Interview Questions

Deep learning on traffic sign detection using public data.

Dharmashloka Debashis (ddebashis3)

Part1: Data Preparation

Given data:

The GTSDB [1] dataset consists of 600 training images and 300 test images. Each image is 1360x800. The annotation file (gt.txt) contains on each line, **<image_name>;<leftCol>;<topRow>;<rightCol>;<bottomRow>;<ClassID>** .

For example: “00000.ppm;774;411;815;446;11”

There are a total of 43 classes.

Note: On their website, the download page says that the gt.txt given outside all the folders contains the annotations for the test data. However, I found that it just contains the same annotations as the train data. So, I downloaded the 900 images from the full dataset (FullIJCNN2013.zip), and took the first 600 images as the train set and the last 300 images as the validation (test) set. I found that this is how they have constructed the train/test split from the full dataset.

Required data format:

For Yolo family of detectors, we need to convert them into one txt file per image. The txt file should contain on each line, the annotation for one bounding box in the following format:

<class id> <x_center> <y_center> <width> <height>, separated by spaces. Moreover, these quantities should be normalized, i.e., between 0 to 1. I wrote the following jupyter notebook to do this conversion:

“yolov5/notebooks/prepare_data.ipynb”

I am also attaching a pdf version of this jupyter notebook in this project folder for your perusal.

After running the above script, it organizes the data as follows:

- Train
  - Images
    - 00000.ppm
    - 00001.ppm
    - …
  - Labels
    - 00000.txt
    - 00001.txt
    - …
- Valid
  - Images
    - 00600.ppm
    - 00601.ppm

        - …
  - Labels
        - 00600.txt
        - 00601.txt
        - …

Further analysis:

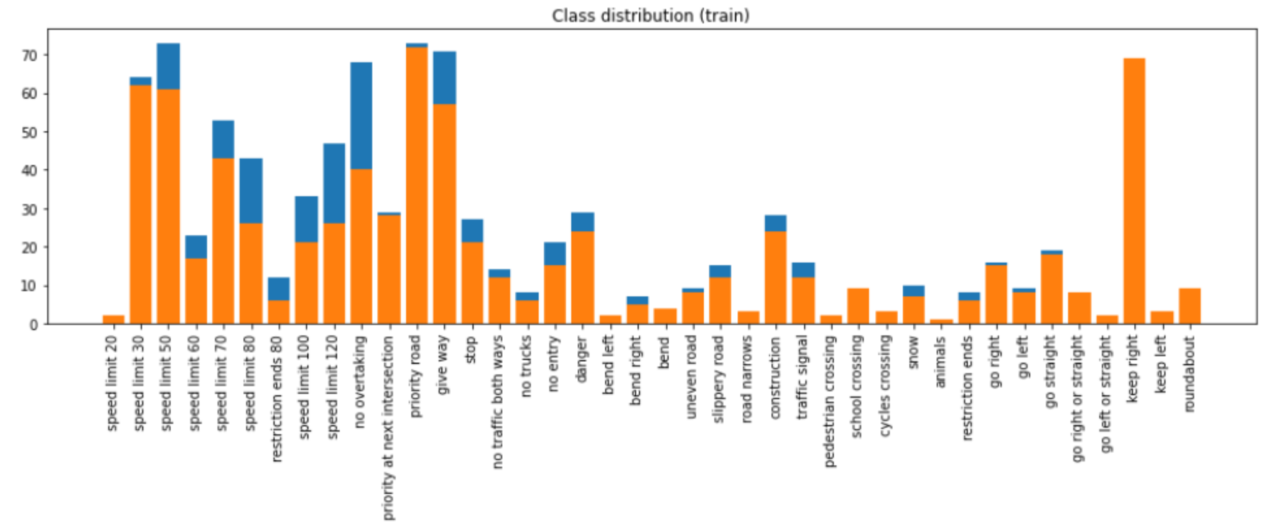
The 43 classes are numbered as follows:

