CIS 581 Fall 2020, Project Report

Traditional & Deep Learning based Lane Detection

- Lejun Jiang; Email: lejunj@seas.upenn.edu
- Han Shao; Email: hanshao@seas.upenn.edu
- Ying Yang; Email: yingya@seas.upenn.edu

---

1 Introduction

Lane detection is one of the critical elements that ensure the proper function of autonomous vehicles. With the fast development of computer vision, cameras have become increasingly important for assisting autonomous driving due to its high accuracy and relatively low cost. However, the images recorded by cameras cannot be used directly unless we apply lane detection algorithms to extract the useful information.

Therefore, in this project, we investigated and/or implemented several lane detection algorithms. Two are traditional methods to detect straight/curved lane and one is a deep learning method to detect multiple lanes. We will use TuSimple dataset [1] which contains 11GB compressed training data, mostly having good weather and light conditions. For traditional methods, we implemented straight lane detection through canny edge approach [2] we learned in the class combined with hough transformation and curved lane detection through sliding window algorithm [3]. For the deep learning method, we picked PiNet [4] since it is open-sourced and easy to customize to our dataset. Moreover, this model can detect multiple lanes regardless of the number of them. And it can be clipped and transferred without additional training to save space during prediction phase. Meanwhile, we compared and analyzed the pros and cons of each method and the situations that they are good/not good at.

2 Data Set

We used a dataset released by TuSimple, a self-driving truck company, in 2017. The dataset contains 7,000 one-second-long video clips of 20 frames each, including 3626 video clips of training set, and 2782 video clips for testing. This dataset is relatively comprehensive, including images from both good and medium weather conditions, captured at different daytime, at highway roads in 2-lane/3-lane/4-lane and more, and different traffic conditions. The TuSimple dataset also provides the ground truth labels for the training data which is useful for supervised deep learning methods.

3 Methods

Here we introduce our three approaches to this problem. Our goal is to take the images from the TuSimple dataset (20 frames per video clip) in, detect the lanes for each frame, and concatenate the results into a video.

3.1 Straight Lane Detection

We first tried straight lane detection with canny edge approach that we learned in class. We filtered the grayscale image by derivative of Gaussian, then compute its gradient magnitude and orientation. After comparing the gradient magnitude with its neighbors on the edge orientation, we performed non-maximum suppression. Then we used edge linking to construct tangent to edge curve and predict next points.

Since we only care about the lanes on the road, and ignore all other background information, we need to define the region of interest to avoid the influence of complex background. We manually cropped the image by focusing on the lower part that contains the edges.

After image cropping, we then apply Hough Transform to determine the two straight lines. We used OpenCV [5] module for the transformation, which finds lines through maximum voting of (p, θ) value. Among detected straight lines, we differentiated left lane and right lane boundaries by their slopes - negative slopes are for left lane boundary and positive slope are for right lane boundary, and fitted them into single left and right lane boundaries.

3.2 Curved Lane Detection

The straight lane detection can provide reasonable results for simple scenarios. However, in the real world, roads are usually curved and the straight lane detection cannot produce satisfactory results for those cases. Therefore, we introduced the curved lane detection algorithm to conquer the limitations [3].

Start with perspective transformation, we convert the input image to a bird's-eye view. We used the OpenCV [5] module for the transformation, and hand-tuned the parameters to ensure good correspondence between the views. To detect edges, we used a different method from straight lane detection's. We transformed the image to the HSL color space and applied sobel filtering to the lightness channel to extract the gradient w.r.t. the x axis. We obtained a binary map representing lane boundaries by applying a threshold to the gradient and the saturation channel [3].

With the potential lane boundaries found in the last step, we used sliding window search algorithm to filter out the noises. We started with two small sliding windows (left and right) at the bottom of the image and adjusted their x positions according to the center of detected edges inside the current window while moving them to the top of the image by decreasing y value by a certain amount. Ultimately, we found two sets of points that represent the left and right lane boundaries. Then, we applied second-order polynomial fit to the two sets of points individually, which gives us the expressions for the two lane boundaries in the bird's-eye view. We converted the result back to the camera's view by using the same parameters for the transformation (but inverted).

3.3 PiNet

Though curved lane detection has improved the performance of straight lane detection, it can detect only one lane right in front of the car. Moreover, it cannot handle some extreme cases such as highway exit or lane change. Therefore, we introduced a deep learning algorithm, Point Instance Network (PiNet) [4] which can detect multiple lanes automatically and has a great performance on extreme cases.

