2021 IEEE Intelligent Transportation Systems Conference (ITSC)
Indianapolis, USA. September 19-21, 2021

A Deep Learning Approach for Lane Detection

Tesfamchael Getahun, Ali Karimoddini, and Priyantha Mudalige

2021 IEEE International Intelligent Transportation Systems Conference (ITSC) | 978-1-7281-9142-3/21/$31.00 ©2021 IEEE | DOI: 10.1109/ITSC48978.2021.9564965

***Abstract*— State-of-the-art lane detection methods use a variety of deep learning techniques for lane feature extraction and prediction, demonstrating better performance than conventional lane detectors. However, deep learning approaches are computationally demanding and often fail to meet real-time requirements of autonomous vehicles. This paper proposes a lane detection method using a light-weight convolutional neural network model as a feature extractor exploiting the potential of deep learning while meeting real-time needs. The developed model is trained with a dataset containing small image patches of dimension 16×64 pixels and a non-overlapping sliding window approach is employed to achieve fast inference. Then, the predictions are clustered and fitted with a polynomial to model the lane boundaries. The proposed method was tested on the KITTI and Caltech datasets and demonstrated an acceptable performance. We also integrated the detector into the localization and planning system of our autonomous vehicle and runs at 28 fps in a CPU on image resolution of 768×1024 meeting real-time requirements needed for self-driving cars.**

***Index Terms*— Lane Detection, Deep learning, Convolutional Neural Network, Self-driving Cars, Autonomous Vehicles**

I. Introduction

Vision-based lane detection systems often consist of three main components: preprocessing, feature extraction, and post-processing including curve fitting [1]. The feature extraction stage is considered as a critical component of a lane detector since features such as edges and color contain information about the lane [1]–[4].

Lane detection systems based on conventional image processing techniques usually use predetermined parameters such as threshold values to extract edge and color features. These categories of algorithms are vulnerable to various challenging scenarios like illumination and road texture changes which could result in low performance. Recognizing the issues with fixed parameter values, works such as [5] and [6] attempted to improve robustness by introducing adaptive thresholding and steerable filters. Similarly, the method proposed in [7] incorporates SVM classifier with HOG features to verify whether an image patch has a valid lane marking to improve the performance of the lane detector.Despite the improvements reported on the performance of conventional image processing techniques in the literature, lane detection systems using manually-tuned feature extractors produce unreliable results when tested in different settings such as in urban environments where lane markings may be covered by shadows and other vehicles.

Recent developments on lane detection systems show growing interest in deep learning based approaches such as Convolutional Neural Networks (CNN). This is due to the ability of the CNN methods to learn from example images (training data) and automatically fine tune the parameters required to extract lane features and perform classification tasks. For instance, the method proposed in [8] predicts the location of the vanishing point to enhance the lane detection process to classify the lane boundary types and colors. Similarly, in [9] a modified region based convolutional neural network (RCNN) with the popular VGG16 back-end for feature extraction is used to determine the presence of a lane marking in a small image patch. At the post-processing stage the predictions are analyzed to create the final lane boundary result which can be very complex as indicated in [10]. Semantic segmentation using an encoder-decoder architecture is also used in [11] and [12] where each pixel in the input image is grouped into lane marking or background. Semantic segmentation approaches often require an extensive post-processing effort by employing clustering methods to put the lane marking pixels into their respective lane boundaries. In an attempt to avoid or minimize the post-processing efforts, end-to-end architectures have been proposed in [13]–[15]. Long short-term memory (LSTM) deep learning architecture is also used in [16] and [17] to exploit temporal correlation between consecutive image frames.

The main issue with most of the deep learning based lane detectors is their slow speed due to the large number of convolution operations in the CNN architecture which often leads to delayed inference results. To address this challenge, we propose a lane detection system that employs a lightweight CNN model as a lane marking or feature extractor. We used a non-overlapping sliding window approach with our CNN model predicting the presence of lane marking for each image patch. The proposed method is implemented in C++ using OpenCV API for image processing in an Ubuntu-18 machine with Intel Xeon CPU (88 cores) and 32GB RAM resulting in inference speed of 28 frames per second(fps), without GPU acceleration, at a reasonable accuracy.

