Real-Time Lane Instance Segmentation Using SegNet and Image Processing

Gad Mohamed Gad[1], Ahmed Mahmoud Annaby[1], Nermin K. Negied[2], and M. Saeed Darweesh[2]

[1]Undergraduate Student, School of Engineering and Applied Sciences, Nile University, Giza 12677, Egypt
[2]Wireless Intelligent Networks Center (WINC), Nile University, Giza 12677, Egypt

Abstract—The rising interest in assistive and autonomous driving systems throughout the past decade has led to an active research community in perception and scene interpretation problems like lane detection. Traditional lane detection methods rely on specialized, hand-tailored features which is slow and prone to scalability. Recent methods that rely on deep learning and trained on pixel-wise lane segmentation have achieved better results and are able to generalize to a broad range of road and weather conditions. However, practical algorithms must be computationally inexpensive due to limited resources on vehicle-based platforms yet accurate to meet safety measures. In this approach, an encoder-decoder deep learning architecture generates binary segmentation of lanes, then the binary segmentation map is further processed to separate lanes, and a sliding window extracts each lane to produce the lane instance segmentation image. This method was validated on a tusimple data set, achieving competitive results.

Keywords—Deep Learning, Autonomous Driving, ADAS, Lane Detection, tuSimple, SegNet.

I. Introduction

Nowadays, autonomous vehicles and advanced driver-assistive systems (ADAS) perception problems like obstacle detection and lane detection are among the hot areas in computer vision. Ultimately, in each task, the target is to reach a sufficient understanding of the scene around the vehicle that is enough to make safe and efficient control decisions on behalf of the driver. What differentiates autonomous vehicles perception problems from other computer vision problems is the required method of quality in terms of accuracy and speed. From one side, safety measures demand highly accurate algorithms to ensure reliability, and from the other side, the severely limited resources on vehicle-based systems demand computationally inexpensive algorithms.

Lane detection is one of these challenging perception tasks today. There are many reasons to even consider lane detection as one of the hardest of these perception tasks. The simple appearance that doesn't distinguish lane features from other similar objects, like road marks as well as the variant lane patterns, like solid, dashed, split, and merging lanes or the variation in lane curvature from straight to curved lanes, which make handwritten rules for identifying lanes inefficient. These challenges made many current lane detection solutions focus on improving accuracy without much care to the computational cost.

This paper's remain is organized as follows: A literature survey for previous related work can be found in Section 2. The approach proposed by this work from training the neural network to postprocessing is explained in Section 3. introduced. The simulation results are demonstrated and discussed in Section 4. Finally, the paper is concluded in Section 5.

II. Literature Review

Due to variations in environments where lane detection algorithms are applied, some assumptions always made might not be valid, e.g., lanes are parallel [1] [2], lanes are distinguishable by color [3], lanes are edges [4] thus, a scalable algorithm should depend on features that are general to most environments.

Recently, deep learning-based lane detection methods, like the work done at [5], have shown outstanding performance due to their ability to extract lane features that are not predefined yet achieve state-of-the-art results on complex scenes particularly, convolutional neural network (CNN) based methods which is especially used in computer vision for feature extraction [6] [7] and semantic segmentation [8]. Due to the unique shape of lanes, there are limitations to the detection methods that can be applied. For example, lanes can't be detected using bounding boxes, pixel-wise segmentation is the most appropriate approach to localize and parameterize lanes.

Lane instance segmentation done by [9] [10] has also yielded promising results. While [11] proposed a multi-task encoder-decoder architecture consisting of a branch for lane binary segmentation and a branch for lane pixel-embeddings. Lane embeddings disentangle lanes by being trained using a clustering loss function [12] proposed earlier by the same authors. At inference time, lane embeddings are clustered using the DBSCAN clustering algorithm. Since DBSCAN has a high computational cost, the binary segmentation branch is used to mask lane embeddings to limit the space on which to apply clustering to the pixels labeled as a lane in the binary segmentation branch.