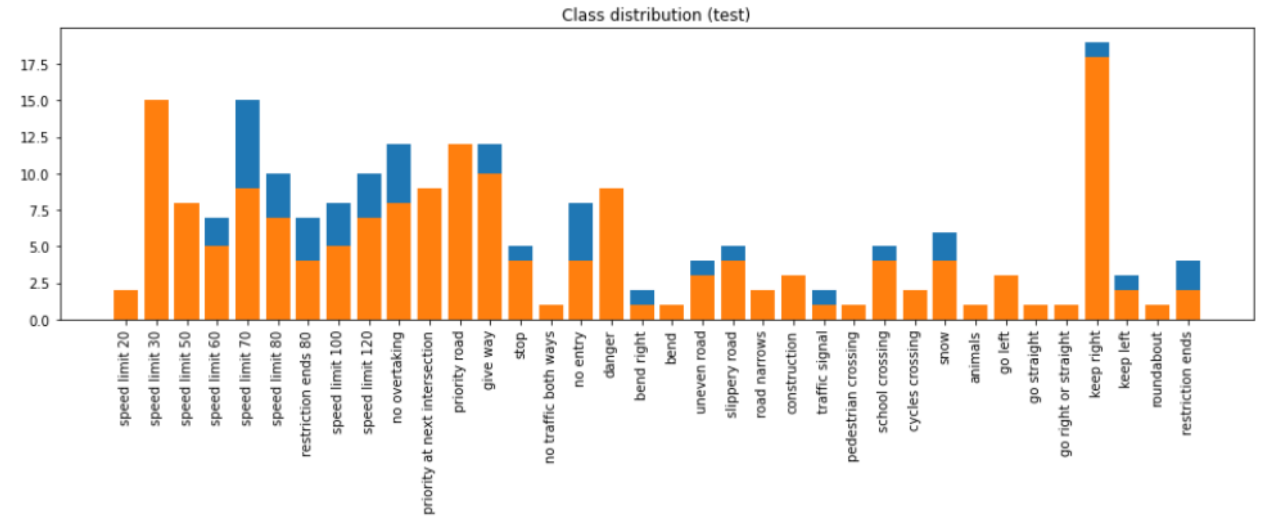
0 = speed limit 20 (prohibitory)
1 = speed limit 30 (prohibitory)
2 = speed limit 50 (prohibitory)
3 = speed limit 60 (prohibitory)
4 = speed limit 70 (prohibitory)
5 = speed limit 80 (prohibitory)
6 = restriction ends 80 (other)
7 = speed limit 100 (prohibitory)
8 = speed limit 120 (prohibitory)
9 = no overtaking (prohibitory)
10 = no overtaking (trucks) (prohibitory)
11 = priority at next intersection (danger)
12 = priority road (other)
13 = give way (other)
14 = stop (other)
15 = no traffic both ways (prohibitory)
16 = no trucks (prohibitory)
17 = no entry (other)
18 = danger (danger)
19 = bend left (danger)
20 = bend right (danger)
21 = bend (danger)
22 = uneven road (danger)
23 = slippery road (danger)
24 = road narrows (danger)
25 = construction (danger)
26 = traffic signal (danger)
27 = pedestrian crossing (danger)
28 = school crossing (danger)
29 = cycles crossing (danger)
30 = snow (danger)
31 = animals (danger)
32 = restriction ends (other)
33 = go right (mandatory)
34 = go left (mandatory)
35 = go straight (mandatory)
36 = go right or straight (mandatory)
37 = go left or straight (mandatory)
38 = keep right (mandatory)
39 = keep left (mandatory)
40 = roundabout (mandatory)
41 = restriction ends (overtaking) (other)
42 = restriction ends (overtaking (trucks)) (other)

In the data preparation jupyter notebook, I also converted these class names into a list that goes into the config file for training the yolo model.

```
names: ['speed limit 20', 'speed limit 30', 'speed limit 50', 'speed limit 60', 'speed limit 70', 'speed limit 80',
        'restriction ends 80', 'speed limit 100', 'speed limit 120', 'no overtaking', 'no overtaking',
        'priority at next intersection', 'priority road', 'give way', 'stop', 'no traffic both ways', 'no trucks',
        'no entry', 'danger', 'bend left', 'bend right', 'bend', 'uneven road', 'slippery road', 'road narrows',
        'construction', 'traffic signal', 'pedestrian crossing', 'school crossing', 'cycles crossing', 'snow',
        'animals', 'restriction ends', 'go right', 'go left', 'go straight', 'go right or straight', 'go left or straight',
        'keep right', 'keep left', 'roundabout', 'restriction ends', 'restriction ends']
```

The class distribution is highly skewed. I have plotted the class distribution of the training set as well as the validation set below. The orange bars denote the number of images per class, the blue bars (higher than orange) denote the number of bounding boxes per class. Blue is >= Orange since there can be multiple boxes per image.