II. Proposed Method

Lane detection systems that are based on conventional image processing techniques rely on manually-tuned processing techniques which may not adapt to scene changes (see e.g. [3], [18]). To minimize the effect of environmental factors on the detector we developed a data driven approach

T. Getahun and A. Karimoddini are with the Department of Electrical and Computer Engineering, North Carolina Agricultural and Technical State University, Greensboro, NC 27411 USA, and P. Mudalige is with General Motors (GM).

Corresponding author: A. Karimoddini. Address: 1601 East Market Street, Department of Electrical and Computer Engineering North Carolina A&T State University Greensboro, NC, US 27411. Email: akarimod@ncat.edu (Tel: +13362853313).

(using CNNs) for feature extraction. The general block diagram representation of the proposed lane detection system is shown in Figure 1. As it can be seen in this figure, the proposed method consists of a CNN block for lane marking extraction followed by the post-processing block for analyzing the detected lane markings and performing polynomial curve fitting to represent the lane boundaries.

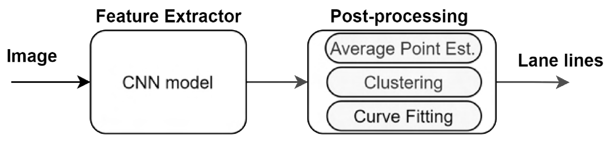

Fig. 1: Block diagram representation of the proposed lane detection framework.

A. Feature Extraction/classification

One of the main objectives of this paper is to replace handcrafted lane marking feature extraction method by an equivalent deep convolutional neural network to make the lane feature extraction invariant to various environmental factors.

Unlike instance or segmentation models which classify every pixel, our method uses a sliding window approach by considering a small image patch and performing predictions on each image patch. The size of the selected window size is small enough that in one image patch only a single segment of lane marking can be captured while sliding it across the image width as shown in Figure 2. On the other hand, the size of the window selected is big enough that a segment of lane marking will not fill the width of the window. Although overlapping of windows is possible while sliding, we intentionally avoided it to minimize the computation cost.

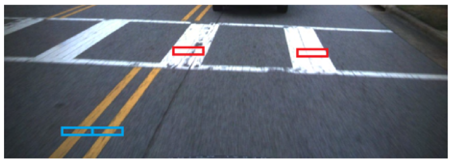

Fig. 2: The size of image patch is selected to better capture lane marking lines. Red window indicates the window filled by the zebra crossing line whereas the blue windows capture a segment of the lane line.

The architecture of our proposed feature detection method is shown in Figure 3. It consists of three convolutional layers, one maximum pooling layer, and three fully connected layers. The Rectified Linear Unit (RELU) is used as the activation function in between each layer and the Softmax function is applied to the output layer. The input to the model is an image patch of size 16×64 pixels and the output is the binary classification result. For each image frame, the output of the feature extractor block is an array in the form of

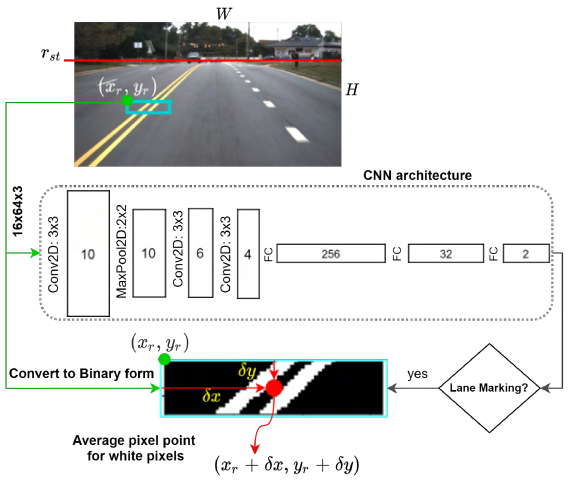

Fig. 3: The proposed CNN Model Architecture. When a lane marking is detected in the image patch, the average pixel position is calculated for the image patch, $(\delta x, \delta y)$, and then, the reference position, (x_r, y_r), is added to find the exact location in the original image.

