Recognize Chinese Characters on Traffic Signs: Part 2

Qifan Zhang

March 2019

1 Dataset

After doing some training using ctw project's code with its dataset[9], I realized that using this dataset has some inconvience, especially it is difficult to filter out only chinese characters on traffic signs. If we use the original dataset to train, the model will detect all chinese characters on the road (on the billboards for example) and is not what we want.

I decided to use another dataset in this project: traffic panel dataset[4] of National Nature Science Foundation of China(NSFC) because it is more appropriate for the use case of this project: it contains images taken on the road of chinese cities, with traffic signs on which simplified Chinese characters are labelled, and others (chinese characters on billboard for example) are not labelled.

However it also has some drawbacks: although it also contains images of different illumination/weather conditions, the camera positions are fixed. Besides, the chinese characters are labelled grouped by expression instead of single characters. In the real use cases, we may need to detect characters on images taken by people using different camera positions. For these reasons, the dataset is relatively simple, and the validation mAP is very high (see next parts).

The dataset contains 2329 images, I splited it to 85% for training and 15% for validation. An important pre-preprocessing step is to transform the dataset to darknet compatible format because I decided to use yolo-v3 as training algorithm. I converted all different Chinese character classes into one class: Chinese character because in this project the goal is detection instead of classification. More detail can be found on Github repository.

2 Network Structure

R-CNN[2], which was appeared in 2014 outperformed traditional computer vision detection algorithms in terms of performance (mAP). However it does have some drawbacks: The training process has multiple stages and is complexe, it requires a lot of time. Besides, during detection stage, comparing to state-of-art methods, the speed is not satisfying even using a GPU with high performance. There have been some refined version of RCNN such as Fast R-CNN[1], Faster R-CNN[8]. As their names indicate they are faster than the original RCNN.

YOLO[3], as its name indicates, needs only one stage training, and is extremely fast comparing to RCNN methods. The first version of YOLO however, is not as good in terms of mAP. But in yolo-v2[6] and yolo-v3[7] integrated newer techniques such as anchor boxes, batch normalization etc. which enables it to have a better mAP by keeping its fast speed during both training and detection stage.

In this project I chose tiny yolo-v3 as NN architecture. According to the paper[7], it achieves 33.1% as mAP@0.5 on the coco dataset. As a comparaison, yolo-v3 has 57.9% mAP@0.5. Nevertheless, in terms of detection speed, it is much faster comparing to complete version: 220 FPS vs 20 FPS.

The reason I used tiny version is that my GPU does not have enough RAM for complete yolo-v3, and tiny yolo-v3 is largely enough given the simplicity of dataset.

For the choose of framework, I used Darknet[5] and I modified it for precision-recall curve.

3 Hyper parameters

One important thing in deep learning is to tweak different parameters. In yolo-v3 (tiny) network, some important parameters are:

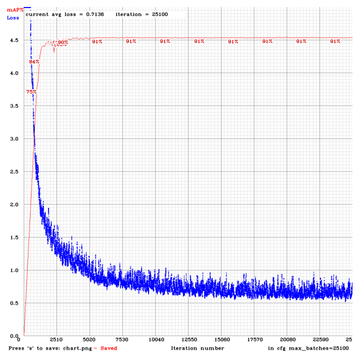
- learning rate and update policy: One of the most important parameters in machine learning. By using a big learning rate, the model updates faster but is more likely to get stuck. Here I used a fixed start learning rate 0.001 and used steps based learning rate update policy.
- Batch size: The number of samples calculated for one back propagation. Usually by taking a relatively big number allows us to train faster but requires more GPU RAM. I took 64.
- Width and height: The resolution of image passed to the network. In yolo-v3, the author gave the possibility to make this value dynamic. The default value was 418. By taking a higher resolution the network predicts more bounding box and if it is appropriately chosen, gives higher mAP. I took 608 in this project.
- Anchor: A new feature introduced in yolo-v2 and v3, it predicts the size of the bounding box. To calculate its value, a common way is to use K-means on the data set. In tiny yolo-v3, we can use up to 6 different anchor boxes. This value is critical, if it is too far away from real value, the training can become slow and can get stuck.
- Activation function: The function used in activation layer. In yolo-v3 by default leaky ReLU is used.

4 Training

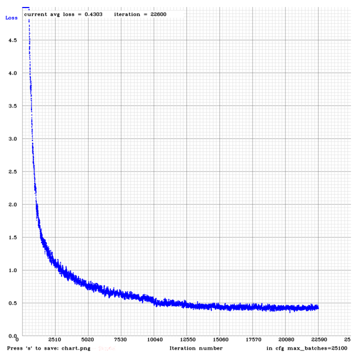
In this project, the goal is to detect Chinese characters without classifying them, thus only one output class, and also given that there are really few outliers in both training/testing data set, the training process is relatively straightforward.

During training, we have to take care of overfitting problem. With batch normalization layer the problem appears less frequently. And also with the dataset I used, the problem did not seem to appear.

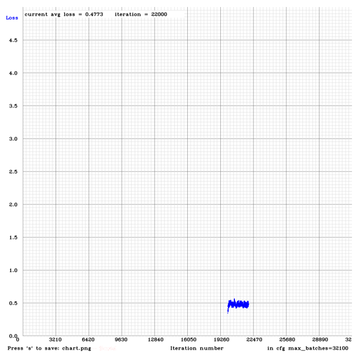
By applying the default value of different parameters, we can see that at the end of training, validation mAP (on the figure it is mAP@0.5) is already at 91% and the loss is at 0.7.



