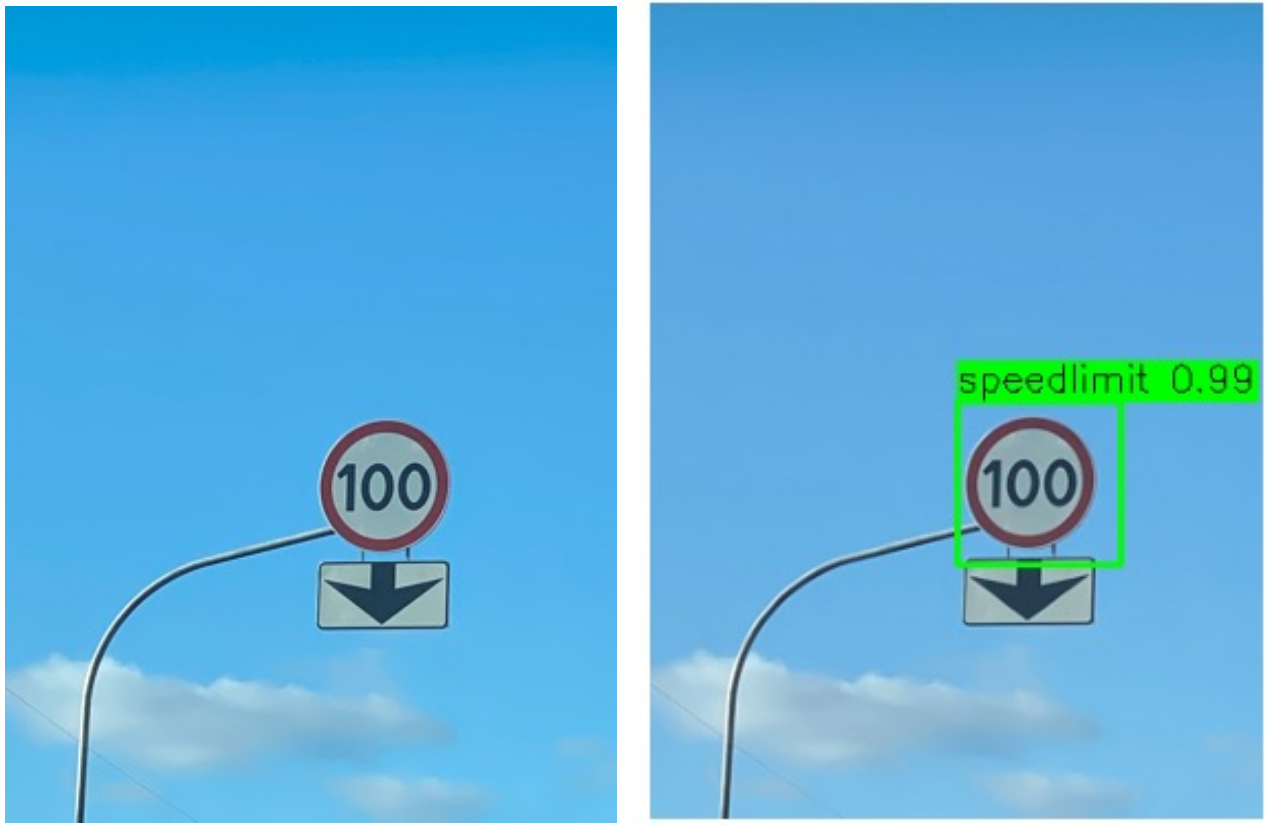
## ❖ Apply Sliding Window on Pyramid Images


Source: https://pyimagesearch.com/wp-content/uploads/2015/03/sliding-window-animated-adrian.gif

# Traffic Sign Localization

## ❖ Visualize bounding box

With this list of bounding box, we need to put them on the image for a better understanding of the final results.



