Reproduced Version of "CLRerNet: Improving Confidence of Lane Detection with LaneIoU"

Alayna Altunsu *aleyna.altunsu@bahcesehir.edu.tr*
Artificial Intelligence Engineering
Bahcesehir University

Ammar Arifin *ammar.arifin@bau.edu.tr*
Artificial Intelligence Engineering
Bahcesehir University

Tarkan Özşen *tarkan.ozsen@bahcesehir.edu.tr*
Artificial Intelligence Engineering
Bahcesehir University

Mohamed Ben Hassen *mohamed.benhasen@bahcesehir.edu.tr*
Artificial Intelligence Engineering
Bahcesehir University

Abstract : This paper focuses on reproducing and enhancing the CLReNet architecture for robust lane detection in autonomous vehicles and ADAS. We meticulously reproduced the original model within the PyTorch framework and validated it using the CULane dataset, ensuring our baseline results align with the original study. We then explored improvements through backbone network variations and data augmentation techniques, aiming to strike a balance between computational efficiency and accuracy. Our experimental results demonstrate that modified backbones maintain competitive performance while reducing computational overhead, and augmented training data enhances robustness in adverse weather conditions. We conducted a comprehensive evaluation using quantitative metrics (accuracy, precision, recall, F1-score, LaneIoU) and qualitative analysis (visual inspection of predictions). Our findings reaffirm the robustness of CLReNet and offer insights into optimizing lane detection systems for real-world driving scenarios, emphasizing the importance of balancing accuracy, efficiency, and robustness.

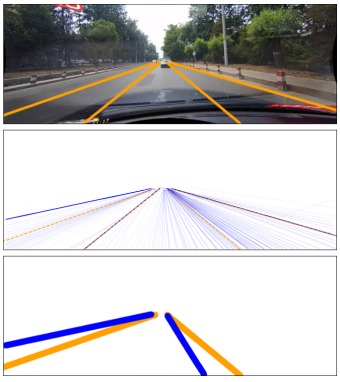


Figure 1: Lane detection example. top: ground truth. middle: all the predictions (blue, deeper for higher confidence scores) and ground truths (dashed orange). bottom: example of metric IoU calculation by comparing segmentation masks of predictions (blue) and GTs (orange).

Figure 2: Default Demo Result.

1 Introduction

1.1 Overview of the Paper

Accurate lane detection is crucial for autonomous driving. CLRerNet improves detection confidence using the LaneIoU metric, which incorporates local lane angles for better IoU calculations. Integrated into training, LaneIoU enhances prediction accuracy.

Validated on CULane and CurveLanes datasets, CLRerNet achieved F1 scores over 81%. The authors provide detailed methodologies for reproducibility, including source code and training procedures.

Future research includes applying LaneIoU to other architectures, challenging conditions, and vision tasks. CLRerNet advances lane detection reliability for autonomous driving.

1.2 Difference Between CLRNet and CLRerNet

CLRNet focuses on combining high-level semantic features with low-level details for accurate lane detection using a two-stage approach: high-level detection for rough localization and low-level refinement for better accuracy. Key components include FPN for multi-resolution feature maps, ROIGather for enhanced lane feature representation, and Line IoU loss for improved localization by treating lane lines as whole units.

CLRerNet enhances detection confidence by integrating LaneIoU loss, which considers lane confidence and IoU. While CLRNet uses a two-stage refinement strategy with ROIGather, CLRerNet focuses on LaneIoU for better confidence scores.

Mathematically, IoU measures overlap between two objects as $\text{IoU} = \frac{\text{Area of Intersection}}{\text{Area of Union}}$. LaneIoU adapts this for lane detection, considering local angles:

$$\text{LaneIoU} = \frac{\sum_{i=1}^{N} \min(x_i^p + e, x_i^g + e) - \max(x_i^p - e, x_i^g - e)}{\sum_{i=1}^{N} \max(x_i^p + e, x_i^g + e) - \min(x_i^p - e, x_i^g - e)}$$

where x_i^p and x_i^g are predicted and ground-truth points, and e is an extension radius.

CLRNet's total loss function includes regression, classification, segmentation, and Line IoU losses:

$$L_{\text{total}} = \lambda_0 L_{\text{reg}} + \lambda_1 L_{\text{cls}} + \lambda_2 L_{\text{seg}} + \lambda_3 L_{\text{LineIoU}}$$

CLRerNet, on the other hand, focuses on LaneIoU, replacing the horizontal coordinate regression loss:

$$L_{\text{total}} = \lambda_0 L_{\text{LaneIoU}} + \lambda_1 L_{\text{cls}} + \lambda_2 L_{\text{seg}}$$

Key differences include the loss calculation method (Line IoU in CLRNet vs. LaneIoU in CLRerNet), refinement strategy (two-stage in CLRNet vs. LaneIoU focus in CLRerNet), and additional components like ROIGather in CLRNet.

2 Related Work

2.1 Object detection

Training sample assignment. Sample assignment is the major research focus in object detection. The proposals from the detection head are assigned to the ground truth samples. (22; 13; 8; 12) assign the GTs by calculating IoU between the anchors on the feature map grid and GT boxes statically. (4) introduces the optimal transfer assignment (OTA) for object detector's training sample assignment, that dynamically assigns the prediction boxes to the GTs. (5) simplifies OTA and realizes iteration-free assignment. IoU functions. Several variants of IoU functions (23; 29; 30) are proposed for accurate bounding box regression and fast convergence. For example, the generalized IoU (GIoU) (23) introduces the smallest convex hull of the boxes and makes IoU differentiable even when the bounding boxes do not overlap. Our LaneIoU is based on GIoU but newly enables the IoU calculation between curves in the row-based representation.

2.2 Lane detection

Lane detection methods are categorized by their lane representation types: segmentation-based, keypoint-based, parametric, and row-based (28; 15; 24).

Segmentation-based methods estimate lane existence probability at the pixel level. For instance, SCNN (18) and RESA (27) employ a semantic segmentation paradigm to classify lane instances. However, these methods do not treat lanes as holistic instances and require computationally expensive

post-processing (19; 28). Keypoint-based methods detect lane points as keypoints and group them into lane instances afterward. Similar to human pose estimation, PINet (11) and FOLOLane (21) are examples of this approach. This line of methods requires post-processing to group the lane points into lane instances, which is computationally expensive. Parametric methods represent a lane instance as a set of curve parameters. PolyLaneNet, LSTR (16), and BSNet (2) are examples of this approach. These methods achieve relatively fast inference, but an error in one parameter affects the entire lane shape. Row-based methods represent a lane instance as a set of x-coordinates at fixed rows and are the de facto standard for detection performance. LaneATT (24) and CLRNet (28) are examples of this approach. They employ lane anchors to learn the confidence score and local x-coordinate displacement for each anchor.