$$P_{pers} = \begin{bmatrix} p_{0,0} & p_{0,1} & .. & p_{0,m-1} \\ p_{1,0} & p_{1,1} & .. & p_{1,m-1} \\ .. & .. & & .. \\ .. & .. & & .. \\ p_{n-1,0} & p_{n-1,1} & .. & p_{n-1,m-1} \end{bmatrix} \quad (1)$$

where each element $P_{i,j} = (x_{rj}, y_{ri}, \alpha_{mi,j}, \alpha_{bi,j})$, x_{rj} and y_{ri} represent reference pixel coordinates, $\alpha_{mi,j}$ and $\alpha_{bi,j}$ represent the prediction results for each image patch whether it is *marking* or *background*. The number of rows, n, of P_{pers} is determined by the image height, H, and the starting row, r_{st}, of the the region of interest (ROI), which can be calculated as $n = (H - r_{st})/16$. The number of columns, m, of P_{pers} depends on the image width, W, and patch/window width, which can be calculated as $m = W/64$.

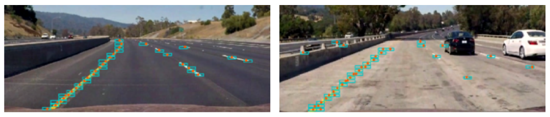

Fig. 4: Predicted lane marking points overlaid on the original images with different textures. Orange points are the average pixel point for the image patch colored in cyan.

B. Post-processing

1) Compute Average Lane Marking Point: The first stage of the post-processing block is filtering out the patches in P_{pers} with low confidence value for lane marking class. The remaining matrix elements that satisfy conditions $\alpha_m > \alpha_b$ and $\alpha_m > MIN\_THR$, are considered to contain lane marking in the image patch. Then, we compute the average pixel point, $(\delta x, \delta y)$, of the lane marking in the patch followed by converting its location with respect to the original image by adding the reference coordinate (x_r, y_r) to it as shown in Figure 3. Therefore, the coordinates of accepted predictions in the same row of P_{pers} (with the same y_r-value) can be represented as a vector $row_i = [[x_0^i, y_0^i]^T, [x_1^i, y_1^i]^T, ..., [x_{m_i-1}^i, y_{m_i-1}^i]^T]$, where (x_j^i, y_j^i) is the coordinate of the average pixel point of the jth lane marking in row i, and m_i is the number of image patches considered to have lane markings in the i^{th} row (of thickness 16 pixels).

At this stage, lane marking features are available in a 2D vector/list containing the spatial information of pixel coordinates as shown in Figure 4. For easier curve fitting and analysis, we first transform each of the vector elements in row_i from the perspective view to the bird's eye view. The vector dimensions are first adjusted to make it compatible with the 3×3 homography/transformation matrix, as $\widetilde{row}_i = [[x_0, y_0, 1]^T, [x_1, y_1, 1]^T, ..., [x_{mi-1}, y_{mi-1}, 1]^T]$. The modified vectors are put together by the $\tilde{P}_{pers}$ as follows:

$$\tilde{P}_{pers} = \begin{bmatrix} \widetilde{row}_0 \\ \widetilde{row}_1 \\ .. \\ \widetilde{row}_{n-1} \end{bmatrix} \tag{2}$$

Transforming $\tilde{P}_{pers}$ to the bird's eye view is done by multiplying each element in the $\widetilde{row}_i$ as follows

$$P_{bev} = \begin{bmatrix} H \times \widetilde{row}_0 \\ H \times \widetilde{row}_1 \\ .. \\ H \times \widetilde{row}_{n-1} \end{bmatrix} = \begin{bmatrix} \overline{row}_0 \\ \overline{row}_1 \\ .. \\ \overline{row}_{n-1} \end{bmatrix} \tag{3}$$

where P_{bev} represents the detected points in the bird's eye view and H is a transformation matrix which can be estimated from the camera calibration parameters and orientation of the camera on the vehicle [2] or from correspondence points in the original and bird's eye view images. The latter approach works well if the road is flat. Otherwise, roads with some slope can introduce an error specially on the top side of the image (further points from the camera).