# Traffic Sign Localization

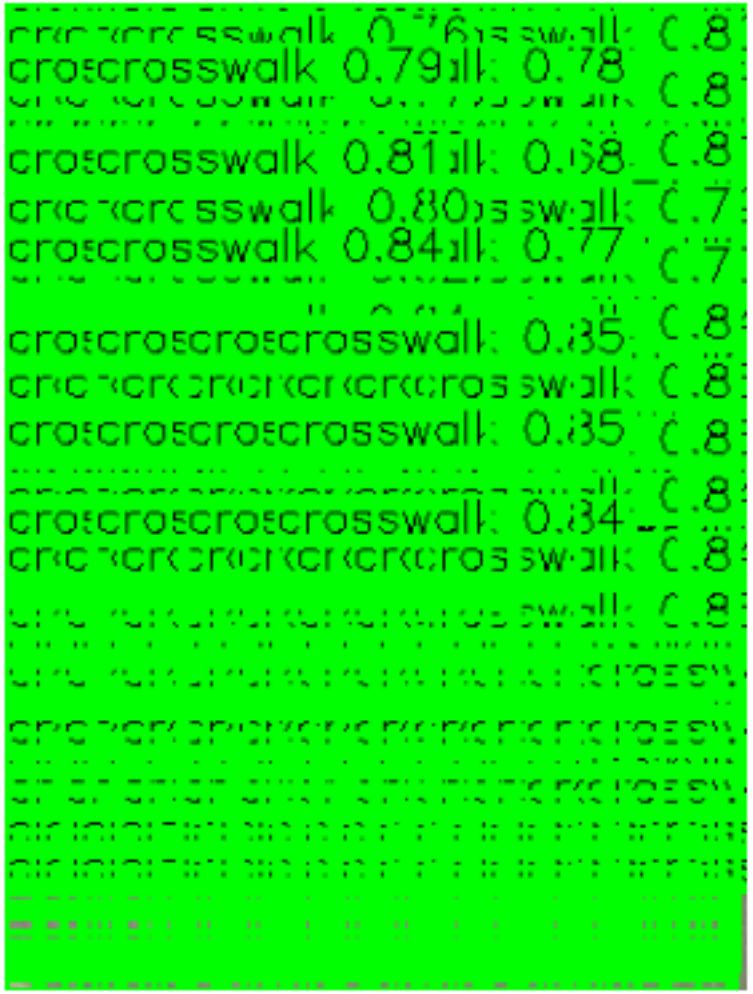
## ❖ Visualize bounding box

```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()
```

# Traffic Sign Localization

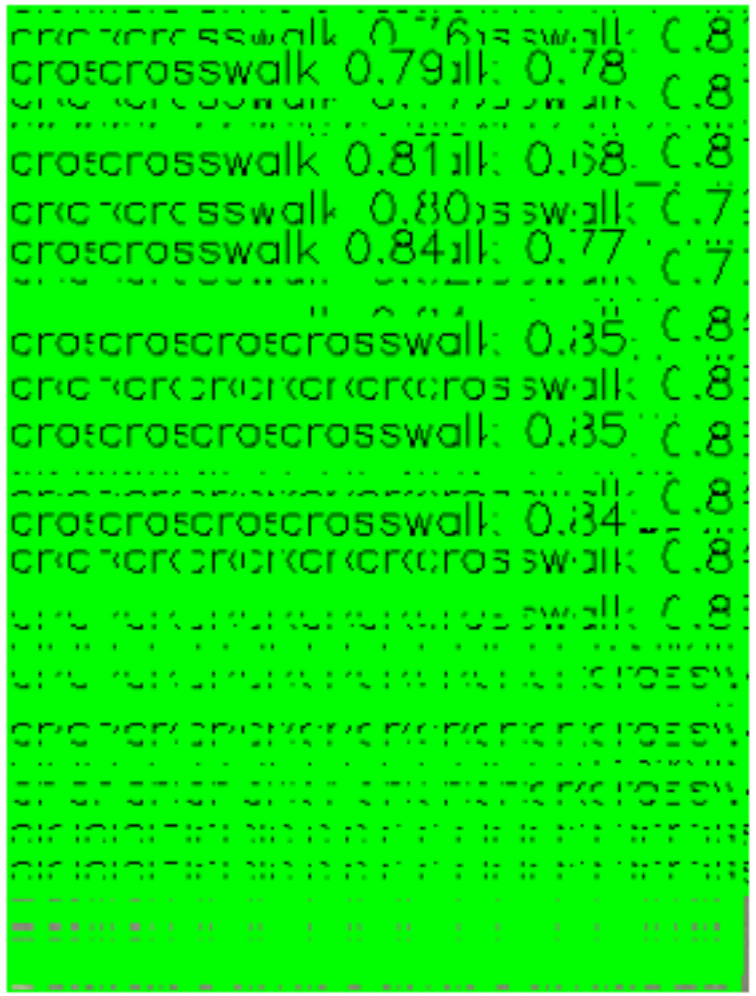
## ❖ Prediction

```
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]
        )
```



