Bounding Box from Scratch using PyTorch

Vashie Garan

March 2021

1 Introduction

This article talks about the case when there is only one object of interest present in an image. The focus here is more on how to read an image and its bounding box, resize and perform augmentations correctly, rather than on the model itself. The goal is to have a good grasp of the fundamental ideas behind object detection, which you can extend to get a better understanding of the more complex techniques. Since we are focusing on the Data rather than the machine learning model, there will be less explanation on the training.

2 Problem Statement

Given an image consisting of a road sign, predict a bounding box around the road sign and identify the type of road sign.

There are four distinct classes these signs could belong to:

- Traffic Light
- Stop
- Speed Limit
- Crosswalk

3 Dataset

It consists of 877 images. It's a pretty imbalanced dataset, with most images belonging to the speed limit class, but since we're more focused on the bounding box prediction, we can ignore the imbalance.

3.1 Loading the Data

The annotations for each image were stored in separate XML files. I followed the following steps to create the training dataframe:

Figure 1: The information in .xml files

Figure 1 shows information of bounding box of the corresponding image to the file name.

By using that information:

- Create a dictionary consisting of filepath, width , height , the bounding box coordinates (xmin , xmax , ymin , ymax) and class for each image and append the dictionary to a list.
- Create a pandas dataframe using the list of dictionaries of image stats.

```
def filelist(root, file_type):
    """Returns a fully-qualified list of filenames under root directory"""
    return [os.path.join(directory_path, f) for directory_path, directory_name,
            files in os.walk(root) for f in files if f.endswith(file_type)]

def generate_train_df (anno_path):
    annotations = filelist(anno_path, '.xml')
    anno_list = []
    for anno_path in annotations:
        root = ET.parse(anno_path).getroot()
        anno = {}
        anno['filename'] = Path(str(images_path) + '/'+ root.find("./filename").text)
        anno['width'] = root.find("./size/width").text
        anno['height'] = root.find("./size/height").text
        anno['class'] = root.find("./object/name").text
        anno['xmin'] = int(root.find("./object/bndbox/xmin").text)
        anno['ymin'] = int(root.find("./object/bndbox/ymin").text)
        anno['xmax'] = int(root.find("./object/bndbox/xmax").text)
        anno['ymax'] = int(root.find("./object/bndbox/ymax").text)
        anno_list.append(anno)
    return pd.DataFrame(anno_list)
```

Figure 2: The information in .xml files

- Create Label encode the class column

```
class_dict = {'speedlimit': 0, 'stop': 1, 'crosswalk': 2, 'trafficlight': 3}
df_train['class'] = df_train['class'].apply(lambda x:  class_dict[x])
```

Figure 3:

3.2 Resizing Images and Bounding Boxes

Since training a computer vision model needs images to be of the same size, we need to resize our images and their corresponding bounding boxes. Resizing an image is straightforward but resizing the bounding box is a little tricky because each box is relative to an image and its dimensions.

Here's how resizing a bounding box works:

- Convert the bounding box into an image (called mask) of the same size as the image it corresponds to. This mask would just have 0 for background and 1 for the area covered by the bounding box.

