Report - Autonomous Driving: Object Detection

Zhen Li
Department of Computer Science
University of Toronto
Toronto, ON, M5S 2E4
zhen@cs.toronto.edu

Zhicong Lu
Department of Computer Science
University of Toronto
Toronto, ON, M5S 2E4
luzhc@cs.toronto.edu

1 Introduction

Object detection has always been one of the core problems of Computer Vision and Machine Learning. Based on object detection, autonomous driving has become the most challenging topic in this field which has a great impact on our life. The goal of this project is to detect the cars and pedestrians in the real road images, and locate the 2D bounding boxes of the objects correctly. Such detection techniques would make vision-based autonomous driving systems more robust and accurate, hence increasing the possibility of adopting them in the real life.

It is hard to apply classical machine learning techniques on this topic directly. Compared with the traditional object recognition database, for example, the MNIST Database [1], the real road images have more occlusions, more complicated background textures, and as a result, objects from the real road images are harder to detect.

The basic idea of this project came from The KITTI Vision Benchmark Suite [2]. It provides a well labeled training set containing 7481 color images, including several classes such as cars, pedestrians, cyclists, vans, trucks, etc. Inspired by the ranking of different methods on the KITTI website, we would try to implement those with very good performance and without using other sensor data (or dual camera information), for the reason that such methods are more accessible and have better potential to be implemented even on mobile devices. We would try to improve the training model based on the findings of the state-of-the-art to come up with our own method, and evaluate the performance of it. We expect that our method can reach a high accuracy on the test set with an optimized speed.

In this project, we focus on the performance of SVM, Regionlets [3], and CNN techniques as well as their differences. Our code is available on GitHub[1].

2 Related work

The object detection, especially cars and pedestrians detection for the autonomous driving system, has been a hot topic in computer vision for recent years. In many tasks, since the number of images and windows to evaluate is huge, we often rely on a weak classifier to get proposals for the more expensive classifier. Selective Search [4] is a successful algorithm, which emphasize recall to include all image fragments of potential relevance. It can be combined with many feature representation techniques, such as the histograms of oriented gradients (HOG) [5], and SIFT [6].

[TODO: introduce Regionlets]

Convolutional Neural Network (CNN or ConvNet) [7] is able to learn the features of the object and handle variations such as poses, viewpoints, and light conditions, with high accuracy and high efficiency. However, it doesn't perform well when occlusion occurs, which is often the case in pedestrian and cyclists detection.

[1]https://github.com/CommanderLee/ObjectDetection

Recently, the Fully Convolutional Neural Network (FCN) based methods[5], with end-to-end approach of learning model parameters and image features, further improves the performance of object detection. DenseBox [8] is a unified end-to-end FCN that directly predicts bounding boxes and object class confidences through all locations and scales of an image with great accuracy and efficiency. It also incorporates with landmark localization during multitask learning and further improves object detection accuracy. It has the best accuracy on car detection on KITTI by the time the proposal is finished. However, it has not been tested on the tasks of pedestrian or cyclists detection.

Felzenszwalb et al [9] combined a margin-sensitive approach for data-mining hard negative samples, which can be used by iteratively adding hard negative examples (false positive errors) [4].

