CS307 Final Project Report

Zitong Huang, Zengrui Lu, Kai Li
12012710@mail.sustech.edu.cn
12013022@mail.sustech.edu.cn
12013029@mail.sustech.edu.cn

1. Introduction

In traffic systecm, traffic sign recognization is a debatable problem. By recognizing the traffic sign, risks of accidents can be significant decline while processing auto-driving problem. With the application of darknet[8] based on yolov4[1] and yolov5 [13], Chen has introduced TSR-SA(traffic-sign recognition small-aware) to imporove the performance of model while processing small traffic-sign[2].

In this project, we have reproduction Chen's experiment, train the model and use it on traffic-sign detection and classfication on dataset collected by ourselves, to test the module's performance and efficiency.

2. Related Works

2.1. Object Detection

Pre-existing domain-specific image object detectors usually can be divided into two categories, the one is two-stage detector, including R-CNN[4], Fast R-CNN[3], Faster R-CNN[12]. Two-stage detectors have high localization and object recognition accuracy. The other one is one-stage, including YOLO[9], SSD[7]. The one-stage detectors achieve high inference speed but low accuracy for tiny object.

YOLO(You Only Look Once) is first one-stage method, which divides each image into a $S \times S$ grid, and then, each grid is responsible for detecting those objects whose center points. The author has already proposed proposed YOLOv2[10], v3[11], v4[1].

2.2. Traffic-signs recognition

Traditional traffic-signs recognition methods can be divided into two categories: color-based and shape-based[6]. Now, it has become consensus that machine learning-based methods are superior to traditional methods. Among the machine learning-based methods, the deep learning-based or CNN-based methods have the best performance and have become the mainstream method of traffic-signs recognition.

3. Method

Inspired by the related Deep Learning Network methods, the paper propose TSR-Small-Aware(TSR-SA), to solve small object detection problems. In this section, I will intoduce four parts: Firstly,I would briefly introduce YOLOv4, the baseline of TSR-SA. Secondly, I introduce receptive field block-cross (RFB-c), which is designed to fuse low-level detailed and high-level semantic features. Thirdly, we describe the details of the proposed TSR-SA. Then finally introduce the data augmentation method Random Erasing-Attention(RE-A).

The figure 1 shows the complete TSR-SA's network structure.

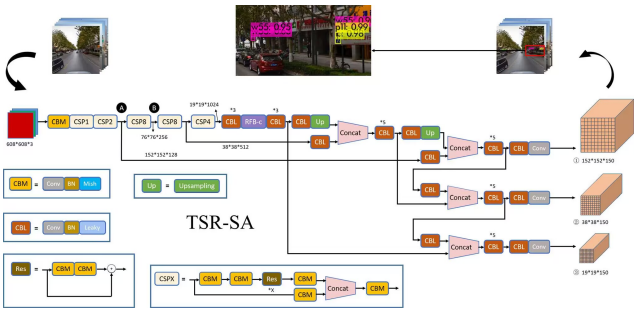


Figure 1.

3.1. Baseline-YOLO method

YOLO is a one stage object detection method. In the YOLO detection pipeline, the network output are all bounding boxes and probabilities for each classification. The YOLO method realized an end-to-end detection. The

YOLO network's input is an image, and YOLO would divides the input image into an $S \times S$ grids. The grids are the main part of detection: each grid will detect the target whose centers are within that grid and make a predict B bounding boxes and the classification C. YOLO defines the size and position of bounding boxes are a tuple (x, y, w, h), which respectively stands for x coordination, y coordinarion, width and height. So we can get the prediction is a tensor of $S \times S \times B \times (5 + C)$, 5 stands for p, x, y, w, h.

