Course Project: Autonomous Driving

Jiahuang Lin linjiah6
Yaolong Wang wangyaol
Zilun Zhang zhan1381

1. Introduction

Autonomous driving is one of the major research venues these days and self-driving vehicles have expanded dramatically over the last few years. Computer vision provides a very cost effective solution to improve the safety level autonomous self-driving cars.

In this paper we are using deep neural networks to solve the object detection and object orientation estimation problems for autonomous driving. we explore two different models to perform high quality classification of road, cars and pedestrians. We apply knowledge from convolutional neural network to the domain of autonomous driving based on a region proposal approach[5]. We present and evaluate our methods in fine-tuning pre-trained neural network models for a classification task. Finally, we visualize and analyse the results of our model on test set. We believe that after experiencing with enough samples, the deep learning algorithm should be able to learn the key features of the road, cars and pedestrians and successfully classify them under complex environment.

Besides the classical object detection task, we want to further estimate the 3D orientation from the 2D images[3]. And our wish is that our system could produce a good result within a limited runtime such that it can be used in practices.

2. Dataset and Features

2.1. Dataset

The dataset that we are using is the KITTI Vision Benchmark Suite.[2] It's developed for use in mobile robotics and autonomous driving research. So it contains several novel challenging benchmarks for the tasks of stereo, optical flow, visual odometry/SLAM and 3D object detection. In our project, we mainly focus on the object detection and orientation estimation task. The corresponding benchmark [1] consists of 7481 training images and 7518 test images, comprising a total of 80,256 labeled objects (up to 15 cars and 30 pedestrians are visible per image). All images are color and saved as png.

[1] http://www.cvlibs.net/datasets/kitti/eval_object.php

For object detection of this project, we use and divide KITTI left training set to our training set and testing set since the original testing set does not have labels. The train test split proportion is 6000:1481.

For car orientation detection in this project, we create 12 folder for 12 classes (12 * 30 degrees) base on the division of object detection. We crop car batch from each image based on its label and put them into their corresponding category. There are 23,097 training car batches and 5647 testing car batches.

Object	Avg Number of Object
Car	3.8429
Pedestrians	0.5998
Cyclists	0.2175

Table 1. Average Number of Objects Per Image

2.2. Feature Extraction

2.2.1 SLIC Segmentation

Intuitively, detecting roads is similar to image segmentation problem of separating foreground and backgrounds. There are models and algorithms in the field of background subtraction such as mixture model and graph-cut segmentation etc. However, they does not fit well into road detection since normally they are used for detecting moving objects in videos from static cameras. Under some circumstances, the foreground is complex, *e.g.* trees, side parking cars and possibly some bill boards on the road area. So instead of trying to segment out road regions, classifying roads and non-roads with the involvement of learning algorithms comes in more natural. While the classification could be performed for each pixel, the approach taken in this work is to first segment each image into a set of superpixels. There are several advantages to this, the first and foremost is that it heavily reduces training time; there are approximately 46 million pixels per dataset. If we can segment the image into a set of similar looking patches, we heavily reduce the number of training examples without losing much information. In addition, the information provided by individual pixels is limited by the RGB color intensities and the pixel position. A set of pixels allows us to compute a variety of features. Su-

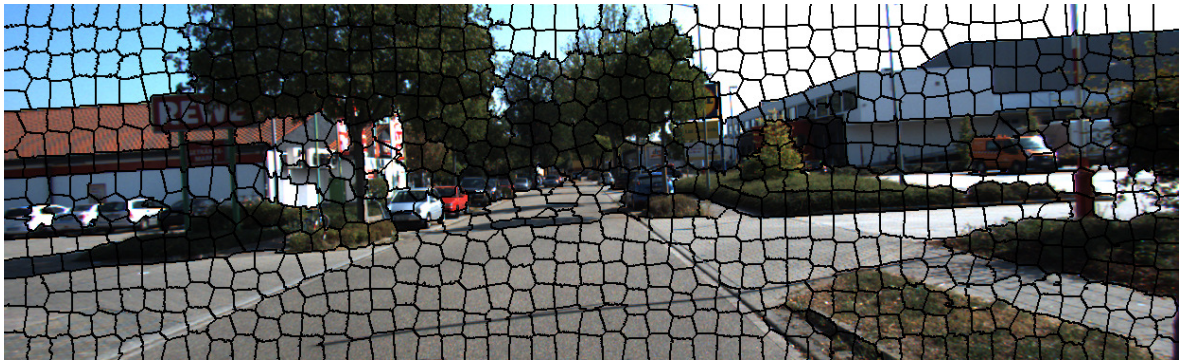

Figure 1. Example of SLIC Superpixel Segmentation Result

perpixel segmentation essentially allows us reduce the number of examples while increasing the dimensionality of each examples.

