Recent Tesla Model X AutoPilot Accident Analysis and Possible Solution via Traffic Sign Detection

Yufan Xue
Stanford SCPD
xueyufan@stanford.edu

Abstract

Tesla Autopilot is an advanced driver-assistance system feature offered by Tesla that has lane centering, adaptive cruise control, self-parking, ability to auto change lanes without requiring driver steering. Recent report shows a Tesla Model X was involved into a fatal crash while the autopilot mode was on.

In this paper, we analysis the car accident and come up with a possible solution to avoid car crash by using traffic sign detection. To detect traffic sign, we implement Faster R-CNN[1] and trained on GTSDB[2] Dataset. The overall model is applied on the GTSDB benchmark and achieves 86.35%, 81.35% and 81.00% AUC (area under the precision-recall curve) for Prohibitory, Danger and Mandatory signs, respectively.

1. Introduction

1.1. Problem Statement

One tesla model x was involved into a crash in California which the car hit concrete barrier while its Autopilot feature was on. Few days later, another Tesla Model X owner reproduced the same hitting scenario at the place where the accident happened. The car driver turned on Autopilot feature while driving on the freeway, while the car was getting closed to the road bifurcation, it didn't follow the correct pavement markings and direct driving toward the concrete barrier showed on Figure 1. There is a clear divider sign on Figure 1, but the car did not capture it and made the stop or slow down decision. Based on the posted experiment video on YouTube, the Tesla driver has to push the brake to stop the car manually.

Therefore our problem will be focusing on traffic sign detection, which means the system should be able to recognize traffic signs, localize it and draw the bounding box around it. Detected a traffic sign in front of the car would provide helpful perception information for the Autopilot system.

1.2. Plan

Based on the steps mentioned above, the entire project will be focusing on implementing traffic sign detection algorithm and train on GTSDB dataset. There are a lot of methods doing this job but we will stick on leveraging Faster R-CNN, implement the code from sketch, train and fine-tuning the model on GTSDB dataset.

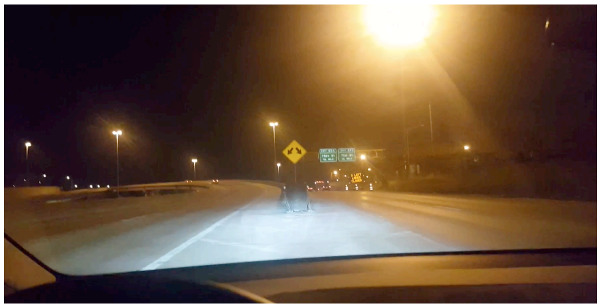
Figure 1: Divider sign on top of the concrete barrier

2. Related Works

Traffic sign detection has been an active research area since the development of computer vision, and many classical and deep learning approaches have been applied in this field. Particularly, similar to many other fields in computer vision, deep learning approach using neural network has achieved significant success in detecting traffic sign as a subclass of object classification, localization and detection. R-CNN's have proved highly effective in detecting and classifying objects in natural images, achieving mAP scores far higher than previous techniques. The R-CNN method is described in the following series paragraphs.

2.1. GTSDB dataset

The German Traffic Sign Detection Benchmark is a single-image detection assessment for researchers with interest in the field of computer vision, pattern recognition and image-based driver assistance. It is introduced on the IEEE International Joint Conference on Neural Network

2013. It has 900 images, which are divided in 600 training image and 300 evaluation images. The 900 training images are 1360 x 800 pixels in PPM format. It has a file in CSV format containing ground truth. The images contain zero to six traffic signs. The size of the traffic signs in the image very from 16x16 to 128x128

2.2. R-CNN

R-CNN[2] uses Selective Search[3] to extract ~2000 boxes that likely contain objects and evaluates the ConvNet on each of them, followed by non-maximum suppression within each class. This usually takes on order of 12 seconds per image with Tesla K80 GPU.

3. Methods

State-of-the-art object detection methods are in two major groups: region proposal based methods like Faster R-CNN, and regression based methods like YOLO[4]. In this work, we will focus on using Faster R-CNN to detect traffic signs. Faster R-CNN is composed of two modules. The first module is a Region Proposal Network (RPN), which proposes regions for the second module, Fast R-CNN detector, to inspect. In RPN module, a small network slides over convolution feature map with multiple anchors at each sliding window location. The RPN outputs a bounding boxes (region proposals) and predicted class as a Fast R-CNN module does.

