Traffic Sign Detection Final Project Report

April 20, 2018

1 Introduction

1.1 Background

Image Understanding is a subject which has a lots of connection with practical applications, and the research of many other disciplines also be based on the data come from Image Understanding. For example, an artificial intelligence must understand what it has seen first, and then, it can make the decision based on that. Autopilot is another field which is full of applications of Image Understanding. When people want to let computer to drive the car, the computer must detect and recognize the objects, vehicles, and traffic signs first. Therefore, the research of traffic sign detection has a great practical application value. In the real situations, we expect to detect traffic signs in a continuous video, however, in this project, we will only research how to detect and recognize traffic signs in a static image.

1.2 Group Information

This project is a team project, my partner is Jian Yao. We separate this project to two parts. In my part, I am responsible for the research of how to detect traffic signs, and in his part, he will research how to recognize specific traffic signs using the detection result. In this report, I will only focus on what I did for the traffic sign detection.

2 Design

2.1 Overall Workflow

The purpose of this project is to detect single or multiple traffic signs inside an image. In general, first, we need to find the dataset. This dataset should include positive training data, negative training data and testing data. Then, we need to using positive training data and negative training data to train the classifier. Finally, after an appropriate training and reinforcing process, we should run our classifier in the testing data and analyze results.

2.2 Training Data

There are two kinds of training data: positive training data and negative training data. The positive training data should be a bundle of image which only contain traffic signs and a small part of the background. The

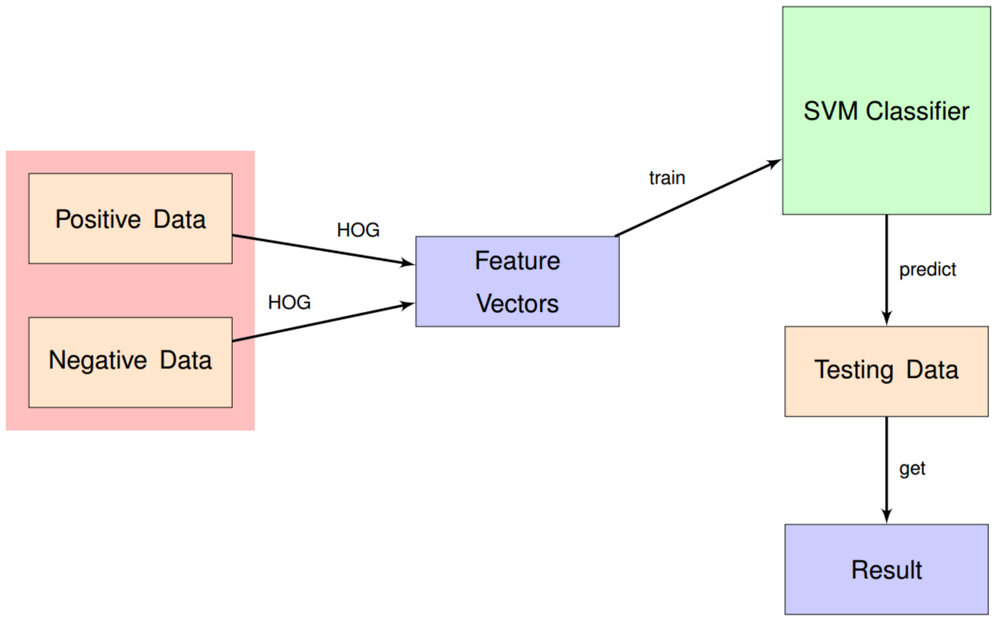


Figure 1: Overall Workflow

negative training data should only contain background information without detection object. In practice of this project, I randomly choose some block which has the same size as positive training data from a few pictures to generate my negative training data.

2.3 Feature Vector

Extract feature vector is the most important part of training data processing. I choose Histogram of Oriented Gradients(HOG) vector as the feature vector in the traffic sign detection process[1]. This idea come from. Using HOG vector as feature vector has many advantages. For example, the normalized HOG feature vector has a good adaptability in the changes of illumination and shadowing. It can accurately describe the shape information of object in different illumination situation without the effect from color space.

2.4 Classifier

The function of a classifier is to distinguish input images to different sets. In this project, we only care about two sets: traffic signs and non-traffic signs. Therefore, this is a binary classify problem. Support Vector Machine(SVM) is a nice tool to solve this problem. Giving SVM two sets of data, one marked as -1(positive training data), one marked as +1(negative training data), after training, it can effective predict the category of next coming testing data and the probability of it. Using this probability value as a score, we can easily set a threshold for our detection result to improve the detection success rate.

