Traffic Sign Detection and Information Extraction

Zhiyu Lei
University of Rochester
zlei6@u.rochester.edu

Xiaojun Min
University of Rochester
xmin2@u.rochester.edu

Abstract

While driving on the roads, drivers have to frequently shift focus and often miss out the signs on the sides. In an attempt to solve this problem, we present a system that can detect and recognize signs, which is a key feature of autonomous driving that can improve the safety of transportation effectively. Instead of using RCNN-family (R-CNN, Fast R-CNN, Faster R-CNN), we use YOLO (You Only Look Once) which is a completely different approach to detect objects. After that, we use Tesseract and OpenCV to do OCR (Optical Character Recognition) for those traffic signs containing texts. Our method of traffic sign detection has reached an average recognition precision of 0.9697 and an average CPU time of 3.129 seconds on American traffic signs. The main code structure is available at this link to Github (training data and model weights excluded).

1. Introduction

We want to design a model that can detect the traffic signs from images taken on streets or highways and then can extract the meaning and major information from the signs, including but not limited to the highway number, the speed limit, the distance or direction to a place, etc. We try to train the model by simulating how a human driver learns the traffic signs. Our system not only allows detection in normal situations but also with different conditions and time frames.

There are two parts involved: first, we train YOLO to detect the custom images, then we use OCR to recognize the text on the traffic signs. Traditionally, the R-CNN family of algorithms use regions to localize the objects in images. High scoring regions of the images are considered as object detected. However, YOLO follows a completely different approach. It takes an entire image, splits it into an $S \times S$ grid, and applies a neural network to conduct classification on the complete image instead of each bound box. Within each grid, there are m bounding boxes and each bounding box has probability and offsets values. Objects with probabilities that are higher than certain thresholds are considered as objects detected. There is a good tradeoff between speed and accuracy by adjusting the size of the models. In general, the YOLO algorithm is faster than other object detection algorithms.

2. Related Works

In previous works, people already developed approaches that can detect or recognize traffic signs in real-time under various situations like different weather, illumination, and visibility, and have tested the results of the models on datasets such as the Tsinghua-Tencent 100K benchmark or German Traffic Signs Detection Benchmark (GTSDB) dataset.

Vennelakanti *et al*. use TensorFlow to implement CNN for traffic sign recognition and have achieved higher than 99% recognition accuracy for circular signs on Belgium and German data sets [2].

Arcos-Garcia *et al*. proposed a paper that analyses the state-of-the-art of several object-detection systems including Faster R-CNN, R-FCN, SSD, and YOLO V2 with various feature extractors such as Resnet V1 50, Resnet V1 101, Inception V2, Inception Resnet V2, Mobilenet V1, and Darknet-19, that are previously developed. They showed Faster R-CNN Inception Resnet V2 has the best mean average precision. They also showed that YOLO V3 (the one that we are presenting in this paper) can achieve a competitive advantage [4].

Zaki *et al*. have described a way that can detect and recognize traffic signs in real-time, considering various weather, illumination, and visibility through transfer learning. They use Faster Recurrent Convolutional Neural Networks (F-RCNN) and Single Shot Multi-Box Detector (SSD) with feature extractors such as MobileNet V1 and Inception V2, and also Tiny-YOLO V2. Their results have also shown that YOLO V2 achieved the best result [3].

In Pon *et al*., a deep hierarchical architecture with a mini-batch proposal selection mechanism is presented to detect both traffic lights and signs from training on different traffic lights and sign datasets. They measure their network on the Tsinghua-Tencent 100K benchmark and make instances from one dataset labeled in the other dataset. Their network

has low graphics processing unit (GPU) memory and thus more suitable for autonomous cars [1].

