CLRNet: Cross Layer Refinement Network for Lane Detection

Tu Zheng\textsuperscript{1,2}*, Yifei Huang\textsuperscript{1,2}*, Yang Liu\textsuperscript{1,2} Wenjian Tang\textsuperscript{1} Zheng Yang\textsuperscript{1} Deng Cai\textsuperscript{1,2} Xiaofei He\textsuperscript{1,2} 
\textsuperscript{1}Fabu Inc., Hangzhou, China \textsuperscript{2}State Key Lab of CAD&CG, Zhejiang University, China

Abstract

Lane is critical in the vision navigation system of the intelligent vehicle. Naturally, lane is a traffic sign with high-level semantics, whereas it owns the specific local pattern which needs detailed low-level features to localize accurately. Using different feature levels is of great importance for accurate lane detection, but it is still under-explored. In this work, we present Cross Layer Refinement Network (CLRNet) aiming at fully utilizing both high-level and low-level features in lane detection. In particular, it first detects lanes with high-level semantic features then performs refinement based on low-level features. In this way, we can exploit more contextual information to detect lanes while leveraging local detailed lane features to improve localization accuracy. We present ROIGather to gather global context, which further enhances the feature representation of lanes. In addition to our novel network design, we introduce Line IoU loss which regresses the lane line as a whole unit to improve the localization accuracy. Experiments demonstrate that the proposed method greatly outperforms the state-of-the-art lane detection approaches. Code is available at: https://github.com/Turoad/CLRNet.

1. Introduction

Lane detection is an important yet challenging task in computer vision, which requires the network to predict lanes in an image. Detecting lanes can benefit many applications, such as autonomous driving and the Advanced Driver Assistance System (ADAS), which helps intelligent vehicles localize themselves better and drive safer.

Benefiting from the effective feature representation of CNN, many approaches [17, 19, 33] have obtained promising performance. However, there are still some challenges for detecting accurate lanes. Lane has high-level semantics, whereas it owns the specific local pattern which needs detailed low-level features to localize accurately. How to utilize different feature levels effectively in CNN remains a problem. As we can see in Fig. 1(a), the landmark and lane line have different semantics, but they share the similar feature (e.g., the long white line). It is hard to distinguish them without high-level semantics and global context. On the other hand, the locality is also essential since lane is long and thin with the simple local pattern. We show the detection result of high-level features in Fig 1(b), though the lane is detected, its location is not precise. Thus, the low-level and high-level information are complementary for accurate lane detection. Previous works either model local geometry of lanes and integrate them into global results [20] or construct a fully-connected layer with global features to predict lanes [19]. These detectors have demonstrated the importance of local or global features for lane detection, but they don’t take advantage of both features, yielding inaccurate detection performance.

Another common problem in lane detection is no visual evidence for the presence of lanes. As shown in Fig. 1(c), the lane is occupied by the car while in Fig. 1(d), the lane is blurred by the extreme lighting condition.
is hard to recognize due to the extreme lighting condition. In the literature, SCNN [17] and RESA [33] propose a message-passing mechanism to gather global context, but these methods perform pixel-wise prediction and don’t take lane as a whole unit. Thus their performances lag behind many state-of-the-art detectors.

In this paper, we propose a new framework, Cross Layer Refinement Network (CLRNet), which fully utilizes low-level and high-level features for lane detection. Specifically, we first perform detection in high semantic features to coarsely localize lanes. Then, we perform refinement based on fine-detail features to get more precise locations. Progressively refining the location of lane and feature extraction leads to high accuracy detection results. To solve the problem of non-visual evidence of lane, we introduce ROIGather to capture more global contextual information by building the relation between the ROI lane feature and the whole feature map. Moreover, we define the IoU of lane by building the relation between the ROI lane feature and ROIGather to capture more global contextual information, the problem of non-visual evidence of lane, we introduce the message-passing mechanism to address no visual evidence problem, which captures the strong spatial relationship for lanes. SCNN significantly improves the lane detection performance, but the method is slow for real-time application. RESA [33] proposes a real-time feature aggregation module, enabling the network to gather the global feature and improve performance. In CurveLane-NAS [28], they use neural architecture search (NAS) to find a better network for capturing accurate information to benefit the detection of curve lanes. However, the NAS is extremely expensive computationally and costs huge GPU hours. These segmentation-based methods are ineffective and time-consuming since they perform pixel-wise prediction on the whole image and don’t consider lanes as a whole unit.