3 Methods

3.1 Object detection system

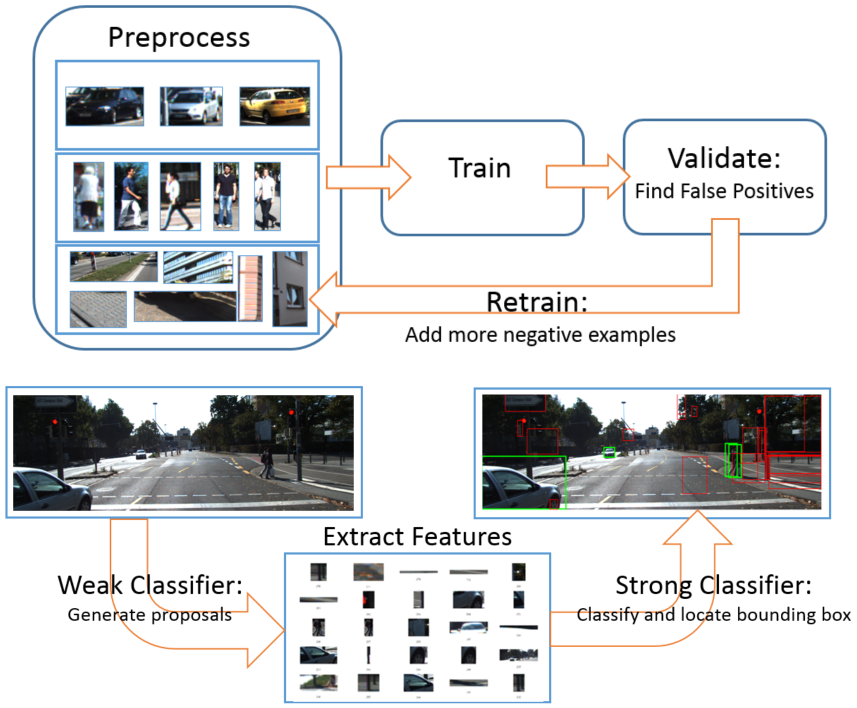

Figure 1: Top: The pipeline for training procedure. Bottom: The pipeline for testing procedure.

The whole architecture of our detection system is illustrated in Figure 1. In the training procedure, cropped images with ground truth labels are utilized to train the initial model. After validation, false positive errors are added to the negative samples, which iteratively emphasize the hard cases.

In the testing procedure, there are mainly three steps:

Step 1 Weak classifier: Generate proposals and save the bounding box.

Step 2 Extract features: Calculate feature matrix from the cropped image box.

Step 3 Strong classifier: Find out cars and pedestrians, and save the results.

We use the Selective Search [4] as the common weak classifier. For the strong classifier after it, we use SVM, Regionlets, and CNN.

3.2 Image multiplication for ground truth generation

In training, image segments of cars and pedestrians are cropped and resized to 64×64, according to the corresponding labels. Then we adjust the original image by adding a random bias to the histogram map (from -20% to $+20\%$) repeatedly. We also reverse the original image horizontally to make full use of the training set. This adjustment will not only balance the different light conditions in the original images, but also make full use of the hidden information.

3.3 Selective Search for negative samples generation

We use the Selective Search [4] to generate the negative samples. Selective Search is a segmentation algorithm that focus on recall rate and ensure its results do not stem from parameter tuning. It starts from initial regions generated by [10], and use a greedy algorithm to repeatedly merge similar segments. The similarity function is defined to combine the size similarity, by calculating the fraction of joint area, and the texture similarity, by calculating the histogram intersection.

We randomly select images from the training set, run Selective Search, and find out background segments, which are defined to have a 0% to 30% overlap with a positive sample (car or pedestrian). We generated 20000 negative samples for training.

3.4 Retrain procedure

Generating the negative samples is a important part of road object detection, since the background textures are too complicated to cluster easily. Too enhance the model's ability to solve hard problems, we add retrain procedure [9] to iteratively adding false positive errors to the negative sample set.

3.5 Regionlets with Adaboost for generic object detection

To try some different features as well as different classifiers, we implemented Regionlets [3] for generic object detection to compare with the SVM classifier. We also use Selective Search [4] to get candidate bounding boxes which may contain target object, and classify the candidate boxes with the boosted classifiers.