3. Methods

3.1. Data set

We trained the model using our dataset. First, we downloaded hundreds of images with traffic signs from various forums and blogs of transportation lovers. And then we used an external software called LabelImg, an image annotation tool with GUI access, to set labels for our dataset, which was a very time-consuming process since we had 500 pictures in total. We labeled the target objects (i.e., traffic signs) on each image by selecting the area where the objects are located and input their categories. The categories include Stop Signs, Yield Signs, Do Not Enter, No Left Turn, No Right Turn, No U-Turn, One Way, Railway Crossing, Speed Limit, Big Green Guiding Boards, Interstate Highway, and US Highway. Each image has a labeling result, inside the file, there are five parameters: classes, centers of x, centers of y, widths, and heights.

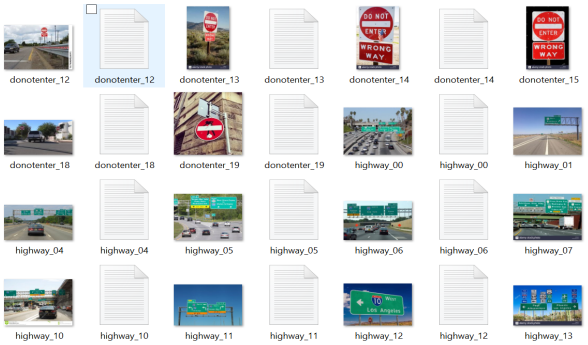

Figure 1. Images and labels

```
15 0.435535 0.506289 0.198113 0.396226
```

Figure 2. Inside the labelling file looks like this

3.2. YOLO Object Detection

YOLO is a state-of-art, real-time object detection system. Comparing to previous detection systems that applying the model to an image at different locations and scales, YOLO applies a single neural network to a full image, makes it one of the best models in object recognition (100 times faster than Fast R-CNN and 1000 times faster than R-CNN).

At the first step of detecting traffic signs, we ignore the specific numbers or words on the signs such as Speed Limit, Guiding Board, Interstate Highway labels, and US Highway labels, because we will use OCR to extract this information once they are detected. We used Google Colab with GPU to train the datasets, which allows us to write and execute arbitrary python code and shell commands through the browser. By connecting with Google Drive, we would not lose files in case of disconnection.

There were several steps before we started the training. First, connect with Google Drive and check if the NVIDIA GPU server was enabled. If it was, we downloaded the open-source Darknet model, which is a real-time object detection system and the framework we would use to train YOLO. Then we uploaded all the images and the corresponding labeling files in a single zip file to Google Drive. We also updated the YOLO V3 configuration file to match the number of categories we want. After Darknet was compiled using Nvidia GPU, the training process began.

It took several hours to train the models. After it finished training, the file storing the weights and configuration file for the YOLO model could be downloaded from the server to the local machine and could be read to the Deep Neural Networks module in OpenCV package using Python, which is used to detect traffic signs from local machines. The bounding boxes will be added to the images and their corresponding categories will be returned. This process is also called transfer learning, which is a technique to reuse the weights in one or more layers from a pre-trained network model in a new model by fine-tuning the weights and adapting that to a new model.

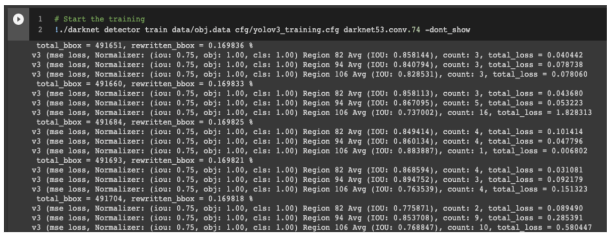

Figure 3. A screenshot of the training process

Figure 4. Image with bounding boxes

Figure 5. YOLO also works for photo taken at night

3.3. Optical Character Recognition

Once we have detected all the target traffic signs in the image, information of their categories, as well as bounding boxes, is returned just like the format in the label. We use Tesseract and OpenCV to do optical character recognition (OCR) on the returned bounding boxes if the corresponding category is Speed Limit, Guiding Board, Interstate Highway, or US Highway, which contains text information. Specifically, PyTesseract and OpenCV will be implemented to convert the text in images to string data. PyTesseract is a wrapper for Tesseract-OCR Engine. It can read all image types supported by the Pillow and Leptonica imaging libraries, including jpeg, png, gif, bmp, tiff, etc. The OCR engine is built on a Long Short-Term Memory (LSTM) network, which is a kind of Recurrent Neural Network (RNN).