The paper use the YOLOv4 which is proposed on April 2020 by Bochkovskiy et al. From the YOLOv3, YOLOv4 improves in these fields: Mosaic data augmentation, Cross-stage partial connections (CSP) , Mish activation, SPP-block, PAN path-aggregation block. And YOLOv4 is an efficient and powerful object detection model, so they chose YOLOv4 as baseline.[2]

3.2. Receptive field block cross(RBF-c)

The paper proposed RFB-c[2] to build the most significant contextual features. RFB-c block is composed by Input layer, four branches and Output layer. Each branch consists of three components: CBL, CBL-d and shortcut connection from input layer. Finally, four branches' outputs are concatenated to Output layer. the four feature tensors ($19 \times 19 \times 512$) are concatenated into a multi-scale contextual feature tensor ($19 \times 19 \times 2048$). The figure 2 shows the figures RBF-c block's structure.

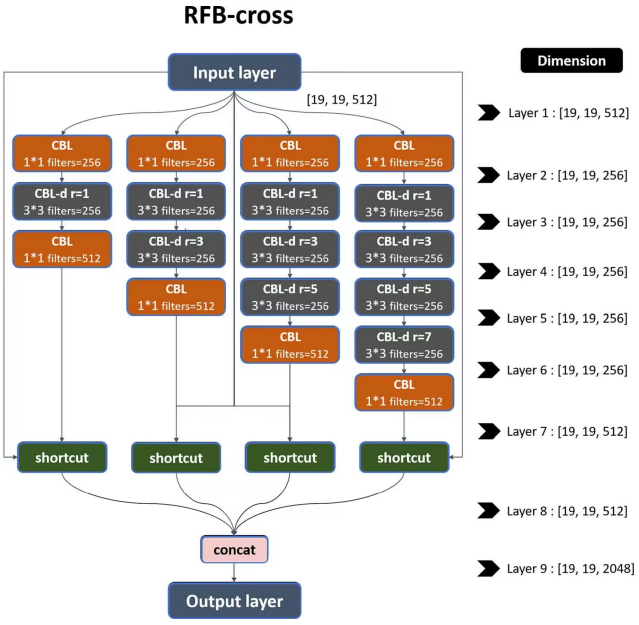


Figure 2.

1. CBL block is composed of convolutional layer, batch normalization, and leaky activation function.

2. The components of a CBL-d block are the dilated convolutional layer, batch normalization, and leaky activation function. The only difference of CBL-d from CBL is dilated convolution.

3. The intention of dilated convolution is to generate a multi-scale feature map, capturing contextual information without increasing the amount of parameters.

As shown in figure 3,the blue area represents the size of the receptive field, with 3*3 on the left and 7*7 on the right. we will only compute at where the red points are.

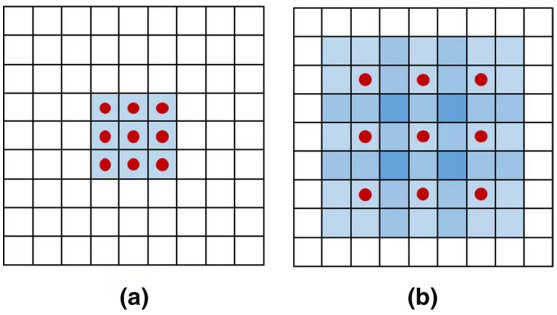


Figure 3.

RF_n is the receptive field of layer,and recursive inference formula is

$$RF_{n+1} = RF_n + (k_m - 1) * \left(\prod_{i=1}^{n-1} s_i \right)$$

We have $k_m = k + (k-1)(r-1)$ is is equivalent kernel size after dilation, $k = 3$ is the origin kernel size,$s_i = 1$ is stride of layer i.

3.3. TSR-SA

"YOLOv4 has a poor performance in small object detection.Because after multiple downsampling, the feature map has little spatial information for small instances.",Chen, J., Jia, K., Chen, W. et al said[2]. The pipeline of TSR-SA consists of 4 stages: backbone for extracting base features, RBF-c for building contextual features, Neck forfusing low-level and high-level features, Head for making predictions. We can see the figure 4 to see the pipeline.