2.2. Anchor-based methods

Anchor-based methods in lane detection can be divided into two classes, e.g., line anchor-based methods and row anchor-based methods. Line anchor-based methods adopt predefined line anchors as references to regress accurate lanes. Line-CNN [8] is the pioneering work to use line anchors in lane detection. LaneATT [24] proposes a novel anchor-based attention mechanism that aggregates global information. It achieves state-of-the-art results and shows both high efficacy and efficiency. SGNet [22] introduces a novel vanish-point guided anchor generator and adds multiple structural guidance to improve performance. As for the row anchor-based method, it predicts the probable cell for each predefined row on images. UFLD [19] first proposes a row anchor-based lane detection method and adopts lightweight backbones to achieve high inference speed. Albeit simple and fast, its overall performance is not good. CondLaneNet [12] introduces a conditional lane detection strategy based on conditional convolution and row anchor-based formulation, i.e., it first locates start points of lane lines then performs row anchor-based lane detection. However, start points are hard to recognize in some complex scenarios, which results in relatively inferior performance.

2.3. Parameter-based methods

Different from points regression, parameter-based methods model the lane curve with parameters and regress these parameters to detect lanes. PolyLaneNet [25] adopts a polynomial regression problem and achieves high efficiency. LSTR [13] takes road structures with camera pose into account to model the lane shape, then introduces the transformer to lane detection task to get the global feature. Parameter-based methods have fewer parameters to regress, but they are sensitive to the predicted parameters, e.g., the error prediction on high-order coefficient may cause shape change of lanes. Though these methods have fast inference speed, they still struggle to achieve higher performance.
3. Approach

3.1. The Lane Representation

**Lane Prior.** Lanes are thin and long with strong shape priors, thus a predefined lane prior can help the network better localize lanes. In common object detection, objects are represented by rectangular boxes. Nevertheless, the box is not appropriate for the representation of the long line. Following [8] and [24], we use equally-spaced 2D-points as lane representation. Specifically, lane is expressed as a sequence of points, i.e., \( P = \{(x_1, y_1), \cdots, (x_N, y_N)\} \). The \( y \) coordinate of points is equally sampled through image vertically, i.e., \( y_i = \frac{H - 1}{N - 1} \times i \), where \( H \) is image height. Accordingly, the \( x \) coordinate is associated with the respective \( y_i \in Y \). In our paper, we call this representation **Lane Prior**. Each lane prior will be predicted by the network and consists of four components: (1) foreground and background probabilities, (2) the length of lane prior, (3) the start point of the lane line and the angle between the \( x \)-axis of the lane prior (termed as \( x, y, \) and \( \theta \)), (4) The \( N \) offsets, i.e., the horizontal distance between the prediction and its ground truth.

3.2. Cross Layer Refinement

**Motivation.** In neural networks, deep high-level features strongly respond to entire objects with more semantic meanings, while the shallow low-level features are with more local contextual information. Allowing lane objects to access high-level features can help exploit more useful context information, e.g., to distinguish lane lines or landmarks. In the meantime, fine-detail features help detect lanes with high localization accuracy. In object detection [9], it builds the feature pyramid to leverage the pyramid shape of a ConvNet’s feature hierarchy and assigns different scales of objects to different pyramid levels. However, it is hard to directly assign a lane to only one level since high-level and low-level features are both critical for lanes. Inspired by Cascade RCNN [3], we can assign lane objects to all levels and detect lanes sequentially. In particular, we can detect lanes with high-level features to localize lanes coarsely. Based on the detected lanes, we can refine them with more detailed features.

**Refinement structure.** Our goal is to leverage a ConvNet’s pyramidal feature hierarchy, which has semantics from low to high levels, and build a feature pyramid with high-level semantics throughout. We take ResNet [6] as the backbone and use \( \{L_0, L_1, L_2\} \) to denote feature levels generated by FPN. As shown in Fig. 2, our cross layer refinement starts from the highest level \( L_0 \) and gradually approaches \( L_2 \). We use \( \{R_0, R_1, R_2\} \) to denote the corresponding refinements. Then we can build a sequence of refinements

\[
P_t = P_{t-1} \circ R_t(L_{t-1}, P_{t-1}),
\]

where \( t = 1, \cdots, T \), \( T \) is the total number of refinements. Our method performs detection from highest level layer with high semantics. \( P_t \) is the parameter of lane prior (start point coordinate \( x, y \) and angle \( \theta \)), which is learnable inspired by [23]. For the first layer \( L_0 \), the \( P_0 \) is uniformly distributed on image plane. The refinement \( R_t \) takes the \( P_t \) as input to get the ROI lane features and then performs two FC layers to get the refined parameter \( P_t \). Progressively refining the lane prior and feature extraction is important for the success of cross layer refinement. Note that, our method is not limited to FPN structure, only using ResNet [6] or adopting PAFPN [14] is also suitable.