2.5 Reinforce Training

To improve the success rate of our traffic sign classifier, we need to reinforce our classifier after the first time training. The technology I used in this project is called hard negative mining. Specifically, I choose a subset of testing data as mining dataset, and run the current classifier on it and get the results which should be negative but current classifier gave a positive result. Then, add these hard negative data inside my negative training dataset and train SVM again[2]. After a few times of reinforce, I expect that the success rate of classifier will improve.

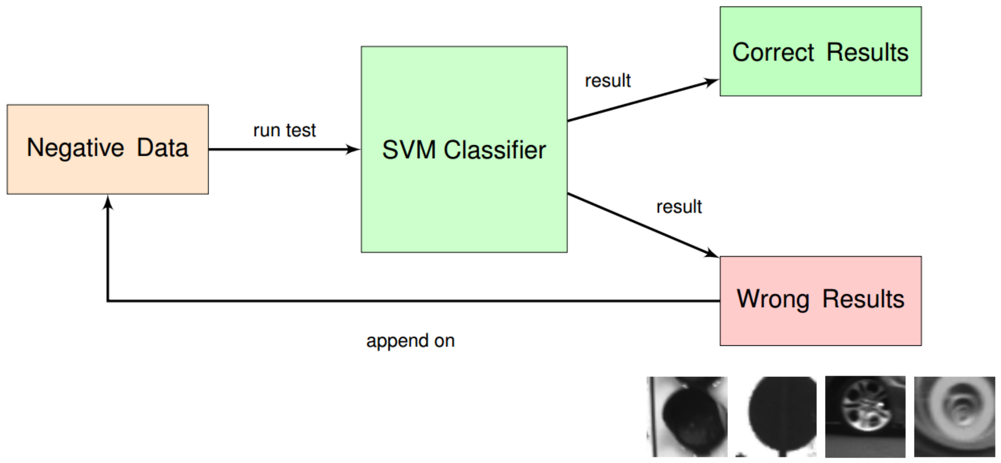


Figure 2: Hard-Negative Mining and Reinforcement Training

2.6 Testing

In the testing process, I used sliding window and pyramid technologies to adapt the different location and size of traffic signs. When the program test every single testing image, it will first generate multiple images of different scales and then divide every image in specific scale to some little windows. These windows have the same size as positive training data. The classifier will gain the HOG feature vector from these little windows and detect them to two sets(traffic signs or non-traffic signs). Moreover, washing detection results also important. For example, before washing detection results, there are always many detection results gather at same location. To washing these repeated results, I used a algorithm called non-maximum suppression[3]. This algorithms basically will treat detection results whose coverage area greater than 1 as a group of results, and choose the result has the biggest probability value as the representative of this group.

3 Implement

3.1 Files

This project is implemented by `Python` and many third party libraries, the main third party libraries includes `numpy`, `skimage`, `sklearn` and `matplotlib`.
Based on these libraries, I implemented 5 `Python` files:

`main.py`:
The main driver program, it is the entrance of whole project. I implemented `init` function to initial the running environment and `train, test, reinforce` functions to do the specific action.

`data.py`:
Defined data io and process functions, including `loadInData, calPositive` and `calNegative`. These functions used to load in data and calculate positive feature vectors and negative feature vectors.

`train.py`:
Functions about training classifier, including `trainSVM, testSVMPosData` and `minePositiveNegtiveData` functions. The function `testSVMPosData` will test a classifier using positive feature vectors to see the effect of training. However, even though using positive feature vectors to test it, the success rate cannot be 100%.

`detect.py`:
Defined functions used to run the test, including `generateTestWindow`, `generatePyramid`, `detect`, `washDetectionResult`, `showDetectionResult` and `doTest` functions. The generate functions will using `Python` generator to gennerate images of different scales and little windows, and the `washDetectionResult` funciton will wash detection results using threshold and non-maximum suppression technologies to wash detection results. The `detect` and `doTest` function will test a single image or a set of images.

`tools.py`:
In this file, I implemented some useful tool functions for project, including functions to judge whether or not two rectangle has intersection and format input data.

`default.config`:
This is the configure file of project, I defined all parameters of this project inside this file. Before run the `main.py`, user should customize their own configuration.

3.2 Commands

There are three commands defined inside `main.py`. User must use one of these commands to run the program. Usually, user will use them in this orser:

```
python main.py train && python main.py reinforce && python main.py test
```

`train`:

The command used to train classifier, it should be ran in the first order of process.

`reinforce`:

The command used to mine hard negative data and re-train classifier. Run this command will take a long time, therefore, before run this command, best modify the reinforce time parameter inside configuration file `default.config`.