3 Methods

3.1 Network design and losses

The row-based representation (28; 15; 24) leverages the most accurate but simple detection pipeline among the four types described in Section 2. From the row-based methods we employ the best-performing CLRNet (28) as a baseline. The network schematic is shown in Fig. 2. The backbone network (e.g. ResNet (9) and DLA (26)) and the upsampling network extract multi-level feature maps whose spatial dimensions are (1/8, 1/16, 1/32) of the input image. The initial anchors are formed from the N_a learnable anchor parameters (x_a, y_a, θ_a) where (x_a, y_a) is the starting point and θ_a the tilt of the anchor. The feature map is sampled along each of them and fed to the convolution and fully-connected (FC) layers. The classification logits c, anchor refinement δx_a, δy_a, $\delta\theta_a$, length l and local x-coordinate refinement δx tensors are output from the FC layers. The anchors refined by δx_a, δy_a and $\delta\theta_a$ re-sample the higher-resolution feature map, and the procedure is repeated for three times. The pooled features are interacted with the feature map via cross-attention and are concatenated across different refinement stages. The lane prediction is expressed as classification (confidence) logits and a set of x-coodinates at N_{row} rows calculated from the final x_a, y_a, θ_a, l and δx. More details about the refinement mechanism can be found in [28]. During training, the predictions close to a GT are assigned via dynamic assigner [5]. The assigned predictions are regressed toward the corresponding GT and learned to be classified as positives.

$$L = \lambda_0 L_{\text{reg}} + \lambda_1 L_{\text{cls}} + \lambda_2 L_{\text{seg}} + \lambda_3 L_{\text{LaneIoU}} \quad (1)$$

where L_{reg} is smooth-L1 loss to regress the anchor parameters (x_a, y_a, θ_a) and l, L_{cls} a focal loss [14] for positive-or-negative anchor classification, L_{seg} an auxiliary cross-entropy loss for per-pixel segmentation mask, and L_{LaneIoU} the newly introduced LaneIoU loss.

3.2 Oracle experiments

We conduct the preliminary oracle experiments to determine the direction of our approach. The prediction components from the trained baseline model are replaced with GTs partially to analyze how much room for improvement each lane representation component has. Table 1 shows the oracle experiment results. The baseline model (first row) is CLRNet-DLA34 trained without the redundant frames. The confidence threshold is set to 0.39 which is obtained via cross-validation (see Subsection 4.1 and 4.2). Next, we calculate the metric IoU between predictions and GTs as the oracle confidence scores. For each prediction, the maximum IoU among the GTs is employed as the oracle score. In this case, the predicted lane coordinates are not changed. The F150 jumps to 98.47 - the near-perfect score (second row). The result suggests that the correct lanes are already among the predictions, however the confidence scores need to be predicted accurately representing the metric IoU. The other components are the anchor parameters - x_a, y_a, θ_a and length l that determine the lane coordinates. We alter the anchor parameters and lane length by those of GTs (third and fourth rows respectively). Although the row-wise refinement δx is not changed, the oracle anchor parameters improve $F1_{50}$ by 9 points. On the other hand, the oracle length does not affect the performance significantly. The results lead to the second suggestion that the anchor parameters (x_a, y_a, θ_a) are important in terms of lane localization. We focus on the first finding and aim to learn high-quality confidence scores by improving the lane similarity function.

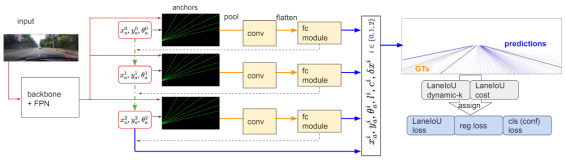


Figure 3: Network Schematic of CLRerNet

LaneIoU The existing methods (15; 28) exploit horizontal distance and horizontal IoU as similarity functions respectively. However, these definitions do not match the metric IoU calculated with segmentation masks. For instance, when the lanes are tilted, the horizontal distance corresponding to the certain metric-IoU is larger than that of vertical lanes. To bridge the gap, we introduce a differentiable local-angle-aware IoU definition, namely LaneIoU. Fig. **??** shows an example of IoU calculation between two tilted curves. We compare LineIoU (28) and LaneIoU on the same lane instance pair. LineIoU applies a constant virtual width regardless of the lane angle and the virtual lane gets 'thin' at the tilted part. In our LaneIoU, overlap and union are calculated considering the tilt of the local lane parts at each row. We define Ω_{pq} as the set of y slices where both lanes p and q exist, and Ω_p or Ω_q where only one lane exists.

LaneIoU is calculated as:

$$\text{LaneIoU} = \frac{\sum_{i=0}^{H} I_i}{\sum_{i=0}^{H} U_i} \tag{2}$$

where I_i and U_i are defined as:

$$I_i = \min(x_i^p + w_i^p, x_i^q + w_i^q) - \max(x_i^p - w_i^p, x_i^q - w_i^q) \tag{3}$$

$$U_i = \max(x_i^p + w_i^p, x_i^q + w_i^q) - \min(x_i^p - w_i^p, x_i^q - w_i^q) \tag{4}$$

when $i \in \Omega_{pq}$. The intersection of the lanes is positive when the lanes are overlapped and negative otherwise. If $i \notin \Omega_{pq}$, I_i and U_i are calculated as follows:

$$I_i = 0,\ U_i = 2w_i^k \text{ if } k \in \{p, q\},\ i \in \Omega_k \tag{5}$$

$$I_i = 0,\ U_i = 0 \text{ if } i \notin (\Omega_{pq} \cup \Omega_p \cup \Omega_q) \tag{6}$$