Before running each bounding box on OCR, there are several steps we need to follow for image preprocessing, which ensures us to detect the text better. First is, of course, to crop the bounding box out from the original image. The cropped image is then resized to 2 times its original size so the texts have the maximum resolution and are clear for our eyes and to feed to our algorithms. However, they should not be too large because Tesseract does not work well with large images. Next, the image is converted to a binary image, which is gained by first grayscaling and then executing a bitwise-not operation, so that each pixel will only be 1 or 0, thus the image only possesses a white background and a black foreground. After that, we use a 3×3 kernel to slide over the image and apply erosion and diversion operations, which are two morphological operations. Erosion is used to detect whether the kernel has white or black pixels. For instance, if the kernel has at least a black pixel, the white pixel is eroded and the pixel will be considered black; otherwise, it will be considered as white. Dilation is the opposite of erosion. Erosion and dilation are combined to remove noises.

Now we can use OCR to extract the text from the preprocessed bounding boxes, but the output strings directly returned from the OCR model are somewhat messy, so we need to do some cleaning to the output. There are two things: first, all the letters are turned to upper cases, and all the beginning and ending white spaces as well as blank lines are trimmed; second, since some arrow symbols at the bottom of the guiding boards are very likely to be weirdly interpreted as letter "W" or "W" with one random letter, we also need to find and get rid of these weird things. Finally, the cleaned output strings will be printed out to the console after each preprocessed bounding box.

Figure 6. The original vs preprocessed bounding box

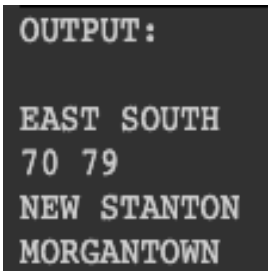

Figure 7. Cleaned OCR output

4. Experiments

We tested our model by randomly selecting 10 photos and checking if their information is accurate for YOLO (object detection) and OCR (text recognition) separately.

4.1. YOLO Object Detection

To measure the performance of YOLO, we use three metrics: precision, recall and $F_{measure}$. Specifically:
Precision $= \frac{TP}{TP+FP}$; Recall $= \frac{TP}{TP+FN}$;
and $F_{measure} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$.

Here, TP stands for True Positive (correctly detected traffic signs); FP stands for False Positive (incorrectly detected traffic signs, or signs/regions that are not in the models but identified incorrectly); FN stands for False Negative

(incorrectly not detected traffic signs, or signs that the system fails to detect). We do not have False Negative (correctly not detected) because there is no way to determine that we do not detect a traffic sign at a position with no signs. Also notice that we ignore signs in the images that are too far to be seen clearly by human eyes.

The table below shows the results for the 10 images using our model:

Img	TP	FP	FN	Wall Time	CPU Time
1	4	1	0	3.12s	4.18s
2	3	0	0	2.91s	4.02s
3	1	0	0	3.97s	2.49s
4	5	0	0	4.17s	3.32s
5	1	0	0	3.89s	2.39s
6	6	0	0	4.10s	3.26s
7	2	0	0	4.04s	2.77s
8	3	0	1	4.06s	2.89s
9	5	0	0	4.17s	3.38s
10	2	0	1	3.89s	2.59s

Table 1. YOLO performance on test images

To summarize the results, the average precision is 0.9697; the average recall is 0.9412, and the average $F_{measure}$ is 0.9552. In terms of time efficiency, the average wall time is 3.832 seconds, and the average CPU time is 3.129 seconds.