https://www.overleaf.com/project/605f119758576cde09ea83d1

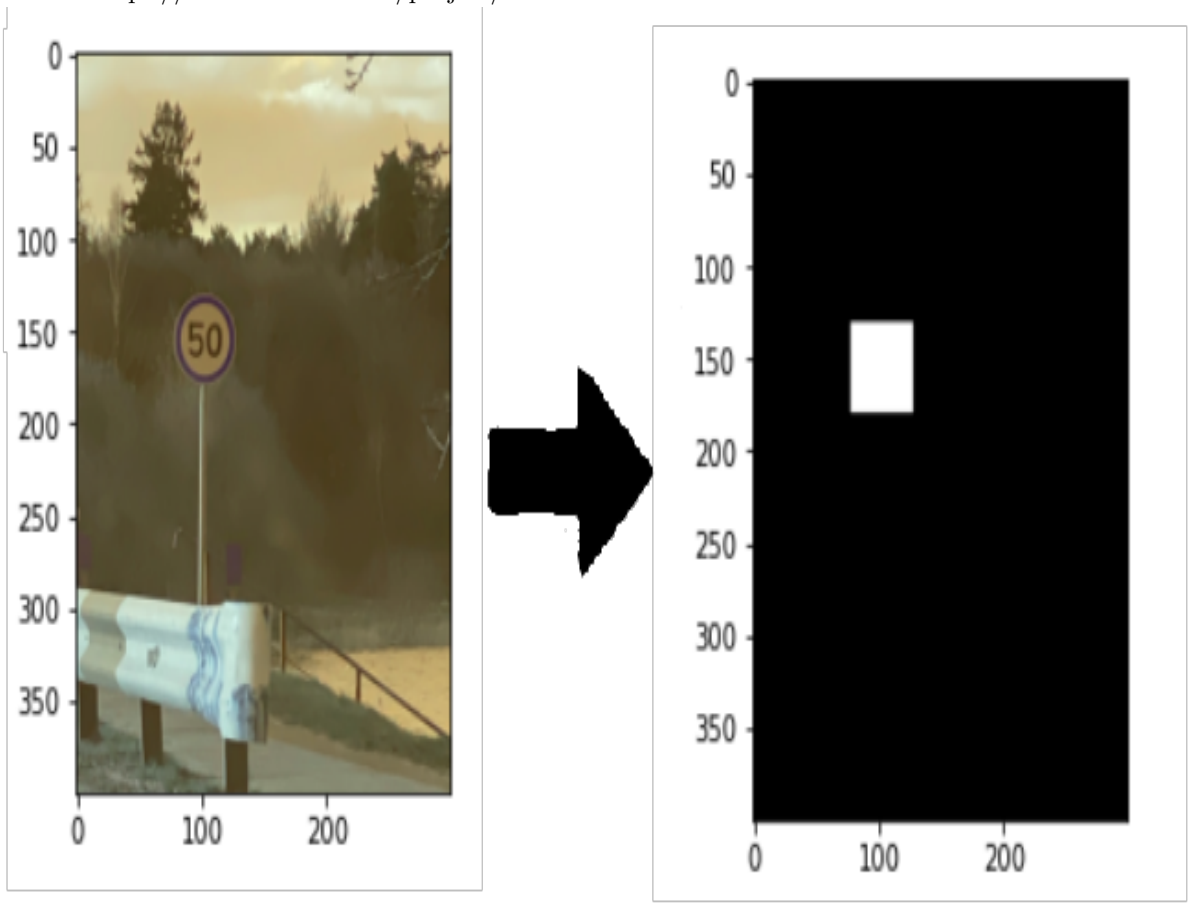

Figure 4: The area of bounding box is filled with 1(white colour)

- Resize the mask to the required dimensions.
- Extract bounding box coordinates from the resized mask.

```python
def create_mask(bb, x):
    """Creates a mask for the bounding box of same shape as image"""
    rows,cols,*_ = x.shape
    Y = np.zeros((rows, cols))
    bb = bb.astype(np.int)
    Y[bb[0]:bb[2], bb[1]:bb[3]] = 1.
    return Y

def mask_to_bb(Y):
    """Convert mask Y to a bounding box, assumes 0 as background nonzero object"""
    cols, rows = np.nonzero(Y)
    if len(cols)==0:
        return np.zeros(4, dtype=np.float32)
    top_row = np.min(rows)
    left_col = np.min(cols)
    bottom_row = np.max(rows)
    right_col = np.max(cols)
    return np.array([left_col, top_row, right_col, bottom_row], dtype=np.float32)

def create_bb_array(x):
    """Generates bounding box array from a train_df row"""
    return np.array([x[5],x[4],x[7],x[6]])
```

```python
def resize_image_bb(read_path,write_path,bb,sz):
    """Resize an image and its bounding box and write image to new path"""
    im = read_image(read_path)
    im_resized = cv2.resize(im, (int(1.49*sz), sz))
    Y_resized = cv2.resize(create_mask(bb, im), (int(1.49*sz), sz))
    new_path = str(write_path/read_path.parts[-1])
    cv2.imwrite(new_path, cv2.cvtColor(im_resized, cv2.COLOR_RGB2BGR))
    return new_path, mask_to_bb(Y_resized)
```

Figure 5: Code

Lets go through the code. The first function is to create a black background . The 2nd function is to fill the bounding box are with white colour. The 3rd function is to extract the information of bounding box such as coordinates. The 4th function is to resize the whole image with bounding box.

subsectionData Augmentation Data Augmentation is a technique to generalize our model better by creating new training images by using different variations of the existing images. We have only 800 images in our current training set, so data augmentation is very important to ensure our model doesn't overfit. For this problem, I've used flip, rotation, center crop and random crop. I've talked about various data augmentation techniques in this article:

The only thing to remember here is ensuring that the bounding box is also transformed the same way as the image. To do this we follow the same approach

as resizing — convert bounding box to a mask, apply the same transformations to the mask as the original image, and extract the bounding box coordinates.