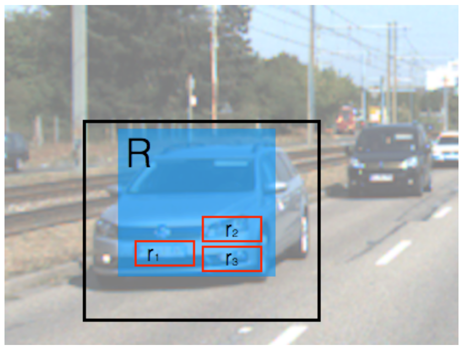

Figure 2: Illustration of Regionlets definition

3.5.1 Regionlet definition

Regionlets are defined as sub-parts of a region in the image. They are introduced to act as basic units to extract appearance features, and are organized into small groups to describe features with different degrees of deformation [3]. In this way, features extracted from small regions and a big region can work together, which can provide a good localization ability as well as tolerate more variations.

Figure 2 shows the definition of regionlets. The outer black rectangle is a candidate bounding box. The blue rectangle inside the bounding box is a feature extraction region denoted as R, which will contribute a weak classifier to the boosting classifier. Within the region R, some small sub-regions are selected and defined as a group of regionlets, as shown in Figure 2. The features of these regionlets will be aggregated to a single feature for a region R during training, and a bounding box will be represented by a number of such regions.

By introducing regionlets, it is straightforward to think that it would improve object recognition especially for occluded situations, which are common in automated driving, because the discerning features are extracted while irrelevant appearance from background is largely discarded. Besides, more regionlets in a single region R will increase the capacity to model deformations.

3.5.2 Regionlet feature extraction

According to [3], feature extraction from R takes 2 steps. First, we extract appearance features of HOG descriptors [5] from each regionlets respectively. Second, we generate the representation of R based on regionlets features. We apply max-pooling over regionlet features for the feature of region R. Denote $T(R)$ as the feature representation for region R, $T(r_j)$ as the feature extracted from the jth region r_j in R, then the operation is defined as follows:

$$T(R) = \max_{j} T(r_j) \tag{1}$$

The max-pooling happens for each feature dimension independently. For each regionlet, we first extract HOG feature for the regionlet. Then we pick a 1D feature from the same dimension of HOG feature in each regionlet and apply max-pooling to get the feature for region R. We have millions of such 1D features in a detection window and the most discriminative ones are determined through a boosting learning process.

3.5.3 Training with boosting regionlet features

We use the ensemble method of boosting to learn the discriminative regionlet groups and their configurations from a huge pool of candidate regions and regionlets.

To deal with deformation at different scales, we first build a largely over-complete pool for regions and regionlets with various positions, aspect ratios and sizes, as in [3]. We denote the 1D feature of a region relative to a bounding box as $R' = (l', t', r', b', k)$, where k denotes the k-th element of the low-level feature vector (HOG features) of the region. The region pool is spanned by $X \times Y \times W \times H \times F$, where X and Y are respectively the space of horizontal and vertical position of region R' in the detection window, W and H are the width and height of the region R', and F is the space of HOG feature. Enumerating all possible regions is impractical and not necessary. We employ a sampling process to reduce the pool size. The algorithm is the same as Algorithm 1 in [3].

After getting the region pool, we propose a set of regionlets with random positions inside each region. Although the sizes of regionlets in a region could be arbitrary in general, we restrict regionlets in a group to have the identical size because the regionlets are designed to capture the same appearance in different possible locations due to deformation[3]. The sizes of regionlets in different groups could be different. A region contains 5 regionlets in our implementation.

The final feature space used as the feature pool for boosting is spanned by $R \times C$, where R is the region feature prototype space and C is the configuration space of regionlets.

We use Gentle Adaboost [11] to train classifiers for our object detector. One boosting classifier consists of a set of selected weak classifiers. We define the weak classifier as a decision tree. Gentle Adaboost puts less weight on outlier data points, which would make the classifiers more robust.

4 Experiments

4.1 Data set

We use the KITTI data set for Object Detection [2], which contains 7481 color images with ground truth bounding box labels, including 28782 cars and 4487 pedestrians. We partition the data set to a training set contains 5237 images (70%), and a testing set contains 2244 images (30%). Then we run different algorithms on the data set and analyze the results.