This work is inspired by the work of [11] with modifications in network architecture and postprocessing methodology to strive for a lower computational cost while preserving high accuracy.

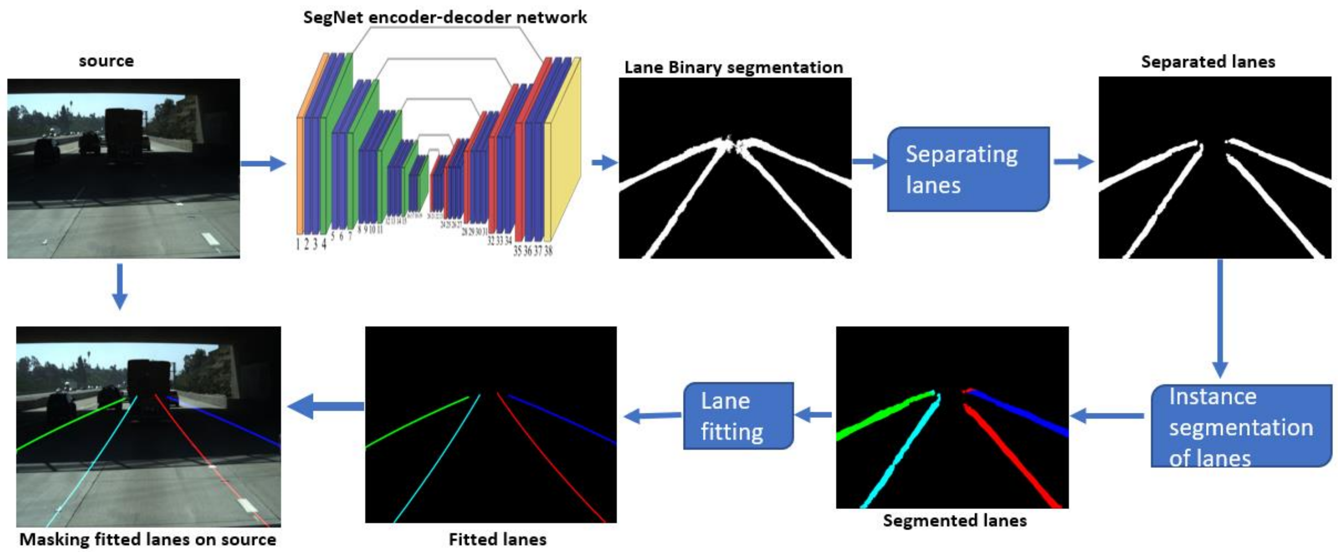

Fig. 1: Overview of the proposed lane detection pipeline

III. Methodology

A fully convolutional neural network is trained to produce a binary segmentation map, labeling each pixel as a lane pixel or non-lane pixel (background). After that, in the post-processing phase, a collection of image processing techniques is applied on the binary segmentation map to separate lanes and assign each lane pixel a lane ID. Finally, all lane pixels are fitted in a 2nd or 3rd order polynomial function to get the lane parameterization. An overview of the proposed lane detection pipeline is shown in Fig. 1.

A. *Network Architecture*

SegNet [13] is a deep convolutional encoder-decoder architecture for robust semantic pixel-wise labeling. Input size is (256, 512, 3) and output size is (256, 512, 2). The output is a two-channel map with each one representing one of the two labels: lane pixel, as 1, and non-lane pixel as 0. Table 1 shows a comparison between the number of parameters of the SegNet used in this work and the dual-decoder ENET used in [11] and other deep learning-based methods. Although ENET has far less parameters than SegNet, we chose the later based on its good results on similar tasks.