In this section, we will describe in details the steps involved in training a R-CNN and inference to do object detection. From high-level perspective, RPN is to produce promising ROIs and classification network is to assign object class scores to each ROI. Therefore to train these network, we need the corresponding ground truth, which includes the coordinates of the bounding boxes around the objects and the class of those objects. The ground truth comes from free to use GTSDB dataset. The gt.txt from the dataset contains the coordinates of the bounding box and the object class label for each object present in the image.

3.1. Image Pre-Processing

Pre-processing steps are applied to an image before it got fed through the networks. These steps must be identical for both training and inference. First, we read the image and subtract each pixel by the fixed "mean vector". The mean vector (3x1, one number per color channel) is not the mean of the pixel values in current image. It is a configuration value that is identical across all training and test images.

Next, we will need to rescale the image using following logic, we choose the desired target size, which is used for the shorter image dimension, choose the max size that the image dimension is not allowed to exceed, then calculate the scale by using target size divided by the minimum number between image width and height, rescale the image using scale number calculated upon so that the rescaled image will fit the desired max size and target size.

3.2. Network Architecture

A R-CNN uses neural networks to solve two main problems. First, to identify Region of Interest – ROI in an input image that are likely to contain the foreground objects. Second, to compute the object class probability distribution of each ROI – i.e., compute the probability that the ROI contains an object of a certain class. The user who uses the network can then select the object class with the highest probability as the classification result.

A R-CNNs consist of three main types of networks: Head (Feature extractor), Region Proposal Network (RPN), Classification Network (Tail)

R-CNNs use the first few layers of a pre-trained network such as RestNet 50 to identify promising features from an input image. Using a network trained on one dataset on a different problem is possible because neural networks exhibit "transfer learning" [11]. The first few layers of the network learn to detect general features such as edges and color blobs that are good discriminating features across many different problems. The features learned by the later layers are higher level, more problem specific features. These layers can either be removed or the weights of these layers can be fine-tuned during back propagation. The first few layers that are initialized from a pre-trained network constitute the "head" network. The convolutional feature maps produces by the head network are then passed through the Region Proposal Network (RPN) which uses a series of convolutional and fully connected layers to produce promising ROIs that are likely to contain a foreground object. These promising ROIs are then used to crop out corresponding regions from the feature maps produced by the head network. The regions produced by the crop pooling are then passed through a classification network, which learns to classify the object contained in each ROIs. Figure 2 below shows the individual components of the three networks described above. We show the dimensions of the input and out of each layer of the end-to-end network, which helps understand how data is flowing through each layer. *w* and *h* represent the width and height of the input image.

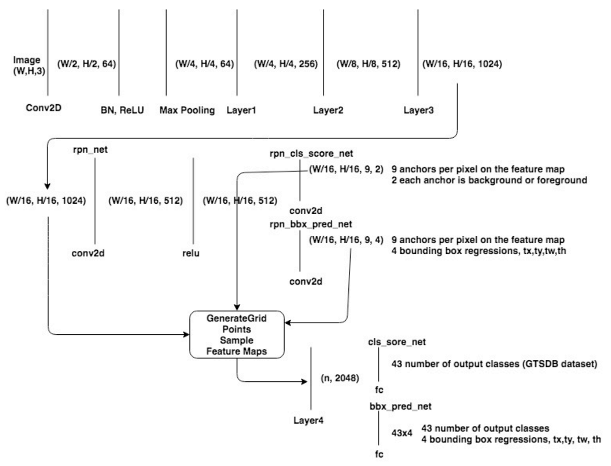

Figure 2: Faster R-CNN Architecture

3.3. Bounding Box Regression Coefficients

One of the goals of R-CNN is to produce good bounding boxes that closely fit object boundaries. R-CNN produces these bounding boxes by taking a given bounding box (defined by top-left corner, width and height) and tweaking its top left corner, width and height by applying a set of "regression coefficients". These coefficients are computed as follows. Let x, y coordinates of the top left corner of the target and original bounding box be donated by *Tx*, *Ty*, *Ox*, *Oy* respectively and width/ height of the target and original box by *Tw*, *Th*, *Ow*, *Oh* respectively. Then the regression targets (coefficients of the function that transform the original bounding box to the target box) are given as:

$$t_x = \frac{(T_x - O_x)}{O_x}, t_y = \frac{(T_y - O_y)}{O_y}, t_w = log(\frac{T_w}{P_w}), t_h = log(\frac{T_h}{P_h})$$

Given the regression coefficients and coordinates of the top left corner and the width and height of the original bounding box, the top left corner and width and height of the target box can be easily calculated.