Part 2: Training and Testing

I used the Yolov5 PyTorch implementation by UtraLytics [2] for training the model. I chose this model as the codebase was in PyTorch and well-implemented and documented. I picked the yolov5s model, which is the smallest one of the family, and trained using 16-bit float precision for faster training.

Steps taken to prepare script for training:

1. I first prepared the data as described in Part 1.
2. Appended ".ppm" format as an allowed image format in utils/datasets.py as follows.

   ```
   IMG_FORMATS = ['bmp', 'jpg', 'jpeg', 'png', 'tif', 'tiff', 'dng', 'webp', 'mpo', 'ppm']
   ```
3. Created a data config file data/gtsdb.yaml with the following contents.

```
path: /home/ubuntu/Shlok/gtsdb/data  # dataset root dir
train: train/images # train images (relative to 'path') 128 images
val: valid/images  # val images (relative to 'path') 128 images
test:  train/images # test images (optional)

# Classes
nc: 43  # number of classes
names: ['speed limit 20', 'speed limit 30', 'speed limit 50', 'speed limit 60', 'speed limit 70', 'speed limit 80',
          'restriction ends 80', 'speed limit 100', 'speed limit 120', 'no overtaking', 'no overtaking',
          'priority at next intersection', 'priority road', 'give way', 'stop', 'no traffic both ways', 'no trucks',
          'no entry', 'danger', 'bend left', 'bend right', 'bend', 'uneven road', 'slippery road', 'road narrows',
          'construction', 'traffic signal', 'pedestrian crossing', 'school crossing', 'cycles crossing', 'snow',
          'animals', 'restriction ends', 'go right', 'go left', 'go straight', 'go right or straight', 'go left or straight',
          'keep right', 'keep left', 'roundabout', 'restriction ends', 'restriction ends']  # class names
```

4. Launched the training script using

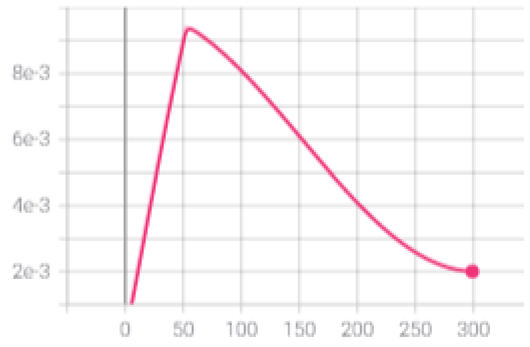
   ```
   python train.py --img 640 --batch 32 --epochs 300 --data
   data/gtsdb.yaml --weights yolov5s.pt --name run_name --noautoanchor --
   hyp data/hyps/hyp.scratch.yaml
   ```

   When yolov5s.pt is given as the weights argument, the script will automatically download their pre-trained yolov5 small model trained on COCO 2017 dataset. The train script finetunes the model starting from that pretrained model (the Detection layers are randomly initialized)

   I trained on image-size of 640 first then, at 1360 (full-size). More on that later.

Answers to the questions asked:

1. **How do you tweak your hyperparameters? What are they all about?**
   The hyperparameters are the various learning rate parameters, the batch-size, the image size, the number of epochs.
   This implementation using a one cycle learning schedule, where the learning rate rises at first (warm-up iterations), then it falls according to a cosine function. See the following.