4.2 SVM

4.2.1 Training

We noticed that there are far more cars than pedestrians in the KITTI data set. To balance the difference, the multiplication trick in Section 3.2 can be used to increase the number of pedestrians, which makes better use of the training set. In addition, since the KITTI benchmark only evaluate objects larger than 25 pixels (height), we ignore these small objects in the training set. After all of these augmenting and filtering strategies, we have 22576 cars and 20960 pedestrians as positive samples for training.

The SVM classifier is trained with the cropped positive samples with ground truth labels, and negative background segments generated by the Selective Search. Since SVM is a binary classifier, we trained 3 classifiers: (1) Car vs All, (2) Pedestrian vs All, and (3) Car vs Pedestrian. We use the `fitcsvm` function from the MATLAB toolbox to train out model.

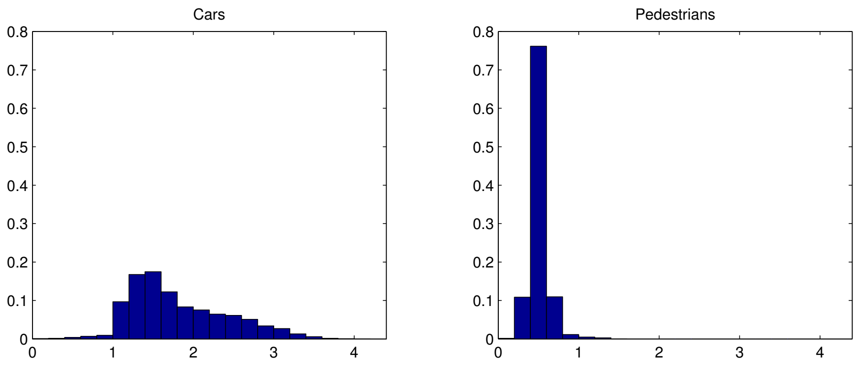

Figure 3: Normalized aspect ratio (width/height) distributions on segments of cropped cars and pedestrians.

As illustrated in Figure 1, there are 3 steps in our detector: (1) Generate proposals with Selective Search, (2) Extract HOG features, and (3) Predict with a SVM classifier. Since the last two steps are relatively expensive, we add some filters to remove the irrelevant proposals after step (1). First, we can remove the small objects (height less than 25 pixels) according to the requirement of KITTI benchmark. Second, according to the distribution of the aspect ratio of cars and pedestrians in Figure 3, we can early stop the proposal segments which are obviously not cars or pedestrians.

4.2.2 Validation

We use the k-fold ($k = 7$) cross validation to evaluate our model, as required by the project guide (60% training, 10% validation, and 30% testing). According to our description in Section 3.2, we multiply the data set by changing the contrast and brightness of the images, as well as reversing the images horizontally. Then we can compare the cross loss before and after the multiplication, using the `crossval` function from the MATLAB toolbox.

We can observe from Figure 4 that the cross loss gets lower when we multiply the data set from 13908 samples (11288 cars and 2620 pedestrians) to 43536 samples (22576 cars and 20960 pedestrians). With this trick, we can make full use of the limited data set when we lack certain classes of objects. On the other hand, this trick also helps to balance the different light conditions among the images by adding random brightness bias. The final cross loss range from 2.75% to 6.65%, which is satisfying.

In addition to the randomly selected negative samples ($N_1 = 20000$), we add retrain step to enhance the training procedure. Though the number of positive samples is limited by the training set, we can crop much more negative samples from the existing images. But we need to balance that with the training cost. To be more effective, we save the false positive image segments to files during the validation, and add these images to the training set, labeled as the background image. These are considered to be the hard samples ($N_2 = 10500$), so finally we have 30500 negative samples.