3.3. ROIGather

**Motivation.** After we assign lane priors to each feature map, we can get features of lane priors with ROIAlign [5]. However, the contextual information of features is still
not sufficient. In some cases, the lane instance may be occupied or blurred with extreme lighting conditions. Thus there is no local visual evidence for the presence of lane. To determine whether a pixel belongs to a lane, we need to look at nearby features. Some recent studies [27, 32] also indicate that the performance could be improved if making sufficient use of long-range dependencies. Thus, we can gather more useful contextual information to better learn lane feature. To this end, we add convolutions along the lane prior. In this way, each pixel in the lane prior can gather information of nearby pixels, and occupied parts can be reinforced from that information. Moreover, we build relations between features of lane priors and the whole feature map. Thus, it can exploit more contextual information to learn better feature representations.

**ROI Gather Structure.** The ROI Gather module is lightweight and easy to implement. It takes feature map and lane priors as input, each lane prior has $N$ points. For each lane prior, we follow ROIAlign [5] to get the ROI feature of lane prior ($X_p \in \mathbb{R}^{C \times N}$). Unlike ROIAlign for bounding box, we uniform sample $N_p$ points from the lane prior and use bilinear interpolation to compute the exact values of input features at these locations. For ROI features of $L_1, L_2$, we concatenate the ROI features of previous layers to enhance feature representations. Convolutions are performed on the extracted ROI features to gather nearby features for each lane pixel. To save memory, we use fully-connected to further extract the lane prior feature ($X_p \in \mathbb{R}^{C \times 1}$). The feature map is resized to $X_f \in \mathbb{R}^{C \times H \times W}$ and flattened to $X_f \in \mathbb{R}^{C \times HW}$. Details are in Sec. 4.2.

To gather the global context for features of lane priors, we first compute the attention [27] matrix $W$ between ROI lane prior feature ($X_p$) and the global feature map ($X_f$), which is written as:

$$ W = f\left(\frac{X_f^T X_p}{\sqrt{C}}\right), \quad (2) $$

where $f$ is a normalize function softmax. The aggregated feature is written as:

$$ G = WX_f^T. \quad (3) $$

The output $G$ reflects the bonus of $X_f$ to $X_p$, which is selected from all locations of $X_f$. Finally, we add the output to the original input $X_p$.

### 3.4 Line IoU Loss

**Motivation.** As discussed above, the lane prior consists of discrete points needed to be regressed with its ground truth. The commonly used distance loss like smooth-$l_1$ can be used to regress these points. However, this kind of loss takes points as separate variables, which is an oversimplified assumption [31], resulting in less accurate regression.

In contrast to distance loss, Intersection over Union (IoU) can take the lane prior as a whole unit to regress and it is tailored for evaluation metric [21, 31, 34]. In our work, we derive an easy and effective algorithm to compute the Line IoU (LIoU) loss.

**Formula.** We introduce Line IoU loss starting from the definition of the line segment IoU, which is the ratio of interaction over union between two line segments. For each point in the predicted lane as shown in Fig. 3, we first extend it ($x_p^g$) with a radius $e$ into a line segment. Then IoU can be calculated between the extended line segment and its ground truth, which is written as:

$$ IoU = \frac{d_{C}^{p}}{d_{C}^{g}} = \frac{\min(x_i^p + e, x_i^g + e) - \max(x_i^p - e, x_i^g - e)}{\max(x_i^p + e, x_i^g + e) - \min(x_i^p - e, x_i^g - e)} \quad (4) $$

where $x_i^p - e, x_i^p + e$ are the extended points of $x_i^p$, $x_i^g - e, x_i^g + e$ are the corresponding ground truth points. Note that, $d_{C}^{g}$ can be negative, which can make it feasible to optimize in case of non-overlapping line segments.