3.4. Intersection over Union (IoU) Overlap

To measure of how close a given bounding box is to another bounding box that is independent of the units used (pixels). The IoU is calculated by the result of area of overlap divided by area of union.

3.5. Anchor Generation Layer

The anchor generation layer produces a set of bounding boxes of different sizes and aspect ratios spread all over the input image. These bounding boxes are the same for all images. Soma of these bounding boxes will contain foreground objects but most of others won't. The goal of the RPN network is to learn to identify which of these boxes are foreground boxes and background boxes and to produce target regression coefficients which will be applied to an anchor box to turns the anchor box into a better bounding box (to fits the closest foreground object more closely), Figure 3 shows the generated anchors

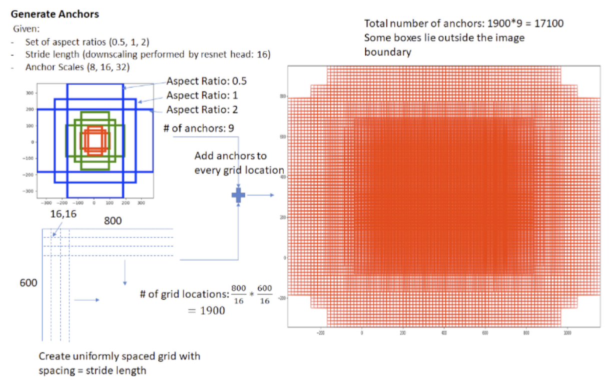
Generate Anchors
Given:
- Set of aspect ratios (0.5, 1, 2)
- Stride length (downscaling performed by resnet head: 16)
- Anchor Scales (8, 16, 32)
Aspect Ratio: 0.5
Aspect Ratio: 1
Aspect Ratio: 2
# of anchors: 9
Add anchors to every grid location
Total number of anchors: 1900*9 = 17100
Some boxes lie outside the image boundary
16,16
800
600
# of grid locations: 800/16 * 600/16 = 1900
Create uniformly spaced grid with spacing = stride length

Figure 3: RPN generated anchors

3.6. Region Proposal Layer

Object detection methods need a "region proposal system" that produces a set of sparse (like selective search [12]) set of features. The first version of the R-CNN system used the selective search to generate ROIs. In Faster R-CNN, a "sliding window" based technique is used to generate a set of dense candidate regions and then a neural network driven region proposal network is used to rank region proposals according to the probability of a region containing a foreground object. Overall the region proposal layer has to goals: First, from a list of anchors, identify background and foreground anchors. Second, modify the position, width and height of the anchors by applying regression coefficients to make the anchor more close to the foreground on the ground truth. Figure 4 shows the architecture of RPN network

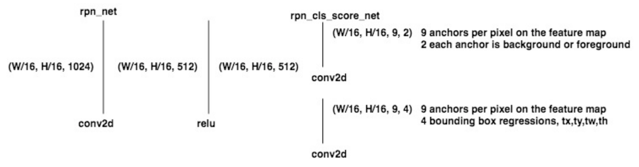
rpn_net
rpn_cls_score_net
(W/16, H/16, 9, 2)
9 anchors per pixel on the feature map
2 each anchor is background or foreground
(W/16, H/16, 1024)
(W/16, H/16, 512)
(W/16, H/16, 512)
conv2d
(W/16, H/16, 9, 4)
9 anchors per pixel on the feature map
4 bounding box regressions, tx,ty,tw,th
conv2d
relu
conv2d

Figure 4: RPN network

The region proposal layer runs feature maps produced by the head network through a convolutional layer followed by ReLU. The output of rpn_net run through two (1,1) kernel convolutional layers to produce background/foreground class scores and corresponding bounding box regression coefficients.