Table 1. Models analysis

Model	Parameters (M)
SCNN [18]	20.72
LaneNet(+H-net) [11]	15.98
PointLaneNet(MoblieNet) [17]	5.33
ENet-SAD [16]	0.98
KeyPointsEstimation(32x16) [15]	4.4
KeyPointsEstimation(64x32) [15]	4.39
Proposed Model	21.67

B. *Binary Segmentation*

SegNet is trained to output a lane binary segmentation map, which indicates which pixels in the input image belong to a lane, any lane, and which pixel doesn't. tuSimple is the dataset used to train the model for binary segmentation. Still, it doesn't have its labels in the form of binary segmentation maps. Still, x, y coordinates of lanes, as shown in Fig. 2 thus, we had to construct the ground truth binary segmentation map by connecting each lane's points, forming a connected line per lane.

Fortunately, the dataset marks lane points even through objects, like cars, or when lanes are dashed or faded. Likewise, the constructed line per lane also passes through objects and on partially or totally pale lanes. This resulted in the network being able to identify lanes as a single, connected line even if it's occluded or faded.

As mentioned previously, the size of the output segmentation map is (256, 512, 2). Usually, *argmax* function is applied on segmentation maps to reduce channels number to one with the highest value across all channels:

$$argmax(f(x)) \coloneqq \{x \mid \forall y : f(y) \leq f(x) \qquad (1)$$

In our case, $f(x)$ is the binary segmentation map B, and the function is applied channel-wise to output 0 or 1.

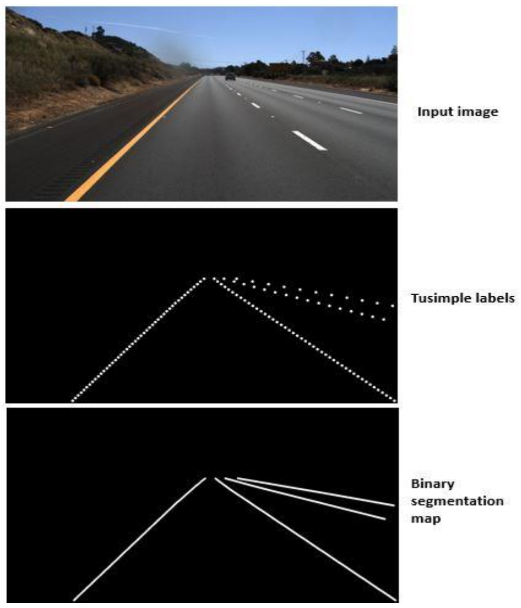

Fig. 2: tusimple dataset image, tusimple labels (points), and SegNet labels (connected lines)

However, working on the raw lane channel and dismissing the non-lane channel gives more flexibility and options on the criteria used to consider a pixel as a lane or not. As shown in Fig. 3, applying argmax resulted in interconnected lanes that are thus harder to the segment. While applying a threshold on the lane channel allows us to control lanes thickness in the binary segmentation map.

C. *Postprocessing*

This phase aims to classify each lane pixel in the binary segmentation map to one lane instance exclusively i.e., instance segmentation of lanes. The input to the postprocessing phase is a (256, 512, 1) binary segmentation maps with lane pixels labeled as 1 and non-lane pixels labeled as 0.

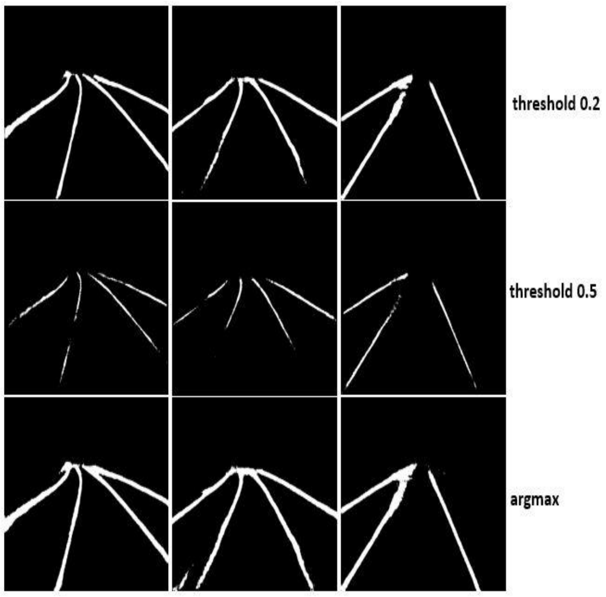

Fig. 3: Top row: using threshold 0.2 on lane channel. Middle row: using threshold 0.5 on lane channel. Bottom row: using argmax on both channels to reduce them to one channel.