Then LIoU can be considered as the combination of infinite line points. To simplify the expression and make it easy to compute, we transform it into a discrete form,

$$ LIoU = \frac{\sum_{i=1}^{N} d_{C}^{p}}{\sum_{i=1}^{N} d_{C}^{g}}. \quad (5) $$

Then, the LIoU loss is defined as

$$ \mathcal{L}_{LinIoU} = 1 - LIoU, \quad (6) $$

where $-1 \leq LIoU \leq 1$, when two lines overlay perfectly, then $LIoU = 1$, $LIoU$ converges to -1 when two lines are far away.

Our Line IoU loss exhibits two advantages: (1) It is simple and differentiable, which is very easy to implement parallel computations. (2) It predicts the lane as a whole unit, which helps improve the overall performance.
3.5. Training and Inference Details

Positive samples selection. During training, each ground truth lane is assigned with one or more predicted lanes dynamically as positive samples, which is inspired by [4]. In particular, we first sort the predicted lanes based on the assigning cost, which is defined as:

\[ C_{\text{assign}} = w_{\text{sim}}C_{\text{sim}} + w_{\text{cls}}C_{\text{cls}}, \]
\[ C_{\text{sim}} = (C_{\text{dis}} \cdot C_{\theta_x} \cdot C_{\theta_t})^2. \]  

Here \( C_{\text{cls}} \) is the focal cost [10] between predictions and labels. \( C_{\text{sim}} \) is the similarity cost between predicted lanes and ground truth. It consists of three parts, \( C_{\text{dis}} \) means the average pixel distance of all valid lane points, \( C_{\theta_x} \) means the distance of start point coordinates, \( C_{\theta_t} \) means the difference of the theta angle, they are all normalized to \([0, 1]\). \( w_{\text{cls}} \) and \( w_{\text{sim}} \) are weight coefficients of each defined component. Each ground truth lane is assigned with a dynamic number (top-k) of predicted lanes based on \( C_{\text{assign}} \).

Training Loss. Training loss consists of classification loss and regression loss. The regression loss is only performed on the assigned samples. The overall loss function is defined as:

\[ L_{\text{total}} = w_{\text{cls}}L_{\text{cls}} + w_{\text{xyt}}L_{\text{xyt}} + w_{\text{LIoU}}L_{\text{LIoU}}. \]

\( L_{\text{cls}} \) is the focal loss between predictions and labels, \( L_{\text{xyt}} \) is the smooth-\( l_1 \) loss for the start point coordinate, theta angle and lane length regression, \( L_{\text{LIoU}} \) is the Line IoU loss between the predicted lane and ground truth. Optionally, we can add an auxiliary segmentation loss following [19]. It is only used in the training period and has no cost in inference.

Inference. We set a threshold with a classification score to filter the background lanes (low score lane priors), and we use nms to remove high-overlapped lanes following [24]. Our method can also be nms-free if we use the one-to-one assignment, i.e., set the top-k = 1.

4. Experiment

4.1. Datasets

We conduct experiments on two widely used lane detection benchmark datasets: CULane [17] and Tusimple [26] and one recently released benchmark (LLAMAS [2]).

CULane [17] is a large scale challenging dataset for lane detection. It contains nine challenging categories, such as crowded, night, cross, etc. The CULane dataset consists of 100,000 images for train, validation, and test sets. All the images have 1640 \times 590 pixels.

LLAMAS [2] is also a large scale lane detection dataset with over 100k images. The lane markers in LLAMAS are automatically annotated with highly-accurate maps. Since the label of test set is not public, we upload the detection result to the website of LLAMAS benchmark for testing.

Tusimple [26] lane detection benchmark is one of the most widely used datasets in lane detection. It contains only highway scenes with 3268 images for training, 358 for validation, and 2782 for testing. All have 1280 \times 720 pixels.

