VRDL Homework 2: Street View House Numbers Detection

Zhi-Yi Chin
joycenerd.cs09@nycu.edu.tw

November 23, 2021

1 Introduction

In this homework, we participate in the SVHN detection competition hosted on CodaLab. The Street View House Numbers (SVHN) dataset [4] contains 33,402 training images and 13,068 testing images. We are required to train not only an accurate but fast digit detector. The submission format should follow COCO results. To test the detection model's speed, we must benchmark the detection model in the Google Colab environment and screenshot the results. Code is available at https://github.com/joycenerd/yolov5-svhn-detection.

2 Methodology

2.1 Data pre-processing

2.1.1 Annotation

The original labels of the data save in a `.mat` file, and the format is not what we want. We want the labels to follow the YOLO annotation format. YOLO annotation format means that for each image, we have a corresponding text file as it labels, and the name of the text file is the same as the image name. Inside each text file consist of numerous lines. There is one detected object information in each line: label id, normalize center x, normalize center y, normalize width, and normalize height.

In the `.mat`, we first extract the name column as the name of the text file. Then we extract the bounding box column. The bounding box column consists of height, left, top, width, and label information. To get the bounding box's center x coordinate, we add left and half of the width. We add the top and half of the height to get the bounding box's center y coordinate. We normalize the center x coordinate and bounding box width by the original image's width and normalize the center y coordinate and bounding box height by the height of the original image. This is the procedure of converting to YOLO format annotation of one object. We repeat this for all the objects in an image and all the images in the training data.

2.1.2 Train validation split

Since this dataset does not have its own validation set, we split the training data into 80% for training and 80% for validation. We separate the training image and validation images into separate folders, so as the labels text files. Then we create a custom data configuration file to state the root directory of our data, the training images directory path, the validation images directory path, the testing images directory path, the number of total classes, and all the class' names. The main program will read this configuration file to find all the data information it needs.

2.1.3 Data augmentation

For data augmentation, there are quite an amount of augmentations applied in training. Here is the list of augmentation: histogram equalization, resize and pad image while meeting stride-multiple constraints, random perspective, copy paste [1] and MixUP [6].

2.2 Model Architecture

We choose YOLOv5 as our detection model. In Figure 1, the goal is to produce an object detector model that is very performant (Y-axis) relative to its inference time (X-axis). Preliminary results show that YOLOv5 does exceedingly well to this end relative to other state-of-the-art techniques. In summary, YOLOv5 derives most of its performance improvement from PyTorch training procedures, while the model architecture remains close to YOLOv4.

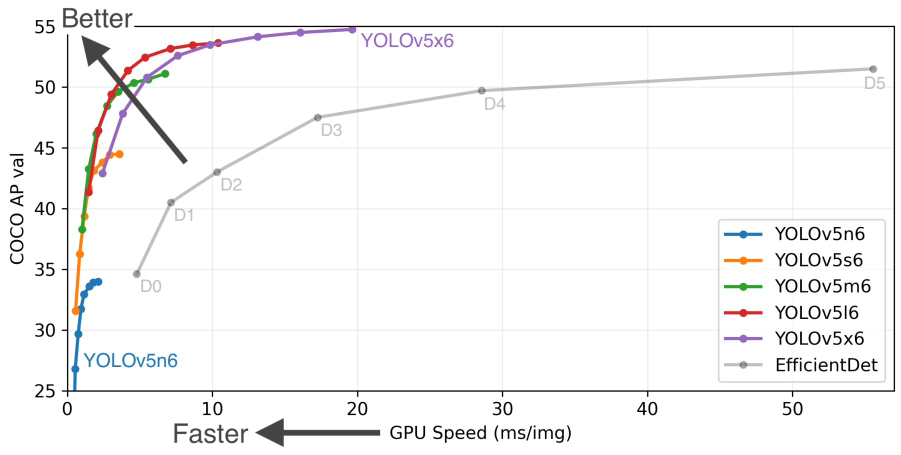


Figure 1: The initial release of YOLOv5 shows promise of state of the art object detection.

2.2.1 Why YOLOv5?

The goal of this homework is to train not only an accurate but fast digit detector. YOLOv5 is fast, precise, and easy to train, which suits all the needs.