2) Clustering Predicted Points: To group points that belong to the same lane boundary, we propose a clustering method that uses Euclidean distance and angle as a criteria. Unlike the K-means clustering algorithm that searches for points closer to a center point in all directions, our method searches for points around an imaginary curve. As illustrated in Figure 5b, the search starts by arbitrarily selecting a point, p_i in $\overline{row}_{n-1}$, from the last row of P_{bev}, followed by calculating the Euclidean distance and angle between p_i and all points in $\overline{row}_{n-2}$. Figure 5a shows the distance d and angle α between the points p_i in $\overline{row}_{n-1}$ and p_j in $\overline{row}_{n-2}$ which will be compared with a predefined distance and angle thresholds. If d and α are in the desired range, then point p_j is considered to belong to the same cluster as p_i. The process continues for points in the next rows until the top row, $\overline{row}_0$, is reached as shown in Figure 5c. This process is detailed in Algorithm 1.

Once the clusters are found, we refine the results as there might exist some overlapping clusters depending on the number of points in P_{bev} 2D vector. To avoid these

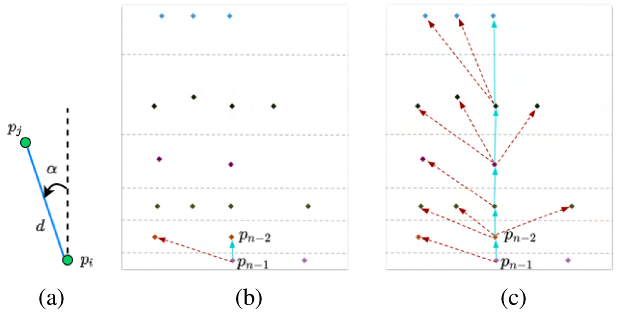

Fig. 5: (5a) The clustering criteria: Euclidean distance, d, and angle, α between points p_i and p_j for $i \neq j$, (5b) finding a point in $\overline{row}_{n-2}$ that belongs to the same lane cluster as P_{n-1}, (5c) selected points belonging to the same lane boundary shown by cyan arrow.

overlapped clusters, we simply sort the clusters based on the number of points in the clusters. Then, the cluster with the largest number of points is considered a good candidate to represent a lane boundary and it can be used as a reference to search for other lane boundaries by taking advantage of information about lane width, L_w, and assuming that lane lines are parallel.Other clusters representing the remaining lane boundaries are searched by defining a search region $\pm$ L_w from the reference cluster as shown in Figure 6.

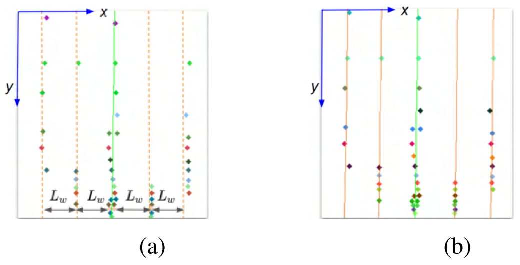

Fig. 6: (6a) The reference cluster is shown in cyan and search regions are average lane width L_w apart from the reference location. (6b) The points around the estimated regions(dotted lines) are collected to form other lane boundaries.

$$f(y_{bev}) = ay_{bev}^2 + by_{bev} + c \tag{4}$$

III. Experimental Results and Discussion

A. Experimental Setup

Our test vehicle is a GM Bolt EV which is equipped with various sensors including a GIGE color camera (by The Imaging Source) as shown in Figure 7. This camera is configured to run at 30 fps streaming an image of resolution 768×1024 pixels and is mounted on the roof along the longitudinal axis of the vehicle as shown in Figure 7. Our computing platform is a Crystal Rugged Server with Intel Xeon Scalable CPU having 88 cores and 32 GB RAM. Algorithms are implemented using Robot Operating System (ROS) framework in Ubuntu-18 OS.