4.2. Implementation details

We adopt the ResNet [6] and DLA [30] as our pre-trained backbones. All input images are resized to 320 \times 800. For data augmentation, similar to [12, 20], we use random affine transformation (translation, rotation, and scaling), random horizontal flips. In the optimizing process, we use AdamW [16] optimizer with an initial learning rate of 1e-3 and cosine decay learning rate strategy [15] with power set to 0.9. We train 15 epochs, 70 epochs, 20 epochs for CULane, Tusimple, and LLAMAS, respectively. Our network is implemented based on Pytorch with 1GPU to run all the experiments. We set the number points of lane prior \( N = 72 \), and the sampled number \( N_s = 36 \). The resized \( H, W \) in ROIGather are 10, 25, respectively, channel \( C = 64 \). The extended radius \( e \) in LIoU is 15. The coefficients of assigning cost are set as \( w_{\text{cls}} = 1 \) and \( w_{\text{sim}} = 3 \).

4.3. Evaluation Metric

We adopt the F1-measure as evaluation metric for CULane [17] and LLAMAS [2]. Intersection-over-union (IoU) is calculated between predictions and ground truth. Predicted lanes whose IoU are larger than a threshold (0.5) are considered as true positives (TP). The F1 is defined as:

\[ F_1 = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}. \]

Following COCO [11] detection metric, we also report a new metric mF1 to better compare the localization performance of algorithms. It is defined as

\[ \text{mF1} = \frac{(F1@50 + F1@55 + \cdots + F1@95)}{10}, \]

where \( F1@50, F1@55, \ldots, F1@95 \) are F1 metrics when IoU thresholds are 0.5, 0.55, \ldots, 0.95 respectively. This is a break from the tradition which will reward detectors with better localization results.

For Tusimple [26] dataset, the evaluation formula is

\[ \text{Accuracy} = \frac{\sum_{\text{clip}} C_{\text{clip}}}{\sum_{\text{clip}} S_{\text{clip}}}, \]

where \( C_{\text{clip}}, S_{\text{clip}} \) are the number of correct points and the number of ground truth points of a image respectively. A predicted lane is a correct one if more than 85% predicted lane points are within 20 pixels the ground truth. Tusimple dataset also reports the rate of false positive (FP) and false negative (FN), where \( FP = \frac{\text{Fpiles}}{\text{N}_{\text{pred}}}, FN = \frac{\text{M}_{\text{pred}}}{\text{N}_{\text{gt}}} \).
LaneNet [25] and LaneATT [24] by 7.7 F1@50 and 2.4 F1@50 respectively on the test set, which is significant improvement. Although LaneAF [1] achieves 96.90 F1@50 in the valid dataset, its inference speed is slow (near 20FPS), which makes it hard for deployment. Moreover, our method achieves near 2 points mF1 higher than LaneAF, which demonstrates our method is more accurate in localization.

Performance on Tusimple. Table 3 shows the performance comparison with state-of-the-art approaches. The performance difference between different methods on this dataset is very small, which shows the result in this dataset seems to be saturated (high value) already. Our method seems to be saturated (high value) already. Our method outperforms PolyLaneNet [25] and LaneATT [24] by 7.7 F1@50 and 2.4 F1@50 respectively on the test set, which is significant improvement. Although LaneAF [1] achieves 96.90 F1@50 in the valid dataset, its inference speed is slow (near 20FPS), which makes it hard for deployment. Moreover, our method achieves near 2 points mF1 higher than LaneAF, which demonstrates our method is more accurate in localization.

**4.4. Comparison with the state-of-the-art results**

Performance on CULane. We show the results of our method on the CULane lane detection benchmark dataset and compare them with other popular lane detection methods. As illustrated in Table 1, our proposed method achieves a new state-of-the-art on CULane with an 80.47 F1@50 measure. The ResNet18 version of our method achieves 79.58 F1@50, which is even higher than CondLaneNet (ResNet101) while getting 1.4 points higher than CondLaneNet (ResNet18). In particular, we surpass CondLaneNet (ResNet18) by 3.4% mF1, which indicates our method better regresses lanes with high localization accuracy. Comparing line anchor-based method LaneATT, our ResNet18 version surpasses 7.88% mF1 and 4.45% F1@50, respectively. In the meantime, CLRNet can achieve 206 FPS in one NVIDIA 1080Ti GPU with TensorRT, which is efficient for real-time lane detection.

We show the qualitative results on the CULane dataset in Fig. 6. Segmentation-based methods like RESA don’t predict the lane as a whole unit, which can not preserve the smoothness of lanes. CondLaneNet only predicts one start point of the lane as the proposal, it is easy to miss some lane instances. Our method can predict continuous and smooth lanes in these challenging scenarios, which demonstrates our method can definitely gather global context and has a strong ability to detect accurate lanes.