The postprocessing phase is divided into 3 stages:

- Separating lanes to make segmentation easier and more accurate
- Applying a sliding window to extract lanes one by one
- Fitting each segmented lane in a polynomial function

Following are the stages explained in detail.

i. *Separating Lanes*

Before applying a sliding window to extract each lane, it is needed to ensure that lanes are separated from each other. Therefore, thresholding lane channel scores were used instead of argmax to reduce each lane's thickness, which practically separated each lane while preserving its location and curvature. However, as seen in Fig. 4, the line of the horizon where all lanes merge usually has all or some lanes overlap, making the sliding window unable to determine which pixels belong to which lane. To further split lanes in this region, erosion is applied with kernel size= 11 on the region starting from the lowest y-index of lane pixels (assuming y-index starts from top to bottom) with a length of 60. This process is repeated with kernel size= 9 in the following region, from 60 to 120. This process is illustrated in Fig. 4. After that, gaussian blurring is applied with kernel size= 9 to smooth lanes and heal holes or disconnections. Finally, erosion with kernel size= 3 is applied to the whole image.

ii. *Sliding Window*

From the bottom of the binary segmentation map, a wide sliding window moves up until the lowest y-index of lane pixels, assuming that the y-index starts from the top to bottom. A region of 5 pixels height and full width is considered at each step where its non-zero pixels (lane pixels) are clustered using DBSCAN clustering with minimum numbers of samples and maximum distance between cluster samples set to 30 and 8, respectively.

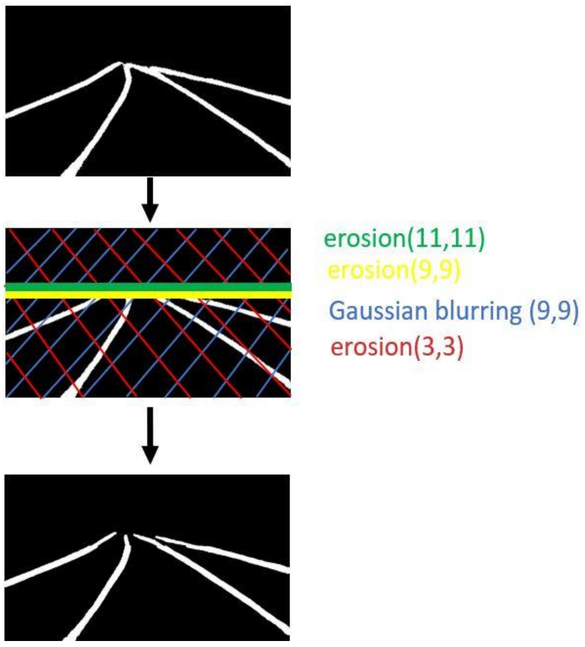

Fig. 4: Top row: the input to the 1st phase of postprocessing. Middle row: the 4 processes applied and the region over which they are applied. Bottom row: the output of the 1st phase

Each acquired cluster is considered as the lowest part of one of the lanes, and another sliding window is applied to extract that lane starting from that cluster as it belongs to the lane the second sliding window is tracking. The found cluster is considered as the lane lowest part, and the initial center of this sliding window is set to equal the center of the first cluster. The second sliding window must move the same distance as the first with the same step size. However, the width of the second sliding window doesn’t fit the whole image width as it was slightly bigger than the width of the cluster and will adapt to the changing width of each cluster it adds to the body of the lane while it’s moving so that the sliding window can fully contain clusters and be able to change its center after each step to also adapt to lane curvature as well as lane width.