`test`:

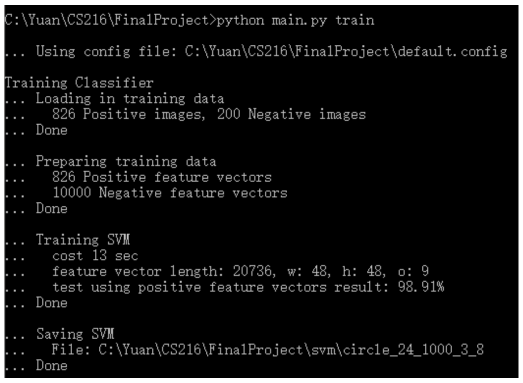
This command will run the current classifier to all images inside directory `TestingData` and give a success ratio.

Optional parameter `-c`:

This optional parameter used to let user specify the location of configure files.

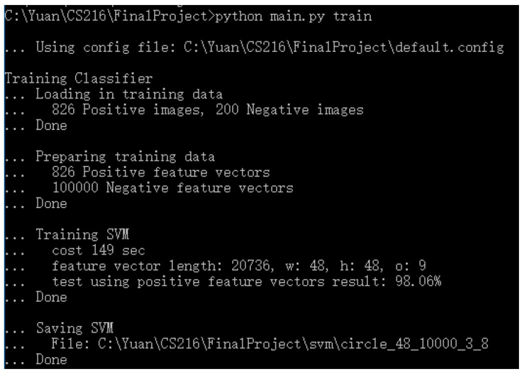
4 Results

Run `train` command to get trained classifier:



```
C:\Yuan\CS216\FinalProject>python main.py train

... Using config file: C:\Yuan\CS216\FinalProject\default.config

Training Classifier
... Loading in training data
...   826 Positive images, 200 Negative images
... Done

... Preparing training data
...   826 Positive feature vectors
...   10000 Negative feature vectors
... Done

... Training SVM
...   cost 13 sec
...   feature vector length: 20736, w: 48, h: 48, o: 9
...   test using positive feature vectors result: 98.91%
... Done

... Saving SVM
...   File: C:\Yuan\CS216\FinalProject\svm\circle_24_1000_3_8
... Done
```

(a) 10000 Negative feature vectors

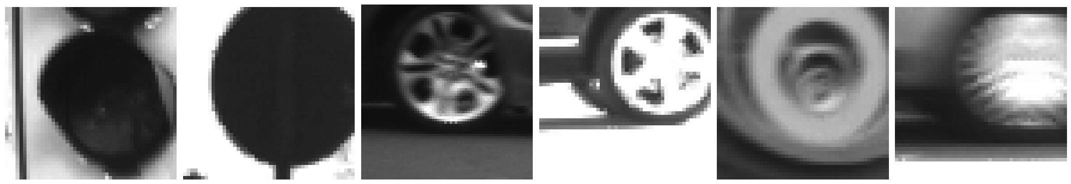


```
C:\Yuan\CS216\FinalProject>python main.py train

... Using config file: C:\Yuan\CS216\FinalProject\default.config

Training Classifier
... Loading in training data
...   826 Positive images, 200 Negative images
... Done

... Preparing training data
...   826 Positive feature vectors
...   100000 Negative feature vectors
... Done

... Training SVM
...   cost 149 sec
...   feature vector length: 20736, w: 48, h: 48, o: 9
...   test using positive feature vectors result: 98.06%
... Done

... Saving SVM
...   File: C:\Yuan\CS216\FinalProject\svm\circle_48_10000_3_8
... Done
```

(b) 100000 Negative feature vectors

Figure 3: Results of training classifiers using different configuration files

Then, run `reinforce` command and `test` command to get the final detection results.



(a) (b) (c) (d) (e) (f)