3.7. Proposal (Generate ROI) Layer

The proposal layer takes the anchor boxes produced by anchor generation layer and prunes the number of boxes by applying non-maximum suppression based on the foreground class scores. Then it will transform bounding box by applying the regression coefficients generated by the RPN to the corresponding anchor boxes. Shown as Figure 5 Proposal (Generate ROI) Layer

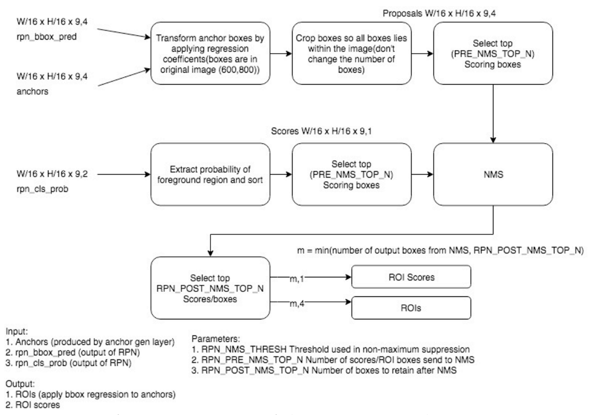
Proposals W/16 x H/16 x 9,4
W/16 x H/16 x 9,4
rpn_bbox_pred
Transform anchor boxes by applying regression coefficents(boxes are in original image (600,800))
Crop boxes so all boxes lies within the image(don't change the number of boxes)
Select top (PRE_NMS_TOP_N) Scoring boxes
W/16 x H/16 x 9,4
anchors
Scores W/16 x H/16 x 9,1
W/16 x H/16 x 9,2
rpn_cls_prob
Extract probability of foreground region and sort
Select top (PRE_NMS_TOP_N) Scoring boxes
NMS
m = min(number of output boxes from NMS, RPN_POST_NMS_TOP_N)
Select top RPN_POST_NMS_TOP_N Scores/boxes
m,1
m,4
ROI Scores
ROIs
Input:
1. Anchors (produced by anchor gen layer)
2. rpn_bbox_pred (output of RPN)
3. rpn_cls_prob (output of RPN)
Parameters:
1. RPN_NMS_THRESH Threshold used in non-maximum suppression
2. RPN_PRE_NMS_TOP_N Number of scores/ROI boxes send to NMS
3. RPN_POST_NMS_TOP_N Number of boxes to retain after NMS
Output:
1. ROIs (apply bbox regression to anchors)
2. ROI scores

Figure 5: Proposal (Generate ROI) Layer

3.8. Anchor Target Layer

The goal of the anchor target layer is to select promising anchors that can be used to train the RPN network to distinguish between foreground and background and generate good bounding box regression coefficients for the foreground boxes.

The goal of the RPN layer is to generate good bounding boxes. To do so from a set of anchor boxes, the RPN layer must learn to classify an anchor box as background or foreground and calculate the regression coefficients to modify the position, width and height of a foreground anchor box to make it fit the ground truth object more closely. The RPN loss is

$$RPNLoss = \text{Classification Loss} + \text{Bounding Box Regression Loss}$$

The ways we select anchors are, we first select the anchor boxes that lie within the image extent. Then good foreground boxes are selected by first computing the IoU overlap all anchor boxes with all ground truth boxes. Two types of boxes are marked as foreground using this overlap information. First is for each ground truth box, all foreground boxes that have the max IoU overlap with the ground truth box. Second are anchor boxes whose maximum overlap with some ground truth box exceeds a threshold. Only anchor boxes whose overlap with some ground truth box exceeds a threshold are selected as foreground boxes. This is done to avoid presenting the RPN with the useless learning task of learning the regression coefficients of boxes are too far from the best match ground truth box. Similarly, boxes whose overlap are less than a negative threshold are labeled background boxes. The input of this layer is the RPN outputs, anchor boxes and ground truth boxes. The output of this layer is the good foreground/background boxes and associated class labels and target regression coefficients.

3.9. Proposal Target Layer

The goal of the proposal target layer is to select from the list of ROIs output by the proposal layer, then perform crop pooling from the feature maps produced by the head network and passed to the tail network to calculates predicted class scores and box regression coefficients. It is important to select good proposals that have large overlap with the ground truth boxes to pass on to the classification layer. The input of this layer is ROIs produced by the proposal layer and ground truth information. The output of this layer is ROIs that meet overlap criteria and class specific target regression coefficients for the ROIs.