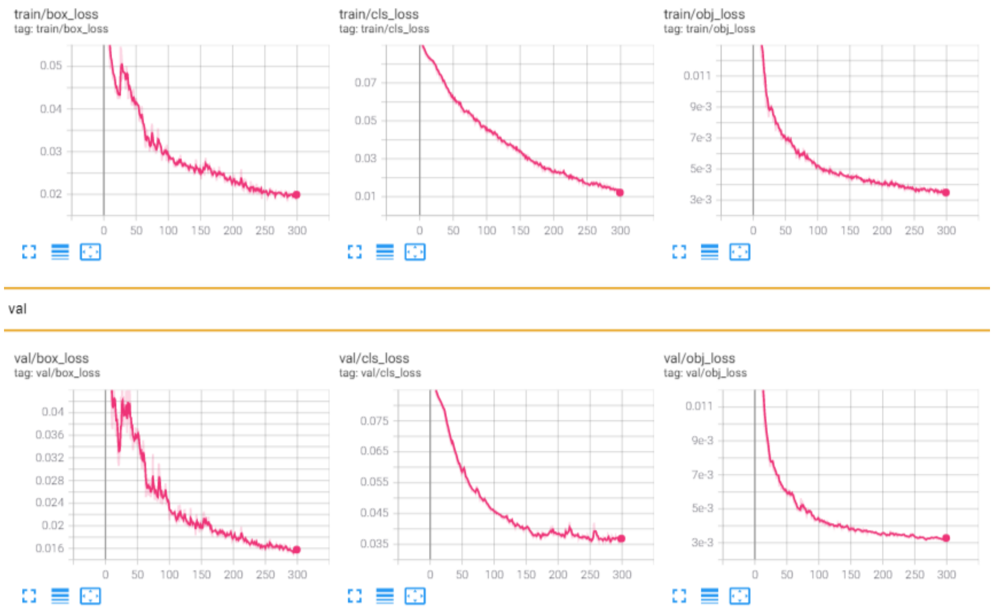


   The momentum of the SGD optimizer on the other hand, first drops, the increases. The hyperparameters are the highest learning rate (lr) in the cycle, the stopping lr, and the number of warm-up iterations.
   My general strategy: I start with a high lr (0.1) and train the model for a few iterations to see if the loss drops, if it doesn't, or if it increases, then I restart the training with a smaller lr, and repeat the process until I find an lr that works, and start the training with an lr which is slightly lower than the good lr found in the previous step.
   Other ways to check if there is no error in the model (like frozen gradients or wrong inputs/outputs), I take a very small subset of the training data and try to overfit on this data. It the model can't overfit on a small dataset, there is something wrong somewhere in the pipeline.

   For this particular case, I started with the learning schedule that was defined in the file:
   `data/hyps/hyp.scratch.yaml`
   These lr hyperparameters worked well. I could see that the training/validation loss was dropping, and the mAP was increasing on the validation set with the number of epochs of training.

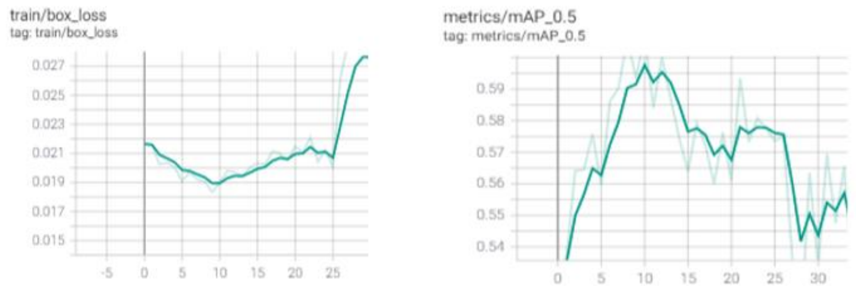
   I trained for 300 epochs (large number of epochs because the dataset is very small). A batch size of 32 seemed to fit on the GPUs I was training on (AWS g4dn.12xlarge instance)

2. **What's your training strategy? How can you tell if it works?**
   I started used the lr schedule given above and trained for 300 epochs on an image size of 640. The best mAP@.5 obtained was around 0.56. I could tell the training strategy was working, as the training loss was dropping (hence the model was able to fit on the data), the validation loss was also decreasing (hence the model was not overfitting), and the mAP@.5 score was decreasing. (see the tensorboard screenshot below)



I also evaluated qualitatively by visualizing the detections on the validation images. I noticed that the detection boundaries were very tight, but the predicted class were not very accurate, especially between different speed limit signs. One possible reason that I could think of was the signs are very small and hence it is difficult for the model to learn it when we train on image size of 640.

I then started from the current best model and finetuned the model on an image size of 1360 (full resolution). I used the same lr hyperparameters to begin with but I found that it did not work. See the figure below.



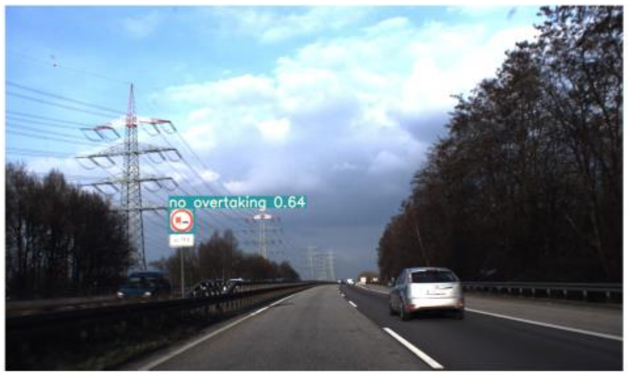
The training loss was increasing and map@.5 score was decreasing. I then restarted the training with a lower lr. I used the lr hyperparameters in the file `data/hyps/hyp.finetune.yaml`