# Traffic Sign Localization

## ❖ Prediction



We want to select certain bounding boxes (e.g: bounding boxes which have a higher or equal than the minimum confidence score => Threshold the confidence score.

```
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]
        )
```

# Traffic Sign Localization

## ❖ Prediction




Results with a confidence score threshold = 0.95

# Traffic Sign Localization

## ❖ Prediction


# Traffic Sign Localization

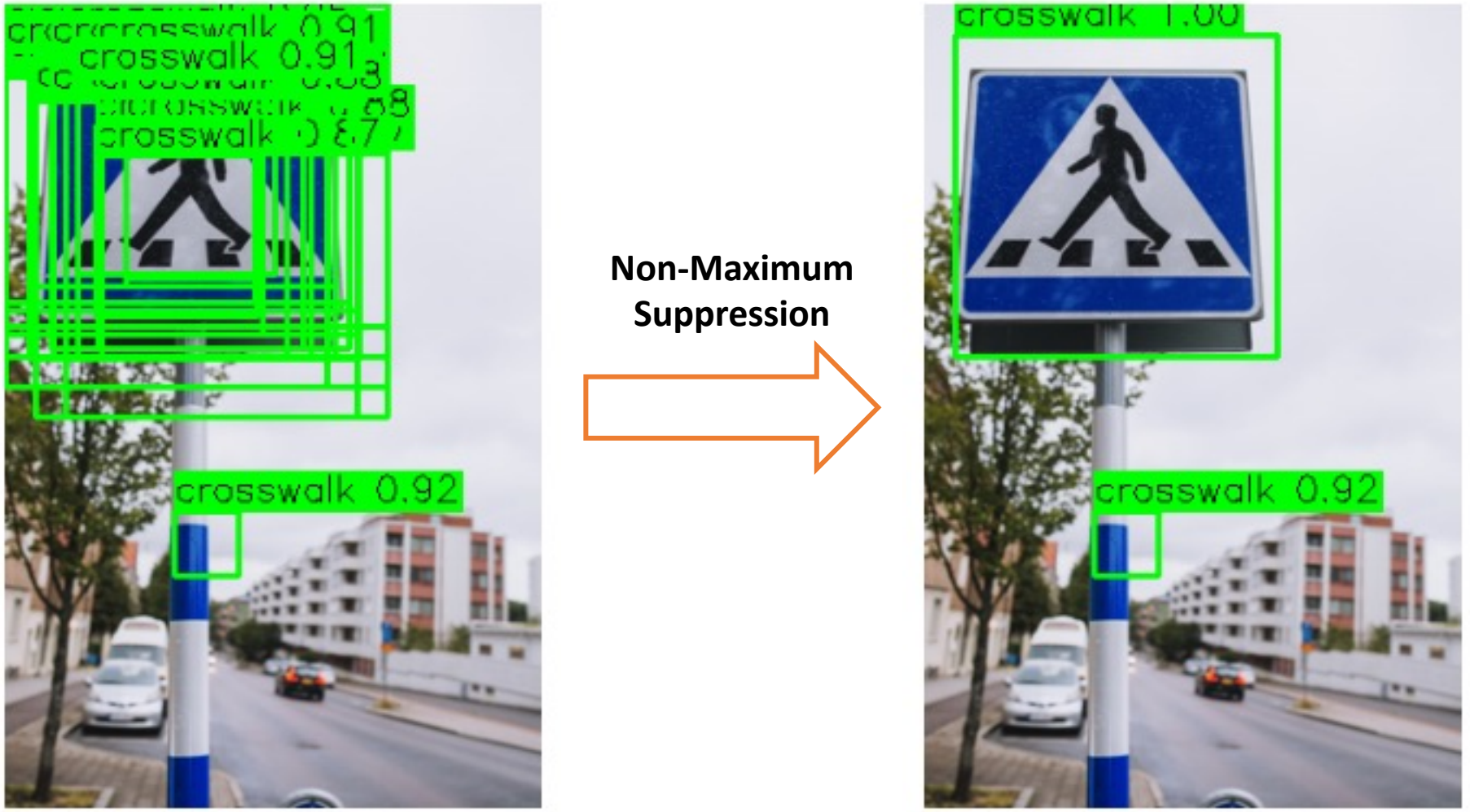
## ❖ Overlapped bounding boxes problem



Usually, the raw outcome of object detection may have a lot of bounding boxes overlapping each other, with almost the same confidence score. We need to compress into the best one.