Here we use the Simple Linear Iterative Clustering (SLIC) [1] algorithm to segment the each image in approximately 800 superpixels. This algorithm doesnt require much computational power and any further knowledge towards the image. It generates superpixels by clustering pixels based on their color similarity and proximity in the image plane and it has a different distance measurement which enables compactness and regularity in the superpixel shapes, and can be used on grayscale images as well as color images. One possible outcome can be found in Figure 1.

2.2.2 Feature Extraction from SLIC

Once we have the superpixels, we only need to extract some information from each superpixel to create a single feature point.

To extract the numerical feature vector, we look at each superpixel and compute the following statistic results of that single region, including the centroid row and col index, the area and perimeter of the superpixel, the 20 bins histogram for each of the RGB channels. So each example has 64 features, encoding the geometric and color information of that superpixel. The final dataset consists of 208693 numerical feature vectors, including 166031 nonroad vectors and 42662 road vectors. To extract the superpixel feature image, we crop out each superpixel, put them into a black background and resize the patch into a 128 X 128 image. We implement this method to take advantage of the convolutional neural network, as we have changed the segmentation problem into an image classification problem. The final dataset consists of 208693 images. This size is too large so we use 2000 from each class to train. To assign a label for each superpixel, we simply check whether the centroid of each superpixel is within the road region in the ground truth image.

Figure 2. Superpixel Resize to 128x128 images

Figure 3. Extracted Numerical Features from SLIC

2.2.3 HOG Descriptor for Feature Extraction

One other task is to classify the orientation of cars. The orientation lies in degree -180 all the way to degree 180, 30 degree for each class. As mentioned on dataset part in the begining, we get 23096 new training data and 5646 new testing data that has 12 categories each for 12 * 30 degrees.

The remaining problem is to describe these orientations differently for each class so that our learning algorithm can learns the difference and thus predict the correct result. The histogram of oriented gradients (HOG) is a feature descriptor used in computer vision and image processing that counts the number of occurrences of gradient orientation in localized portions of an image. It is scale invariant. Note that different orientations of cars result in different lighting condition and edge distributions for cars, which can be recorded by HOG descriptors.

3. Methods and Evaluation

3.1. Faster R-CNN For Object Detection

Faster R-CNN is an elegant and effective solution for object detection.

For Object detection, we chose Faster-RCNN as our detector. We use the tensorflow implemented version by Charles Shang on github.[2]

[2]https://github.com/CharlesShang/TFFRCNN

It is an end-to-end object detection framework using a Region Proposal Network (RPN) and an Object Detection Network. Previous methods such as SPPnet and Fast R-CNN, reply on proposal algorithms to hypothesize object locations, which becomes a bottleneck in order to reducing the running time. In Faster R-CNN, we don't need to pre-generated several region proposals using some inexpensive features and economical inference schemes such as Selective Search. Instead, it has a RPN on top of the convolutional features, which can be used for generating detection proposals and be trained end-to-end.

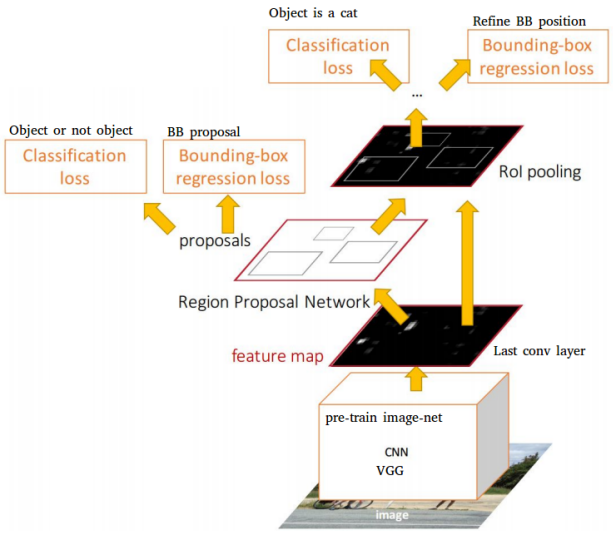

Figure 4. Faster R-CNN Architecture

A Region Proposal Network (RPN)[5] takes an image (of any size) as input and outputs a set of rectangular object proposals, each with an objectness score. Such process is modeled by a fully convolution network.[4] The region proposals are generated by the box-regression layer for predicting the coordinates of the proposal and box-classification layer for classifying whether it's an object or non-object using the features from a small network sliding over the convolutional feature map output.