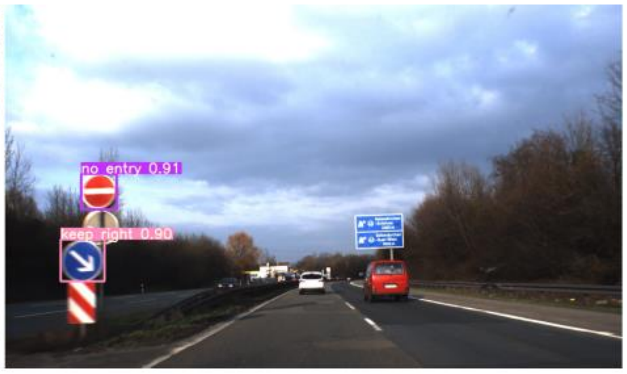
I finetuned the model on full resolution for another 100 epochs using the above lr hyperparameters. I could push the accuracy (map@.5) to ~0.62

The confusion between class predictions improved after training on the full resolution. However, there was still some confusion.

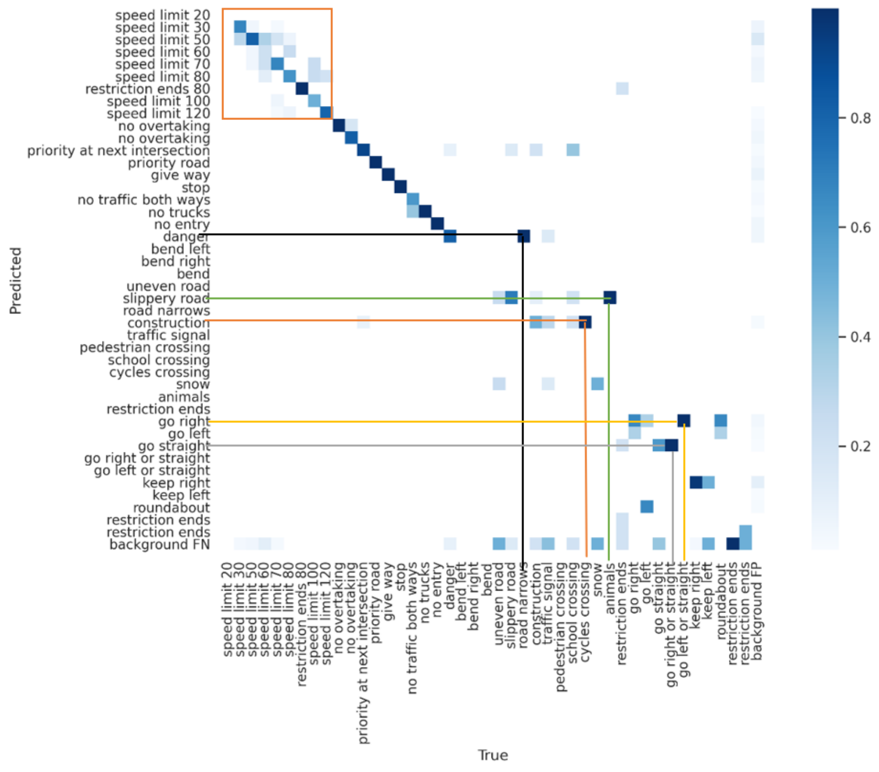
3. What's the performance of your detection system? How do you evaluate it?

The detector's performance was measured using mAP@.5 . This metric was used in the evaluation COCO object detection competition. The way this is measured is as follows. Define a detection when the GT and the prediction overlaps with and IoU (intersection over union) of at least 0.5. For each of the classes, calculate the precision (accuracy among all the positive detections) and recall (fraction of predictions among all the positives ground truths). AP@.5 is defined as the area under the curve of maximum precision vs recall. Now, mAP@.5 is the average across all the classes. Here .5 denotes that the IoU is .5 with a GT for a detection to be considered a hit.
We also evaluate the detector's performance qualitative by visualizing the detections after non-maximum suppression of all the detections.
We also see the confusion matrix to understand for which classes the model is getting confused and if it makes sense. See the example detections below.