In the first test image, the false positive happens where a New York State Highway label (NY-297) is mistakenly detected as a US Highway label. This is mainly because we have not included State Highway labels in the training data set and these two kinds of labels look quite similar.

Figure 8. Test image 1 with one FP marked by red circle

In the eighth test image, the false negative happens where a US Highway label (US-11) fails to be detected. This is probably because the undetected sign is slightly behind an Interstate Highway label (I-81), and there is also another US Highway label (US-52) besides, which might interfere with the successful detection.

Figure 9. Test image 8 with one FN marked by red circle

In the tenth image, the false negative happens where a No Right Turn sign fails to be detected. This is probably because the undetected sign is a little bit far away in the image, and the truck in the background might also interfere with the successful detection.

Figure 10. Test image 10 with one FN marked by red circle

4.2. Optical Character Recognition

As for the performance of OCR, it generally does very well to extract the text information on guiding boards unless the photo is taken poorly and the words are hard to detect even by human eyes. However, it does not perform quite well on extracting the numbers from Speed Limit signs, Interstate Highway, or US Highway labels. The main reasons are that these kinds of signs, as well as the numbers on the signs, are quite small and that the patterns around the numbers on the signs may interfere with the OCR process as well. Since we know there is only one number on these kinds of signs, we can optimize by limiting the output range for these kinds of signs to make the model perform better.

Another drawback of the current model is that for guiding boards with texts in completely different colors (such as a board with white texts on green background and black

texts on yellow background, shown in Figure 8), the current model can only extract texts in one color, typically white texts on green background, since the preprocessing step of converting the bounding boxes to binary images cannot handle texts in completely different colors together. Therefore, to further optimize the current model, we can try to split the guiding board into parts if there are different background colors and then extract the texts separately.

5. Conclusion

Self-driving cars have taken the center stage of Computer Vision, and by drawing bounding boxes around traffic signs, there will be improvements in pose estimation, vehicle detection, surveillance, etc. Although there are abundant literatures on such topics, they mainly focus on other object detection algorithms such as Faster RNN, R-FCN or SSD, and are trained in Belgium, German or Chinese datasets. Our method of traffic sign detection and information extraction using image processing for American traffic signs with YOLO V3 provides a fast alternative way of recognizing objects in the sense of using global features instead of local features and has achieved 0.9697 and 0.9412 in precision and recall respectively within four seconds.

For future directions, there are several aspects that we want to implement. First of all, we want to improve the ability to recognize arrows and symbols because currently, we do not have a good way to recognize arrows or lane splits. Second, we want to try to do YOLO V3 Object Detection for Webcam and Video using Tensorflow, which will be closer to our goal in a sense of helping drivers to detect traffic signs on the road in real-time. Third, it would be interesting to see the results for text detection after applying the Hough transformation or RANSAC algorithm with Homography to vertically align the text characters. Moreover, it will be necessary to include more images in the training dataset to aim for more target signs and better performances.

References

[1] Alex D. Pon, Oles Andrienko, Ali Harakeh, and Steven L. Waslander. A hierarchical deep architecture and mini-batch selection method for joint traffic sign and light detection, 2018.

[2] A. Vennelakanti, S. Shreya, R. Rajendran, D. Sarkar, D. Muddegowda, and P. Hanagal. Traffic sign detection and recognition using a cnn ensemble. In *2019 IEEE International Conference on Consumer Electronics (ICCE)*, pages 1–4, 2019.

[3] Pavly Salah Zaki, Marco Magdy William, Bolis Karam Soliman, Kerolos Gamal Alexsan, Keroles Khalil, and Magdy El-Moursy. Traffic signs detection and recognition system using deep learning, 2020.

[4] Álvaro Arcos-García, Juan A. Álvarez García, and Luis M. Soria-Morillo. Evaluation of deep neural networks for traffic sign detection systems. *Neurocomputing*, 316:332 – 344, 2018.