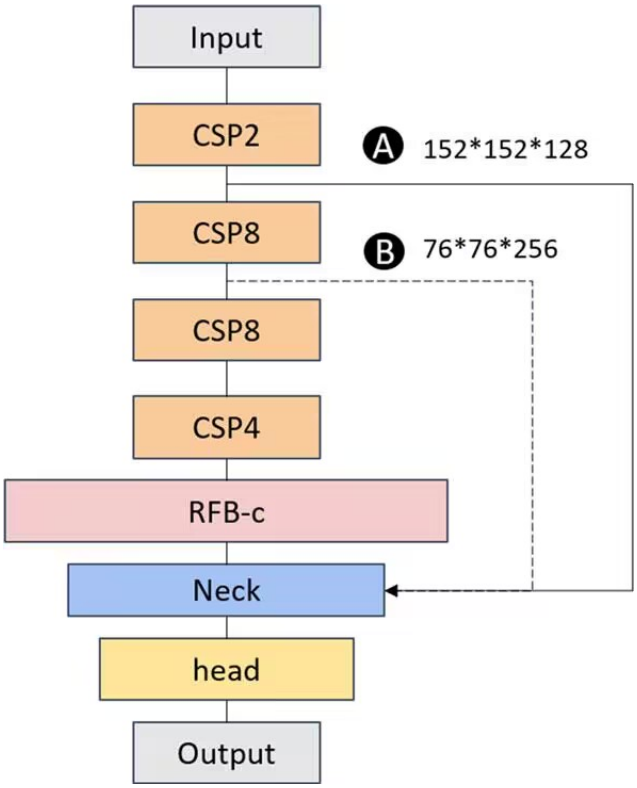


Figure 4.

3.3.1. Backbone

The input image will be resized to a $608*608$ for feature extraction. The TSR-SA used CSPDarknet53 as backbone network.

3.3.2. RBF-C

The paper devise the RFB-c block and add it over the backbone, because it significantly increases the receptive field, builds the good contextual features and doesn't cause reduction of the network operation speed.

As I mentioned above in 3.2,the input of RFB-c is a feature with dimension 19*19*512 , and output is 19*19*2048

3.3.3. Neck

Neck build the top-down structure to fuse low-level and high-level features from different backbone levels for detection in differet levels. And PANet is selected as the neck.

3.3.4. Head

TSR-SA apply three scales to detect objects

on the feature map output by neck.They use 152*152 grids to divide the picture instead of 76*76 in the detector head-1, as illustrated in Fig. 5. Each grid is responsible for predicting objects whose center point falls within its range. Therefore, the 152*152 grids improved the ability to locate small objects. And 150 in the tensor (152 * 152 * 150) represents 3*(5 + 45), as mentioned in Section.3.1, $C = 45, B = 3$

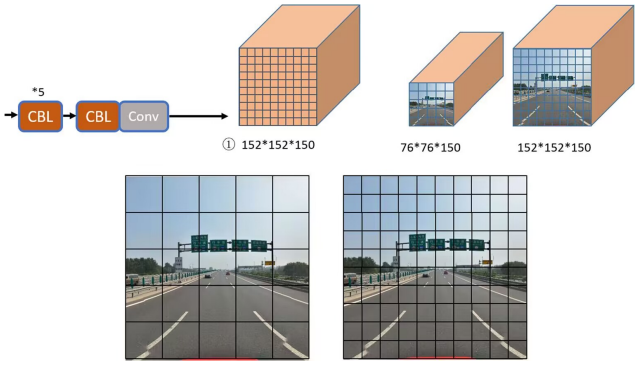


Figure 5.