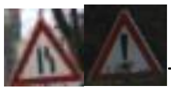
4. **Do you see any pattern in your FP/FN detection? Can you categorize the errors and propose potential solutions?**



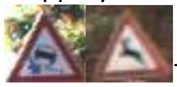
Some patterns can be seen from confusion matrix on the validation dataset as shown above.
We can see (box in the top left in the confusion matrix) that the model gets confused among the speed limit categories. This is understandable, since the boxes are small and not enough images for the model to learn all the numbers inside the signs. The overall shape and design is the same for all speed limit signs.

Some other prominent confusions are as follows:

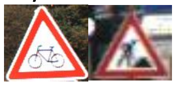
"Road narrows" and "danger"


The signs look very similar.

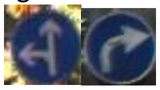
"slippery road" and "animals"


There is only one instance of each in the training data, and they look very similar.

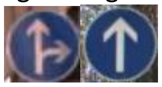
"cycles crossing" and "construction"


They look similar. Although, there are enough differences, so the model can be improved.

"go left or straight" and "go right"


There is only one image for "go left or straight" in the training data, so it is expected.

"go straight or right" and "go straight"

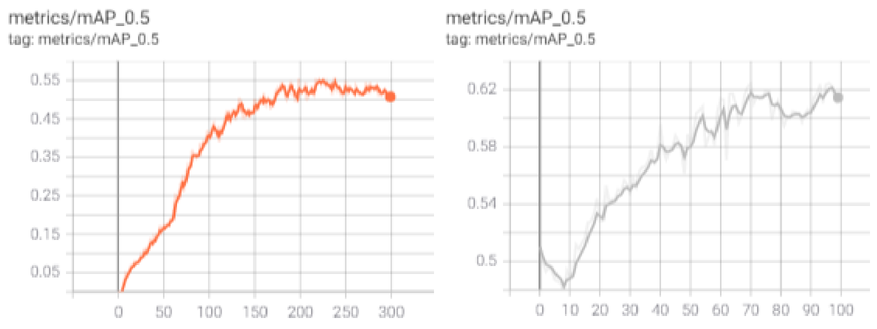

Even though the signs are similar, the model can do better.

Potential solutions:
For "slipper road", "animals" "go left or straight", it would be hard to improve the model without more data since there is only one example each.
For other cases, we notice that the overall shape and color of the sign is identified correctly. We just need the model to see the exact symbol/number inside the signs correctly. We can do that by cleverly augmenting the data. For example, by cropping the image by zoom in into some of these signs, and in general repeating the confusing classes more to handle the data distribution.

We can also use focal loss instead of a regular cross-entropy loss for the classifier head. This loss will compute a weighted loss function, giving more weight to the examples where the loss is worse.

5. **Did you need to go back to Part1 to improve it?**
   No, I would prepare the data just like before. I would handle all the augmentations in the dataloader code or change the loss function to focal loss.

6. **How does your results compare to the state of the art in sign or object detection? (You are not supposed to be as good as state of the art) What are the main differences with your approach?**



The above plots show the accuracy (mAP@.5) at each epoch of our training. The left figure shows the training for image size of 640. The right plot is the training that starts with the best model for the training with image size of 640 and trains for another 100 epochs on full resolution (image size = 1360). We achieve a score of ~0.62 for the best model.

The state of the art is at 95.77 mAP@.5 score using a Faster R-CNN Inception Resnet V2 model [3]. The main difference is a much larger model with ~64M parameters whereas our model has ~4M parameters. Another difference is that it is two-step process for the faster R-CNN. First, the region proposals, then another round bounding box regression and classification. The yolo family is a one-stage object detector.

7. **What is the main challenge you encountered in this project, if there is any, please tell us you thought of how you will overcome it?**

Some of the challenges as follows:

1. Improving the class accuracy since the objects were very small. Had to train on full resolution in the second stage to overcome it.
2. High class imbalance. I haven't taken care of it in this project. But some potential solutions are clever augmentations as discussed in the previous sections and using a focal loss.
3. Finding a good yolo implementation that is easy to use for fast prototyping. And understanding the codebase to figure out what changes needs to be made to train on a new dataset.

References:

1. German Traffic Sign Detection Benchmark GTSDB https://sid.erda.dk/public/archives/ff17dc924eba88d5d01a807357d6614c/published-archive.html
2. Yolo-v5 PyTorch implementation: https://github.com/ultralytics/yolov5
3. Arcos-García, Á.; Alvarez-Garcia, J.; Soria Morillo, L. Evaluation of Deep Neural Networks for traffic sign detection systems. *Neurocomputing* **2018**, *316*, 332–344