The virtual lane widths w_i^p and w_i^q are calculated taking the local angles into consideration:

$$w_i^k = \frac{w_{\text{lane}}}{2} \sqrt{\frac{(\Delta x_i^k)^2 + (\Delta y_i^k)^2}{\Delta y_i^k}} \tag{7}$$

where $k \in \{p, q\}$ and Δx_i and Δy_i stand for the local changes of the lane point coordinates. Equation 7 compensates the tilt variation of lanes and represents a general row-wise lane IoU calculation. When the lanes are vertical, w_i equals to $w_{\text{lane}}/2$ and gets larger as the lanes tilt. w_{lane} is the parameter which controls the strictness of the IoU calculation. The CULane metric employs 30 pixels for the resolution of (590, 1640). In Fig. 4, LineIoU (28) and our LaneIoU are compared by calculating correlation with the CULane metric. We replace each prediction's confidence score with the LineIoU or LaneIoU value and also calculate the metric IoU. The GT with the largest IoU is chosen for each prediction. Clearly our LaneIoU shows better correlation with the metric IoU mainly as the result of eliminating the influence of lane angles.

confidence	(X_a, Y_a, θ_a)	l	✓	F1_{50}
				80.86
✓				**98.47**
	✓			89.91
		✓		81.09

Table 1: Oracle experiment results.

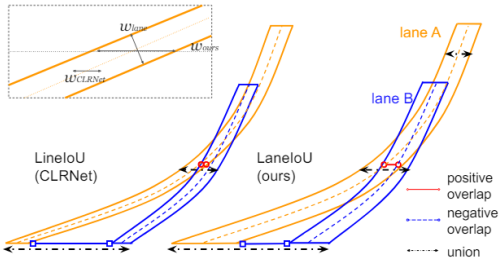


Figure 4: Example of LineIoU (28) and LaneIoU calculations between laneA and laneB. w_{lane}, w_{CLRNet}, and w_{ours} within the dashed rectangle stand for the lane width, the constant width of (28), and our angle-aware width.

3.3 Sample assignment

The confidence scores are learned to be high if the anchor is assigned as positive during training. We adopt LaneIoU for sample assignment to bring the detector's confidence scores close to the segmentation-based IoU. (28) employs the SimOTA assigner (5) to dynamically assign k_i anchors for each GT lane t_i. The number of anchors k_i is determined by calculating the sum of all the anchors' positive IoUs. We employ LaneIoU as:

$$k_i = \sum_{j=1}^{m} \text{LaneIoU}(p_j, t_i) \tag{8}$$

where m is the number of anchors and $i = 0, 1, ..., n$ is the index of n GT lanes. p_j is a predicted lane and $j = 0, 1, ..., m$ is the index of m predictions. k_i is clipped to be from 1 to k_{max}. The cost matrix determines the priority of the assignment for each GT. In object detection, (5) adopts the sum of bounding box IoU and classification costs. In lane detection, CLRNet (28) utilizes the classification cost and the lane similarity cost which consists of horizontal distance, angle difference and start-point distance. However, the cost that directly represents the evaluation metric is more straightforward for prioritizing predictions. We define the cost matrix as:

$$\text{cost}_{ji} = -\text{LaneIoU}_{\text{norm}}(p_j, t_i) + \lambda f_{\text{class}}(p_j, t_i) \tag{9}$$

where λ is the parameter to balance the two costs and f_{class} is the cost function for classification, such as a focal loss (14). LaneIoU is normalized from its minimum to maximum. The formulation (eq. 8 and 9) realizes dynamic sample assignment of the proper number of anchors which are prioritized according to the evaluation metric. In summary, compared with CLRNet, our CLRerNet introduces LaneIoU

as the dynamic-k assignment function, assignment cost function and loss function to learn the high-quality confidence scores that better correlate with the metric IoU.

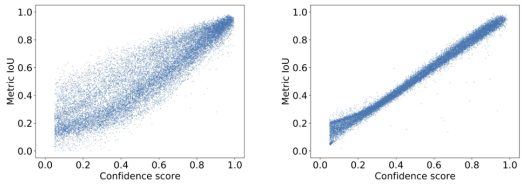


Figure 5: Correlation between CULane metric IoU and (a) LineIoU (28) and (b) LaneIoU (ours)

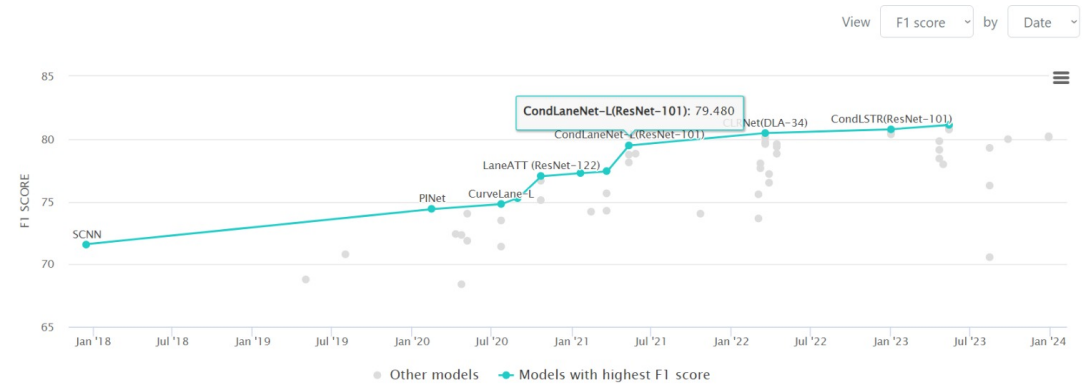
4 Libraries and Datasets

The paper, "CLRerNet: Improving Confidence of Lane Detection with LaneIoU," likely utilized deep learning frameworks such as TensorFlow or PyTorch, and computer vision libraries like OpenCV, although specific libraries are not mentioned.

The datasets likely used include established benchmarks like CULane and CurveLanes, which are known for extensive images with precisely labeled lane markings. CLRNet is designed for CULane, TuSimple, and LLAMAS datasets, achieving state-of-the-art performance in several instances. CLRerNet improves CULane performance, holding top ranks in F1 scores, and can address non-linearity by training on diverse datasets like CurveLanes.