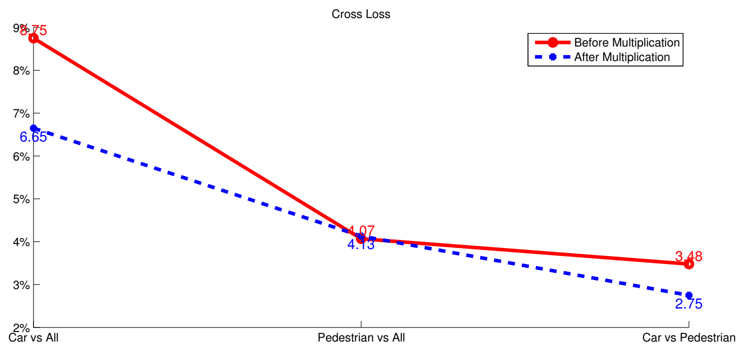

Figure 4: Cross validation ($k = 7$) loss before and after the image multiplication.

4.2.3 Testing

First we can test on the cropped testing set to evaluate the object recognition performance. Similarly, we can compare the different accuracy, precision, and recall rate we got before the image multiplication and after the image multiplication. We use the `predict` function from the MATLAB toolbox to classify with our SVM models.

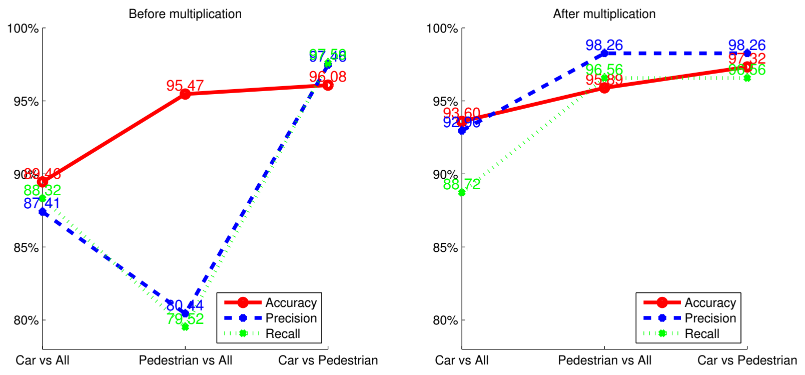

Figure 5: Accuracy, precision, and recall rate of the 3 classifiers (Car vs All, Pedestrian vs All, and Car vs Pedestrian) on the object recognition task before and after the image multiplication.

From Figure 5, we can conclude that multiplying images will also increase the accuracy, precision, and recall rate. Using 43536 positive samples and 20000 negative samples, we obtain a precision rate range from 92.96% to 98.26% and a recall rate range from 88.72% to 96.56%. It is reasonable to consider that if we collect more training data or multiply more images with existing data, we will be able to get better performance. But that will also leads to more expensive training and testing cost. Anyway, we observe a good performance of Selective Search + SVM on the object recognition task.

Table 1: Comparison of performance on object recognition task and object detection task

	Object recognition	**Object detection**
Average precision rate	96.50%	5.19%
Average recall rate	93.95%	30.67%

However, on the real test set, the precision and recall rate in Figure 6 is very low compared to the testing result on the object recognition task in Figure 5. The average performance is compared in

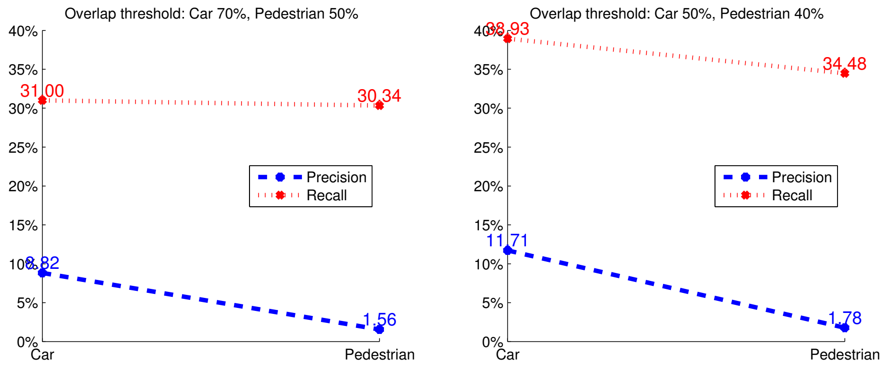

Figure 6: Precision and recall rate of the 2 classes (car and pedestrian) on the final object detection task. Left: results when the overlap threshold is 70% for cars and 50% for pedestrians. Right: results when the overlap threshold is 50% for cars and 40% for pedestrians

Table 1. We try to analyze our false results from two aspects: false positive results and false negative results.