Figure 4: Hard negative results

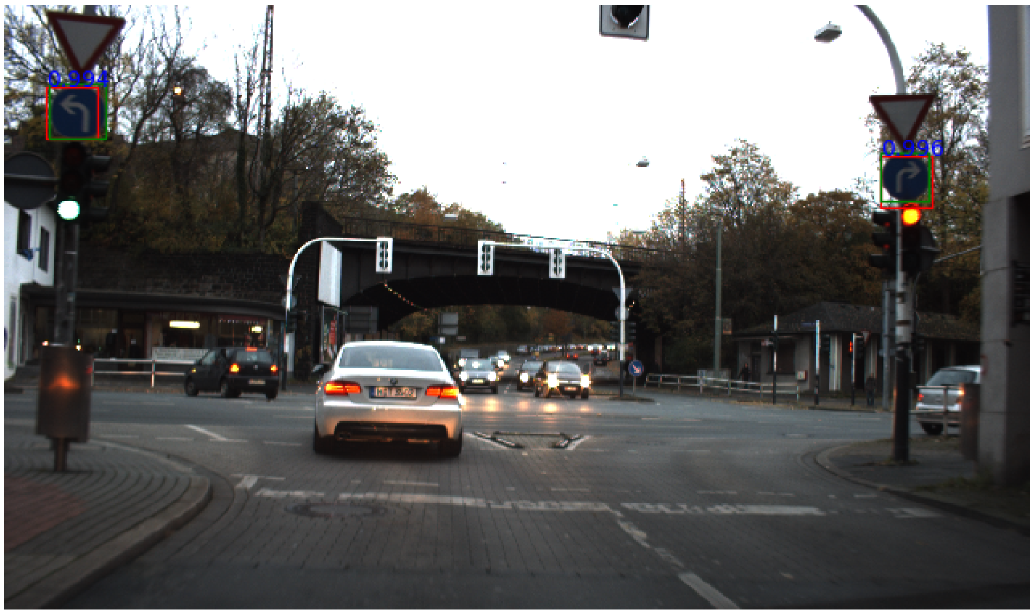


Figure 5: Detection result

5 Analysis

5.1 Detection Success Rates

After testing, I found that reinforcement training can remarkably increase the success rate of detection. Moreover, if we increase the number of orientations in HOG, the success rate of detection will also increase. However, after increase the number of orientations, the time and space overhead of the program will also increase remarkably.

5.2 Reflection

My first version of code have several defects. First, the testing process is pretty slow. Then, it's hard to use when I want to modify running parameters. I have to modify every parameter inside code. Last, the training,

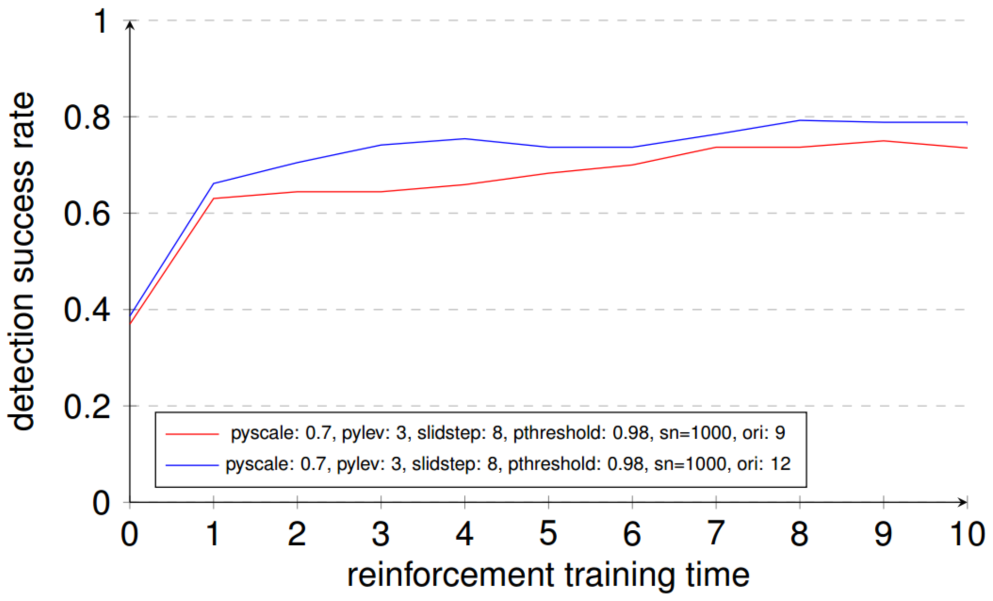


Figure 6: Detection success rate for different setting

reinforcement and testing process are written together. I cannot run specific process separately. Therefore, I improved them defects in my second version. I used multiple processes optimization to decrease the running time. I add configure file system and command system, these system let me modify parameters and run test easily.
However, there are still some problem I can fix in the future. First, maybe I can try to use GPU acceleration to accelerate my code. Then, the SVM classifier is not as good as I imagined, and the sliding window method is too slow. I think I can try to use Neural Networks as my classifier and detector in the future, even though I know nothing about it right now.

References

[1] Dalal N, Triggs B. Histograms of oriented gradients for human detection[C]//Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on. IEEE, 2005, 1: 886-893.

[2] What is Hard Negative Mining and How is it
https://www.reddit.com/r/computervision/comments/2ggc5l/what_is_hard_negative_mining_and_how_is_it/

[3] Hosang J, Benenson R, Schiele B. Learning non-maximum suppression[J]. arXiv preprint arXiv:1705.02950, 2017.