2.2.2 How YOLOv5?

YOLOv5 has five versions; we apply YOLOv5s as our model. YOLO is a single-stage object detector, and it is mainly composed of three parts: model backbone, model neck, model head. YOLOv5 uses CSPNet Cross Stage Partial Networks [5] as the model backbone; it is mainly used to extract important features from the given input image. CSPNet has shown significant improvement in processing time with deeper networks. In YOLOv5, PANet [3] is used as the model neck to get feature pyramids; feature pyramids are very useful and help models to perform well on unseen data. As for the model head, YOLOv5 is the same as the previous YOLOv3 and YOLOv4 versions, and it is mainly used to perform the final detection part.

As for the implementation, we reference heavily from ultralytics/yolov5 [2].

2.3 Hyperparameters

For activation function, Leaky ReLU is used in the hidden layers, and Sigmoid is used in the final detection layer. For the optimization function, SGD is used, and the initial learning rate is set to 0.01 and weight decay set to 0.0005. LambdaLR is used as the scheduler, which is used to gradually decrease the learning rate while training. For loss calculation and object score, binary cross-entropy with logits loss function from Pytorch is used. We have also used the pre-trained weights yolov5s.pt from [2]. We trained the model for 150 epochs and took about 14 hours. Due to the high amount of small objects, we reshape all the images to 640×640 while training. The training batch size is set to 16, although it can be set larger. All of the experiments are run on a machine that has an NVIDIA RTX1080Ti graphics card.