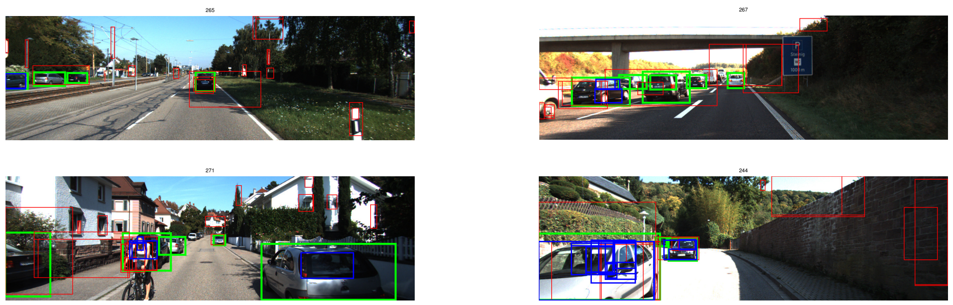

Figure 7: False Positive results. The green thick bounding box is the ground truth, the blue moderate bounding box is relevant output, and the red thin bounding box is the false positive output. In these cases, most of the cars and pedestrians are classified successfully, but some background segments, such as the road signs, are also classified as cars (or pedestrians).

On the one hand, we can analyze the false positive results from Figure 7. We notice that the detector find out most of the cars and pedestrians, while it also put extra bounding box on the background wall or the traffic sign. This problem also indicates the difficulty in training the negative samples, since they are harder to cluster, and full of different textures. And if we simply add more negative samples, then the negative output will dominate the classifier. In addition, Selective Search is a general image segmentation algorithm, so it need to be modified and trained to better fit this road object detection task.

On the other hand, there are many false negative results when the objects are small, as shown in Figure 8. We can observe from Figure 8 that the proposals are relatively big compared to the small cars. However, if we decrease the expected size of bounding box, we will miss the big car that close to the observer. An alternative way to do this is trying multiple times with different segment sizes, and then merge the boxes based on similarity. We try to do this trick, but the running time is not affordable.

4.3 Regionlets feature boosting

The training of the classifiers for regionlets features is described in Section 3.5.3. Since we use one-split decision tree as the weak classifier, the classifiers we get is binary. We again trained 3

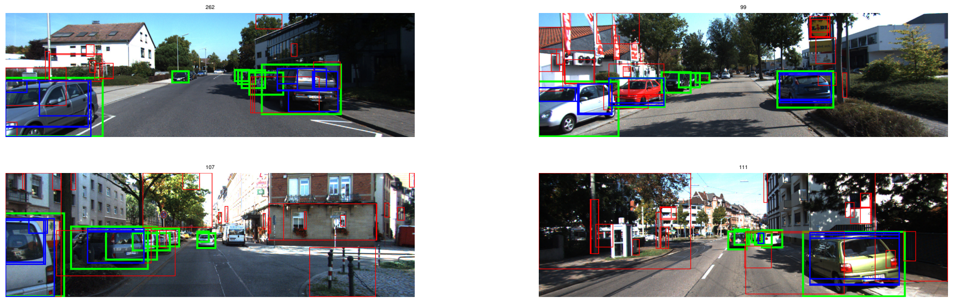

Figure 8: False Negative results. The green thick bounding box is the ground truth, the blue moderate bounding box is relevant output, and the red thin bounding box is the false positive output. In these cases, most of the cars and pedestrians that close to the observer are classified successfully, but those objects that far away from the observer are not found.

classifiers: (1) Car vs All, (2) Pedestrian vs All, and (3) Car vs Pedestrian. We used Gentle Adaboost Algorithm[] to get boosted classifiers. The training set is preprocessed in the same way as SVM.