Algorithm 1: Clustering Algorithm

```
Input: $P_{bev}$, points in bird's eye view
Result: $P_{cl} = []$, list of clusters
1  $\alpha_{thr} = angle\ threshold$
2  $d_{thr} = distance\ threshold$
3  for $r = 1 : n - 1$ do
4      $row = P_{bev}[n - r]$
5      for $i = 0 : length(row) - 1$ do
6          $p_i = row[i]$
7          $temp_{vector} = [p_i]$
8          $m = 1$
9          while True do
10             if $(n - r - m) < 0$ then
11                 break
12             end
13             $\alpha$, $d$, $p_j$ = getNearestPoint($p_i$,
                 $P_{bev}[n - r - m]$)
14             //Get the next point from the row above
                 the current row
15             if $\alpha \leq \alpha_{thr}$ and $d \leq d_{thr}$ then
16                 $temp_{vector} = [temp_{vector}, p_j]$
17                 $p_i = p_j$
18             end
19             $m = m + 1$
20         end
21         $P_{cl} = [P_{cl}; temp_{vector}]$
22     end
23 end
```

Fig. 7: Test vehicle - GM Chevy Bolt EV equipped with a color camera mounted on the roof of the vehicle and other sensors.

B. Training Data

To train the feature detector model, we collected images from various sources such as the TuSimple [19] dataset in addition to the data collected using our own test vehicle in different parts of Greensboro area. To train and validate our model, image patches of about 40,000 are prepared of which half of the image patches contain lane markings and the remaining data has no lane markings or considered as background as shown in Figure 8.

C. Performance Evaluation

To evaluate the performance of the proposed method, we considered two popular datasets: Caltech [2] and the KITTI ego-lane dataset [20]. The KITTI ego-lane dataset already has ground-truth label and performance evaluation script whereas for the Caltech dataset, we manually labeled more than 580 images for ego-lane analysis and the commonly used evaluation metrics, distance between ground-truth and prediction lane boundaries, are used [20], [21]. Each lane boundary in the ground-truth label contains up to 10 discrete points in the perspective view to represent the lane boundaries, $P_{grt} = \{(x_0, y_0), ..., (x_k, y_k)\}$, where $k \leq 10$. To compare the output of our lane detector with the ground-truth data, we convert the estimated polynomial in (4) to the perspective view and compare it with the ground-truth, which is also in the perspective view. This evaluation process is shown in Figure 9. Mathematically, we first discretize the estimated polynomial in (4) by finding points $p_{bev} = [x_{bev} = f(y_{bev}), y_{bev}, 1]^T$. We then transform each sample point to the perspective view using the inverse transformation matrix, H^{-1}, using $\hat{p}_{prd} = H^{-1} * p_{bev}$. Then, the polynomial representing estimated lane boundary in the perspective view is reconstructed from a group of $\hat{p}_{prd}$ points, given by $f'(y_{prd}) = a'y_{prd}^2 + b'y_{prd} + c'$. The error at a sample point is computed as the difference between the ground-truth and the polynomial output along the horizontal axis (row) as:

$$Error = |x_{grt} - x_{prd}| \tag{5}$$

where $(x_{grt}, y_{grt}) \in P_{grt}$ and (x_{prd}, y_{prd}) is a sample point evaluated from $f'(y_{prd})$, i.e. $x_{prd} = f'(y_{grt}) = a'y_{grt}^2 + b'y_{grt} + c'$ and $y_{grt} = y_{prd}$.

For a lane boundary to be considered as correctly detected or true positive (TP), more than 70% of the sample points need to have an *error* less than a defined threshold (in our case 15 pixels, which is approximately the same as the thickness of double yellow lines). Otherwise, the prediction is considered incorrect or false negative (FN). Estimates of lane lines by the lane detector without a matching ground-truth label are counted as false positives (FP).

Sample output of our method and ground-truth label points are shown in Figure 10, where the left image indicates correct detection as the predicted points are within the threshold whereas in the right side image, the right lane boundary is considered as FN since more than 60% of the predicted points have error higher than the threshold. The evaluation results summary for the Caltech dataset ego-lane detection is shown in Table I. Sample outputs of the developed method when applied to the Caltech dataset are shown in Figure 11.

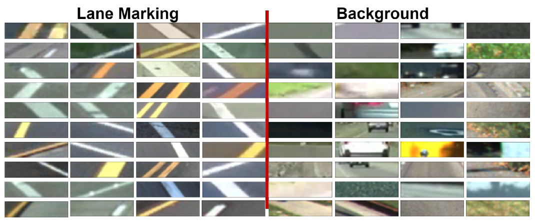

Fig. 8: Image patches containing lane markings and without lane markings/background are used to train the developed model.