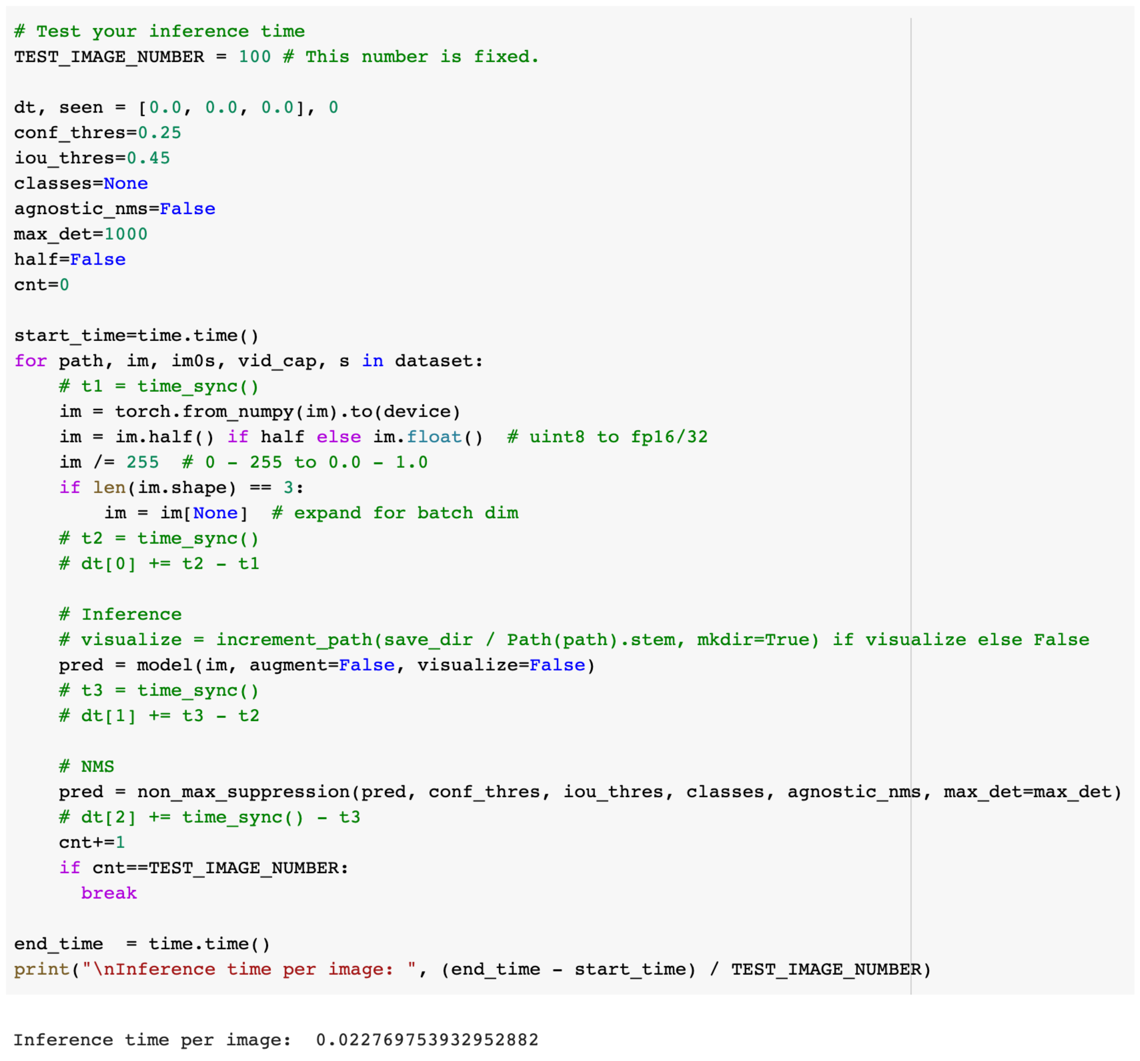
2.4 Post-processing

The detection output is in YOLO format (one image, one text file) as well, so post-processing is needed to transform to the submission format, which follows COCO detection results. The image id is the name of the text file. Since the bounding box of YOLO is normalized, so de-normalize is needed. We multiply the x center coordinate and the width of the bounding box by the image's width. For the y center coordinate and the height of the bounding box, we multiply by the height of the image. Then we will get the left of the bounding box, which is done by subtracting x center coordinate by half of the bounding box width. To get the top of the bounding box, we subtract the y center coordinate by half the bounding box height. The score is the confidence value of the prediction. Then we write all the information into a JSON file for submission.

3 Summary

3.1 Results

We got a score of 0.4067 on the test set when our confidence threshold is set to 0.25. When confidence threshold is set to 0.1, we got score 0.4172. We got the best score 0.4217 when our confidence threshold is set to 0.001. Furthermore, the speed of our detector is 22.7ms per image. The screenshot of the speed testing benchmark is shown in Figure 2. Figure 3 shows some of the qualitative detection results in the validation set.



```
# Test your inference time
TEST_IMAGE_NUMBER = 100 # This number is fixed.

dt, seen = [0.0, 0.0, 0.0], 0
conf_thres=0.25
iou_thres=0.45
classes=None
agnostic_nms=False
max_det=1000
half=False
cnt=0

start_time=time.time()
for path, im, im0s, vid_cap, s in dataset:
    # t1 = time_sync()
    im = torch.from_numpy(im).to(device)
    im = im.half() if half else im.float()  # uint8 to fp16/32
    im /= 255  # 0 - 255 to 0.0 - 1.0
    if len(im.shape) == 3:
        im = im[None]  # expand for batch dim
    # t2 = time_sync()
    # dt[0] += t2 - t1

    # Inference
    # visualize = increment_path(save_dir / Path(path).stem, mkdir=True) if visualize else False
    pred = model(im, augment=False, visualize=False)
    # t3 = time_sync()
    # dt[1] += t3 - t2

    # NMS
    pred = non_max_suppression(pred, conf_thres, iou_thres, classes, agnostic_nms, max_det=max_det)
    # dt[2] += time_sync() - t3
    cnt+=1
    if cnt==TEST_IMAGE_NUMBER:
      break

end_time  = time.time()
print("\nInference time per image: ", (end_time - start_time) / TEST_IMAGE_NUMBER)
```

```
Inference time per image:  0.022769753932952882
```

Figure 2: The speed benchmark screenshot from Google Colab.