3.4. Random Erasing-Attention(RE-A)

The paper[2] proposed the Random Erasing-Attention method to generate hard samples. The difference between RE-A and

Random Erasing(RE) is that the former makes an attention on the traffic signs and RE focuses on the entire picture. The RE-A is better than RE in TSR field because the former only detect traffic signs, so we need to focus on the part of the traffic signs, not the entire picture. Figure 6 illustrates the psedocode of RE-A:

Algorithm 1: Random Erasing-Attention(RE-A)

Input: An image X;
(x, y, w, h) of each labeled object;
Maximum erasing probability P;
Area ratio Ar_l and Ar_h;
Aspect ratio r_l and r_h
Output: A new image X^*

Initialization: $p \leftarrow Rand(0, 1)$
if $p \geq P$ **then**
  $X^* \leftarrow X$
else
  Get the (x, y, w, h) of the labeled object;
  Get the erasing value $\gamma \leftarrow Rand(0, 255)$;
  $S = h \times w$;
  $Ar \leftarrow Rand(Ar_l, Ar_h)$;
  $S_e = S \times Ar$;
  $r_e \leftarrow Rand(r_l, r_h)$;
  $H_e = S_e \times r_e$, $W_e = S_e \times r_e$;
  $\mathcal{M} = (H_e, W_e, h, w)$;
  $X^* \leftarrow Mask(X, \mathcal{M})$
end
return X^*

Figure 6.

For an image, RE-A randomly selects some pixels in a rectangular area of traffic signs and replaces them with random values.

$$X^* = Mask(X, \mathcal{M}) = \mathcal{M} * X + (1 - \mathcal{M}) * X$$

RE-A mask $\mathcal{M}$ is binary in the same shape of the selected region. And the mask is filled with 0 when the region is selected to be erased.

4. Experiments

4.1. Dataset

We use tt100k(Tsinghua-Tencent 100K)[14] as our training dataset. It provides 100000 images containing 30000 traffic-sign instances[14].

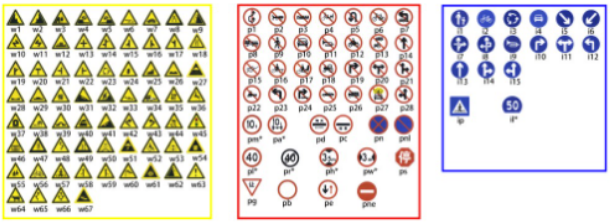


Figure 7. Some Traffic-Sign contained by tt100k

In this project, we choose 45 signs as traffic-signs needs to be classified.

For test, our group collect over 30 pictures and 1 video from wild, to test model's accuracy rate and recall rate while model be applicated in daily life.

4.2. Implementation Details

In this project, most hyper-parameters is same as Chen's work[2]. Some important parameters is list in following table.

Batch size	image size of network input	learning rate
16	608*608	0.001

Table 1.

More details and parameters can be found in TSR-SA/yolov4-tt100k_base.cfg at main · Kunkun-Jia/TSR-SA (github.com)

The hard-ware environment when we test the module in this project is

1. CPU: HUAWEI Cloud 8vCPUs
2. GPU: NVIDIA Tesla V100
3. Memory: 32GiB
4. OS: Ubuntu 16.04 Server

Other environment required and version is listed below

1. OpenCV: 4.6.0
2. CUDA: 10.2
3. cuDNN: 8.0.5.96
4. cmake: 3.25.1
5. GNU Make: 4.1

4.3. Metric

Some parameters is used to evaluate the performance of the module. A list of these parameter is list below:

Statistic	True	False
Positive	2941	1309
Nagetive	Too much	1213

Table 2.

In testing Confidence threshold is set to be 0.25, I can get the following result on the testset.

The TN result don't need to be considered because its number is too big. Model's precision = 0.69, recall = 0.71. If we set IoU threshold to be 50%,using Area-Under-Curve for each unique Recall,the test results mean average precision (mAP@0.50) =52.69 %

For video test, an averge FPS is got as 11. It is worth to be mention that we get the averge FPS with model training to save time. Since our server been shutdown accidently because of bills due, we have no enough time to test model performance on video without model training.

4.4. Experimental design & results

4.4.1. Data Processing

As the requirments list by Jia[2], a VOC-2007 format data is first produced from tt100k format data by using codes provided by Hankerchen[5]. Then, by using format-transform code provided in Jia[2], VOC-2007 format data is transmitted to yolo-format data.

4.4.2. Model Training

Since Jia's work is based on darknet[8], darknet is use to train and test the model.