3.10. Crop Pooling

Proposal target layer produces promising ROIs for us to classify along with the associated class labels and regression coefficients that are used during training. The next is to extract the regions corresponding to these ROIs from the convolutional feature maps produces by the head network. The extracted feature maps are run through the tail network to produce the object class probabilities distribution and regression coefficients for each ROI. The job of Crop Pooling is to do region extraction from the feature maps

3.11. Classification Layer

The crop pooling layer takes the ROI boxes output by the proposal target layer and the convolutional feature maps output by the head network and output cropped feature maps. The feature maps are then passed through classifiers to produce class scores for each bounding box and class specific bounding box regression coefficients, like figure 6 Classification Layer

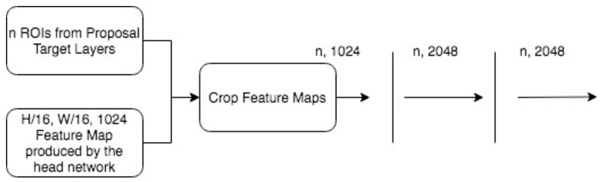
n ROIs from Proposal Target Layers
H/16, W/16, 1024 Feature Map produced by the head network
Crop Feature Maps
n, 1024
n, 2048
n, 2048

Figure 6: Proposal (Generate ROI) Layer

4. Experiments

4.1. Environment Setup

All experiments are running on an Amazon Web Service P2.8xLarge instance with 8 NVIDIA Tesla K80 GPUs. All necessary software was installed including CUDA9.0, CuDNN7.0, PyTorch 0.5.

4.2. Implementation of Faster R-CNN via PyTorch

Referring to origin paper of Faster R-CNN and few open sourced repositories on github, we are able to implement the Faster R-CNN via PyTorch framework, trained, evaluated and tested on GTSDB Dataset.

4.3. Train on GTSDB dataset

We trained on GTSDB train dataset and evaluate it on its test dataset; the training loss is attached below. Since Faster R-CNN has different losses on RPN network and Classification network, the loss is calculated and visualized separated, Figure 7 shows the training loss for rpn_cls_loss, rpn_loc_loss (RPN loss) and Figure 8 shows the roi_cls_loss, roi_loc_loss (Classification loss), Figure 9 is the confusion matrix for each class on the GTSDB train dataset.

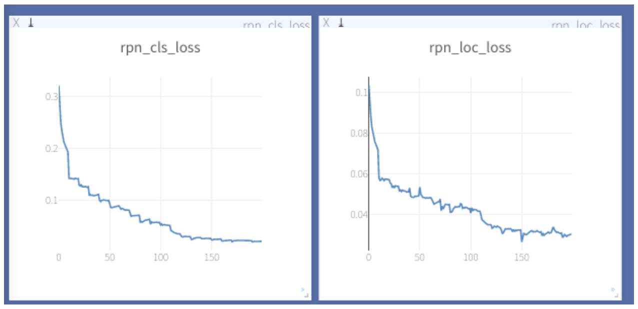

Figure 7: RPN loss (rpn_cls_loss and rpn_loc loss)

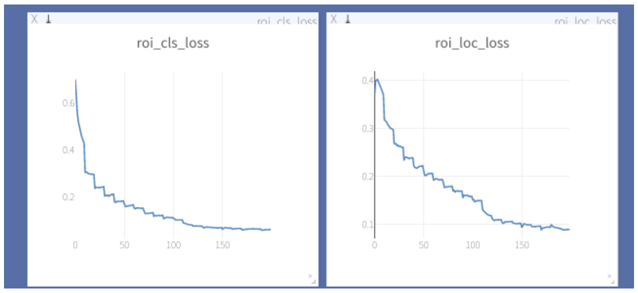

Figure 8: Classification loss (roi_cls_loss and roi_loc loss)

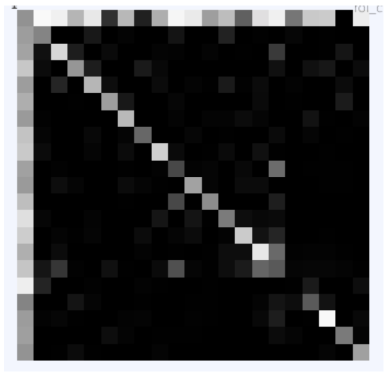
Figure 9: Classification confusion matrix

We tested on GTSDB test dataset by which we could reach a near 80% accuracy and the final detection result is shown on Figure 11

