Generating Dynamic Kernels via Transformers for Lane Detection

Ziye Chen[1] Yu Liu[2] Mingming Gong [1] Bo Du[3] Guoqi Qian[1] Kate Smith-Miles[1] *
[1]School of Mathematics and Statistics, University of Melbourne, Australia
[2]Mach Drive, China [3]School of Computer Science, Wuhan University, Wuhan, China
ziyec1@student.unimelb.edu.au, yu.liu@mach-drive.com, mingming.gong@unimelb.edu.au,
dubo@whu.edu.cn, qguoqi@unimelb.edu.au, kate.smithmiles@gmail.com

Abstract

State-of-the-art lane detection methods often rely on specific knowledge about lanes – such as straight lines and parametric curves – to detect lane lines. While the specific knowledge can ease the modeling process, it poses challenges in handling lane lines with complex topologies (e.g., dense, forked, curved, etc.). Recently, dynamic convolution-based methods have shown promising performance by utilizing the features from some key locations of a lane line, such as the starting point, as convolutional kernels, and convoluting them with the whole feature map to detect lane lines. While such methods reduce the reliance on specific knowledge, the kernels computed from the key locations fail to capture the lane line's global structure due to its long and thin structure, leading to inaccurate detection of lane lines with complex topologies. In addition, the kernels resulting from the key locations are sensitive to occlusion and lane intersections. To overcome these limitations, we propose a transformer-based dynamic kernel generation architecture for lane detection. It utilizes a transformer to generate dynamic convolutional kernels for each lane line in the input image, and then detect these lane lines with dynamic convolution. Compared to the kernels generated from the key locations of a lane line, the kernels generated with the transformer can capture the lane line's global structure from the whole feature map, enabling them to effectively handle occlusions and lane lines with complex topologies. We evaluate our method on three lane detection benchmarks, and the results demonstrate its state-of-the-art performance. Specifically, our method achieves an F1 score of 63.40 *on OpenLane and* 88.47 *on CurveLanes, surpassing the state of the art by* 4.30 *and* 2.37 *points, respectively.*

*Corresponding Author. This work was supported in part by the Australian Research Council for funding of the ARC Training Centre in Optimisation Technologies, Integrated Methodologies and Applications (OPTIMA), under grant IC200100009. This work is also supported in part by the computational resources of the Spartan HPC system at UniMelb.

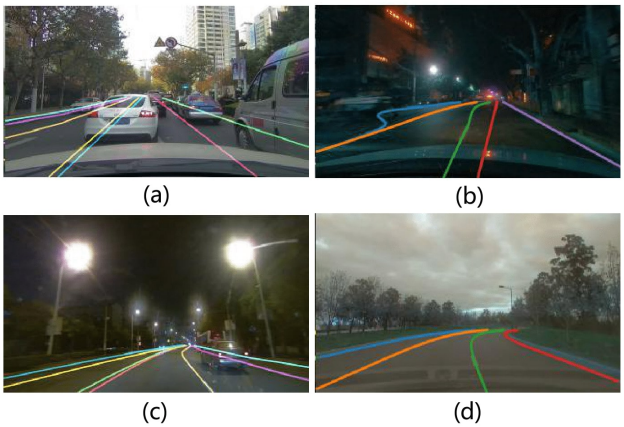

Figure 1. Examples of lanes lines in real scenes. The lane lines often have complex topologies, and are in diverse road scenarios. (a) The lane lines are blocked by vehicles. (b) The lane lines have forked and curved structures. (c) The lane lines are dense and blocked. (d) The lane lines are extremly curved. The lighting and weather conditions in these four pictures are also very different.

1. Introduction

Lane detection is a fundamental task in Autonomous Driving System (ADS) and Advanced Driver Assistance System (ADAS). It plays a crucial role in the downstreaming tasks, such as driving route planning, lane keeping assist, and adaptive cruise control. As shown in Figure 1, lane detection in real scenes is very challenging, since lane lines usually have complex topologies such as dense, curved, and forked structures, and are often blocked by vehicles and pedestrians. Furthermore, to be deployed on real-time vehicle-based systems, the lane detection algorithms need to have high running speed.

Traditional lane detection methods [16, 29, 12, 18, 4] usually rely on hand-crafted features and post-processing techniques like Hough Transform [7]. These methods are limited in representation ability and robustness, making them difficult to handle the diversity of lane lines in different road scenarios. Deep-learning-based lane detection

methods [32, 24, 2, 14, 25, 22, 8, 10, 17, 13] have recently achieved great success, thanks to the powerful representation ability of convolutional neural networks (CNN). Existing methods often rely on specific background knowledge – such as straight line anchors [32, 24, 2] and parametric curves [14, 25] – to detect lane lines.

Although the specific background knowledge can ease the modeling process, it causes difficulty in handling occlusions and lane lines with complex topologies (e.g., dense, forked, curved, etc.), as shown in Figure 1. For example, the anchor-based methods [32, 24, 2] detect lane lines by generating a set of anchors and then regressing the offsets from the anchors to the target lane lines. Unfortunately, the lane lines in real scenes are often curved or forked, which are difficult to be covered by the pre-defined anchors. The parameter-based methods [14, 25] represent lane lines with parametric curves and then detect lane lines by predicting the curve parameters. However, the extremely curved lane lines are difficult to be fit by the parametric curves.

Recently, dynamic convolution-based methods, e.g., CondLaneNet [13], have shown promising performance by considering the features from some key locations of a lane line, such as the starting point, as convolutional kernels, and convoluting them with the whole feature map to detect lane lines. By reducing the reliance on specific knowledge, these methods can achieve better results than the previous anchor-based and parameter-based methods. However, the kernels computed from the key locations fail to capture the lane line's global information due to its long and thin structure, leading to inaccurate detection of lane lines with complex topologies. In addition, the kernels resulting from the key locations are sensitive to occlusions and lane intersections. For instance, detecting starting points can be challenging when they are obscured by vehicles or pedestrians. Additionally, it is difficult to differentiate between multiple lane lines that share the same starting point.

To overcome this limitation, we propose a transformer-based dynamic kernel generation architecture for lane detection, which further reduces the reliance on specific knowledge. It utilizes a transformer to generate dynamic convolutional kernels for each lane line in the input image, and then detects these lane lines by convoluting these kernels with the whole feature map. Compared to the kernels generated from some key locations of a lane line, the kernels generated with transformer can capture the lane line's global information from the whole feature map, enabling them to effectively handle occlusions and lane lines with complex topologies (as shown in Fig. 2(b) and (c)).