This method is based on the key point estimation and instance segmentation approaches. PiNet is consisted by several stacked hourglass networks which can be trained at the same time. It can detect multiple lanes and determine the number of lanes automatically since it converts lane detection problem to an instance

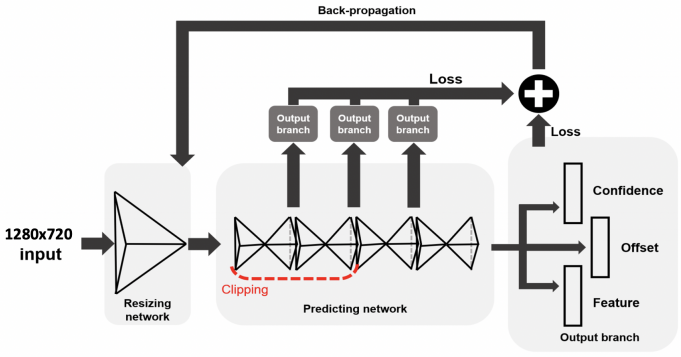


Figure 1: PiNet Architecture [4]

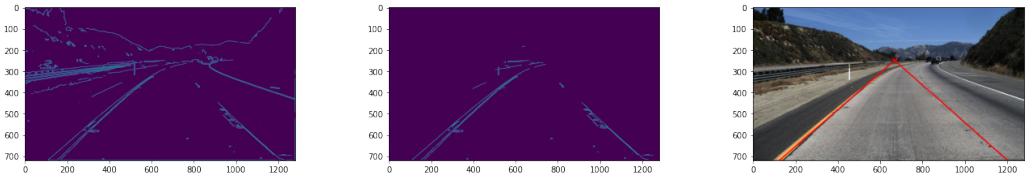
segmentation problem by clustering the predicted key points. Figure 1 shows the detailed architecture of PiNet. Since the shape of our input data is 1280x720, first, the resizing network resizes the image to 512x256. Then the compressed input is fed into 4 hourglass modules. Four output branches are applied following each hourglass module. They compute confidence, offset and feature. Then, the three values are used to calculate the loss. This model takes images as input and ground truth labels as output.

4 Experiments & Results

4.1 Straight Lane Detection

We applied standard Canny Edge Detection method with Gaussian filtering, non-maximum suppression and edge linking, and get results as in Figure 2 (a), where all edges in images were detected. After image cropping as in Figure 2 (b), we only focused on edges on the bottom part of the image. Then, we applied Hough Transformation and get result in Figure 2 (c), where two straight lines are highlighted in red.

We tested straight lane detection on 30 video clips, total 600 frames. It performs well on about 60% of the testing frames (manually check). In some cases, if the car is changing lanes, or there are multiple lines or shadows on the ground, the straight lane detection cannot work properly.



(a) edge detection (b) cropping (c) Hough Transform

Figure 2: Sample Results of Straight Lane Detection algorithm.

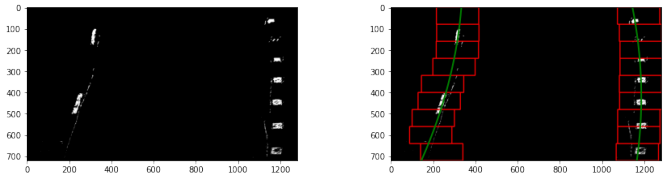
4.2 Curved Lane Detection

After fine tuning our parameters, we got results shown in Figure 3. The blue area is the detected lane that the car is currently driving in. Similarly, we tested curved lane detection on the same 600 frames. It performs well on about 76% of the testing frames (manually check). Compared to the performance of straight lane detection, curved lane detection works better in some cases such as shadows on the ground.

However, there are some scenarios it performs badly. In Figure 3 (b) car is changing lanes, (c) right line is not clear, and (d) the spaces between dashed lines are too large, the results of curved lane detection are bad. Figure 4 shows the intermediate steps of curved lane detection. White color represents detected edges, the red boxes represent the sliding windows, and the green curves represent the fitted polynomials.

(a) Good sample (b) Bad sample 1 (c) Bad sample 2 (d) Bad sample 3

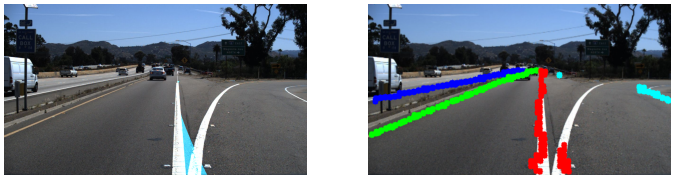
Figure 3: Sample Results of Curved Lane Detection algorithm.