(a) Default



(b) Refined



(c) Resume using big LR

Figure 1: Figures

I then recalculated anchors, and augmented resolution of network. Besides, we can see that for this simple project, the default learning rate worked fine at the begining, but caused a lot of fluctuations at the end. I added some steps in the LR update policy to lower down learning rate during training.

By doing these, the mAP@0.5 is still at 91% (not showed on the figure for technical problem). This is because of the precision of calculation (in AlexeyAB's Darknet, he used 10 points on precision-recall curve). The loss has a significant improvement: 0.4. Furthermore, there are less fluctuations and in the following part we can see there is a great improvement in mAP@0.8.

I also applied a technique: augmenting the learning rate when the loss is no more improving which in some paper and projects significantly ameliorated the result. I manually stopped the training at 17500 iteration and resumed by reusing the initial learning rate, but it did not work: the loss became higher and unstable.

5 Evaluation

To evaluate the performance of detection system, some common statistics are: IoU, precision, recall and mAP.

- IoU evaluates the percentage of intersection area between predicted bounding box and ground truth.
- Precision=TP/(TP+FP), which means among all the detected objects how many are correct.
- Recall=TP/(TP+TN), which evaluates among all the ground truth objects how many are detected.
- mAP which corresponds to Mean Average Precision, since in this project, we only have 1 class (Chinese character) to detect, mAP is equal to Average Precision, which is the surface surrounded by precision-recall curve (see below).I slightly modified the code of Darknet to generate it.

Both precision and recall can be affected by the chosen IoU threshold: above how much IoU the detected area is treated as True.

mAP@0.5 is very high: 91%. This is not astonishing given that this project uses a relatively simple dataset with few outliers, and we only need to detect one class. mAP@0.8 is still really good: 66.79%.

Besides we can also test the detection speed. On AlexeyAB's version of Darknet, my system detects 100 frames/s. Just as said in the paper, it is a faster and stronger system.

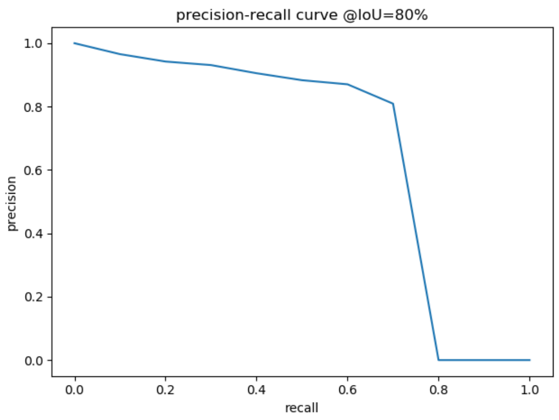
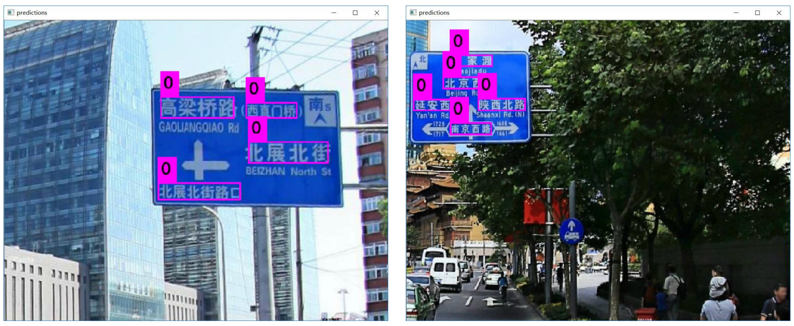


Figure 2: PR curve IoU=80%

Here are some detection results, notice that the second one is extremely important because this is where my home is:



(a) Beijing (b) Shanghai

Figure 3: predictions

6 Future improvement

In this part, I showed some simple techniques to fine tune the CNN's hyper parameters and the result is satisfying because of the simplicity of the dataset. Given more time, it is possible to find a larger and more complexe dataset with different camera positions to train and test on.

Besides, it is possible to test more possibilities in the network such as:

- using different activation function
- using different optimizers
- using other network structure, such as yolo v2 or even faster R-cnn, SSD, RetinaNet etc.

This list is not exhuastive. In fact in deep learning we do have a lot of empirical things, and time and computing power (GPUs) are critical to the final result obtained.

References

[1] Ross B. Girshick. Fast r-cnn. 10.1109/ICCV.2015.169, 2015.

[2] Ross B. Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. 10.1109/CVPR.2014.81, 2014.

[3] Ross Girshick Joseph Redmon, Santosh Divvala and Ali Farhadi. You only look once: Unified, real-time object detection. 10.1109/CVPR.2016.91, 2016.

[4] National Nature Science Foundation of China(NSFC). Chinese traffic sign database. http://www.nlpr.ia.ac.cn/pal/trafficdata/panel.html.

[5] Joseph Redmon. Darknet: Open source neural networks in c. http://pjreddie.com/darknet/, 2013–2016.

[6] Joseph Redmon and Ali Farhadi. Yolo9000: Better, faster, stronger. 10.1109/CVPR.2017.690, 2016.

[7] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. Tech report, 2016.

[8] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster r-cnn: towards real-time object detection with region proposal networks. NIPS'15 Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1 Pages 91-99, 2015.

[9] T. Yuan, Z. Zhu, K. Xu, C. Li, and S. Hu. Chinese text in the wild. *CoRR*, abs/1803.00085, 2018.