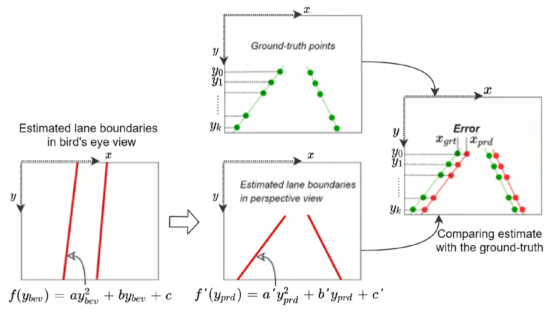
Ground-truth points
x
y
y_0
y_1
y_k
Estimated lane boundaries in bird's eye view
Estimated lane boundaries in perspective view
Error
x_{grt}
x_{prd}
Comparing estimate with the ground-truth
$f(y_{bev}) = ay_{bev}^2 + by_{bev} + c$
$f'(y_{prd}) = a'y_{prd}^2 + b'y_{prd} + c'$

Fig. 9: Performance evaluation in the perspective view. Estimated lane boundaries are first transformed from bird's eye view to perspective view for comparison with ground-truth at discrete points.

TABLE I: Evaluation of the developed method on the Caltech dataset

Dataset	#frames	TP	FN	FP
Cordova1	250	0.9254	0.0746	0.0080
Washington1	337	0.9298	0.0701	0.0267

The results show the robustness of the developed method under strong shadow conditions.

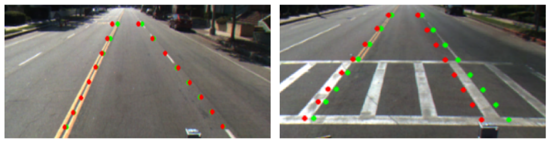

Fig. 10: Green dots indicate ground-truth labels and red dots are predictions. In the left image, the lane boundary predictions are counted as TP whereas in the right image, right lane boundary prediction is counted as FN as the majority of the sample points deviate from the ground-truth beyond the threshold.

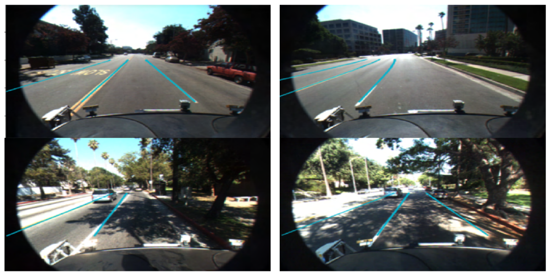

Fig. 11: Sample results of the developed method when applied to the Caltech dataset.

The developed method is also applied to the KITTI ego-lane dataset [20]. The evaluation script of KITTI considers false positive rate (FPR), false negative rate (FNR), precision (PRE), recall (REC), average precision (AP), and the maximized harmonic mean (F-measure or MaxF) parameters as a performance measures. The evaluation results for the KITTI dataset are shown in Table II. The results indicate that the proposed method has a very low FPR and FNR as well as reasonable PRE, REC, AP, and MaxF values. The AP and PRE are significantly affected by missing right side lane boundary in the dataset which is estimated by the lane detector based on left boundary detection. Nevertheless, the result obtained is comparable to the top 40 methods in the leader-boards posted in the KITTI website [20]. Sample outputs of our method when applied to KITTI dataset are shown in Figure 12. The results show robust performance of the detector under challenging conditions such as strong shadow and interference by the rails.

TABLE II: Evaluation of the developed method on the KITTI Ego-lane dataset

MaxF	AP	PRE	REC	FPR	FNR
83.72	70.73	76.98	91.75	2.44	8.25

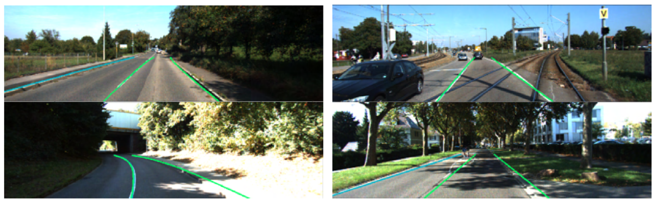

Fig. 12: Sample results of our method when applied to the KITTI dataset where the green curves indicate the ego-lane boundaries.