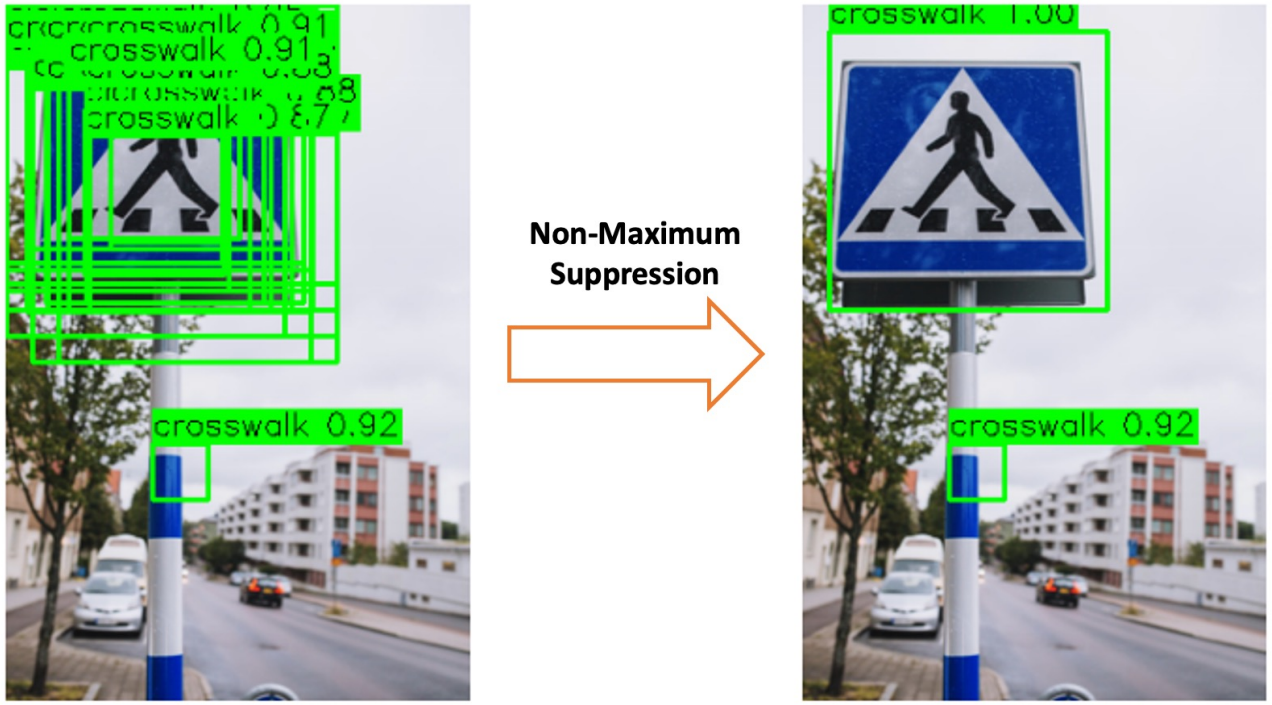
# Traffic Sign Localization

## ❖ Non-maximum Suppression



# Traffic Sign Localization

## ❖ Non-maximum Suppression




**Non-maximum suppression (NMS):** A technique used in object detection tasks to prune multiple bounding boxes that are overlapping and refer to the same object.

# Traffic Sign Localization

## ❖ Non-maximum suppression

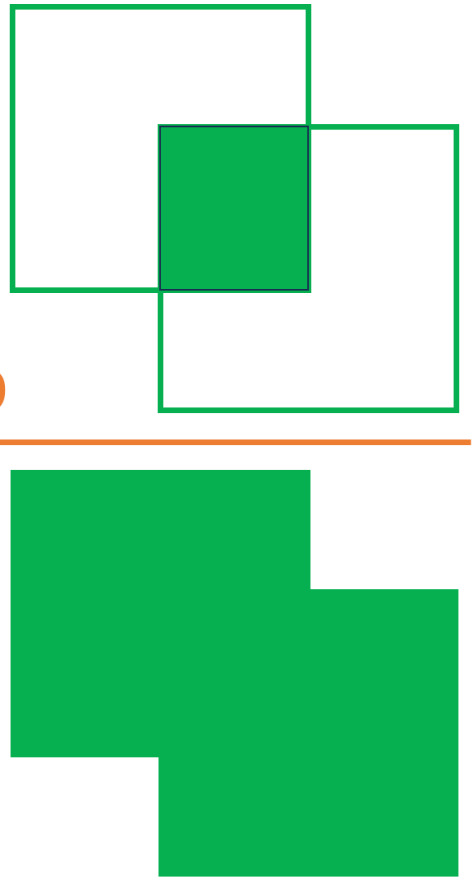

**Non-maximum suppression:**

1. Select the bounding box with highest confidence score (A).
2. Compute IoU between (A) with other images.
3. Only retain images that have IoU less than a threshold (iou_threshold). Otherwise, remove all.