As illustrated in Fig. 2(a), the transformer employs a set of learnable parameter vectors, called lane queries, as lane templates to search for lane points throughout the whole feature map. It then fuses the features of the searched lane points by a weighted sum to generate dynamic kernels for each lane line. Thus, the generated kernels can capture the global information of the lane lines. The lane queries represent different lane prototypes, which are learned from annotated data and can approximate various lane lines. There are two sets of dynamic kernels that generate the heat map and offset map for each lane line, respectively. The final lane points are obtained from the heat maps and offset maps after a post-processing. During training, a bipartite matching loss is calculated between the predicted and ground-truth lane lines based on Hungarian algorithm [9]. We test our method on three lane detection benchmarks, and the results demonstrate its state-of-the-art performance.

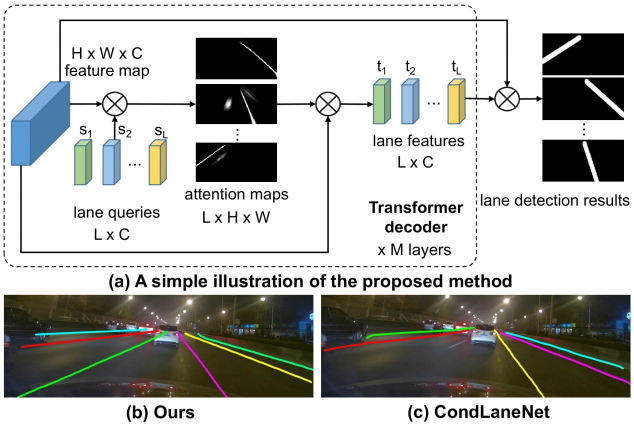

Figure 2. A simple illustration of the proposed method, and the comparison results between our method and CondLaneNet[13] uner the case of occlusion. In (a), the dashed box part corresponds to the transformer decoder part in Fig. 3, which consists of M layers. In each layer, the lane queries are updated with the lane features, and the lane features in the final layer is transformed to the dynamic kernels via MLPs.

2. Related Work

2.1. Anchor-based Methods

The anchor-based methods [11, 2, 24, 23, 32] detect lane lines by generating a set of pre-defined line anchors first, and then extracting the anchor features with pooling operation, and finally regressing the offsets from the anchors to the target lane lines with the extracted features. For example, LaneATT [24] uses the straight lines at different locations with different rotation angles as anchors, whose features are obtained by pooling and enhanced with an attention module, and used for classification and offset regression. SGNet [23] proposes a vanishing point guided anchor generator to produce effective anchors, whose features are enhanced with structural guidance including pixel-level perception, lane-level relation, and image-level attention. CLRNet [32] uses equally-spaced 2D-points as anchors, whose features are obtained with a RoIGather

module that exploits the global contextual information. For these methods, the Non-maximum Suppression (NMS) post-processing is needed to remove duplicate predictions, which decreases the efficiency. Also, the fixed lane anchors can hardly deal with the lane lines with complex topologies, such as the curved and forked lane lines, which influences the lane detection performance.

2.2. Parameter-based Methods

Parameter-based methods [14, 25] represent lane lines with parametric curves and detect lane lines by predicting the corresponding curve parameters. It removes the complex post-processing procedures, such as pixel clustering and non-maximum suppression, thus can achieve fast speed. For example, PolyLaneNet [25] applies a convolutional neural network (CNN) to directly output polynomial coefficients, vertical offsets, and confidence scores for a fixed number of lane lines in left-to-right order. LSTR [14] applies transformer to directly decode the parameters of a lane shape model for each lane line in an image. The lane shape model takes the road structures and camera pose into consideration to better model lane lines. However, in some road scenarios, the lane lines are extremely curved, which are difficult to be represented with cubic curves. Increasing the order of the parametric curves can partially solve this problem; however, the prediction of these high-order terms is difficult and not robust, since a small prediction error on these terms will cause a large change in the shapes of lane lines. Therefore, the parameter-based methods have not exceeded other lane detection methods in accuracy.

2.3. Segmentation-based Methods

Segmentation-based methods [22, 8, 10, 17, 19, 6, 21, 31, 13] detect lane lines with pixel-wise classification. Some of them [22, 8, 10, 17] apply semantic segmentation to detect lane points, which are then processed further to extract lane lines. For example, FOLOLane [22] identify the lane points first, then refines their location in a local range, and then correlate the lane points of the same lane line. HDMapNet [10] and LaneNet [17] identify the lane points first, and then learn the embeddings for these points to cluster them into different lane lines. However, these methods may fail when the lane lines are overlapped, such as the forked lane lines. Some of them [19, 6, 21, 31, 13] apply instance segmentation to detect lane lines. For example, UFLD [21] and RESA [31] label lane lines into a fixed number of classes in left-to-right order, and then directly classify the pixels into different lane lines. However, the multi-instance classification is difficult. Closely related to ours is CondLaneNet [13], which detects the starting points of lane lines first, then generates dynamic kernels from the features of the starting points, and then convolute the kernels with the whole feature map to detect the lane lines. However, this method may fail when the starting points are blocked by vehicles and pedestrians, or shared by multiple lane lines. And the kernels generated from starting points lacking the lane line's global information, making this method not robust for handling lane lines with complex topologies.

3. Method

In this section, we present the proposed transformer-based dynamic kernel generation architecture for lane detection. We first present the overall framework, and then describe each component in detail, including a dynamic kernel head, a lane detection head and a bipartite matching loss.

3.1. Overall Framework

The overall framework is shown in Figure 3. It starts with a CNN backbone to extract a feature map from the input image. Then, the dynamic kernel head uses a transformer to generate dynamic convolutional kernels for each lane line from the feature map. After that, the lane detection head detects lane lines by convoluting the dynamic kernels with the feature map. There are two sets of dynamic kernels that generate the heat map and the offset map for each lane line, respectively. The framework also predicts a vertical range and an object score for each lane line. Finally, the lane points are obtained from the heat maps, offset maps, vertical ranges and object scores after a post-processing. To train the model, a bipartite matching loss is calculated between the predicted and ground-truth lane lines. Compared to the kernels generated from some key locations of a lane line, the kernels generated with transformer can capture the lane line's global information from the whole feature map, enabling them to effectively handle occlusions and lane lines with complex topologies.

3.2. Dynamic Kernel Head

Here we describe how to generate the dynamic convolutional kernels for each lane line in the input image via a transformer. As shown in Figure 3, given an input image $\mathbf{X} \in \mathbb{R}^{H_0 \times W_0 \times 3}$, we first adopt a CNN backbone to extract a feature map $\mathbf{F} \in \mathbb{R}^{H \times W \times C}$, where H, W and C are the height, width, and channels of $\mathbf{F}$, respectively. Then the feature map $\mathbf{F}$ is added with a position embedding $\mathbf{E} \in \mathbb{R}^{H \times W \times C}$ that encodes the 2D pixel locations with cosine and sine functions [20], and then flattened into a sequence $\mathbf{I} \in \mathbb{R}^{HW \times C}$, which is taken as the input of a transformer. The position embedding $\mathbf{E}$ is used to avoid the permutation invariance.