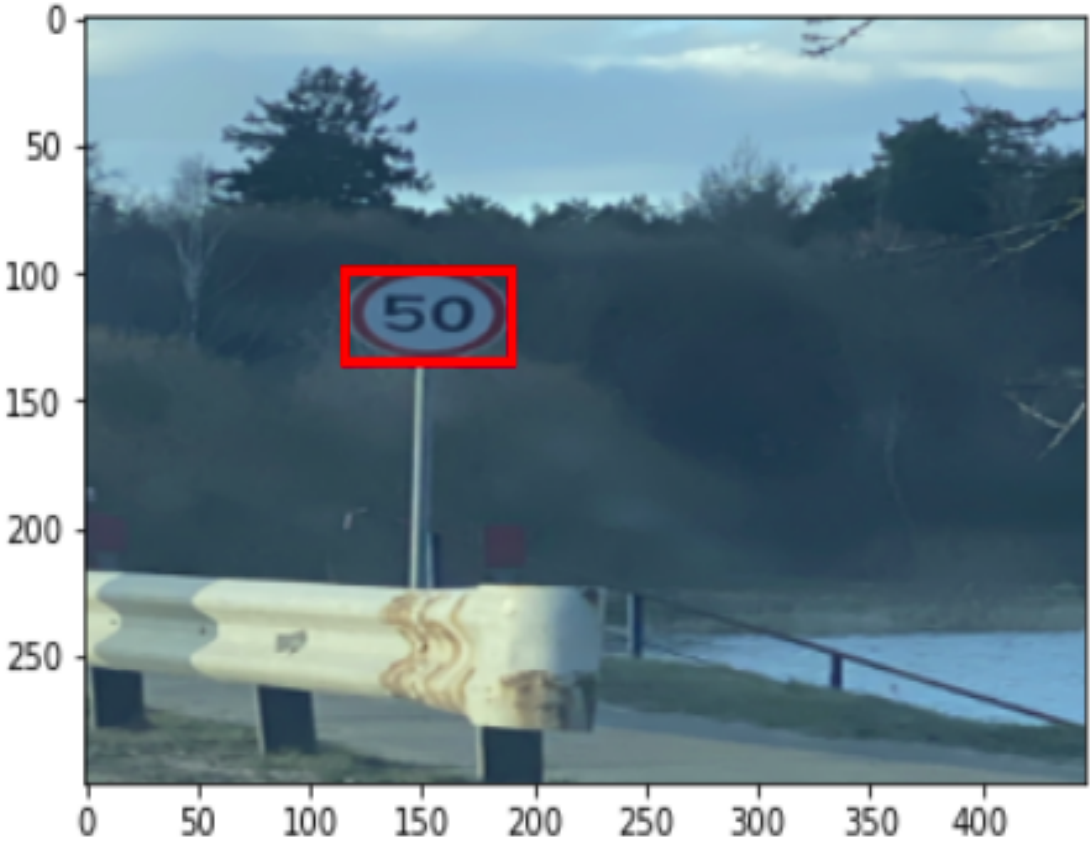

Figure 6: Before Transformation

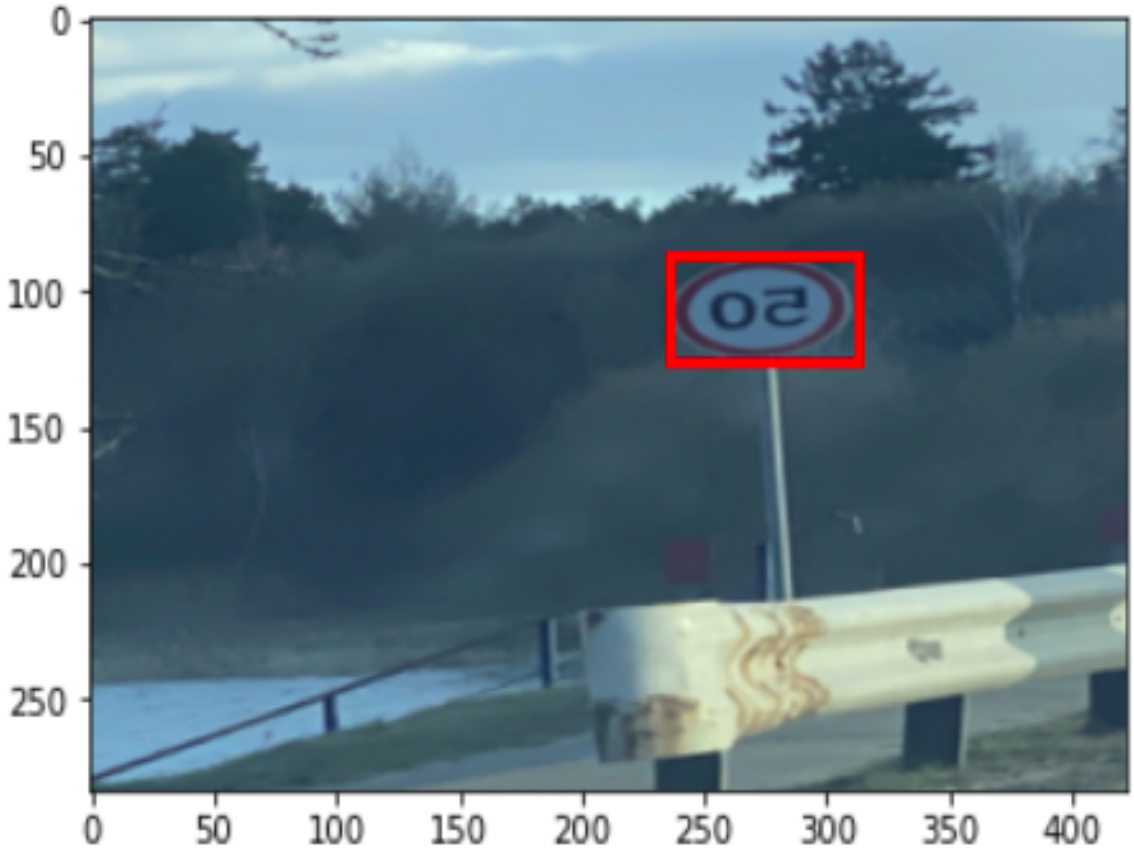

Figure 7: After Transformation

```
def create_corner_rect(bb, color='red'):
    bb = np.array(bb, dtype=np.float32)
    return plt.Rectangle((bb[1], bb[0]), bb[3]-bb[1], bb[2]-bb[0], color=color,
                         fill=False, lw=3)
```

Figure 8: Create the Bounding Box for the image

4 PyTorch Model