**Performance on LLAMAS.** The result on the LLAMAS dataset is shown in Table 2. Our method outperforms PolyLaneNet [25] and LaneATT [24] by 7.7 F1@50 and 2.4 F1@50 respectively on the test set, which is significant improvement. Although LaneAF [1] achieves 96.90 F1@50 in the valid dataset, its inference speed is slow (near 20FPS), which makes it hard for deployment. Moreover, our method achieves near 2 points mF1 higher than LaneAF, which demonstrates our method is more accurate in localization.

**Table 1. State-of-the-art results on CULane.** For a fairer comparison, we re-measure the FPS of the source code available detectors using one NVIDIA 1080Ti GPU on the same machine, * means FPS on TensorRT. In addition, we also evaluate these detectors to report the performance comparison with state-of-the-art approaches. The reported metric of these categories is based on F1@50.

<table>
<thead>
<tr>
<th>Method</th>
<th>Backbone</th>
<th>valid</th>
<th>test</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>mF1</td>
<td>F1@50</td>
</tr>
<tr>
<td>CLRNet (ours)</td>
<td>ResNet18</td>
<td>54.83</td>
<td>79.48</td>
</tr>
<tr>
<td>CLRNet (ours)</td>
<td>ResNet34</td>
<td>55.14</td>
<td>79.73</td>
</tr>
<tr>
<td>CLRNet (ours)</td>
<td>DLA34</td>
<td>55.64</td>
<td>80.13</td>
</tr>
</tbody>
</table>

**Table 2. State-of-the-art results on LLAMAS.** Additionally, we re-run the evaluation for these methods with source code and trained models to get the mF1, F1@50, F1@75. For “Cross” category, only false positives are shown. The reported metric of these categories is based on F1@50.

<table>
<thead>
<tr>
<th>Method</th>
<th>Backbone</th>
<th>valid</th>
<th>test</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>mF1</td>
<td>F1@50</td>
</tr>
<tr>
<td>CLRNet (ours)</td>
<td>ResNet18</td>
<td>70.83</td>
<td>96.93</td>
</tr>
<tr>
<td>CLRNet (ours)</td>
<td>DLA34</td>
<td>71.57</td>
<td>97.06</td>
</tr>
</tbody>
</table>
Figure 4. Latency vs. F1-score of state-of-the-art methods on CULane and Tusimple benchmarks.

Table 3. State-of-the-art results on TuSimple. Additionally, F1 was computed using the official source code.

<table>
<thead>
<tr>
<th>Method</th>
<th>Backbone</th>
<th>F1 (%)</th>
<th>Acc (%)</th>
<th>FP (%)</th>
<th>FN (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCNN [17]</td>
<td>VGG16</td>
<td>95.97</td>
<td>96.53</td>
<td>6.17</td>
<td>1.80</td>
</tr>
<tr>
<td>RESA [33]</td>
<td>ResNet34</td>
<td>96.93</td>
<td>96.82</td>
<td>3.63</td>
<td>2.48</td>
</tr>
<tr>
<td>E2E [29]</td>
<td>ERFNet</td>
<td>96.25</td>
<td>96.02</td>
<td>3.21</td>
<td>4.28</td>
</tr>
<tr>
<td>UFLD [19]</td>
<td>ResNet18</td>
<td>87.87</td>
<td>95.82</td>
<td>19.05</td>
<td>3.92</td>
</tr>
<tr>
<td>UFLD [19]</td>
<td>ResNet34</td>
<td>88.02</td>
<td>95.86</td>
<td>18.91</td>
<td>3.75</td>
</tr>
<tr>
<td>LaneATT [24]</td>
<td>ResNet18</td>
<td>96.71</td>
<td>95.57</td>
<td>3.56</td>
<td>3.01</td>
</tr>
<tr>
<td>LaneATT [24]</td>
<td>ResNet34</td>
<td>96.77</td>
<td>95.63</td>
<td>3.53</td>
<td>2.92</td>
</tr>
<tr>
<td>LaneATT [24]</td>
<td>ResNet122</td>
<td>96.06</td>
<td>96.10</td>
<td>5.64</td>
<td>2.17</td>
</tr>
<tr>
<td>FOLOLane [20]</td>
<td>ERFNet</td>
<td>96.59</td>
<td>96.92</td>
<td>4.47</td>
<td>2.28</td>
</tr>
<tr>
<td>CondLaneNet [12]</td>
<td>ResNet18</td>
<td>97.01</td>
<td>95.48</td>
<td>2.18</td>
<td>3.80</td>
</tr>
<tr>
<td>CondLaneNet [12]</td>
<td>ResNet34</td>
<td>96.98</td>
<td>95.37</td>
<td>2.20</td>
<td>3.82</td>
</tr>
<tr>
<td>CondLaneNet [12]</td>
<td>ResNet101</td>
<td>97.24</td>
<td>96.54</td>
<td>2.01</td>
<td>3.50</td>
</tr>
<tr>
<td>CLRNet (ours)</td>
<td>ResNet18</td>
<td>97.89</td>
<td>96.84</td>
<td>2.28</td>
<td>1.92</td>
</tr>
<tr>
<td>CLRNet (ours)</td>
<td>ResNet34</td>
<td>97.82</td>
<td>96.87</td>
<td>2.27</td>
<td>2.08</td>
</tr>
<tr>
<td>CLRNet (ours)</td>
<td>ResNet101</td>
<td>97.62</td>
<td>96.83</td>
<td>2.37</td>
<td>2.38</td>
</tr>
</tbody>
</table>