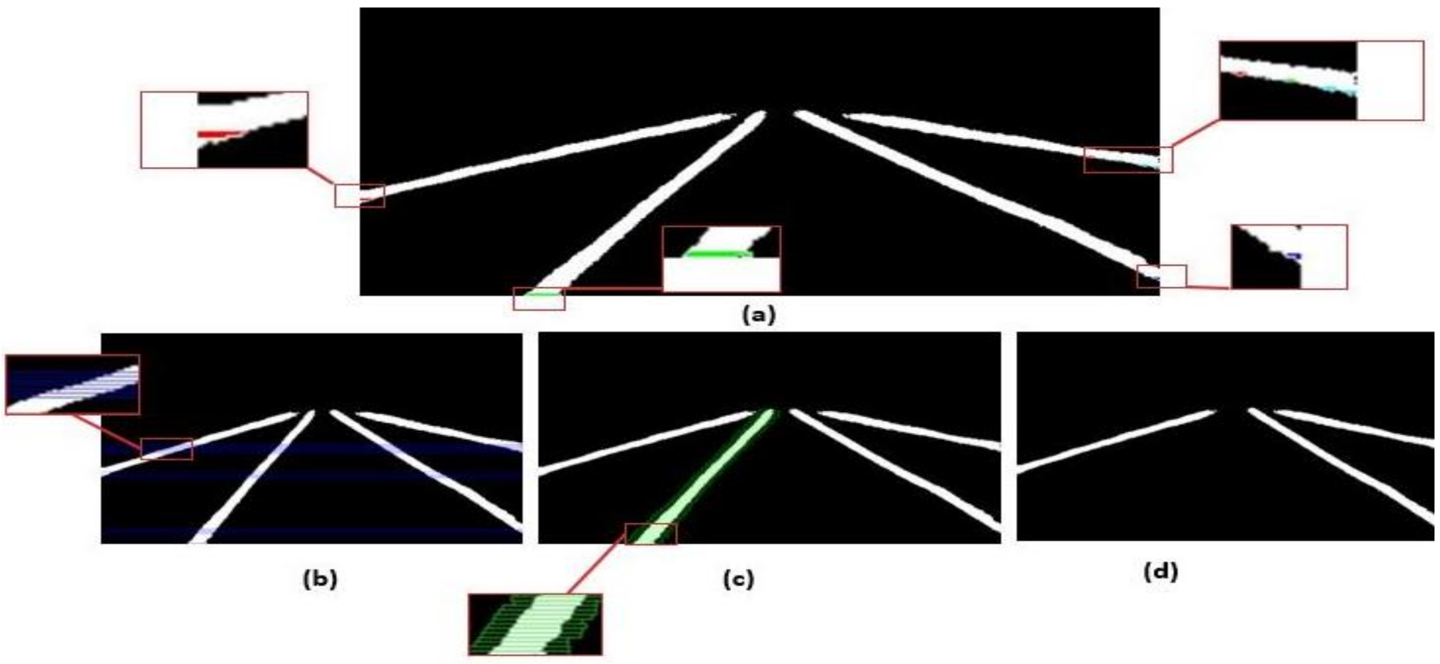

Fig. 5: (a) initial lane clusters captured by the first sliding window. (b) the steps taken by the first sliding window (Note that it started for times, one time for each lane). (c) the steps taken by the second sliding window on one of the lanes after capturing its initial cluster by the first sliding window. (d) After the second sliding window tracked and collected all pixels belonging to some lane, it deletes them from the original image before the first sliding window starts again.

After the second sliding window tracks all clusters of a lane, these clusters are removed from the original binary segmentation map before the first sliding window takes another step cluster of nonzero pixels again.

iii. Fitting Lanes

At this stage, each lane is a set of clusters of pixels collected by the sliding window. Since lane detection is a fundamental perception task in all autonomous driving or driving assistance systems, we must parameterize these lanes for the vehicle to be able to integrate this information, i.e., lane location and curvature, with other perception or control tasks.