D. *Run-time Efficiency*

The proposed feature extraction model is trained in Tensorflow framework; optimization and inference are facilitated by the Intel's OpenVINO model optimizer and inference engine toolkit [22], and implementation is in C++ using the OpenCV library support. We tested the proposed method on three computer platforms targeting only the CPUs. We achieved an average inference speed of 28 fps with a CPU having 6 to 8 CPU cores/threads for an input image of resolution of 768×1024 pixels. Less or more CPU cores causes relatively slower performance due to lack of threads or threading overhead respectively. Although our main computer has 88 cores, maximum performance (28 fps) is achieved when the threading is restricted to only 8 cores as shown in the last row in Table III.

To evaluate the real-time performance of the proposed method, we deployed the lane detector (as a separate ROS node) in our test vehicle where its output is used as a part of the localization and planning modules. The run-time efficiency is unaffected after integration. Our test result for path planning is shown in Figure 13. The video demonstration of the lane detection algorithm (without tracking) implemented on the developed autonomous test vehicle is available at https://youtu.be/O2iV3f7KMMs.

IV. Conclusions

This paper proposed a fast, and robust lane detection system using a light-weight CNN model for lane feature

TABLE III: Run-time Efficiency Result

Platform	**#Cores**	**fps**
Dell Precision 7520, Intel Xeon E3-1505M	8	25
Intel NUC, Intel Core i7-6770HQ	8	29
*Crystal Rugged Server, Intel Xeon Scalable	88	28

**The fps is achieved using only 8 cores.*

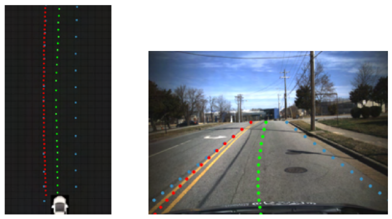
(a) (b)

Fig. 13: (13a) Top view of lane detection results overlaid with the map data: Red indicates detected lane boundary, blue shows map data and green shows the path estimated by combining map data and the lane center output of the lane detector. (13b) Perspective view of all data in (13a).

extraction. The developed lane detector takes a small patch from the input image and performs predictions in each image patch searching for lane markings followed by a clustering algorithm, which collects predictions belonging to the same lane boundary using lane geometry as a reference guide. Finally, the clustered points are fitted with a polynomial. The lane detector is also deployed in our development vehicle to confirm real-time performance when used with other software modules. The experimental results show that the developed method is robust enough for the tested driving scenarios with challenging conditions like interference with other road signs, texture changes and worn out lane markings and fulfills real-time requirements for self-driving cars. Further real-world testing is necessary to assess the robustness of this approach under other variety of driving conditions.

V. ACKNOWLEDGMENTS

The authors also thank the support from the North Carolina Department of Transportation (NCDOT) under the award numbers RP2022-16 and TCE2020-03, the National Science Foundation under award numbers 2018879 and 2000320, as well as the support from The Imaging Source, SAE International and General Motors. The contents do not necessarily reflect the official views or policies of either the NCDOT, NSF, SAE, or GM at the time of publication. This report does not constitute a standard, specification, or regulation.

REFERENCES

[1] J. C. McCall and M. M. Trivedi. Video-based lane estimation and tracking for driver assistance: survey, system, and evaluation. *IEEE Transactions on Intelligent Transportation Systems*, 7(1):20–37, 2006.

[2] M. Aly. Real time detection of lane markers in urban streets. In *2008 IEEE Intelligent Vehicles Symposium*, pages 7–12, 2008.

[3] T. Getahun, A. Karimoddini, L. H. Beni, and P. Mudalige. A robust lane marking extraction algorithm for self-driving vehicles. In *2018 15th International Conference on Control, Automation, Robotics and Vision (ICARCV)*, pages 1779–1784, 2018.

[4] M. Nieto, L. Garcia, O. Scnderos, and O. Otaegui. Fast multi-lane detection and modeling for embedded platforms. In *2018 26th European Signal Processing Conference (EUSIPCO)*, pages 1032–1036, 2018.