The transformer includes an encoder and a decoder, where each of them consists of several stacked layers. The encoder takes the feature sequence $\mathbf{I}$ as input, captures the most relevant input features for each feature in $\mathbf{I}$ through a self-attention mechanism [27], and outputs a feature sequence $\mathbf{M} \in \mathbb{R}^{HW \times C}$. Then we define a lane query se-

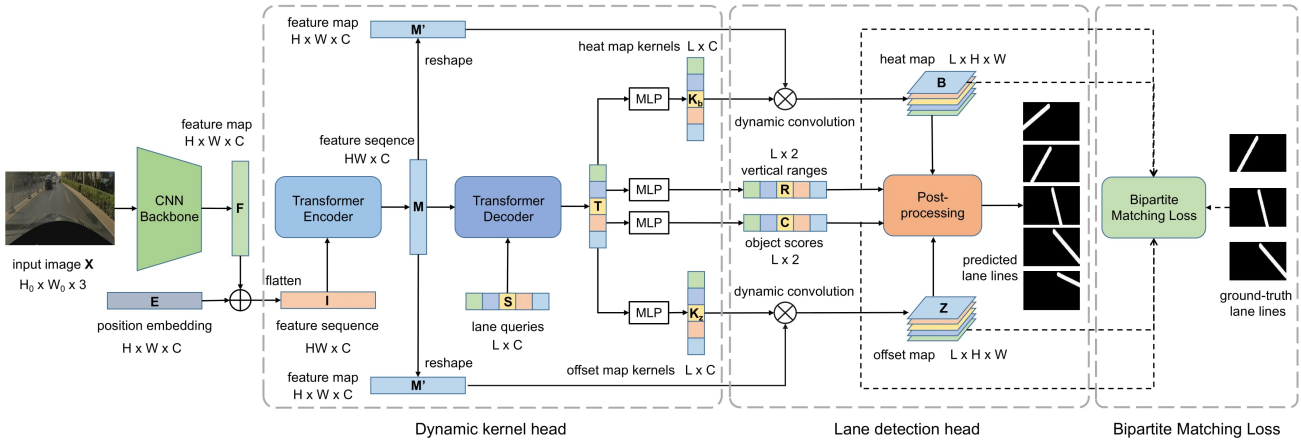

Figure 3. Overview of the lane detection framework. It includes a CNN backbone to extract a feature map from the input image, a dynamic kernel head to generate the dynamic convolutional kernels for lane lines via a transformer, a lane detection head to detect lane lines with dynamic convolution, and a bipartite matching loss for model training. The **dotted** arrow parts are only engaged in training.

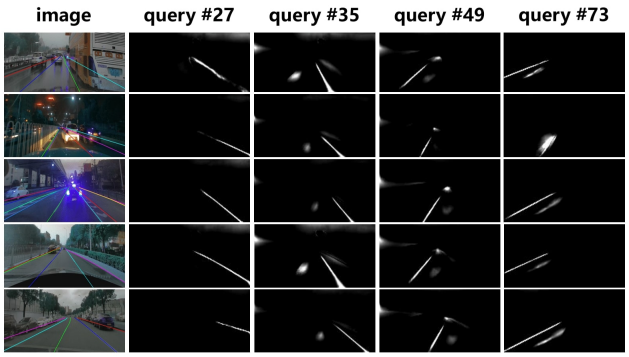

Figure 4. Visualization of attention maps between the the lane query sequence $\mathbf{S}$ and the image feature sequence $\mathbf{M}$. We select 4 out of 80 lane queries for visualization.

quence $\mathbf{S} \in \mathbb{R}^{L\times C}$, which is composed of L learnable parameter vectors of length C, representing L different prototypes of lane lines. Then the decoder takes the feature sequence $\mathbf{M}$ and the lane query sequence $\mathbf{S}$ as input, captures the most relevant features in $\mathbf{M}$ for each lane query in $\mathbf{S}$ through a cross-attention mechanism [27], and outputs a lane feature sequence $\mathbf{T} \in \mathbb{R}^{L\times C}$.

The cross-attention mechanism between the lane query sequence $\mathbf{S}$ and the feature sequence $\mathbf{M}$ is implemented as follows:

$$\mathbf{A}_{i,j} = \frac{\exp(\mathbf{Q}_i^T \mathbf{K}_j)}{\sum_{l=1}^{HW} \exp(\mathbf{Q}_i^T \mathbf{K}_l)}, \quad \mathbf{T}_i = g(\sum_{j=1}^{HW} \mathbf{A}_{i,j}\mathbf{V}_j), \quad (1)$$

where $\mathbf{Q}$ is a query sequence obtained by a linear transformation of $\mathbf{S}$; $\mathbf{K}$ and $\mathbf{V}$ are key and value sequences obtained by linear transformations of $\mathbf{M}$, respectively; $\mathbf{Q}_i$ and $\mathbf{K}_j$ are the features at the i-th and j-th position of $\mathbf{Q}$ and $\mathbf{K}$, respectively; $\mathbf{A}$ is an attention map which describes the pair-wise relationship $\mathbf{A}_{i,j}$ between $\mathbf{Q}_i$ and $\mathbf{K}_j$; $\mathbf{T}_i$ is the i-th output lane feature corresponds to the i-th lane query $\mathbf{S}_i$; $g(\cdot)$ is a non-linear transformation function.

It is worth noting that $\mathbf{M}$ is a sequence of pixel features enhanced with global contextual information, and $\mathbf{S}$ is a sequence of different lane prototypes. The cross-attention mechanism between $\mathbf{S}$ and $\mathbf{M}$ captures the most correlated pixel features in $\mathbf{M}$ for each lane prototype $\mathbf{S}_i$, fuses them by a weighted sum according to the attention map $\mathbf{A}_i$, and then obtains the lane feature $\mathbf{T}_i$. Thus, $\mathbf{T}_i$ can capture the lane line's global information specified by $\mathbf{S}_i$. To further illustrate this, we select 4 out of total 80 lane queries, and visualize the attention map $\mathbf{A}_i$ of each lane query $\mathbf{S}_i$ in Figure 4. We find that each lane query corresponds to a lane line at a specific location, and the corresponding lane pixels are highlighted. This demonstrates that each lane query $\mathbf{S}_i$ is a lane prototype with a specific location, and the cross-attention mechanism can find the most related lane pixels for the lane query $\mathbf{S}_i$. Thus, by applying a weighted sum to the features of these lane pixels, we can obtain the lane feature $\mathbf{T}_i$. Compared to the kernels generated from some key locations of a lane line, the kernels generated from $\mathbf{T}$ can capture the lane line's global information from the whole feature map, enabling them to effectively handle occlusions and lane lines with complex topologies.