Usually, the image is projected into a 'bird's-eye view' which is considered a better representation for lanes before fitting them into a polynomial function because at 'bird's-eye view' lane curvature is reduced thus allowing the lane to fit in a lower order, like 2nd or 3rd, polynomial. Also, the 'bird's-eye view' is frequently used for localization and mapping algorithms, so using it in lane detection makes it easier to integrate the resulting polynomial with other algorithms.

Projecting the image into a 'bird's-eye view' is a linear transformation that uses a transformation matrix as follows: Given a lane pixel $p_i = [x_i, y_i, 1] \in P$, the transformed pixel $p_i' = [x_i', y_i', 1] \in P'$ is obtained by $p_i' = \mathrm{H}p_i$, where H is the transformation matrix.

One of the problems in this approach is that H needs to be adapted to the exact ground-plane on which it's applied. In [11] proposed a network called *H-Net* to output the transformation matrix that is suitable to the input image ground-plane. As a result, we used a 2nd and 3rd order polynomial functions to fit lanes in the normal front-camera view.

IV. Simunlation Results

A. Training

The binary segmentation network is trained using tuSimple lane dataset [14], which is one of the large-scale datasets with 3626 training and 2782 testing images, under different weather conditions, and with up to 5 lanes per image. The dataset also provides the previously unlabeled 19 frames to the labeled frame. Annotations are in the form of x-y pairs for each lane in each image, as shown in Fig. 2, stacked in a json format. As described earlier, the json file is parsed to read the x-y pairs and construct a connected line per lane in a binary image as the label for the binary segmentation network.

The network is trained using Adam optimizer with a batch size of 4 and a learning rate of 4e-4 until convergence.

B. Evaluation

The evaluation was done on tuSimple test set using accuracy as the only metric for ranking while FP and FN are also calculated to give more insight on performance. Accuracy, as defined by tuSimple benchmark [21], is calculated by averaging correct lane points.

$$accuracy = \sum_{clip} \frac{C_{clip}}{S_{clip}} \tag{2}$$

Where C_{clip} is the number of the correctly predicted points in the given image clip, and C_{clip} is the number of requested (ground-truth) points in the given image clip. FP and FN are calculated using the following equations:

$$FN = \frac{M_{pred}}{N_{gt}} \tag{3}$$

$$FP = \frac{F_{pred}}{N_{pred}} \quad (4)$$

Where M_{pred} is the number of missed ground-truth lanes in the prediction, N_{gt} is the number of all the ground-truth lanes in the given image clip, F_{pred} is the number of wrongly predicted lanes, and N_{pred} is the number of all predicted lanes. Table 2 shows the results achieved compared to state-of-the-art methods.

Table 2. Evaluation result on tuSimple dataset

Work	**Acc**	**FP**	**FN**
SCNN [18]	96.53%	0.062	0.0180
LaneNet(+H-net) [11]	96.38%	0.078	0.0244
PointLaneNet(MoblieNet) [17]	96.34%	0.046	0.0518
ENet-SAD [16]	96.64%	0.060	0.0205
KeyPointsEstimation(32x16) [15]	95.75%	0.027	0.0362
KeyPointsEstimation(64x32) [15]	96.62%	0.031	0.0272
Proposed method	91.83%	0.103	0.096

C. Comparing with LaneNet

This work is inspired by LaneNet [11] and the potential to use similar architecture integrated with low-cost postprocessing. However, since LaneNet has no official implementation and since we had to regenerate its result on the platform we have access to (Tesla K80 GPU) to be comparable with our results, we used an unofficial implementation [19] that uses the same method as instructed by [11] except for two differences:

1- The architecture used in [19] for segmentation is a dual-decoder SegNet, while LaneNet uses a dual-decoder ENet.
2- LaneNet uses a second network called H-Net to estimate the perspective transformation matrix conditioned on the input image to fit the lanes more accurately and using a low-order polynomial function. The benefit of perspective transformation to "bird's eye view" in lane detection was discussed in the "fitting lanes" Section.