Region of interest (RoI) pooling is also an important step in Faster RCNN's object detection (It was used in Fast RCNN actually), it is an operation widely used in object detection tasks using convolutional neural networks. For example, to detect multiple cars and pedestrians in a single image. Its purpose is to perform max pooling on inputs of nonuniform sizes to obtain fixed-size feature maps. [3]

In our project, we convert KITTI data set in to Pascal VOC format. We use a pre-trained VGG16 which is trained on ImageNet model and train the Faster-RCNN with 200000 iterations on KITTI. The whole training process

[3]https://blog.deepsense.ai/region-of-interest-pooling-explained/

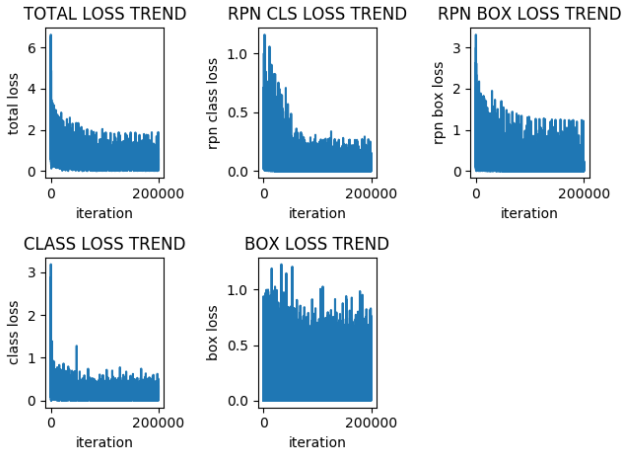

Figure 5. loss trend separate

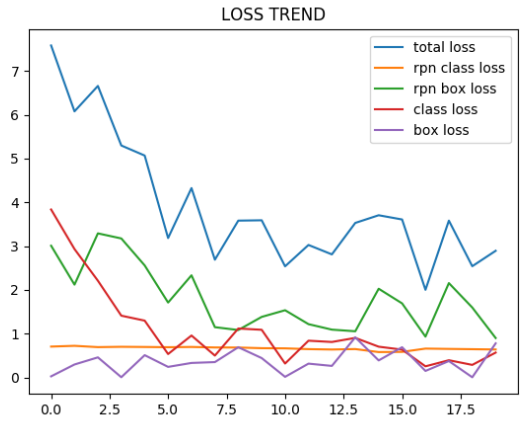

Figure 6. loss trend overall

lasted 8 hours and the loss information as shown in Figure 5 and Figure 6. The loss function in Faster-RCNN is the same multi task loss function in Fast-RCNN. For regression, it is smooth L1 norm.

There are 4 class can be detected: car, pedestrian, cyclist and dontcare. We will only focus on first three in the project. We tuning the nms and threshold for bounding box to get a better represent. Also, we modified some functions, configurations and classes to let KITTI perfectly fit into the Faster-RCNN.

Finally, the result goes pretty good. Figure 7 , Figure 8 and Figure 9 are some demonstration. The confusion matrix for classification result is on Appendix section.

3.2. SVM For orientation detection.

Support Vector Machine is a famous supervised learning model that analyze data used for classification and regression analysis. Given a set of training examples, each marked

Figure 7. A good result

Figure 8. A good result

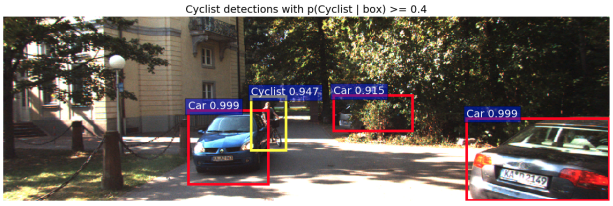

Figure 9. A good result

as belonging to one or the other of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other. We consider it a proper model to classify orientation of cars as SVMs can efficiently perform both linear and non-linear classification task by implicitly mapping their inputs into high-dimensional feature spaces whenever necessary. Note that there are 12 categories for 12 * 30 degrees (-180 to 180).