Table 4. Effects of each component in our method. Results are reported on CULane.

4.5. Ablation study

To validate the effectiveness of different components of the proposed method, we conducted several experiments on the CULane dataset to show the performance.

Overall Ablation Study. To analyze the importance of each proposed method, we report the overall ablation studies in Table 4. We gradually add LIoU loss, Cross Layer Refinement, and ROI Gather on the ResNet18 baseline. LIoU loss improves the mF1 from 51.90 to 52.80. This result validates that the localization accuracy is much improved. Moreover, the refinement further improves the mF1 to 54.74. Results in mF1, F1@50, F1@70, and F1@90 are consistently improved, which validates that leveraging high-level and low-level semantic features to detect lanes is useful and yields consistent improvements. ROI Gather further improves mF1 by 0.5%, which validates rich global context can enhance the representation of lane features.

Analysis for ROI Gather. To further demonstrate how ROI Gather works in the network, we visualize the attention map (Eq. 2) in Fig. 5. It shows attention weight between the ROI feature of the lane prior (orange line) and the whole feature map. The brighter the color is, the larger the weight value is. Notably, the proposed ROI Gather can (i) effectively gather global context with rich semantic information, (ii) capture the feature of foreground lanes even under occlusion. More quantitative results are in Appendix.

Ablation study on Cross Layer Refinement. Ablation studies of Cross Layer Refinement are shown in Table 5. We first implement the detector with only one layer to perform refinement. As we can see from the result (setting $R_0$, $R_1$, $R_2$), these three refinements get similar results. $R_2$ gets relatively high F1@90 while the F1@50 is relatively low, indicating low-level features help regress lanes accurately. However, it may cause false detection due to losing high semantic information. We select the better result $R_0$ and gradually add more refinements. As $R_0 \rightarrow R_0$, it gets slightly improvement. Other fusion feature methods like adding all features still cannot give an improvement.
Figure 6. Visualization results of UFLD, RESA, LaneATT, CondLane and our method on CULane testing set.

Table 5. Ablation studies of on different refinement methods. \( R_i \) is the refinement method discussed in Sec. 3.2. ADD means add all features with refinement iteration=3 for a fairer comparison.

<table>
<thead>
<tr>
<th>Settings</th>
<th>mF1</th>
<th>F1@50</th>
<th>F1@75</th>
<th>F1@90</th>
</tr>
</thead>
<tbody>
<tr>
<td>( R_0 )</td>
<td>52.80</td>
<td>78.27</td>
<td>59.50</td>
<td>16.54</td>
</tr>
<tr>
<td>( R_1 )</td>
<td>52.26</td>
<td>78.05</td>
<td>59.27</td>
<td>16.71</td>
</tr>
<tr>
<td>( R_2 )</td>
<td>52.50</td>
<td>77.82</td>
<td>59.05</td>
<td>16.82</td>
</tr>
<tr>
<td>( R_0 \rightarrow R_0 )</td>
<td>53.02</td>
<td>78.63</td>
<td>59.74</td>
<td>16.88</td>
</tr>
<tr>
<td>( R_0 \rightarrow R_0 \rightarrow R_0 )</td>
<td>53.21</td>
<td>78.61</td>
<td>60.11</td>
<td>16.96</td>
</tr>
<tr>
<td>( R_2 \rightarrow R_1 \rightarrow R_0 )</td>
<td>53.58</td>
<td>78.73</td>
<td>60.44</td>
<td>17.58</td>
</tr>
<tr>
<td>ADD</td>
<td>53.77</td>
<td>78.52</td>
<td>60.45</td>
<td>19.01</td>
</tr>
<tr>
<td>( R_0 \rightarrow R_1 \rightarrow R_2 )</td>
<td><strong>54.74</strong></td>
<td><strong>79.91</strong></td>
<td><strong>61.77</strong></td>
<td><strong>20.09</strong></td>
</tr>
</tbody>
</table>