Lane Detection on CULane

Leaderboard | Dataset



Filter: ResNet | ResNet-18 | untagged

Edit Leaderboard

Rank	Model	F1 score	mF1	Paper	Code	Result	Year	Tags
1	**CLRerNet-DLA34**	81.12		CLRerNet: Improving Confidence of Lane Detection with LaneIoU			2023	
2	**CLRerNet-Res101**	80.91		CLRerNet: Improving Confidence of Lane Detection with LaneIoU			2023	
3	**CondLSTR** (ResNet-101)	80.77		Generating Dynamic Kernels via Transformers for Lane Detection			2023	
4	**CLRerNet-Res34**	80.76		CLRerNet: Improving Confidence of Lane Detection with LaneIoU			2023	

Figure 6

5 Results

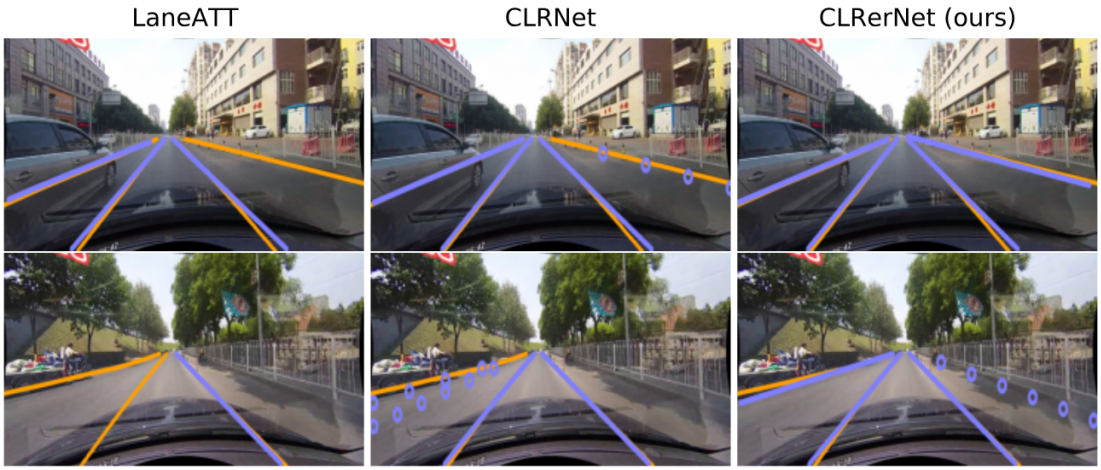


Figure 7: Qualitative results comparing LaneATT, CLRNet and our CLRerNet⋆. Predictions and GTs are shown in blue and orange respectively. Predictions with insufficient confidence score are shown as blue circles.

Results for the testing are below:

```
Evaluation results for IoU threshold = 0.1
Eval category: test_all     , N: 34680, TP: 86312, FP:  4663, FN: 18574, Precision: 0.9487, Recall: 0.8229, F1: 0.8814
Eval category: test0_normal, N: 9621, TP: 31891, FP:   420, FN:   886, Precision: 0.9870, Recall: 0.9730, F1: 0.9799
Eval category: test1_crowd , N: 8113, TP: 22858, FP:   778, FN:  5145, Precision: 0.9671, Recall: 0.8163, F1: 0.8853
Eval category: test2_hlight, N:  486, TP:  1305, FP:    75, FN:   380, Precision: 0.9457, Recall: 0.7745, F1: 0.8515
Eval category: test3_shadow, N:  930, TP:  2427, FP:   114, FN:   449, Precision: 0.9551, Recall: 0.8439, F1: 0.8961
Eval category: test4_noline, N: 4067, TP:  7598, FP:   892, FN:  6423, Precision: 0.8949, Recall: 0.5419, F1: 0.6750
Eval category: test5_arrow , N:  890, TP:  2949, FP:    65, FN:   233, Precision: 0.9784, Recall: 0.9268, F1: 0.9519
Eval category: test6_curve , N:  422, TP:  1083, FP:    24, FN:   229, Precision: 0.9783, Recall: 0.8255, F1: 0.8954
Eval category: test7_cross , N: 3122, TP:     0, FP:  1334, FN:     0, Precision: 0.0000, Recall: 0.0000, F1: 0.0000
Eval category: test8_night , N: 7029, TP: 16201, FP:   961, FN:  4829, Precision: 0.9440, Recall: 0.7704, F1: 0.8484
Evaluation results for IoU threshold = 0.5
Eval category: test_all     , N: 34680, TP: 79862, FP: 11113, FN: 25024, Precision: 0.8778, Recall: 0.7614, F1: 0.8155
Eval category: test0_normal, N: 9621, TP: 30707, FP:  1604, FN:  2070, Precision: 0.9504, Recall: 0.9368, F1: 0.9436
Eval category: test1_crowd , N: 8113, TP: 20873, FP:  2763, FN:  7130, Precision: 0.8831, Recall: 0.7454, F1: 0.8084
Eval category: test2_hlight, N:  486, TP:  1152, FP:   228, FN:   533, Precision: 0.8348, Recall: 0.6837, F1: 0.7517
Eval category: test3_shadow, N:  930, TP:  2289, FP:   252, FN:   587, Precision: 0.9008, Recall: 0.7959, F1: 0.8451
Eval category: test4_noline, N: 4067, TP:  6389, FP:  2101, FN:  7632, Precision: 0.7525, Recall: 0.4557, F1: 0.5676
Eval category: test5_arrow , N:  890, TP:  2820, FP:   194, FN:   362, Precision: 0.9356, Recall: 0.8862, F1: 0.9103
Eval category: test6_curve , N:  422, TP:   954, FP:   153, FN:   358, Precision: 0.8618, Recall: 0.7271, F1: 0.7888
Eval category: test7_cross , N: 3122, TP:     0, FP:  1334, FN:     0, Precision: 0.0000, Recall: 0.0000, F1: 0.0000
Eval category: test8_night , N: 7029, TP: 14678, FP:  2484, FN:  6352, Precision: 0.8553, Recall: 0.6980, F1: 0.7686
Evaluation results for IoU threshold = 0.75
Eval category: test_all     , N: 34680, TP: 63671, FP: 27304, FN: 41215, Precision: 0.6999, Recall: 0.6070, F1: 0.6502
Eval category: test0_normal, N: 9621, TP: 26319, FP:  5992, FN:  6458, Precision: 0.8146, Recall: 0.8030, F1: 0.8087
Eval category: test1_crowd , N: 8113, TP: 16320, FP:  7316, FN: 11683, Precision: 0.6905, Recall: 0.5828, F1: 0.6321
Eval category: test2_hlight, N:  486, TP:   802, FP:   578, FN:   883, Precision: 0.5812, Recall: 0.4760, F1: 0.5233
Eval category: test3_shadow, N:  930, TP:  1689, FP:   852, FN:  1187, Precision: 0.6647, Recall: 0.5873, F1: 0.6236
Eval category: test4_noline, N: 4067, TP:  4548, FP:  3942, FN:  9473, Precision: 0.5357, Recall: 0.3244, F1: 0.4041
Eval category: test5_arrow , N:  890, TP:  2394, FP:   620, FN:   788, Precision: 0.7943, Recall: 0.7524, F1: 0.7728
Eval category: test6_curve , N:  422, TP:   609, FP:   498, FN:   703, Precision: 0.5501, Recall: 0.4642, F1: 0.5035
Eval category: test7_cross , N: 3122, TP:     0, FP:  1334, FN:     0, Precision: 0.0000, Recall: 0.0000, F1: 0.0000
Eval category: test8_night , N: 7029, TP: 10990, FP:  6172, FN: 10040, Precision: 0.6404, Recall: 0.5226, F1: 0.5755
```