For the model, I've used a very simple pre-trained resNet-34 model. Since we have two tasks to accomplish here, there are two final layers — the bounding box regressor and the image classifier.

4.1 Prediction on Test Images

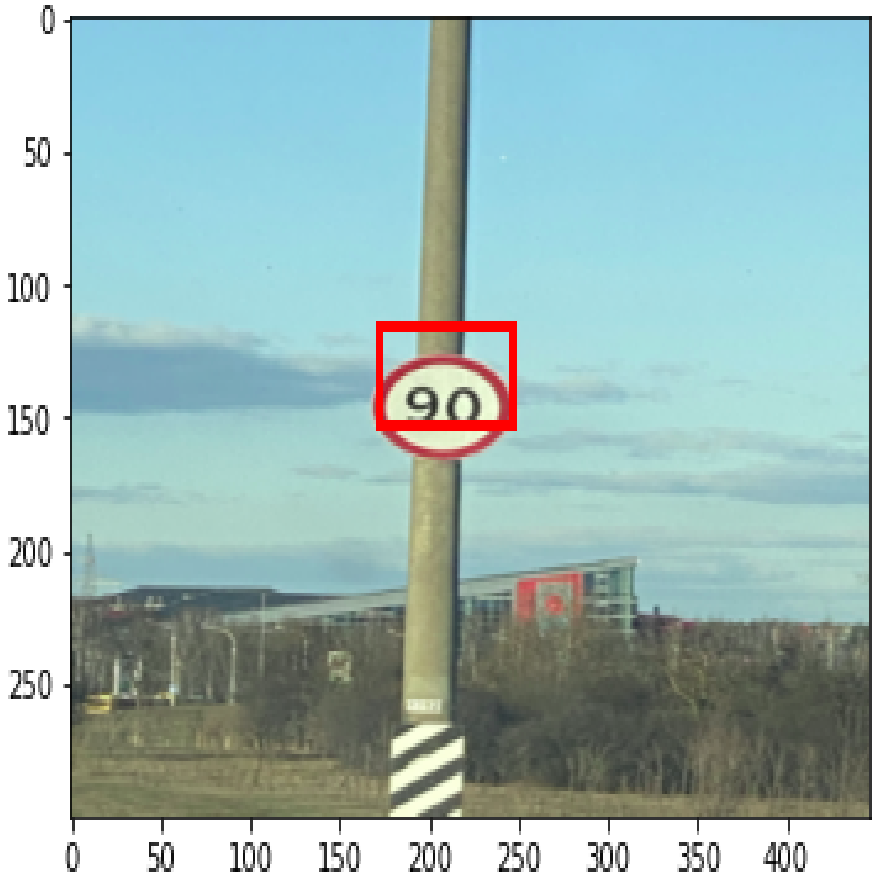

Figure 9: Predicted Bounding Box on Test Image

Since we are focusing on the Data preprocessing, I used only 10 number of epochs to train the model and hence the prediction is not accurate.