We pre-processed our data. First is resize each image in training and testing set to the median number of their shapes ((49, 81) and (49, 83) respectively). Then apply hog to each image in training set. We got a feature vector with size (1, 2592) for each image. Final result for all images is stored in a list of 12 matrices, each corresponds to a category and matrix[i, j] means ith image in this class, jth feature. The number of training batches for each class is as below:
[729, 831, 4820, 4574, 601, 866, 953, 199, 3862, 4254, 319, 1088].

The best kernel of SVM is linear in this project by testing (k-fold). We come up two ways of training classifier. The first one is training all of them together, that is training a single SVM classifier to classify all 12 classes. That will make us be able to predict a single class with most confidence in testing phase. The second one is training them separately, that is, trains 12 classifiers (Ratio between data with positive label and data with negative label is 1 : 1.1. We use all data with positive label and data with negative label is randomly chosen), and each of them in charge of judging whether this image belongs to a specific class.

For the first type of classifier, we test it as normal. For the second one, we use a different measure to test. we test it by convert it as binary vector. For instance, for a single image with true label 1, we make a one-hot vector as ground truth (1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0). Then, we preforms 12 classifiers on this image's feature vector and get a binary vector, (1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0). We count their number of same digits (11, in this case) and divided by 12 to get the accuracy for this single image. Then we use mean accuracy of all images as our test accuracy. The accuracy for the first method is about 60 % and the second method is about 84%. For the second method, we may get some predict vector with more than 1 "1" and it is really hard to determine which one should represent the result since their probability is predicted basing on different sample space. Therefore, we can only get the 2*2 confusion matrix, but we can get a 12*12 confusion matrix by using first method. The Confusion matrices are appended on Appendix section.

Figure 10. Demo for trained classifier (ground true is 88 degrees)

3.3. Neural Network for road detection

As discussed before, given a set of images, we first manipulate data by calling function SLIC, and split the image to 500 to 1000 sub segments to extract features independently. To generate label, we check the corresponding image in the file gt_image and save 'is-road' segments as 1, and 0 otherwise. Then, we build a neural network with two hidden layers, one with 256 nodes, and the other with 128 nodes. The output is 0 or 1, which corresponds the loss of the algorithm. The first two layers' activation function are 'relu' and the third one is 'sigmoid'. We have about 300 images in training set, while each image is segmented into hundreds of segments used as data. First, we trained on 96 images form training set, and tested on 96 images that are also from the training dataset but not formally used. The training accuracy was finally reached 98.9%.

The test accuracy was 92.8% on other 96 images from training dataset and 82% on testing dataset.

Then we add a 3d feature, y, the height information by adding threshold on the dataset. (e.g. if the pixel's 3d y

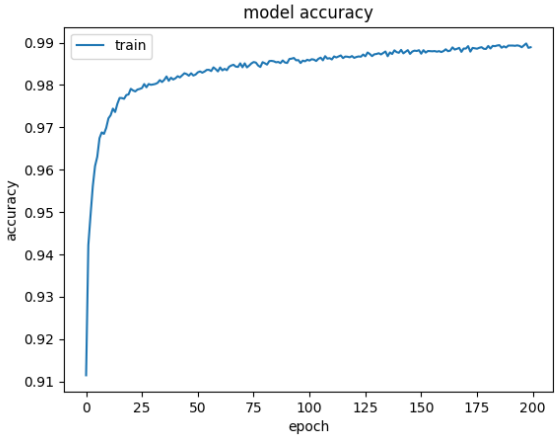

Figure 11. training accuracy with epoch=200

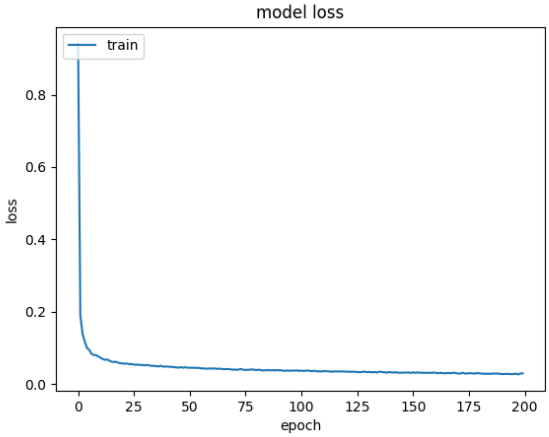

Figure 12. training loss with epoch=200

Figure 13. raw image $umm\_000001.png$

coordinate is greater than T, it's not in road)
After adding 3d feature, the output gets better.