Then we apply two multi-layer perceptrons (MLPs) to $\mathbf{T}$, which generates two sets of dynamic convolutional kernels, respectively, one is $\mathbf{K}_b \in \mathbb{R}^{L\times C}$ for heat map generation, another is $\mathbf{K}_z \in \mathbb{R}^{L\times C}$ for offset map generation. Different from the lane queries $\mathbf{S}$, which are shared by all images, the dynamic kernels $\mathbf{K}_b$ and $\mathbf{K}_z$ contain image-specific information. In addition, we apply another two MLPs to $\mathbf{T}$, which generates vertical ranges $\mathbf{R} \in \mathbb{R}^{L\times 2}$ and object scores $\mathbf{C} \in \mathbb{R}^{L\times 2}$, where $\mathbf{R}$ predicts the start and end rows

of each lane line, and $\mathbf{C}$ predicts the probabilities of foreground and background (hitting a lane line or not) for each lane query $\mathbf{S}_i$.

3.3. Lane Detection Head

Here we describe how to apply the generated dynamic kernels to detect lane lines in a form of dynamic convolution. We reshape the output feature sequence $\mathbf{M} \in \mathbb{R}^{HW \times C}$ of the transformer encoder into a feature map $\mathbf{M'} \in \mathbb{R}^{H \times W \times C}$. Then we apply the dynamic kernels $\mathbf{K}_b \in \mathbb{R}^{L \times C}$ and $\mathbf{K}_z \in \mathbb{R}^{L \times C}$ to convolve with the feature map $\mathbf{M'}$, and obtain the heat map $\mathbf{B} \in \mathbb{R}^{L \times H \times W}$ and the offset map $\mathbf{Z} \in \mathbb{R}^{L \times H \times W}$. The heat map $\mathbf{B}$ is further processed with a row-wise softmax normalization as follows:

$$\mathbf{B}_{ijk} = \exp(\mathbf{B}_{ijk}) / \sum_{m=1}^{W-1} \exp(\mathbf{B}_{ijm}), \tag{2}$$

where $\mathbf{B}_{ijk}$ is the predicted pixel logit/probability of the i-th predicted lane line at the j-row and k-th column of $\mathbf{B}$. For simplicity, we use the same name $\mathbf{B}$ for the output.

As a result, each lane feature $\mathbf{T}_i$ corresponds to a heat map $\mathbf{B}_i \in \mathbb{R}^{H \times W}$ and an offset map $\mathbf{Z}_i \in \mathbb{R}^{H \times W}$. As shown in Figure 5, the heat map predicts the probability of each pixel being a lane point (foreground), and the offset map predicts the horizontal offset from each pixel to the lane point in the same row (each row has one lane point at most for each $\mathbf{T}_i$). The final lane points are obtained from the heat map and the offset map after a post-processing shown as follows:

$$\begin{aligned} n = \left[\sum_{k=0}^{W-1} \mathbf{B}_{ijk} \cdot k \right], \quad \mathbf{P}_{ij} = (n + \mathbf{Z}_{ijn}, j), \\ \mathbf{L}_i = \{\mathbf{P}_{ij} | \mathbf{R}_{i0} \le j \le \mathbf{R}_{i1}\}\}, \quad \mathbf{Y} = \{\mathbf{L}_i | \mathbf{C}_{i1} \ge t\}, \end{aligned} \tag{3}$$

where n is the predicted coarse abscissa of the lane point at the j-th row of the i-th predicted lane line; $\mathbf{Z}_{ijn}$ is the predicted pixel horizontal offset of the i-th predicted lane line at the j-th row and n-th column of $\mathbf{Z}$; $\mathbf{P}_{ij}$ is the refined abscissa of the lane point at the j-th row of the i-th predicted lane line; $\mathbf{L}_i$ is the set of the predicted lane points of the i-th predicted lane line, where only the lane points between the start row $\mathbf{R}_{i0}$ and the end row $\mathbf{R}_{i1}$ are kept; $\mathbf{Y}$ is the set of the predicted lane lines, where only the lane lines with foreground probability $\mathbf{C}_{i1}$ higher than a threshold t are retained.

3.4. Bipartite Matching Loss

Here we introduce the design of loss functions. First, we need to compute a pair-wise matching cost $L_{match}(\mathbf{Y}_i, \mathbf{Y}_j^*)$ between L predicted lane lines $\mathbf{Y} = \{\mathbf{Y}_i\}_{i=1}^{L}$ and M ground-truth lane lines $\mathbf{Y}^* = \{\mathbf{Y}_i^*\}_{i=1}^{M}$, including an object cost, a heat map cost, an offset map cost and a vertical

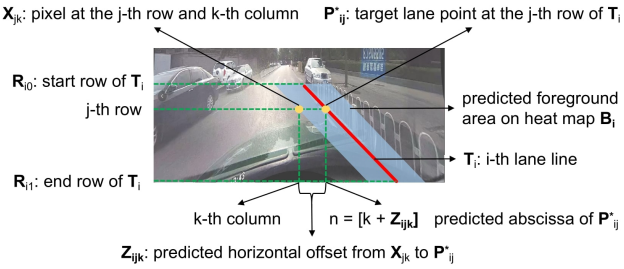

Figure 5. The illustration of the heat map and offset map of each lane line.

range cost. The object cost is defined as follows:

$$L_{obj}(\mathbf{Y}_i, \mathbf{Y}_j^*) = -log(\mathbf{S}_{i1}), \tag{4}$$

where $\mathbf{S}_{i1}$ is the predicted foreground probability of the i-th predicted lane line. The heat map cost is defined as follows:

$$L_{heat}(\mathbf{Y}_i, \mathbf{Y}_j^*) = \frac{1}{\mathbf{R}_{j1}^* - \mathbf{R}_{j0}^*} \sum_{k=\mathbf{R}_{j0}^*}^{\mathbf{R}_{j1}^*} \| \sum_{m=0}^{W-1} \mathbf{B}_{ikm} \cdot m - \mathbf{P}_{jk0}^* \|_1, \tag{5}$$

where $\mathbf{B}_{ikm}$ is the predicted foreground probability of the pixel at the k-th row and m-th column of the i-th predicted lane line, $\mathbf{P}_{jk0}^*$ is the abscissa of the lane point at the k-th row of the j-th ground-truth lane line, $\mathbf{R}_{j0}^*$ and $\mathbf{R}_{j1}^*$ are the start row and end row of the j-th ground-truth lane line. The heat map cost is only valid between $\mathbf{R}_{j0}^*$ and $\mathbf{R}_{j1}^*$. The offset cost $L_{off}(\mathbf{Y}_i, \mathbf{Y}_j^*)$ is defined as follows:

$$\frac{1}{W(\mathbf{R}_{j1}^* - \mathbf{R}_{j0}^*)} \sum_{k=\mathbf{R}_{j0}^*}^{\mathbf{R}_{j1}^*} \sum_{m=0}^{W-1} \| \mathbf{Z}_{ikm} + m - \mathbf{P}_{jk0}^* \|_1, \tag{6}$$

where $\mathbf{Z}_{ikm}$ is the predicted horizontal offset of the pixel at the k-th row and m-th column of the i-th predicted lane line, $\mathbf{Z}_{ikm} + m$ is the abscissa of the predicted lane point at the k-th row. The offset map cost is also only valid between $\mathbf{R}_{j0}^*$ and $\mathbf{R}_{j1}^*$. The vertical range cost is defined as follows:

$$L_{rng}(\mathbf{Y}_i, \mathbf{Y}_j^*) = \|\mathbf{R}_{i0} - \mathbf{R}_{j0}^*\|_1 + \|\mathbf{R}_{i1} - \mathbf{R}_{j1}^*\|_1, \tag{7}$$

where $\mathbf{R}_{i0}$ and $\mathbf{R}_{i1}$ are the predicted start row and end row of the i-th predicted lane line. The final pair-wise matching cost $L_{match}(y_i, y_j^*)$ is defined as follows:

$$\begin{aligned} L_{match}(\mathbf{Y}_i, \mathbf{Y}_j^*) = \lambda_{obj} L_{obj}(\mathbf{Y}_i, \mathbf{Y}_j^*) + \lambda_{heat} L_{heat}(\mathbf{Y}_i, \mathbf{Y}_j^*) \\ + \lambda_{off} L_{off}(\mathbf{Y}_i, \mathbf{Y}_j^*) + \lambda_{rng} L_{rng}(\mathbf{Y}_i, \mathbf{Y}_j^*), \end{aligned} \tag{8}$$

where λ_{obj}, λ_{heat}, λ_{off} and λ_{rng} are the balance weights for L_{obj}, L_{heat}, L_{off} and L_{rng}, respectively. Then we find an optimal injective function $z : \{\mathbf{Y}_i\}_{i=1}^{L} \rightarrow \{\mathbf{Y}_j^*\}_{j=1}^{M}$,

where $z(j)$ is the index of the prediction assigned to the j-th ground-truth, by minimizing the matching cost as follows:

$$\arg\min_{z} \hat{z} = \sum_{j=1}^{N} L_{match}(y_{z(j)}, y_j^*). \tag{9}$$

This objective function can be solved by the Hungarian algorithm [9]. After obtaining the injective mapping z, we can compute the final loss function as follows:

$$\begin{aligned} loss = & \frac{1}{N}\sum_{j=1}^{N}\big[\lambda_{obj}L_{obj}(\mathbf{Y}_{z(j)}, \mathbf{Y}_j^*) + \lambda_{heat}L_{heat}(\mathbf{Y}_{z(j)}, \mathbf{Y}_j^*) \\ & + \lambda_{off}L_{off}(\mathbf{Y}_{z(j)}, \mathbf{Y}_j^*) + \lambda_{rng}L_{rng}(\mathbf{Y}_{z(j)}, \mathbf{Y}_j^*)\big] \\ & + \frac{1}{L-N}\sum_{i \notin V}\lambda_{obj}L_{obj}(\mathbf{Y}_i, \phi), \end{aligned} \tag{10}$$

where $V = \{z(j)\}_{j=1}^{N}$ is the index set of the predicted lane lines matched to the ground-truth lane lines, $L_{obj}(\mathbf{Y}_i, \phi) = -log(\mathbf{C}_{i0})$, where $\mathbf{C}_{i0}$ is the predicted background probability for the i-th predicted lane line.

4. Experiment

4.1. Datasets

To extensively evaluate the proposed method, we conduct experiments on four representative lane detection benchmarks: OpenLane [1], CULane [19], CurveLanes [30] and TuSimple [26]. OpenLane contains contains $160K$ and $40K$ images for training and validation sets, respectively. The validation set consists of six different scenarios, including curve, intersection, night, extreme weather, merge and split, and up and down. It annotates 14 lane categories, including road edges, double yellow solid lanes, and so on. CurveLanes contains 100K, 20K, and 30K images for training, validation, and testing, respectively, including a lot of difficult scenarios such as curved, forked and dense lane lines. CULane contains 88880, 9675, and 34680 images for training, validation, and test sets, respectively. The test set consists of nine different scenarios, including normal, crowd, curve, dazzle light, night, no line, shadow, and arrow in the urban area. Tusimple consists of 3268 images for training, 358 images for validation, and 2782 images for testing, which mainly focus on highway driving scenes.

4.2. Evaluation Metrics

Following [13] and [1], we adopt F1 measure as the metric for OpenLane, CurveLanes and CULane datasets. We apply intersection-over-union (IoU) between the predicted lane line and the ground-truth lane line to judge whether a sample is true positive (TP) or false positive (FP) or false negative (FN). The IoU of two lane lines is defined as the IoU of their masks with a fixed line width (30 pixels). The F1 measure is calculated as follows:

$$\begin{aligned} Precision = \frac{N_{TP}}{N_{TP}+N_{FP}}, \quad Recall = \frac{N_{TP}}{N_{TP}+N_{FN}}, \\ F1 = \frac{2 \times Precision \times Recall}{Precision + Recall}. \end{aligned} \tag{11}$$

For TuSimple dataset, the main evaluation metric is accuracy, which is defined as follows:

$$accuracy = \frac{\sum_{clip} C_{clip}}{\sum_{clip} S_{clip}}, \tag{12}$$

where C_{clip} is the number of the correctly predicted lane points in a clip, and S_{clip} is the total number of lane points in a clip. A lane point prediction is considered correct when it is within 20 pixels of the ground truth. The predicted lane line with accuracy greater than 85% is considered a true positive (TP), otherwise a false positive (FP). Besides, false positive rate (FPR), false negative rate (FNR), and F1 score are also reported, where $FPR = \frac{N_{FP}}{N_{pred}}$, $FNR = \frac{N_{FN}}{N_{gt}}$.