# Traffic Sign Localization

## ❖ Intersection over Union

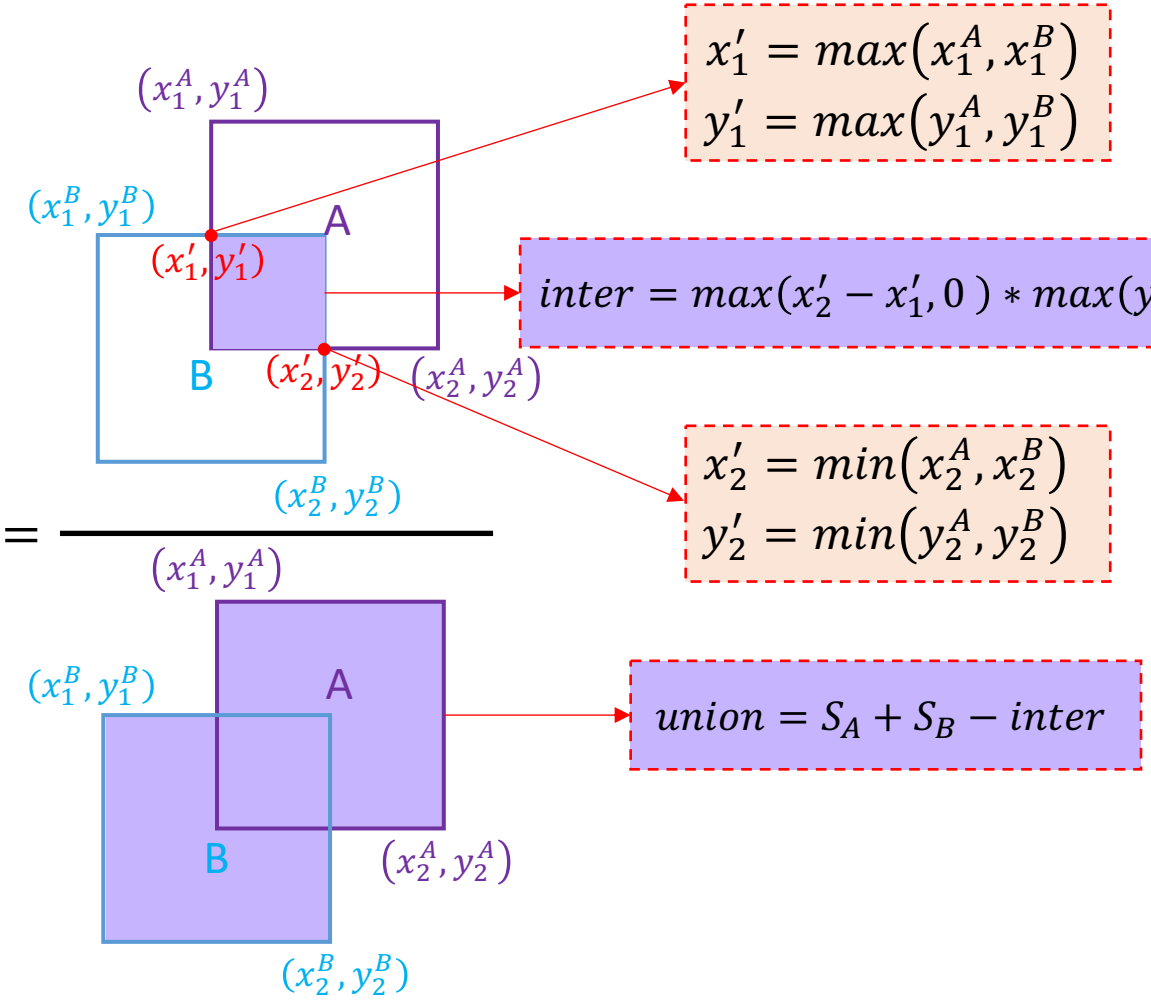
$$IoU = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$



**Intersection over Union (IoU):** A metric used to measure the overlap between two bounding boxes.

# Traffic Sign Localization

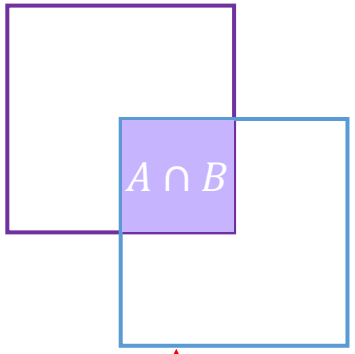
## ❖ Intersection over Union



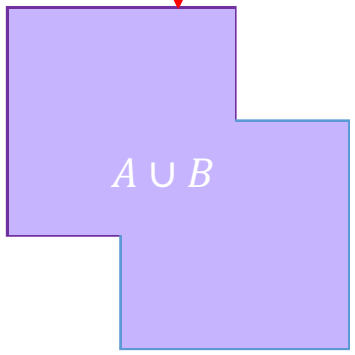

```python
def iou_bbox(box1, box2):
    """
    - Input:
        - box1 (x1, y1, x2, y2)
        - box2 (x1, y1, x2, y2)
    - Output:
        - iou of box1 and box2

    """
    b1_x1, b1_y1, b1_x2, b1_y2 = box1[0], box1[1], box1[2], box1[3]
    b2_x1, b2_y1, b2_x2, b2_y2 = box2[0], box2[1], box2[2], box2[3]

    # Step 1: Compute inter area
    x1 = max(b1_x1, b2_x1)
    y1 = max(b1_y1, b2_y1)
    x2 = min(b1_x2, b2_x2)
    y2 = min(b1_y2, b2_y2)

    inter = max((x2 - x1), 0) * max((y2 - y1), 0)

    # Step 2: Compute union area
    box1Area = abs((b1_x2 - b1_x1) * (b1_y2 - b1_y1))
    box2Area = abs((b2_x2 - b2_x1) * (b2_y2 - b2_y1))

    union = float(box1Area + box2Area - inter)

    iou = inter / union
    return iou
```