Since we used sigmoid activation function in the last layer, the output distribute from 0 to 1. Then we gave a threshold to the output and got the precision and recall curve.

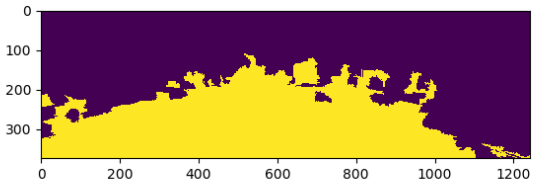

Figure 14. testing result with 2d features only

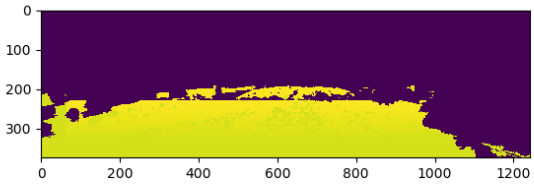

Figure 15. testing result filtered by 3d height

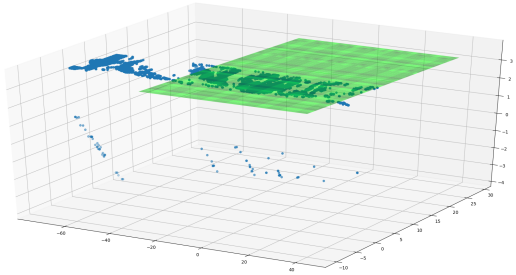

Figure 16. 3d points cloud for road with threshold

4. Future Work

In the future, we want to increase the speed of real-time detection on KITTI dataset without sacrificing the detection accuracy. The road boundary that our model produces may not be smooth and sometimes it could be discontinuous compared with ground truth, meaning there is room for im-

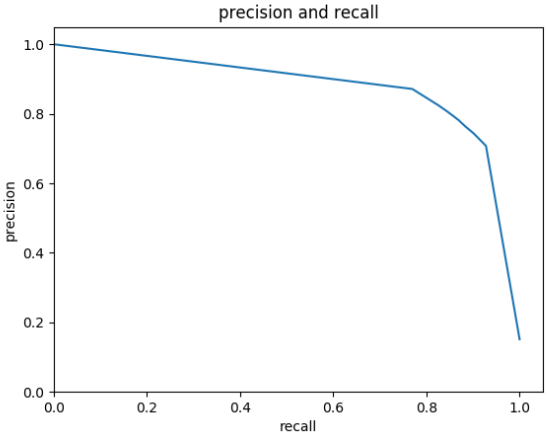

Figure 17. precision vs recall curve

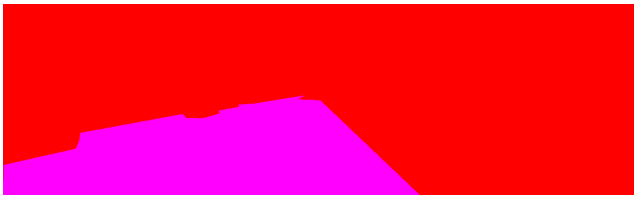

Figure 18. ground truth

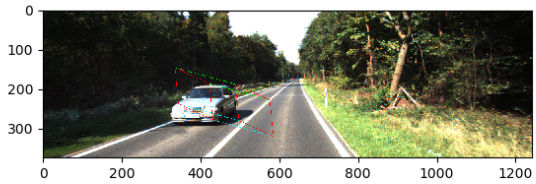

Figure 19. 3d box for car detection

provement. And notice our SVM model for classifying car orientations has accuracy around 85%, which is certainly another thing we should look into. Last but not least, we need to improve our stereo part to produce more accurate 3D coordinates and thus report back the system a more accurate result in terms of location, height and distance etc. since now it looks a bit noisy.

Also, for Faster-RCNN's pedestrian and cyclist detection, the performance is not so good when lots of pedestrian and cyclist appear at same picture. The detector sometimes will be confused by those two. Figure 15 is a good example.