4.3. Implementation Details

We adopt ResNet [5] with the pretrained weights from ImageNet [3] as the CNN backbone. The input images are augmented with random horizontal flipping and random affine transformation including translation, rotation, and scaling. Optimization is done by AdamW [15] with betas of 0.9 and 0.999, and weight decay of $1e^{-4}$. The batch size is set to 16. We train the model for 50 epochs. The base learning rate is initialized at $1e^{-4}$ and decayed to $1e^{-5}$ after 40 epochs. The learning rate for the backbone is set to 0.1 times of the base learning rate. The number of the lane queries L is set to 80, and the number of transformer encoder and decoder layers are set to 2 and 4, respectively. The weights λ_{obj}, λ_{heat}, λ_{off} and λ_{rng} are set to 5, 1, 1, and 10, respectively, to balance the scales of different losses, i.e., bring the losses to the same scale. The object score threshold t to keep the predicted lane lines is set to 0.7. The results are reported on the test set for CULane and TuSimple. For CurveLanes, we report the results on the validation set following [30] and [13].

4.4. Results

Performance on OpenLane Dataset. The comparison results on OpenLane are shown in Table 1. Using ResNet-18, ResNet-34 and ResNet-101 as backbones, our method achieves F1 scores of 60.1, 62.0 and 63.4, surpassing those of CondLaneNet by 7.8, 7.0 and 4.3 points, respectively. As for F1 scores under different scenarios, our method achieves the best performance in all of the six scenarios, showing the robustness of our method. For the "Curve", "Intersection" and "Merge & Split" scenarios, using ResNet-18 as

Method	Backbone	All	Up & Down	Curve	Extreme Weather	Night	Intersection	Merge & Split	FPS	GFlops
LaneATT [24]	ResNet-18	28.3	25.3	25.8	32.0	27.6	14.0	24.3	153	**9.3**
LaneATT [24]	ResNet-34	31.0	28.3	27.4	34.7	30.2	17.0	26.5	129	18.0
PersFormer [1]	EfficientNet-B7	42.0	40.7	46.3	43.7	36.1	28.9	41.2	-	-
CondLaneNet [13]	ResNet-18	52.3	55.3	57.5	45.8	46.6	48.4	45.5	**173**	10.2
CondLaneNet [13]	ResNet-34	55.0	58.5	59.4	49.2	48.6	50.7	47.8	128	19.6
CondLaneNet [13]	ResNet-101	59.1	62.1	62.9	54.7	51.0	55.7	52.3	47	44.8
Ours	ResNet-18	60.1	56.2	63.9	51.5	51.0	54.5	62.5	105	13.7
Ours	ResNet-34	62.0	59.1	65.4	53.4	54.1	57.3	63.2	91	23.2
Ours	ResNet-101	**63.4**	**62.2**	**67.0**	**55.1**	**57.3**	**58.5**	**65.8**	45	50.2

Table 1. Comparison of different methods on OpenLane dataset. We report the F-score for the whole validation set and under different scenarios, including curve, intersection, night, extreme weather, merge and split, and up and down.

Method	Backbone	F1 (%)	Normal	Crowded	Dazzle	Shadow	No line	Arrow	Curve	Cross	Night	FPS	GFlops
LaneATT [24]	ResNet-18	75.13	91.17	72.71	65.82	68.03	49.13	87.82	63.75	**1020**	68.58	153	**9.3**
LaneATT [24]	ResNet-34	76.68	92.14	75.03	66.47	78.15	49.39	88.38	67.72	1330	70.72	129	18.0
LaneATT [24]	ResNet-122	77.02	91.74	76.16	69.47	76.31	50.46	86.29	64.05	1264	70.81	20	70.5
FOLOLane [22]	ERFNet	78.80	92.70	77.80	75.20	79.30	52.10	89.00	69.40	1569	74.50	40	-
CondLaneNet [13]	ResNet-18	78.14	92.87	75.79	70.72	80.01	52.39	89.37	72.40	1364	73.23	**173**	10.2
CondLaneNet [13]	ResNet-34	78.74	93.38	77.14	71.17	79.93	51.85	89.89	73.88	1387	73.92	128	19.6
CondLaneNet [13]	ResNet-101	79.48	93.47	77.44	70.93	80.91	54.13	90.16	75.21	1201	74.80	47	44.8
GANet [28]	ResNet-18	78.79	93.24	77.16	71.24	77.88	53.59	89.62	75.92	1240	72.75	153	-
GANet [28]	ResNet-34	79.39	93.73	77.92	71.64	79.49	52.63	90.37	76.32	1368	73.67	127	-
GANet [28]	ResNet-101	79.63	93.67	78.66	71.82	78.32	53.38	89.86	**77.37**	1352	73.85	63	-
Ours	ResNet-18	80.36	94.11	79.17	73.55	80.39	54.41	90.37	75.89	1214	75.39	105	13.7
Ours	ResNet-34	80.55	94.12	79.72	**77.02**	**82.51**	53.76	90.59	76.65	1370	**75.57**	91	23.2
Ours	ResNet-101	**80.77**	**94.17**	**79.90**	75.43	80.99	**55.00**	**90.97**	76.87	1047	75.11	45	50.2

Table 2. Comparison of different methods on CULane dataset. We also report the F1 score for the whole test set and under different scenarios. For the "Cross" scenario, we report the number of false positives instead of F1, since the images in the "Cross" scenario have no lane lines. For a fair comparison, the FPS of above methods are measured under the same machine and conditions.

Method	Backbone	F1 (%)	Prec.(%)	Rec.(%)
SCNN [19]	VGG-16	65.02	76.13	56.74
Enet-SAD [6]	ENet	50.31	63.60	41.60
PointLaneNet [2]	GoogLeNet	78.47	86.33	72.91
CurveLane [30]	Searched-S	81.12	**93.58**	71.59
CurveLane [30]	Searched-M	81.80	93.49	72.71
CurveLane [30]	Searched-L	82.29	91.11	75.03
CondLaneNet [13]	ResNet-18	85.09	87.75	82.58
CondLaneNet [13]	ResNet-34	85.92	88.29	83.68
CondLaneNet [13]	ResNet-101	86.10	88.98	83.41
Ours	ResNet-18	87.99	90.90	85.27
Ours	ResNet-34	88.23	91.24	85.41
Ours	ResNet-101	**88.47**	91.32	**85.80**

Table 3. Comparison of different methods on CurveLanes dataset. The backbones named Searched-S, Searched-M, and Searched-L are the searched architectures with capacities of small, medium and large in [30].

backbone, our method achieves F1 scores of 63.9, 54.5 and 62.5, respectively, surpassing those of CondLaneNet by 6.4, 6.1 and 17.0 points, respectively. The results demonstrate that the proposed transformer-based dynamic kernel generation architecture can deal with the lane lines with complex topologies very well. This is because the dynamic kernels generated with transformer can capture the lane line's global information from the whole feature map, which enabling them to distinguish different lane lines better than the kernels generated from some key locations of a lane line. As for speed, the ResNet-18 version of our method achieves 105 FPS and 13.7 GFlops with a F1 score of 60.1, which ensures real-time efficiency with a good performance.