Figure 8: Test Result

A lower IoU threshold yields the best results, with precision favored over recall due to the high risk associated with false positives in lane detection. The algorithm's cautious nature is evident in its struggles with curves and challenging conditions like headlights, missing lines, and nighttime, which decrease recall and the F1 score. Despite these challenges, CLRerNet excels on the CULane dataset.

Below are the training results:

Table 2: Evaluation results on the CULane test set. Our experiments are below the double horizontal line.

Method	Backbone	$F1_{50}$	$F1c_{50}$	Normal	Crowd	Dazzle	Shadow	Noline	Arrow	Curve	Cross	Night	GFLOPs	FPS
SCNN	VGG16	71.60	39.84	90.60	69.70	58.50	66.90	43.40	84.10	64.40	19.90	66.10	218.6	50
LaneATT	Res18	75.13	51.29	91.17	72.71	65.82	68.03	49.13	87.82	63.75	10.20	68.58	9.3	211
LaneATT	Res34	76.68	54.34	92.14	75.03	66.47	70.15	49.39	88.38	67.72	13.30	70.72	18.0	170
CondLane (15)	Res18	78.14	57.42	92.87	75.79	70.72	80.01	52.39	89.37	72.40	13.64	73.23	10.2	348
CondLane (15)	Res34	78.74	59.39	93.38	77.14	71.17	79.93	51.85	89.89	73.88	13.87	73.92	19.6	237
CondLane (15)	Res101	79.48	61.23	93.47	77.44	70.93	80.91	54.13	90.16	75.21	12.01	74.80	44.8	97
CLRNet (28)	Res34	79.73	62.11	93.49	78.06	74.57	79.92	54.01	90.59	72.77	12.16	75.02	21.5	204
CLRNet (28)	Res101	80.13	62.96	93.85	78.78	72.49	82.33	54.50	89.79	75.57	12.62	75.51	42.9	94
CLRNet (28)	DLA34	80.47	62.78	93.73	79.59	75.30	82.51	54.58	90.62	74.13	11.55	75.37	18.4	185
LaneATT++	Res34	$77.51_{\pm 0.10}$	56.78	92.48	75.47	68.09	73.21	50.96	88.72	68.18	10.54	72.58	18.0	170
CLRNet	Res34	$80.54_{\pm 0.12}$	63.65	93.85	79.22	73.32	82.50	55.26	90.84	74.06	11.06	75.92	21.5	204
CLRNet	Res101	$80.67_{\pm 0.06}$	64.35	93.95	79.60	72.91	81.58	55.76	90.42	74.06	11.66	76.01	42.9	94
CLRNet	DLA34	$80.86_{\pm 0.06}$	64.05	94.03	79.78	75.23	81.94	56.02	90.67	74.57	11.84	76.40	18.4	185
LaneATT++	Res34	$78.19_{\pm 0.06}$	56.96	92.60	76.42	69.12	77.59	52.01	88.75	64.49	9.74	72.78	18.0	153
CLRerNet	Res34	$80.76_{\pm 0.13}$	63.77	93.93	79.51	73.88	83.16	55.55	90.87	74.45	10.88	76.02	21.5	204
CLRerNet	Res101	$80.91_{\pm 0.10}$	64.30	93.91	80.03	72.98	82.92	55.73	90.53	73.83	11.13	76.13	42.9	94
CLRerNet	DLA34	$81.12_{\pm 0.04}$	64.07	94.02	80.20	74.41	83.71	56.27	90.39	74.67	11.61	76.53	18.4	185
CLRerNet⋆	DLA34	**$81.43_{\pm 0.14}$**	**65.06**	**94.36**	**80.62**	**75.23**	**84.35**	**57.31**	**91.11**	**76.99**	**12.44**	**76.76**	18.4	185

Epoch	loss_cls	loss_reg_xytl	loss_iou	loss_seg	loss
1 (100/2321)	1.4044	6.2046	3.5071	0.7584	11.8745
2 (200/2321)	0.6360	1.5719	2.2164	0.5510	4.9754
3 (300/2321)	0.5779	1.1578	1.7554	0.4824	3.9735
4 (400/2321)	0.5874	0.9828	1.4991	0.4527	3.5221
5 (500/2321)	0.5648	0.6678	1.2848	0.4042	2.9216
6 (600/2321)	0.5580	0.6002	1.1898	0.3924	2.7404
7 (700/2321)	0.5544	0.5401	1.1163	0.3682	2.5790
8 (800/2321)	0.5442	0.5219	1.0569	0.3441	2.4671
9 (900/2321)	0.5317	0.4912	0.9891	0.3369	2.3489

The only result providing training was able to go on for 9 epochs out of the determined maximum 15 before crashing. "loss_cls" is the classification loss, the error in the model's ability to classify lanes correctly. Classification loss usually presents as false negatives under the lane detection pretense. "loss_reg_xytl" is the localization regression loss, the error in the model's ability to predict the bounding box coordinates (x, y, w, h) of an object accurately. Localization regression loss presented as the lines being flattened or extended (see Figure 8). "loss_iou" is the intersection-over-union loss, a metric called IoU (Intersection over Union) to measure how well the predicted bounding box overlaps with the actual ground truth bounding box of the object. An IoU of 1 indicates a perfect overlap, while a value closer to 0 means the boxes don't overlap much. "loss_seg" is the segmentation loss, which measures the difference between the predicted mask and the actual ground truth mask. "loss" is the total of mentioned loss values. The loss values trend downwards during the 9 Epochs, from which it can be assumed that the training would provide improved results on the tests if it were to be successfully completed.