# Traffic Sign Localization

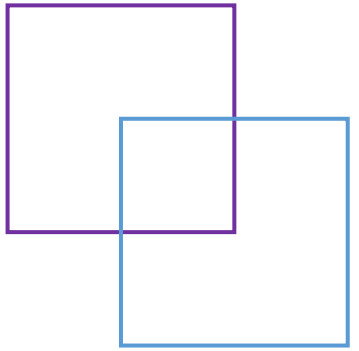
## ❖ Intersection over Union

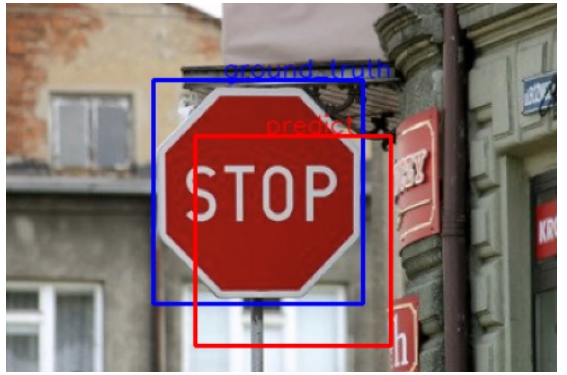



$$IoU = \frac{Area\ of\ Overlap}{Area\ of\ Union}$$

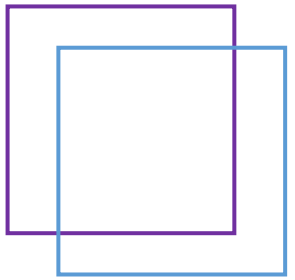


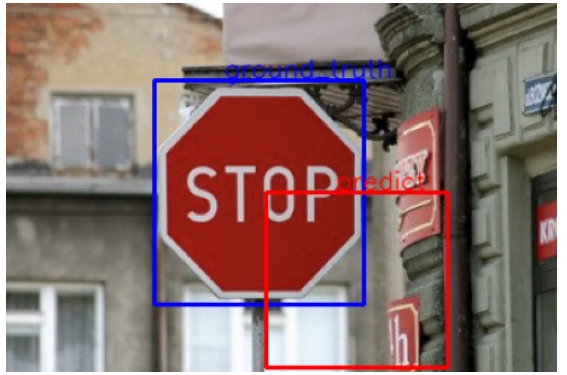

Poor



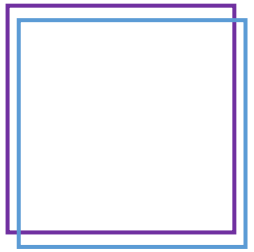



Good






Excellent



# Traffic Sign Localization

## ❖ Non-maximum suppression

**Non-maximum suppression:**

1. Select the bounding box with highest confidence score (A).
2. Compute IoU between (A) with other images.
3. Only retain images that have IoU less than a threshold (iou_threshold). Otherwise, remove all.