(a) Results of perspective transform and sobel filtering

(b) Results of sliding window search and second-order polynomial fitting

Figure 4: Sample Intermediate Steps of Curved Lane Detection algorithm.



(a) Curved Lane Detection Result

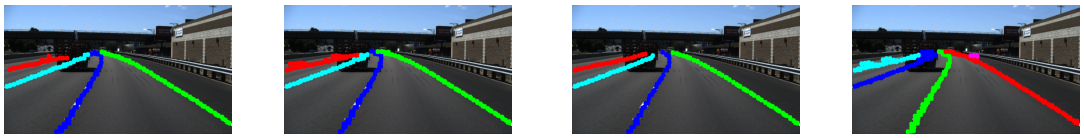
(b) PiNet Result

Figure 5: Curved Lane Detection v.s. PiNet on Extreme Case

4.3 PiNet

The code of PiNet is published on the GitHub [6]. It reads data from local machine. However, since we used Colab and the training data is about 11GB, we uploaded the training images to the Google Cloud platform and modified the code to load the data from the cloud. We trained the model with 2 different configurations. For both models, we used 4 hourglass modules. First, we used a constant learning rate, 1e-5 and trained the model for 50 epochs. The best model achieved the loss, 1.2532, at epoch 42. For the second attempt, we tried a larger learning rate, 1e-3, and decayed the learning rate to 1e-4 after 30 epochs. We trained the model on TuSimple dataset for 40 epochs and achieved the lowest loss, 0.6383, at epoch 36. Figure 5 compares the results of curved lane detection and PiNet on an extreme case. From image (a), we noticed that curved lane detection cannot handle the situation where car is changing lanes. The result of PiNet is not perfect, but much better than curved lane detection. The straight lane detection could not produce a result for this case.

Since PiNet is built by stacked hourglass blocks, it can be clipped and transferred without additional training. For this experiment, we trained the model with 4H and tested transferred 1H, 2H, 3H, 4H models. In Figure 6, we noticed all 4 models produced acceptable results. Moreover, we can see the transferred model with 3 hourglass blocks has the best result which implies that clipping and transfer learning can not only save space but also improve performance by conquering overfitting.



(a) 1H

(b) 2H

(c) 3H

(d) 4H

Figure 6: Results of transferred 1H, 2H, 3H, 4H PiNet

5 Conclusion and Future Work

From this project, we found that traditional methods only works well in the most simplified situations and would make mistakes when road condition get complicated. Deep learning significantly improves performance by detecting multiple lanes, and is able to deal with extreme cases. Having said that, the deep learning methods take a huge amount of time to train and require a lot of data, which may be a drawback in a certain number of real-world applications.

In the future, we can try different deep learning methods and evaluate each model's results. For example, in real world scenarios, real-time efficiency is key to autonomous driving, where the vehicle need to make decisions in seconds. Real-time Attention-guided Lane Detection (LaneATT) [7] assumes that using global information can help infer positions, and uses anchor-based attention mechanism to aggregates global information.

In addition, our project only explored road conditions in daytime under good to medium weather, however, nighttime driving and driving at adverse weather are often the case. Under rainy and low illumination conditions, wet roads cause light reflection and distort the appearance of lines. Vision based algorithm [8] aims to conquer poor visibility cases like vague lane patterns, perspective consequence, low visibility, shadows, incomplete occlusion, brightness and light reflection. Gradient and HSL (Hue Saturation Lightness) thresholding, perspective transform, combined with other methods we implemented in this project, such as sliding window search and cropping, are the central components of this algorithm. Therefore, our current work gives us a good start on this kind of algorithms.

References

[1] Tusimple dataset. https://github.com/TuSimple/tusimple-benchmark. (Accessed 2020-12-19).

[2] J. Canny, "A computational approach to edge detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-8, no. 6, pp. 679–698, 1986.

[3] Curved lane detection. https://www.hackster.io/kemfic/curved-lane-detection-34f771. (Accessed 2020-12-14).

[4] Y. Ko, Y. Lee, S. Azam, F. Munir, M. Jeon, and W. Pedrycz, "Key points estimation and point instance segmentation approach for lane detection," 2020.

[5] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

[6] Y. Ko, Y. Lee, S. Azam, F. Munir, M. Jeon, and W. Pedrycz. Pinet_new. https://github.com/koyeongmin/PINet_new. (Accessed 2020-12-19).

[7] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[8] M. Haque, M. Islam, K. Alam, H. Iqbal, and M. Shaik, "A computer vision based lane detection approach," *International Journal of Image, Graphics and Signal Processing*, vol. 11, pp. 27–34, 03 2019.