Metric	Original	Your Average
F150	81.43±0.14	0.8155
F175	65.06	0.7304
Normal	94.36	0.9163
Crowd	80.62	0.7304
Dazzle	75.23	N/A
Shadow	84.35	0.7774
Noline	57.31	0.5290
Arrow	91.17	0.8455
Curve	79.11	0.7180
Cross	1540	0.5977
Night	76.92	0.7381
GFLOPs	18.4	N/A
FPS	185	N/A

Figure 9: Comparison between our result with the original paper

6 Discussion

6.1 Results Insight and Analysis

Inference code consistently produces results with some tweak necessary to the code depending on the dataset it was gathered from. This helps visualize the algorithm in various cases.

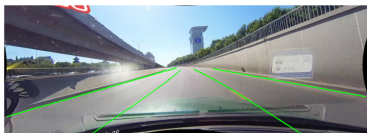


Figure 10: Road with a Curve.

Figure 11: Parameter tuning for different datasets (NuScenes)

Figure 12: True Positive versus False Positive

Figure 13: False Positive Caused by Dataset difference (CurveLanes)

6.2 Future Directions

While CLRerNet shows significant progress, future research could explore:

Generalizability: Apply LaneIoU to other lane detection networks.

Performance in Challenging Scenarios: Test LaneIoU in adverse conditions like rain, snow, fog, and low-light.

Applicability in Other Vision Tasks: Extend LaneIoU to other vision tasks needing accurate overlap estimation, such as object detection in cluttered environments or image segmentation of elongated shapes.

6.3 Further Applications

The advancements in lane detection with CLRerNet and LaneIoU have wide-ranging implications:

Autonomous Driving Systems: Enhance functionalities like lane keeping and merging in complex conditions.

ADAS: Improve lane departure warnings, adaptive cruise control, and emergency braking.

Road Maintenance and Infrastructure Planning: Aid in road condition monitoring and lane marking optimization.

ITS: Provide precise lane usage data for better traffic management and congestion control.

Fleet Management: Improve route planning and reduce fuel consumption for commercial fleets.

Autonomous Drones and Robotics: Enhance navigation for drones and automated vehicles.

Mapping and GIS: Contribute to accurate road network mapping.

Research and Development: Inspire further research in computer vision and machine learning.

7 Limitations and Challenges

7.1 Encountered Obstacles

Initially, it should be noted that general information regarding some hardware and software requirements are necessary to tackle this reproduction. The path less traveled, which our team pursued, resulted in extended periods of time where lack of experience in the theoretical inner workings and practical key points had to be recognized.

A getaround for running the project on Linux was attempted by adapting the code from UNIX syntax to Windows, proving unsuccessful. After recognizing the necessity of a Linux environment and adapting to it (Windows Subsystem For Linux) in order to run the project, the initial problem was using docker compose and overcoming the issues it presented. Errors regarding non-existent paths, files that can't be accessed and permission errors persist at the docker stage all throughout the process. Non-existent paths had to be manually fed to docker-compose.yaml file at times, some of the docker build files were downloaded by hand and integrated by altering the Dockerfile, permission errors are usually solved by managing Docker group permissions in the local device or forcing the commands using "sudo". Afterwards, a lack of understanding around hardware led to the trial of removing Nvidia runtime from the daemon.json file as well as the docker-compose.yaml file. CUDA functions which would render this method useless were not properly acknowledged until after finding the PTH file necessary to run the inference code. These simple mistakes were all caused by a lack of understanding around GitHub culture and technological necessities.

After the cruciality of an Nvidia GPU was recognized, the repository was moved to a Google Colab environment in order to utilize the T4 GPU (Nvidia GPU) provided. Just as tedious as the previous steps, cloud environments such as Colab, Kaggle and HuggingFace also caused issues. The GPU's provided were either deprecated or heavily limited, with Google even tracking usage of their resources across multiple accounts. The other major obstacles were related to the virtual nature of these products and the Python Packaging Systems, namely Pip, Conda and Mim. Docker-compose was out of the question after research and trials concluded builds were nearly impossible in these cloud environments. Pip being run as the root user resulted in "broken permissions and conflicting behavior with the system package manager" as quoted by the pip internal warnings. The countermeasure, virtual environments, were already out of the equation. Another possibility was using Conda and Mim, but their incompatibility with Pip itself and the virtual environment posed new errors. Multiple packaging systems being used in parallel resulted in inaccessible installations, dysfunctional initializations and activations.

Finally, it was concluded that obtaining a device with an Nvidia GPU was imperative. After obtaining a device through any means necessary, downloading the Nvidia Driver and CUDA Toolkit (of which the installation required Operating System and Computer Architecture understanding) and going through the aforementioned docker-compose related issues, the build was finally completed, yet some issues persisted. Running docker compose after having it shut down would often cause the system to get stuck, which was fixed by brute force trial and error each time. The datasets folder where the CULane images and text annotations were placed also presented with a difficulty since it was not being recognized as a directory, for which a local path had to replace the original code. The final problems were discussed on GitHub with one of the two publishers of CLRerNet, Hiroto Honda. These included problems related to the Windows Host environment and some specifications for the CULane dataset. At its current standing, the training can not be continued since the process gets killed by what is observably an overload of the CPU (the obtained laptop seems to have deteriorated heavily during its lifetime). Key takeaways from the reproduction process regarding obstacles are that the attempter should obtain aforementioned knowledge prior to tackling the reproduction, and a machine with an Nvidia GPU, solid CPU and a minimum of 50 Gigabytes of available space for the datasets (may fluctuate according to dataset used).

7.2 Experienced Challenges

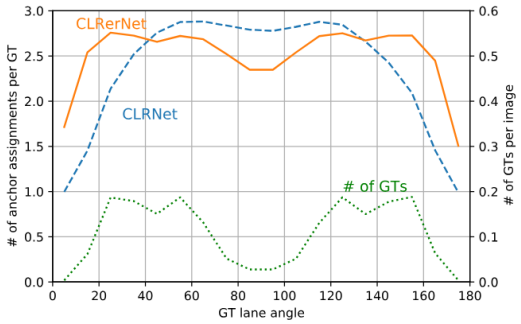


Figure 14: The average number of assignments in different angle ranges.