Since we use a server to train the model, command `./darknet detector train cfg/tt100k.data cfg/yolov4-tt100k_base.cfg backup/yolov4-tt100k_base.weights -dont_show -map` is used to do train operation. Command line parameters and their meaning is explained following:

`detector`: Command to do object detection

`train`: Runing type, train means it should train a model

`cfg/tt100k.data`: Information of data to be input to the network

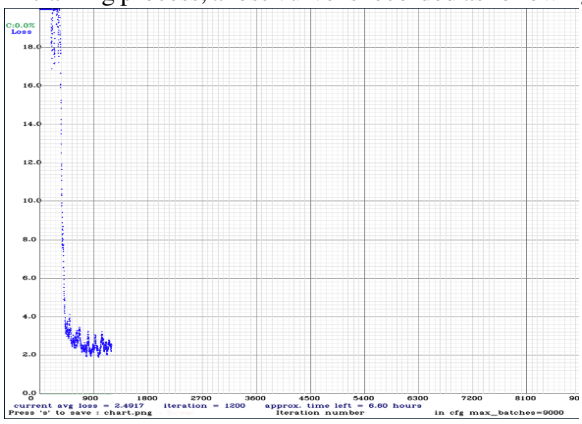
`cfg/yolov4-tt100k_base.cfg`: Network configuration file, hyper-parameters is assigned here.

`yolov4.conv.137`: pre-train parameter. It can be replaced by model weight to continue last shut-downed train.

`-dont_show`: Don't show picture window. Sicne we are using server, no GUI can be used.

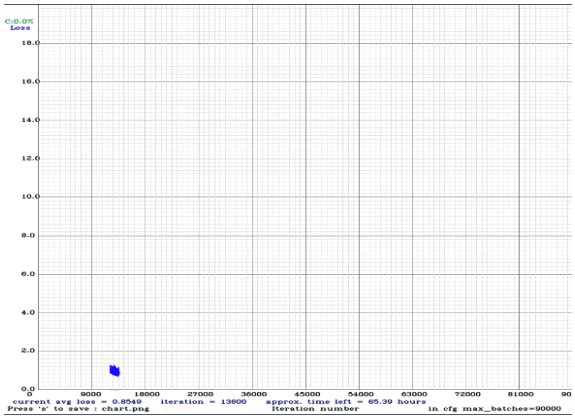
`-map`: train the model with mAP

In training process, a loss vurve is recorded as following



loss curve between batches 0-1200

Figure 8.



loss curve between batches 12000-13000

Figure 9.

Our GPU Memory crashed while test batches reaches 1200, and because of our negligence, we lost loss information while training batches between 1200 and 14000, only two figure can be shown here.

mAP(Mean Average Precision)while training batches reaches 6000, 8000, 10000, 13000,14000 is list below:

Training Batch	6000	8000	10000	13000	14000
mAP(%)	36.72	41.67	39.51	48.81	52.69

mAP of different stage while training

Table 3.

As model's performance improved highly between batch 10000 to 13000, a possible reason is model jump out from a local optimum.

4.4.3. Model Testing

We use command provided by Jia[2] to test our models performance on image:

```
./darknet detector test cfg/tt100k.data
cfg/yolov4-tt100k_base.cfg backup/yolov4-
tt100k_base_last.weights -ext_output
test.jpg
```

Same as train command, cfg/yolov4-tt100k_base.cfg is configuration file of model(contains hyper-parameters of network), backup/yolov4-tt100k_base_last.weights is model parameters weight. test.jpg is image to be test. A sample output is following:

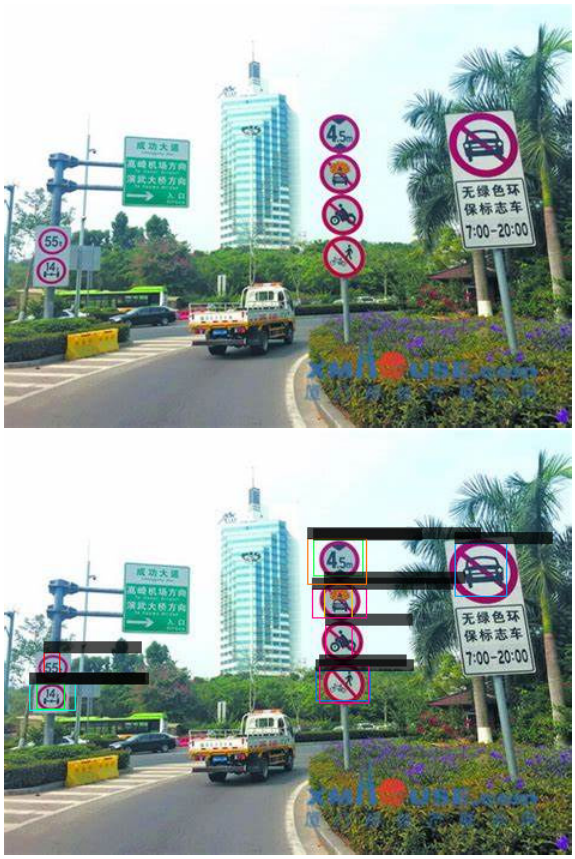


Figure 10. Sample test image and result

Also, we use

```
./darknet detector test cfg/tt100k.data
cfg/yolov4-tt100k.cfg backup/yolov4-
tt100k_base.weights -ext_output test.mp4
```

to test model performance on video. It have same parameter with image-test command. We will show it in our persentation.

5. Conclusion

The effort of the paper is to improve the speed and accuracy of small traffic-signs recognition. In this project, we reimplemented the model of TSR-SA and get the 0.54 mAP with test dataset. The average FPS is 11 in training process. The result is satisfied and the model can detect traffic-signs in most situation. Limited to time and server hardware environment, we can not do the training better. Though, we learned how to use dataset and darknet to train and test the model. Based on the YOLOv4, the model can apply to the automatic driving, serve to driver assistance system. If we have more time to do project, we will do some attempts for the training time and video test.

Bibliography

[1] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. Yolov4: optimal speed and accuracy of object detection. 2020.

[2] Junzhou Chen, Kunkun Jia, Wenquan Chen, Zhihan Lv, and Ronghui Zhang. A real-time and high-precision method for small traffic-signs recognition. *Neural Computing and Applications*, 34(3):2233–2245, 2022.

[3] Ross Girshick. Fast r-cnn. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1440–1448. 2015.

[4] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 580–587. 2014.

[5] Hankerchen. 小技巧（5）：将TT100K数据集转成VOC格式. https://blog.csdn.net/Hankerchen/article/details/120727299?spm=1001.2014.3001.5502. (2021,Obt 12).

[6] Chunsheng Liu, Shuang Li, Faliang Chang, and Yinhai Wang. Machine vision based traffic sign detection methods: review, analyses and perspectives. *IEEE Access*, 7:86578–86596, 2019.

[7] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott E. Reed, Cheng⟨keepcase|-Yang⟩ Fu, and Alexander C. Berg. SSD: single shot multibox detector. *CoRR*, abs/1512.02325, 2015.

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

[9] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: unified, real-time object detection. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 779–788. 2016.

[10] Joseph Redmon and Ali Farhadi. Yolo9000: better, faster, stronger. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6517–6525. 2017.

[11] Joseph Redmon and Ali Farhadi. Yolov3: an incremental improvement. *CoRR*, abs/1804.02767, 2018.

[12] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: towards real-time object detection with region proposal networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(6):1137–1149, 2017.

[13] Xingkui Zhu, Shuchang Lyu, Xu Wang, and Qi Zhao. Tph-yolov5: improved yolov5 based on transformer prediction head for object detection on drone-captured scenarios. 2021.

[14] Zhe Zhu, Dun Liang, Songhai Zhang, Xiaolei Huang, Baoli Li, and Shimin Hu. Traffic-sign detection and classification in the wild. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.