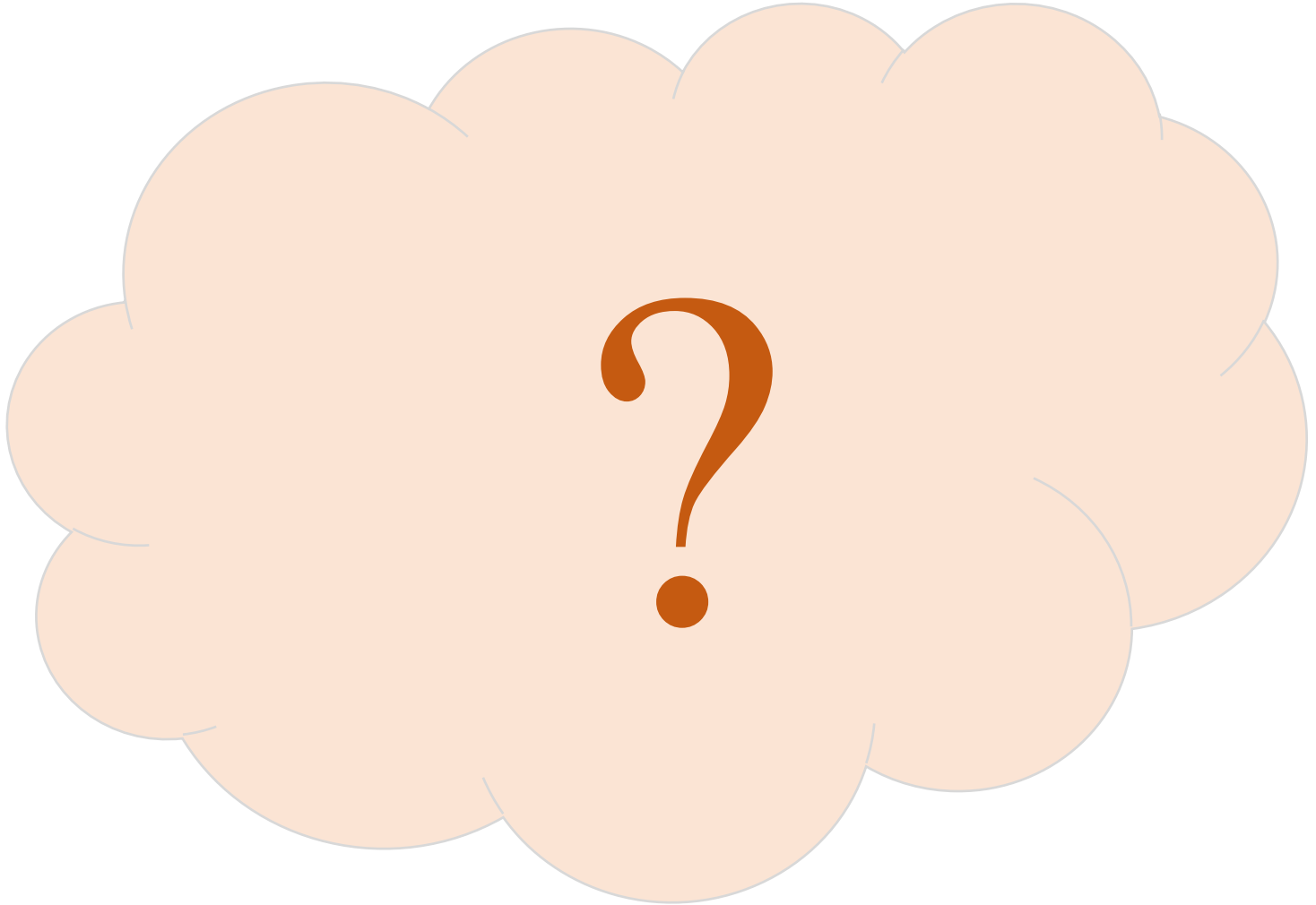
```python
def nms(bboxes, iou_threshold):
    if not bboxes:
        return []
    scores = np.array([bbox[5] for bbox in bboxes])
    sorted_indices = np.argsort(scores)[::-1]

    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]
```

# Traffic Sign Localization

## ❖ Final Results



# Question




Thank you!