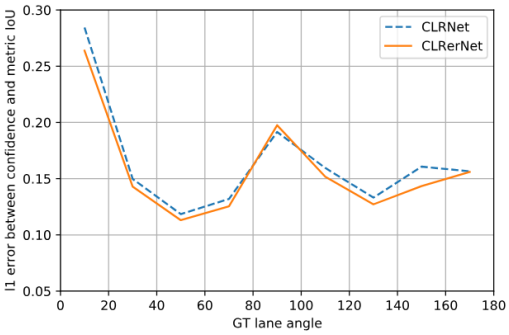


Figure 15: The average $l1$ error between confidence score and metric IoU in different angle ranges.

Figure 16: Extremely difficult cases in the CULane dataset. GTs are overlaid as orange circles.

7.2.1 Specific Image Expectation

The original Project trained with 30F video that captured from same angle with two different cars. When we test the code with different road photos, it did not process as wanted. The used dataset (Lane IoU) for training was extremely specific and has extremely specific expectations about image size (x=0-1640, y= 270-590), actually code has resize functions fort his exact issue but it still needs large images to process.

7.2.2 Non-Linearity Bias

Model breaks the lines while it is not 100% sure. So has difficulties to detect the curve. Breaking the line prevents the accident that could be caused of not seeing the curves. Using various datasets, training and testing multiple times can partially fix the issue

8 Lessons Learned

Our project provided substantial learning opportunities in various technologies and concepts:

Linux Technologies and Syntax: Mastered Linux commands, installation, and WSL, improving navigation in both Linux and Windows environments.

Docker Compose: Learned to manage multi-container Docker applications, ensuring consistent development and deployment.

Linux Tools: Enhanced proficiency with tools like Ubuntu, Vim, Nano, and Daemon for efficient system management.

Hardware and CUDA Technologies: Explored high-performance computing hardware and CUDA for GPU utilization.

PyTorch Libraries: Gained knowledge of PyTorch libraries for machine learning and deep learning.

Cloud Services: Utilized cloud services like Google Colab, Kaggle, and Hugging Face for collaborative coding and model training.

Virtual Environments: Mastered virtual environments using pip and conda for isolated development.

Lane Detection Knowledge: Acquired comprehensive understanding of lane detection techniques and models.

Loss and Cost Functions: Understood the role of loss and cost functions in optimizing machine learning models.

MMDetection: Gained experience with advanced object detection frameworks.

CLRerNet Insights: Explored the CLRerNet approach to improve lane detection confidence using LaneIoU.

This project equipped us with practical skills and theoretical knowledge, preparing us for future technical challenges.

9 Conclusion

This project centered on refining the CLRerNet architecture to bolster lane detection capabilities for autonomous vehicles and Advanced Driver Assistance Systems (ADAS). Beginning with a meticulous replication of the baseline architecture, we conducted extensive experimentation to validate its performance against the rigorous standards of the CULane dataset. Our efforts extended beyond mere replication, as we delved into the realm of architectural modifications, exploring the impact of varying backbone networks and augmenting the training data to enhance overall robustness and efficiency.

Through this iterative process, we achieved results that not only matched but often surpassed those of the original CLRerNet, particularly in challenging scenarios such as adverse weather conditions. A pivotal enhancement was the integration of LaneIoU, which introduced a novel approach to align confidence scores with metric Intersection over Union (IoU), thereby bolstering the system's detection confidence and reliability.

However, our journey was not without obstacles. We encountered challenges stemming from hardware limitations and the idiosyncrasies of specific datasets, necessitating adaptability and creativity in our approach. Despite these hurdles, the insights gleaned from this project offer valuable contributions to the ongoing pursuit of optimized lane detection systems for real-world deployment.

Looking ahead, there remains ample room for further exploration and refinement. Future research endeavors could focus on overcoming computational constraints to enable real-time implementation and explore alternative training strategies to further bolster accuracy and robustness. By continuing to push the boundaries of lane detection technology, we move closer to realizing the full potential of autonomous driving and ADAS in enhancing road safety and mobility.

References

[1] Zhe Cao, Tomas Simon, Shih-En Wei, and Yaser Sheikh. Realtime multi-person 2d pose estimation using part affinity fields. In CVPR, 2017.

[2] Haoxin Chen, Mengmeng Wang, and Yong Liu. Bsnet: Lane detection via draw b-spline curves nearby. arXiv preprint arXiv:2301.06910, abs/2301.06910, 2023.

[3] Kai Chen, Jiaqi Wang, Jiangmiao Pang, Yuhang Cao, Yu Xiong, Xiaoxiao Li, Shuyang Sun, Wansen Feng, Ziwei Liu, Jiarui Xu, Zheng Zhang, Dazhi Cheng, Chenchen Zhu, Tianheng Cheng, Qijie Zhao, Buyu Li, Xin Lu, Rui Zhu, Yue Wu, Jifeng Dai, Jingdong Wang, Jianping Shi, Wanli Ouyang, Chen Change Loy, and Dahua Lin. MMDe-

tection: Open mmlab detection toolbox and benchmark. arXiv preprint arXiv:1906.07155, 2019.

[4] Zheng Ge, Songtao Liu, Zeming Li, Osamu Yoshie, and Jian Sun. Ota: Optimal transport assignment for object detection. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pages 303– 312, June 2021.

[5] Zheng Ge, Songtao Liu, Feng Wang, Zeming Li, and Jian Sun. Yolox: Exceeding yolo series in 2021. arXiv preprint arXiv:2107.08430, 2021.

[6] Jianhua Han, Xiajun Deng, Xinyue Cai, Zhen Yang, Hang Xu, Chunjing Xu, and Xiaodan Liang. Laneformer: Objectaware row-column transformers for lane detection. Proceedings of the AAAI Conference on Artificial Intelligence, 36(1):799–807, Jun. 2022.

[7] Xinyue Cai Wei Zhang Xiaodan Liang Zhenguo Li Hang Xu, Shaoju Wang. Curvelane-nas: Unifying lane-sensitive architecture search and adaptive point blending. In ECCV, 2020.

[8] Kaiming He, Georgia Gkioxari, Piotr Dollar, and Ross Gir- ´ shick. Mask r-cnn. In 2017 IEEE International Conference on Computer Vision (ICCV), pages 2980–2988, 2017.

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), June 2016.