Performance on CurveLanes Dataset. The comparison results on CurveLanes are shown in Table 3. CurveLanes contains lane lines with complex topologies and occlusions, such as the curved, forked, dense and blocked lane lines. Using ResNet-18, ResNet-34 and ResNet-101 as backbones, our method achieves F1 scores of 87.99, 88.23 and 88.47, surpassing those of CondLaneNet by 2.90, 2.31, 2.37 points, respectively. We also show some qualitative comparison results on CurveLanes in Figure 6, where the results are divided into 4 categories, including the curved, forked, dense and blocked lane lines. The comparison results demonstrate that our method can cope with occlusions and lane lines with complex topologies very well.

Performance on CULane Dataset. The comparison results on CULane are shown in Table 2. Our method achieves a new state-of-the-art result of a 80.77 F1 score. Using ResNet-18, ResNet-34 and ResNet-101 as backbones, our method achieves F1 scores of 80.36, 80.55 and 80.77, surpassing those of CondLaneNet by 2.22, 1.81 and 1.29 points, respectively. As for F1 scores under different scenarios, our method achieves the best performance in seven of nine scenarios, showing the robustness of our method. For the "Curve" scenario, using ResNet-18, ResNet-34 and

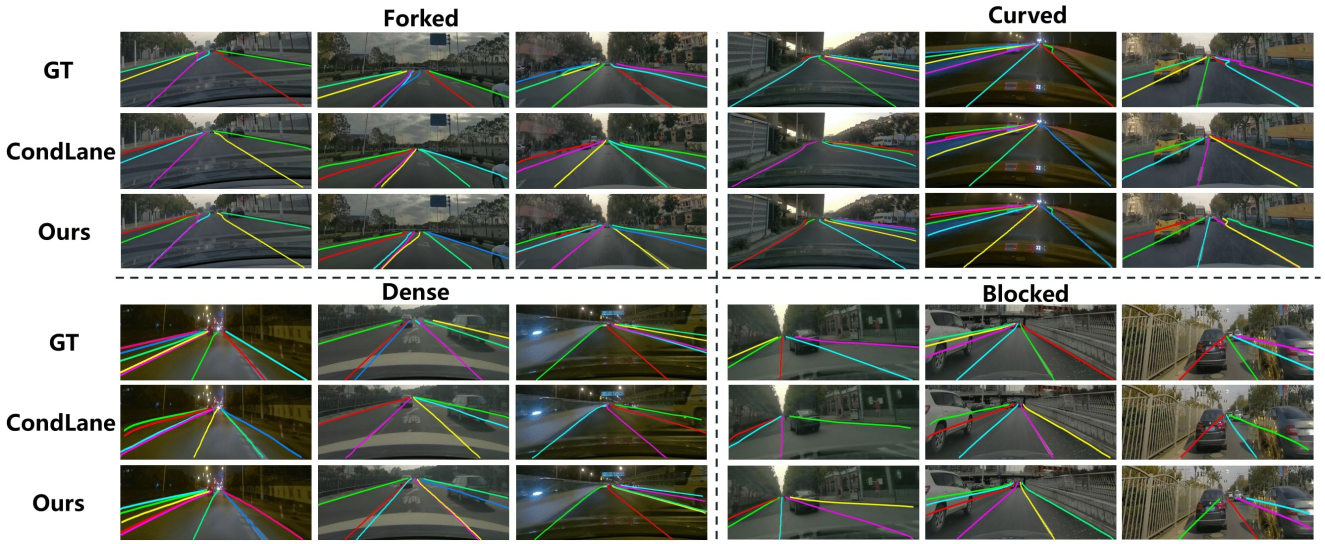

Figure 6. Visualization results on CurveLanes. We choose CondLaneNet [13] as our comparison method, which was the previous SOTA on CurveLanes. We visualize the comparison results under different scenarios, including the forked, curved, dense and blocked lane lines. Different lane lines are represented by different colors.

ResNet-101 as backbones, our method achieves F1 scores of 75.89, 76.65 and 76.87, respectively, surpassing those of CondLaneNet by 3.49, 2.77 and 1.66 points, respectively. The results demonstrates that our method can deal with the lane lines with complex topologies well.

Performance on Tusimple Dataset. The results on Tusimple is shown in Table 4. The performance gap between different methods on this dataset is not obvious due to the small amount of data and the relatively simple scenes.

Method	Backbone	F1(%)	Acc(%)	FPR(%)	FNR(%)
LaneATT [24]	ResNet-18	96.71	95.57	3.56	3.01
LaneATT [24]	ResNet-34	96.77	95.63	3.53	2.92
LaneATT [24]	ResNet-122	96.06	96.10	5.64	**2.17**
FOLOLane [22]	ERFNet	96.59	**96.92**	4.47	2.28
CondLaneNet [13]	ResNet-18	97.01	95.48	2.18	3.80
CondLaneNet [13]	ResNet-34	96.98	95.37	2.20	3.82
CondLaneNet [13]	ResNet-101	97.24	96.54	2.01	3.50
GANet [28]	ResNet-18	97.71	95.95	1.97	2.62
GANet [28]	ResNet-34	97.68	95.87	1.99	2.64
GANet [28]	ResNet-101	97.45	96.44	2.63	2.47
Ours	ResNet-18	97.71	96.06	1.79	2.82
Ours	ResNet-34	97.64	96.06	1.84	2.92
Ours	ResNet-101	**97.94**	96.02	**1.55**	2.56

Table 4. Comparison of different methods on Tusimple dataset. In addition to accuracy, we also report FPR, FNR and F1 score.

4.5. Ablation Studies

Number of lane queries. Here we investigate the influence of different number of lane queries. As shown in Table 5, when we increase the number of lane queries from 20 to 80, the performance is improved. This demonstrates that increasing the number of lane queries is beneficial to capture lane lines with diverse topologies. When we further increase the number of lane queries from 80 to 100, the performance only increases slightly, which is because too many lane qeuries can cause redundancy.

Num of lane queries	CuLane			CurveLanes		
	F1	Prec.	Rec.	F1	Prec.	Rec.
20	80.13	86.59	**74.56**	87.37	89.30	**85.52**
40	80.24	87.01	74.44	87.57	89.82	85.43
80	80.36	87.44	74.35	**87.99**	90.90	85.27
100	**80.37**	**87.50**	74.31	87.97	**91.11**	85.04

Table 5. Comparison of the results with different number of lane queries on CuLane and CurveLanes datasets. The backbone we use is ResNet-18.

Number of decoder layers. Here we investigate the influence of different number of decoder layers. As shown in Table 6, by increasing the number of decoder layers from 1 to 4, we observe an improvement in performance. This demonstrates that increasing the number of decoder layers enhances the ability of dynamic kernels to effectively capture the lane lines' global information. Increasing the number of decoder layers from 4 to 6 results in only a marginal improvement in performance.