[5] Yeongho Son, Elijah S. Lee, and Dongsuk Kum. Robust multi-lane detection and tracking using adaptive threshold and lane classification. *Machine Vision and Applications*, 30:111–124, 2018.

[6] Joel C McCall and Mohan M Trivedi. Video-based lane estimation and tracking for driver assistance: survey, system, and evaluation. *IEEE transactions on intelligent transportation systems*, 7(1):20–37, 2006.

[7] Jingchun Piao and Hyunchul Shin. Robust hypothesis generation method using binary blob analysis for multi-lane detection. *IET Image Processing*, 11(12):1210–1218, 2017.

[8] Seokju Lee, Junsik Kim, Jae Shin Yoon, Seunghak Shin, Oleksandr Bailo, Namil Kim, Tae-Hee Lee, Hyun Seok Hong, Seung-Hoon Han, and In So Kweon. Vpgnet: Vanishing point guided network for lane and road marking detection and recognition. In *Proceedings of the IEEE international conference on computer vision*, pages 1947–1955, 2017.

[9] Yan Tian, Judith Gelernter, Xun Wang, Weigang Chen, Junxiang Gao, Yujie Zhang, and Xiaolan Li. Lane marking detection via deep convolutional neural network. *Neurocomputing*, 280:46–55, 2018.

[10] Weiwei Zhang, Hui Liu, Xuncheng Wu, Lingyun Xiao, Yubin Qian, and Zhi Fang. Lane marking detection and classification with combined deep neural network for driver assistance. *Proceedings of the Institution of Mechanical Engineers, Part D: Journal of Automobile Engineering*, 233(5):1259–1268, 2019.

[11] K. Behrendt and J. Witt. Deep learning lane marker segmentation from automatically generated labels. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 777–782, 2017.

[12] J. Kim and C. Park. End-to-end ego lane estimation based on sequential transfer learning for self-driving cars. In *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 1194–1202, 2017.

[13] Alexandru Gurghian, Tejaswi Koduri, Smita V Bailur, Kyle J Carey, and Vidya N Murali. Deeplanes: End-to-end lane position estimation using deep neural networksa. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 38–45, 2016.

[14] Z. Chen, Q. Liu, and C. Lian. Pointlanenet: Efficient end-to-end cnns for accurate real-time lane detection. In *2019 IEEE Intelligent Vehicles Symposium (IV)*, pages 2563–2568, 2019.

[15] Shriyash Chougule, Nora Koznek, Asad Ismail, Ganesh Adam, Vikram Narayan, and Matthias Schulze. Reliable multilane detection and classification by utilizing cnn as a regression network. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 0–0, 2018.

[16] Q. Zou, H. Jiang, Q. Dai, Y. Yue, L. Chen, and Q. Wang. Robust lane detection from continuous driving scenes using deep neural networks. *IEEE Transactions on Vehicular Technology*, 69(1):41–54, 2020.

[17] Ze Wang, Weiqiang Ren, and Qiang Qiu. Lanenet: Real-time lane detection networks for autonomous driving. *arXiv preprint arXiv:1807.01726*, 2018.

[18] C. Lee and J. Moon. Robust lane detection and tracking for real-time applications. *IEEE Transactions on Intelligent Transportation Systems*, 19(12):4043–4048, 2018.

[19] TuSimple. Tusimple dataset. https://github.com/TuSimple/tusimple-benchmark. Accessed - March 2021.

[20] Jannik Fritsch, Tobias Kuehnl, and Andreas Geiger. A new performance measure and evaluation benchmark for road detection algorithms. In *16th International IEEE Conference on Intelligent Transportation Systems (ITSC 2013)*, pages 1693–1700. IEEE, 2013.

[21] Ravi Kumar Satzoda and Mohan M Trivedi. On performance evaluation metrics for lane estimation. In *2014 22nd International Conference on Pattern Recognition*, pages 2625–2630. IEEE, 2014.

[22] Intel Corporation. Openvino toolkit overview. https://docs.openvinotoolkit.org/latest/index.html. Accessed - March, 2021.