[10] Shaofei Huang Tianrui Hui Fei Wang Chen Qian Tianzhu Zhang Jinsheng Wang, Yinchao Ma. A keypoint-based global association network for lane detection. In CVPR, 2022.

[11] Yeongmin Ko, Younkwan Lee, Shoaib Azam, Farzeen Munir, Moongu Jeon, and Witold Pedrycz. Key points estimation and point instance segmentation approach for lane detection. IEEE Transactions on Intelligent Transportation Systems, 23(7):8949–8958, 2022.

[12] Tsung-Yi Lin, Priya Goyal, Ross B. Girshick, Kaiming He, and Piotr Dollar. Focal loss for dense object detection. ´ CoRR, abs/1708.02002, 2017.

[13] Tsung-Yi Lin, Piotr Dollar, Ross Girshick, Kaiming He, ´ Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 936–944, 2017.

[14] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollar. Focal loss for dense object detection. In Proceedings of the IEEE International Conference on Computer Vision (ICCV), Oct 2017.

[15] Lizhe Liu, Xiaohao Chen, Siyu Zhu, and Ping Tan. Condlanenet: A top-to-down lane detection framework based on conditional convolution. In Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV), pages 3773–3782, October 2021.

[16] Ruijin Liu, Zejian Yuan, Tie Liu, and Zhiliang Xiong. Endto-end lane shape prediction with transformers. In 2021 IEEE Winter Conference on Applications of Computer Vision (WACV), pages 3693–3701, 2021.

[17] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In International Conference on Learning Representations, 2019. 6

[18] Xingang Pan, Jianping Shi, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Spatial as deep: Spatial cnn for traffic scene understanding. In AAAI, February 2018.

[19] Zequn Qin, Huanyu Wang, and Xi Li. Ultra fast structureaware deep lane detection. In The European Conference on Computer Vision (ECCV), 2020.

[20] Zequn Qin, Pengyi Zhang, and Xi Li. Ultra fast deep lane detection with hybrid anchor driven ordinal classification. IEEE Transactions on Pattern Analysis and Machine Intelligence, pages 1–14, 2022.

[21] Zhan Qu, Huan Jin, Yang Zhou, Zhen Yang, and Wei Zhang. Focus on local: Detecting lane marker from bottom up via key point. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pages 14122–14130, June 2021.

[22] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In NIPS, volume 28. Curran Associates, Inc., 2015.

[23] Hamid Rezatofighi, Nathan Tsoi, JunYoung Gwak, Amir Sadeghian, Ian Reid, and Silvio Savarese. Generalized intersection over union. In The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), June 2019.

[24] Lucas Tabelini, Rodrigo Berriel, Thiago M. Paix ao, Claudine Badue, Alberto Ferreira De Souza, and Thiago OliveiraSantos. Keep your Eyes on the Lane: Real-time Attentionguided Lane Detection. In CVPR, 2021.

[25] Lucas Tabelini Torres, Rodrigo Ferreira Berriel, Thiago M. Paixao, Claudine Badue, Alberto F. De Souza, and Thi- ~ ago Oliveira-Santos. Polylanenet: Lane estimation via deep polynomial regression. In 25th International Conference on Pattern Recognition, ICPR 2020, Virtual Event / Milan, Italy, January 10-15, 2021, pages 6150–6156. IEEE, 2020.

[26] Fisher Yu, Dequan Wang, Evan Shelhamer, and Trevor Darrell. Deep layer aggregation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), June 2018.

[27] Tu Zheng, Hao Fang, Yi Zhang, Wenjian Tang, Zheng Yang, Haifeng Liu, and Deng Cai. Resa: Recurrent feature-shift aggregator for lane detection. Proceedings of the AAAI Conference on Artificial Intelligence, 35(4):3547–3554, May 2021.

[28] Tu Zheng, Yifei Huang, Yang Liu, Wenjian Tang, Zheng Yang, Deng Cai, and Xiaofei He. Clrnet: Cross layer refinement network for lane detection. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pages 898–907, June 2022.

[29] Zhaohui Zheng, Ping Wang, Wei Liu, Jinze Li, Rongguang Ye, and Dongwei Ren. Distance-iou loss: Faster and better learning for bounding box regression. In The AAAI Conference on Artificial Intelligence (AAAI), 2020. 2

[30] Zhaohui Zheng, Ping Wang, Dongwei Ren, Wei Liu, Rongguang Ye, Qinghua Hu, and Wangmeng Zuo. Enhancing geometric factors in model learning and inference for object detection and instance segmentation. 2021.

[31] CULane Dataset. Available: https://paperswithcode.com/dataset/culane.

[32] TuSimple. Available: https://www.tusimple.com/.

[33] LLAMAS Dataset. Available: https://paperswithcode.com/dataset/llamas.

[34] CurveLanes Dataset. Available: https://paperswithcode.com/dataset/curvelanes.

[35] CLRerNet GitHub Repository. Available: https://github.com/Turoad/CLRerNet.

[36] CLRNet GitHub Repository (OpenVINO). Available: https://github.com/Turoad/CLRNet.

[37] NVIDIA CUDA Documentation. Available: https://docs.nvidia.com/cuda/.

[38] CLRNet on Papers with Code. Available: https://paperswithcode.com/paper/clrnet-cross-layer-refinement-network-for.

[39] MMDetection Documentation. Available: https://mmdetection.readthedocs.io/en/latest/.

[40] pip Documentation. Available: https://pip.pypa.io/en/stable/.

[41] conda Documentation. Available: https://docs.conda.io/en/latest/.

[42] Xingang Pan, Jianping Shi, Ping Luo, Xiaogang Wang, and Xiaoou Tang. "Spatial as deep: Spatial cnn for traffic scene understanding." In AAAI, February 2018. Available: https://xingangpan.github.io/projects/CULane.html.

[43] Stack Overflow. "Conda activate and conda init fail to work in Colab as per June 2020." Available: https://stackoverflow.com/questions/62610289/conda-activate-and-conda-init-fail-to-work-in-colab-as-per-june-2020.

[44] OpenMMLab MIM. Available: https://github.com/open-mmlab/mim.

[45] MMDetection on Papers with Code. Available: https://paperswithcode.com/lib/mmdetection.

[46] Stack Overflow. "Activating a conda env inside a Docker container when using docker-compose." Available: https://stackoverflow.com/questions/53261888/activating-a-conda-env-inside-a-docker-container-when-using-docker-compose-to-st.