Num of decoder layers	CuLane			CurveLanes		
	F1	Prec.	Rec.	F1	Prec.	Rec.
1	79.48	86.43	73.57	86.82	89.75	84.08
2	79.87	86.89	73.90	87.21	90.09	84.51
4	80.36	87.44	74.35	87.99	90.90	85.27
6	**80.44**	**87.51**	**74.43**	**88.17**	**91.12**	**85.41**

Table 6. Comparison of the results with different number of transformer decoder layers on CuLane and CurveLanes datasets. The backbone we use is ResNet-18.

5. Conclusion

In this paper, we propose a transformer-based dynamic kernel generation architecture for lane detection, which utilizes a transformer to generate dynamic convolutional kernels for each lane line in the input image, and detects lane lines with dynamic convolution. Compared to the kernels generated from some key locations of a lane line, the kernels generated with transformer can capture the lane line's global information, enabling them to effectively handle occlusions and lane lines with complex topologies. The proposed method is validated on four benchmarks of lane detection, i.e., OpenLane, CurveLanes, CUlane and Tusimple. The experimental results shows the state-of-the-art performance of the proposed method.

References

[1] Li Chen, Chonghao Sima, Yang Li, Zehan Zheng, Jiajie Xu, Xiangwei Geng, Hongyang Li, Conghui He, Jianping Shi, Yu Qiao, et al. Persformer: 3d lane detection via perspective transformer and the openlane benchmark. *arXiv preprint arXiv:2203.11089*, 2022.

[2] Zhenpeng Chen, Qianfei Liu, and Chenfan Lian. Pointlanenet: Efficient end-to-end cnns for accurate real-time lane detection. In *2019 IEEE intelligent vehicles symposium (IV)*, pages 2563–2568. IEEE, 2019.

[3] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[4] Jie Guo, Zhihua Wei, and Duoqian Miao. Lane detection method based on improved ransac algorithm. In *2015 IEEE Twelfth International Symposium on Autonomous Decentralized Systems*, pages 285–288. IEEE, 2015.

[5] 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*, pages 770–778, 2016.

[6] Yuenan Hou, Zheng Ma, Chunxiao Liu, and Chen Change Loy. Learning lightweight lane detection cnns by self attention distillation. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 1013–1021, 2019.

[7] Paul VC Hough. Method and means for recognizing complex patterns, Dec. 18 1962. US Patent 3,069,654.

[8] 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*, 2021.

[9] Harold W Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955.

[10] Qi Li, Yue Wang, Yilun Wang, and Hang Zhao. Hdmapnet: An online hd map construction and evaluation framework. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 4628–4634. IEEE, 2022.

[11] Xiang Li, Jun Li, Xiaolin Hu, and Jian Yang. Line-cnn: End-to-end traffic line detection with line proposal unit. *IEEE Transactions on Intelligent Transportation Systems*, 21(1):248–258, 2019.

[12] Zuo-Quan Li, Hui-Min Ma, and Zheng-Yu Liu. Road lane detection with gabor filters. In *2016 International Conference on Information System and Artificial Intelligence (ISAI)*, pages 436–440. IEEE, 2016.

[13] 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*, pages 3773–3782, 2021.

[14] Ruijin Liu, Zejian Yuan, Tie Liu, and Zhiliang Xiong. End-to-end lane shape prediction with transformers. In *Proceedings of the IEEE/CVF winter conference on applications of computer vision*, pages 3694–3702, 2021.

[15] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[16] Chunyang Mu and Xing Ma. Lane detection based on object segmentation and piecewise fitting. *TELKOMNIKA Indonesian Journal of Electrical Engineering*, 12(5):3491–3500, 2014.

[17] Davy Neven, Bert De Brabandere, Stamatios Georgoulis, Marc Proesmans, and Luc Van Gool. Towards end-to-end lane detection: an instance segmentation approach. In *2018 IEEE intelligent vehicles symposium (IV)*, pages 286–291. IEEE, 2018.

[18] Jianwei Niu, Jie Lu, Mingliang Xu, Pei Lv, and Xiaoke Zhao. Robust lane detection using two-stage feature extraction with curve fitting. *Pattern Recognition*, 59:225–233, 2016.

[19] Xingang Pan, Jianping Shi, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Spatial as deep: Spatial cnn for traffic scene understanding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[20] Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Lukasz Kaiser, Noam Shazeer, Alexander Ku, and Dustin Tran. Image transformer. In *International conference on machine learning*, pages 4055–4064. PMLR, 2018.

[21] Zequn Qin, Huanyu Wang, and Xi Li. Ultra fast structure-aware deep lane detection. In *European Conference on Computer Vision*, pages 276–291. Springer, 2020.

[22] 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*, pages 14122–14130, 2021.

[23] Jinming Su, Chao Chen, Ke Zhang, Junfeng Luo, Xiaoming Wei, and Xiaolin Wei. Structure guided lane detection. *arXiv preprint arXiv:2105.05403*, 2021.

[24] Lucas Tabelini, Rodrigo Berriel, Thiago M Paixao, Claudine Badue, Alberto F De Souza, and Thiago Oliveira-Santos. Keep your eyes on the lane: Real-time attention-guided lane detection. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 294–302, 2021.

[25] Lucas Tabelini, Rodrigo Berriel, Thiago M Paixao, Claudine Badue, Alberto F De Souza, and Thiago Oliveira-Santos. Polylanenet: Lane estimation via deep polynomial regression. In *2020 25th International Conference on Pattern Recognition (ICPR)*, pages 6150–6156. IEEE, 2021.

[26] TuSimple. Tusimple: Lane detection benchmark. 2017.

[27] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[28] Jinsheng Wang, Yinchao Ma, Shaofei Huang, Tianrui Hui, Fei Wang, Chen Qian, and Tianzhu Zhang. A keypoint-based global association network for lane detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1392–1401, 2022.

[29] Pei-Chen Wu, Chin-Yu Chang, and Chang Hong Lin. Lane-mark extraction for automobiles under complex conditions. *Pattern Recognition*, 47(8):2756–2767, 2014.

[30] Hang Xu, Shaoju Wang, Xinyue Cai, Wei Zhang, Xiaodan Liang, and Zhenguo Li. Curvelane-nas: Unifying lane-sensitive architecture search and adaptive point blending. In *European Conference on Computer Vision*, pages 689–704. Springer, 2020.

[31] Tu Zheng, Hao Fang, Yi Zhang, Wenjian Tang, Zheng Yang, Haifeng Liu, and Deng Cai. Resa: Recurrent feature-shift aggregator for lane detection. *arXiv preprint arXiv:2008.13719*, 5(7), 2020.

[32] 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*, pages 898–907, 2022.