So it's important to explicitly state that the speed results of LaneNet were re-generated using an unofficial implementation [19] to be able to compare the two methods on the same platform. The proposed implementation is available at [20]. Although accuracy was reduced in this method, Table 3 shows the boost in speed, which gives us room for further improving accuracy.

Table 3. Comparing performance. measured on Tesla K80 GPU

	LaneNet [11]	**Proposed method**
Forward pass time (ms)	13.6	10
Postprocessing time (ms)	1352	56
Total time (ms)	1365.6	66
FPS	0.7	15

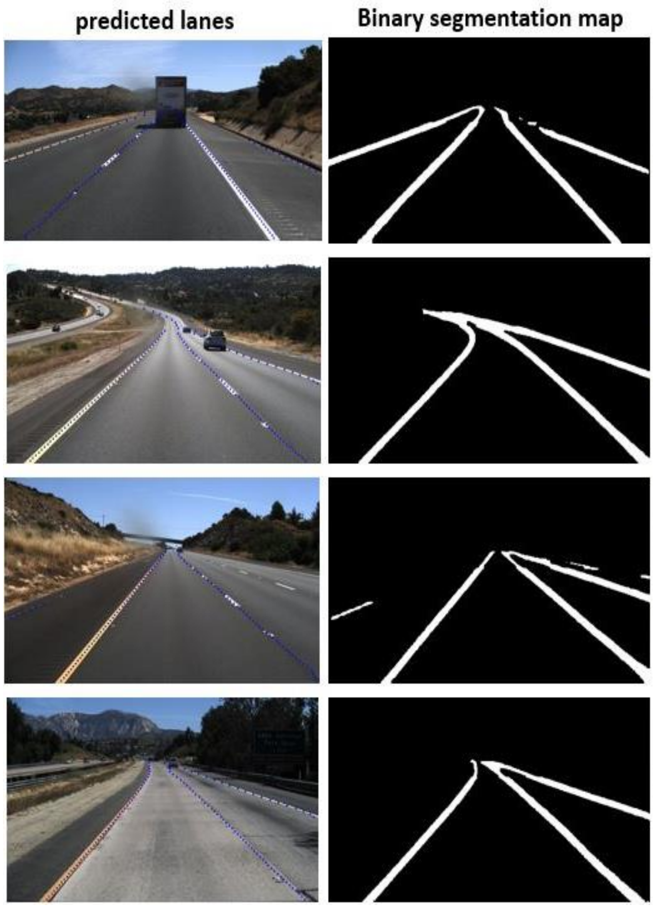

Figure 6 Samples of the predictions on tuSimple test set

V. CONCLUSION

In this paper, a real-time method for lane instance segmentation is proposed, which runs at 15 FPS on Tesla K80 GPU and achieves competitive accuracy on tuSimple benchmark. Many lane detection algorithms, especially deep learning-based methods, focus on accuracy but lack speed. Lane detection is a fundamental perception task in autonomous systems. Thus, considering the limited resources and real-time requirements for such systems must be a critical consideration for practical solutions. Inspired by recent instance segmentation techniques, our goal from this work is to integrate the use of powerful deep-learning architectures and techniques with inexpensive postprocessing to address both accuracy and speed challenges. Results and comparison with other state-of-the-art, mostly deep learning-based methods show that our method is notably faster. However, the current version needs optimization to increase its accuracy. We expect this to be achieved in future work, giving the tolerance obtained in speed, which can be partially traded to improve accuracy.

REFERENCES

[1] Aly, M., "Real time detection of lane markers in urban streets," in *IEEE Intelligent Vehicles Symposium*, 2008, doi: 10.1109/ivs.2008.4621152.
[2] Jiang, Y., Gao, F., & Xu, G., "Computer vision-based multiple-lane detection on straight road and in a curve," in *International Conference on Image Analysis and Signal Processing*, pp. 114-117, 2010.
[3] Kuo-Yu Chiu and Sheng-Fuu Lin., "Lane detection using color-based segmentation," in *IEEE Proceedings. Intelligent Vehicles Symposium*, 2005. doi: 10.1109/ivs.2005.1505186

[4] Lee, C., and Moon, J., "Robust Lane Detection and Tracking for Real-Time Applications," in *IEEE Transactions on Intelligent Transportation Systems*, vol. 19, pp. 4043-4048, 2018.