Adopting the refinement from \( R_0 \) to \( R_2 \) is much better than others, which validates our cross layer refinement can utilize high-level and low-level features better.

Ablation Study on Line-IoU Loss. Ablation studies of Line IoU loss are shown in Table 6. We first turn loss weight to select the best regression weight of smooth-l_1. We observe that the regression loss of smooth-l_1 is much larger than classification loss when regression weight is 1.5. Results show decreasing weight to 0.5 is relatively better. In contrast, LIoU loss is more stable and improves the performance by near 1 point mF1. To be more specific, the improvements mostly come from high overlapping metrics, like F1@80 and F1@90. These experimental results validate that our Line IoU loss can achieve better performance and make the model better converged. We show the proposed Line IoU loss can also improve the performance of LaneATT [24], details can be found in Appendix.

Table 6. Ablation studies of Line IoU loss on CULane.

<table>
<thead>
<tr>
<th>Loss Weight</th>
<th>mF1</th>
<th>F1@50</th>
<th>F1@60</th>
<th>F1@70</th>
<th>F1@80</th>
<th>F1@90</th>
</tr>
</thead>
<tbody>
<tr>
<td>smooth-l_1</td>
<td>0.1</td>
<td>54.15</td>
<td>79.23</td>
<td>75.00</td>
<td>67.35</td>
<td>51.47</td>
</tr>
<tr>
<td></td>
<td>0.5</td>
<td>54.22</td>
<td>79.05</td>
<td>74.94</td>
<td>67.21</td>
<td>52.03</td>
</tr>
<tr>
<td></td>
<td>1.0</td>
<td>53.48</td>
<td>78.33</td>
<td>74.09</td>
<td>66.63</td>
<td>50.85</td>
</tr>
<tr>
<td></td>
<td>1.5</td>
<td>52.88</td>
<td>78.23</td>
<td>73.57</td>
<td>65.43</td>
<td>49.38</td>
</tr>
<tr>
<td>( \mathcal{L}_{LIOU} )</td>
<td>2</td>
<td>55.23</td>
<td><strong>79.58</strong></td>
<td><strong>75.53</strong></td>
<td>68.26</td>
<td>53.62</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>55.22</td>
<td>79.31</td>
<td>75.43</td>
<td><strong>68.54</strong></td>
<td>53.78</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td><strong>55.31</strong></td>
<td>79.35</td>
<td>75.45</td>
<td>68.48</td>
<td><strong>53.82</strong></td>
</tr>
</tbody>
</table>

5. Conclusion

In this paper, we present Cross Layer Refinement Network (CLRNet) for lane detection. CLRNet can exploit high-level features to predict lanes while leveraging local-detailed features to improve localization accuracy. To solve the no visual evidence for the presence of lane, we propose ROI Gather to enhance the representation of lane features by building relations with all pixels. To regress lane as a whole unit, we propose Line IoU loss tailored for lane detection, which considerably improves the performance compared with standard loss, i.e., smooth-l_1 loss. Our method is evaluated on three lane detection benchmark datasets, i.e., CULane, LLamas, and Tusimple. Experiments show our proposed method outperforms current state-of-the-art lane detection methods.

6. Acknowledgement

This work was supported in part by The National Key Research and Development Program of China (Grant Nos: 2018AAA0101400), in part by The National Nature Science Foundation of China (Grant Nos: 62036009, U1909203, 61936006, 62133013), in part by Innovation Capability Support Program of Shaanxi (Program No. 2021TD-05).
References


IEEE conference on computer vision and pattern recognition, pages 7794–7803, 2018. 4