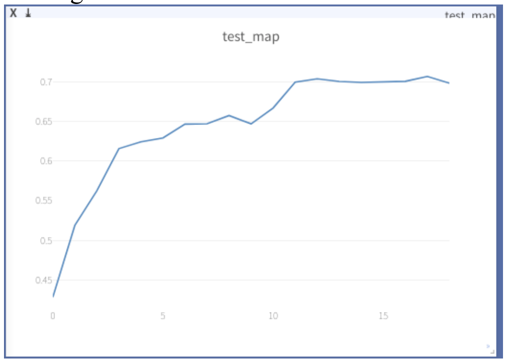

Figure 10: Test accuracy

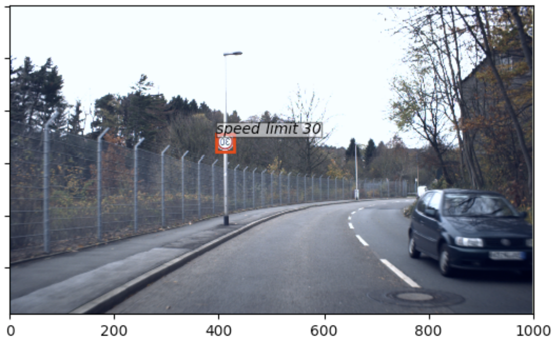

Figure 11: Traffic sign detection on GTSDB test dataset

5. Conclusion

We were able to implement the Faster R-CNN and train and evaluated on the GTSDB test dataset. However there are many problems we haven't solved yet. First is we are training the Faster R-CNN model on GTSDB dataset, this dataset has no sample for divider traffic sign which is shown on Figure 1, which means by using model trained on GTSDB dataset, we won't be able to detect the divider sign. To detect the divider sign, we will have to collect the image with the divider sign and labeling it correctly. Moreover, it would not be enough to just detect the traffic sign but also detect the distance between the car and the sign so that the Autopilot would be able to know when and how much it should slow or stop the car. To detect the distance between the car and sign, a binocular camera is needed [14].

References

[1] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In Advances in neural information processing systems, pages 91–99, 2015.

[2] Sebastian Houben and Johannes Stallkamp and Jan Salmen and Marc Schlipsing and Christian Igel. Detection of Traffic Signs in Real-World Images: The {G}erman {T}raffic {S}ign {D}etection {B}enchmark. International Joint Conference on Neural Networks

[3] Ross Girshick, Jeff Donahue, Trevor Darrell, Jitendra Malik: Rich feature hierarchies for accurate object detection and semantic segmentation, arXiv:1311.2524

[4] Jasper R. R. Uijlings, Koen E. A. van de Sande, Theo Gevers, Arnold W. M. Smeulders. Selective Search for Object Recognition. International Journal of Computer Vision, Volume 104 (2), page 154-171, 2013

[5] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 779–788, 2016.

[6] B. Huval, T. Wang, S. Tandon, J. Kiske, W. Song, J. Pazhayampallil, M. Andriluka, P. Rajpurkar, T. Migimatsu, R. Cheng-Yue, et al. An empirical evaluation of deep learning on highway driving. arXiv preprint arXiv:1504.01716, 2015.

[7] Ross Girshick, Fast R-CNN, arXiv:1504.08083

[8] B. He, R. Ai, Y. Yan, and X. Lang. Accurate and robust lane detection based on dual-view convolutional neutral network. In IV, 2016.

[9] O. Bailo, S. Lee, F. Rameau, J. S. Yoon, and I. S. Kweon. Robust road marking detection and recognition using densitybased grouping and machine learning techniques. In WACV, 2017.

[10] J. Matas, O. Chum, M. Urban, and T. Pajdla. Robust widebaseline stereo from maximally stable extremal regions. Image and Vision Computing, 22(10):761–767, 2004.

[11] T.-H. Chan, K. Jia, S. Gao, J. Lu, Z. Zeng, and Y. Ma. Pcanet: A simple deep learning baseline for image classification IEEE Transactions on Image Processing (TIP), 24(12):5017–5032, 2015.

[12] Yosinski, Jason, Jeff Clune, Yoshua Bengio, and Hod Lipson. 2014. How transferable are features in deep neural networks *arXiv.org*. November 6.

[13] Uijlings, J.R.R., van de Sande, K.E.A., Gevers, T. et al. Int J Comput Vis (2013) 104: 154. https://doi.org/10.1007/s11263-013-0620-5

[14] C. Häne, T. Sattler, M. Pollefeys, Obstacle Detection for Self-Driving Cars Using Only Monocular Cameras and Wheel Odometry IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) 2015