[5] Li, J., Mei, X., Prokhorov, D., Tao, D., "Deep Neural Network for Structural Prediction and Lane Detection in Traffic Scene," in *IEEE Transactions on Neural Networks and Learning Systems*, vol. 28, no. 3, pp. 690-703, 2017, doi: 10.1109/tnnls.2016.2522428

[6] Van Gansbeke, W., De Brabandere, B., Neven, D., Proesmans, M., Van Gool, L., "End-to-end lane detection through differentiable least-squares fitting," in *IEEE International Conference on Computer Vision Workshops*, 2019.

[7] Zou, Q., Jiang, H., Dai, Q., Yue, Y., Chen, L., Wang, Q., "Robust Lane Detection From Continuous Driving Scenes Using Deep Neural Networks," in *IEEE Transactions on Vehicular Technology*, vol. 69, no. 1, pp. 41-54, 2020, doi: 10.1109/tvt.2019.2949603

[8] Yang, W., Cheng, Y., Chung, P., "Improved Lane Detection With Multilevel Features in Branch Convolutional Neural Networks," in *IEEE Access*, vol. 7, pp. 173148-173156, 2019, doi: 10.1109/access.2019.2957053

[9] Romera-Paredes, B., and Torr, P. H. S., "Recurrent instance segmentation," in *European Conference on Computer Vision*, pp. 312-329, 2016.

[10] Zhang, Z., Schwing, A. G., Fidler, S., and Urtasun, R., "Monocular object instance segmentation and depth ordering with cnns," in *IEEE International Conference on Computer Vision*, pp. 2614-2622, 2015.

[11] Neven, D., Brabandere, B.D., Georgoulis, S., Proesmans, M., and Gool, L.V, "Towards End-to-End Lane Detection: an Instance Segmentation Approach," in *IEEE Intelligent Vehicles Symposium (IV)*, pp. 286-291, 2018.

[12] De Brabandere, B., Neven, D., and Van Gool, L., "Semantic instance segmentation with a discriminative loss function," *in arXiv preprint*, arXiv:1708.02551, 2017.

[13] Badrinarayanan, V., Kendall, A., and Cipolla, R., "Segnet: A deep convolutional encoder-decoder architecture for image segmentation," in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 12, pp. 2481-24, 2017.

[14] The tuSimple lane challange, http://benchmark.tusimple.ai/

[15] Ko, Y., Jun, J., Ko, D., & Jeon, M., "Key Points Estimation and Point Instance Segmentation Approach for Lane Detection," in *arXiv preprint*, arXiv:2002.06604, 2020.

[16] Hou, Yuenan, et al., "Learning lightweight lane detection cnns by self-attention distillation," in *IEEE International Conference on Computer Vision*, 2019.

[17] Chen, Zhenpeng, Qianfei Liu, and Chenfan Lian. "PointLaneNet: Efficient end-to-end CNNs for Accurate Real-Time Lane Detection," in *IEEE Intelligent Vehicles Symposium (IV)*, 2019.

[18] Pan, Xingang, et al., "Spatial as deep: Spatial cnn for traffic scene understanding," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[19] MaybeShewill-CV/lanenet-lane-detection., Retrieved 8 July 2020, from https://github.com/MaybeShewill-CV/lanenet-lane-detection

[20] gadm21/Real-time-lane-instance-segmentation., Retrieved 8 July 2020, from https://github.com/gadm21/Real-time-lane-instance-segmentation.

[21] TuSimple/tusimple-benchmark. GitHub. (2020). Retrieved 29 August 2020, from https://github.com/TuSimple/tusimple-benchmark/tree/master/doc/lane_detection.