Besides, as shown on figure 19. The 3d box fitting on car is not such good. Our 2d bounding box and plane fitting are almost perfect, but because of disparity, the 3d bounding box fitting for cars is not so good. We used cv2 to calculate the disparity and maybe we can improve this by changing the dataset (City Scape for instance) or tools to calculate the disparity.

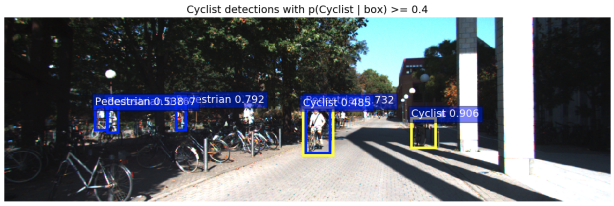

Figure 20. pedestrain and cyclist detection

This problem maybe fixed by using bootstrapping, rewight or other method to punishment wrong weight.

5. Conclusion

In conclusion, our models perform well in terms of the three major tasks: classifying roads, cars and people. Road even can achieve an accuracy around 95% under testing. We should also note that the high accuracy might be biased by the small and unbalanced dataset we use, we should test further to see more performance that our models produce under a more general circumstances. But we believe our models should be able to learn more and get the job done with high accuracy after exposing to enough training samples.

6. Appendix

6.1. Confusion Matrix For Object Detection And Classification using Faster-RCNN:

There are four class in total: We ignored Don't Care class when testing images since it is irrelevant. We treat objects in ground truth label but don't cannot find on any one of Car, Pedestrian and Cyclist's predict result as Don't care (you could see them in confusion matrix)

	Car	Pedestrian	Cyclist	Don't Care
Car	4275	6	0	1365
Pedestrian	1	504	3	342
Cyclist	2	19	182	85
Don't Care	636	176	56	0

Table 2. Confusion Matrix for Faster-RCNN Classification

6.2. Confusion Matrix For Car Orientation Classification Using Method 1

The order is [-30, -60, -90, -120, -150, -180, 30, 60, 90, 120, 150, 180] left to right, top to bottom.

607	36	31	35	23	189	33	6	33	16	9	29
4	93	18	2	1	6	5	1	4	4	1	3
5	10	116	11	2	3	8	2	45	12	2	8
1	6	4	108	8	10	1	2	8	34	0	4
6	3	1	13	143	6	2	0	5	8	3	4
204	29	40	55	50	645	39	10	25	29	15	55
27	18	32	19	13	25	646	24	19	18	4	172
0	1	3	0	1	0	4	53	6	2	0	3
13	10	43	8	5	13	7	17	191	8	4	16
8	6	10	29	5	6	4	1	3	143	6	4
1	3	3	5	1	6	0	0	2	5	32	8
28	18	25	15	8	32	236	17	28	21	12	498

Figure 21. Confusion Matrix for Method 1 Classification

6.3. Confusion Matrix For Car Orientation Classification Using Method 2

12 * 5647 count in total (12 classifier for each image (output "0" or "1") and 5647 images in total), this is what we demoed during presentation, the revised version with 12 classes (12 * 12 confusion matrix) according to what professor suggested is shown in Method 1.

	0	1
0	52678	9428
1	1855	3791

Table 3. Confusion Matrix for Method 2 Classification

References

[1] R. Achanta. Slic superpixels compared to state-of-the-art superpixel methods. *IEEE transactions on pattern analysis and machine intelligence 34.11*, 2012.

[2] P. L. Andreas Geiger and R. Urtasun;. Are we ready for autonomous driving? the kitti vision benchmark suite. *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[3] P. S. C. Geiger, Andreas; Lenz. Vision meets robotics: The kitti dataset. *International Journal of Robotics Research (IJRR) 32*, 12(1):12291235, 2013.

[4] E. S. Jonathan Long and T. Darrell;. Fully convolutional networks for semantic segmentation. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

[5] R. G. Shaoqing Ren, Kaiming He and J. Sun;. Faster r-cnn: Towards real-time object detection with region proposal networks. 2016.

There are so many source code and data (about 26GB), we only uploaded .py file to markus when the project due. If you need data of TFRCNN and CarSVM, please contact Zilun Zhang (zilun.zhang@mail.utoronto.ca) and go to https://github.com/zilunzhang/zhan1381.git to download. Thanks!