4.4 CNN

We try to employ a binary classifier with CNN using the DeepLearnToolbox [12]. We train and test a $6C - 2S - 12C - 2S$ CNN on the MNIST Database [1], and we got a 11.13% error rate after 1 epoch, and it will drop to 4.56% after 5 epochs. However, it cannot even work with the object recognition task. We tried different parameters and structers of the CNN, but it does not perform well even on the cropped images. It is hard to tune the structure of the CNN based on the cross validation, and the sample code for MNIST should be modified carefully to fit this project.

5 Conclusion

From our experiment results in Table 1, our project performs well on the object recognition task (average $precision = 96.50\%, recall = 93.95\%$), but performs terrible on the object detection task (average $precision = 5.19\%, recall = 30.67\%$). The object recognition task used the cropped images to train and test, while the real road images are more complicated, more truncated, and of different scales.

In the future, we could run the code on GPU to speed up. There are many limitations of our work which can be improved by parallelize the code on GPU. For example, it will help to iteratively add more false positive samples to the retrain procedure in a short time. It will also be helpful if we want to combine several different classifier together and output a weighted result.

What's more, in order to get more training sets with different features, we could use some pedestrian data sets in the future, including Caltech Pedestrian Detection Benchmark [13] and Daimler Pedestrian Segmentation Benchmark Data Set [14]. The Caltech Pedestrian data set also includes sequential image flow generated from street view videos, which will contribute to the utilization of context information for autonomous driving.

Generally speaking, each of us had fun in this course project, and we learned some first hand experience by trying different techniques and solve unexpected problems. Since we are still interested in this field, we will take CSC2523 and CSC2541 respectively, continue working on this topic.

References

[1] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

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

[3] X. Wang, M. Yang, S. Zhu, and Y. Lin, "Regionlets for Generic Object Detection," in *2013 IEEE International Conference on Computer Vision*. IEEE, dec 2013, pp. 17–24.

[4] K. E. Van de Sande, J. R. Uijlings, T. Gevers, and A. W. Smeulders, "Segmentation as selective search for object recognition," in *Computer Vision (ICCV), 2011 IEEE International Conference on*. IEEE, 2011, pp. 1879–1886.

[5] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, vol. 1. IEEE, 2005, pp. 886–893.

[6] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International journal of computer vision*, vol. 60, no. 2, pp. 91–110, 2004.

[7] A. Krizhevsky, I. Sutskever, and G. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in Neural Information Processing Systems 25*, 2012.

[8] L. Huang, Y. Yang, Y. Deng, and Y. Yu, "DenseBox: Unifying Landmark Localization with End to End Object Detection," *arXiv*, sep 2015. [Online]. Available: http://arxiv.org/abs/1509.04874

[9] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan, "Object detection with discriminatively trained part-based models," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 32, no. 9, pp. 1627–1645, 2010.

[10] P. F. Felzenszwalb and D. P. Huttenlocher, "Efficient graph-based image segmentation," *International Journal of Computer Vision*, vol. 59, no. 2, pp. 167–181, 2004.

[11] A. Torralba, K. P. Murphy, and W. T. Freeman, "Sharing features: Efficient boosting procedures for multiclass object detection," in *IN CVPR*, 2004, pp. 762–769.

[12] R. B. Palm, "Prediction as a candidate for learning deep hierarchical models of data," *Technical University of Denmark*, 2012.

[13] P. Dollar, C. Wojek, B. Schiele, and P. Perona, "Pedestrian detection: A benchmark," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, jun 2009, pp. 304–311. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5206631

[14] F. Flohr and D. M. Gavrila, "Pedcut: an iterative framework for pedestrian segmentation combining shape models and multiple data cues," in *Proc. BMVC*, 2013, pp. 66–1.