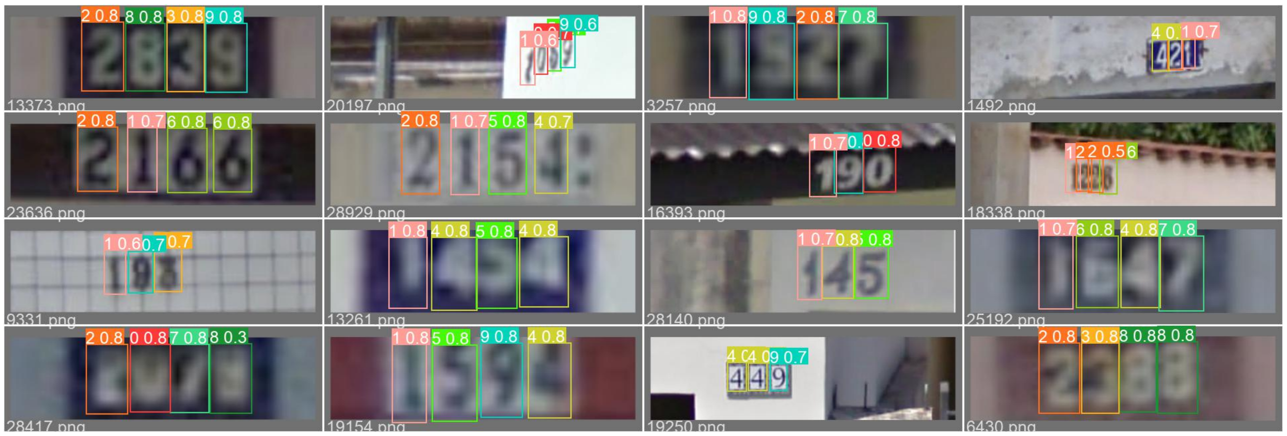


Figure 3: Qualitative detection results in the validation set.

3.2 Discussion

Before YOLOv5, we have also tried YOLOv3. We got higher validation mAP in YOLOv3 than YOLOv5, but the testing score of YOLOv3 is terrible; it only got 0.1424. We think the problem is in post-processing, but we look up other people's YOLOv3 detection post-processing online, but no different than ours. At last, we admit that YOLOv3 is not robust enough, so we switch to YOLOv5. All the hyperparameters are from the default setting, and the results come out pretty good. We have also tried different confidence threshold, and we found but that the score decrease when we use higher confidence threshold and the score increase when we decrease the confidence threshold. The highest score we get is when the confidence threshold is set to 0.001. We think this is because there are many small objects; predicting these objects is hard, the confidence will not be high. If we use a higher confidence threshold, these objects will not be in the final detection results. Therefore, we think confidence threshold is what needed to concern as well when doing detection.

3.3 Conclusion

In this homework, we have applied YOLOv5 to train a accurate and fast digit detector. The advantages of our method are:

- Easy to train.
- Need little computation resource.
- Fast detection, since YOLO is a one stage detection method.

References

[1] G. Ghiasi, Y. Cui, A. Srinivas, R. Qian, T.-Y. Lin, E. D. Cubuk, Q. V. Le, and B. Zoph. Simple copy-paste is a strong data augmentation method for instance segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2918–2928, 2021.

[2] G. Jocher, A. Stoken, A. Chaurasia, J. Borovec, NanoCode012, TaoXie, Y. Kwon, K. Michael, L. Changyu, J. Fang, A. V, Laughing, tkianai, yxNONG, P. Skalski, A. Hogan, J. Nadar, imyhxy, L. Mammana, AlexWang1900, C. Fati, D. Montes, J. Hajek, L. Diaconu, M. T. Minh, Marc, albinxavi, fatih, oleg, and wanghaoyang0106. ultralytics/yolov5: v6.0 - YOLOv5n 'Nano' models, Roboflow integration, TensorFlow export, OpenCV DNN support, Oct. 2021. URL https://doi.org/10.5281/zenodo.5563715.

[3] S. Liu, L. Qi, H. Qin, J. Shi, and J. Jia. Path aggregation network for instance segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, pages 8759–8768, 2018.

[4] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng. Reading digits in natural images with unsupervised feature learning. 2011.

[5] C.-Y. Wang, H.-Y. M. Liao, Y.-H. Wu, P.-Y. Chen, J.-W. Hsieh, and I.-H. Yeh. Cspnet: A new backbone that can enhance learning capability of cnn. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops (CVPRW)*, pages 390–391, 2020.

[6] H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz. mixup: Beyond empirical risk minimization. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=r1Ddp1-Rb.