TECHNICAL UNIVERSITY OF MUNICH

TUM Data Innovation Lab

Multi-lane Detection in Urban Scenarios with Deep Learning

Authors	Alberto Gonzalo Rodriguez Salgado, Alessandro Ferrenti, Ebraheem Abdelhafez, Yezi Yang
Mentor(s)	Nikolay Kadrileev, MSc. Robert Bosch GmbH
Co-Mentor	Michael Rauchensteiner
Project Lead	Dr. Ricardo Acevedo Cabra (Department of Mathematics)
Supervisor	Prof. Dr. Massimo Fornasier (Department of Mathematics)

Jul 2021

Abstract

Lane estimation plays a crucial role in driver assistance systems. Moreover, the reliable detection of driving lines is an important milestone in enabling autonomous driving. Most of the existing state-of-the-art algorithms already achieved very remarkable results in simple scenarios.
This project aims to achieve good results also in urban scenarios. In addition to the classical lane detection challenges, dealing with bad weather conditions, urban driving, and construction site driving is another challenge. Furthermore, for high autonomous driving purposes, the detection of the lanes is not enough. It is really important to extract the meaning associated with the lines, for example, if the lane line is dashed or solid. Even the color is valuable information that needs to be taken into consideration, in construction site driving, for example.
For these purposes, an algorithm is implemented to detect the sequence of points that make up the lines and at the same time is able to extract high-level features such as the type of lines. Conducted experiments show that this method leads to promising results and reliable properties estimation thanks to the usage of affinity fields. Furthermore, the proposed models are also able to extract affinity fields that are not limited to this project but also viable for incoming projects by Robert Bosch GmbH.

Contents

1 Introduction

Highly automated driving (Level 2 to Level 5, SAE-J3016 [14]) demands a range of functionality of advanced driver assistance systems: *Lane Assistance, Predictive Emergency Braking, Turn and Crossing Assistance, Park and Maneuver Assistance, etc.* In this report, the terms "Lane Markings", "Lanes", and "Lines" were used interchangeably. However, they all refer to the same meaning which is driving lines, consisting of connected road markings.
Lane Detection is just one of the detection components, but its role is paramount to build a solid and reliable autonomous driving system. State-of-the-art algorithms already achieved high lane detection rates in simple situations, for example, on highways. However, urban scenarios still represent a major challenge for lane detection algorithms for several reasons:

- **Bad weather conditions:** In some scenarios, due to snow, heavy rain, or even direct sunlight is very hard to detect driving lanes either because the image contains artifacts, or because it is not properly visible.
- **Urban Driving:** Especially in city driving, it is easy to stumble across dense traffic, static obstacles, pedestrian or advanced lane types like roundabouts and intersections along the way.
- **Construction sites:** In order to develop a high automated guidance system, some exception scenarios must be considered. For example, on construction sites lines have a different meaning than usual. Thus, the goal is to understand its semantics and prioritize the lines accordingly.

1.1 Project Goal

This project aims to develop a deep learning algorithm that operates on front video camera images. With respect to the latter, was installed behind the rear view mirror. The algorithm also can detect multiple driving lines along with their associated semantic meaning in challenging urban scenarios.

2 Datasets

Numerous state-of-the-art algorithms already achieve high lane detection rates in simple situations like in *TuSimple* [6], which only contains frames from US highways with no bad weather conditions. Later on, some other more challenging datasets came out [7], among them *CULane* [11] which is the largest available dataset, volume-wise.

2.1 CULane

CULane [11] is a large scale challenging dataset for academic research on traffic lane detection. It is collected by cameras mounted on six different vehicles driven by different drivers in Beijing. More than 55 hours of videos were collected and 133,235 frames were

extracted. Data examples are shown below. The dataset is divided into 88,880 for training set, 9,675 for validation set, and 34,680 for test set. The test set is divided into normal and 8 challenging categories, as shown in Figure 1. This dataset covers challenging scenarios,

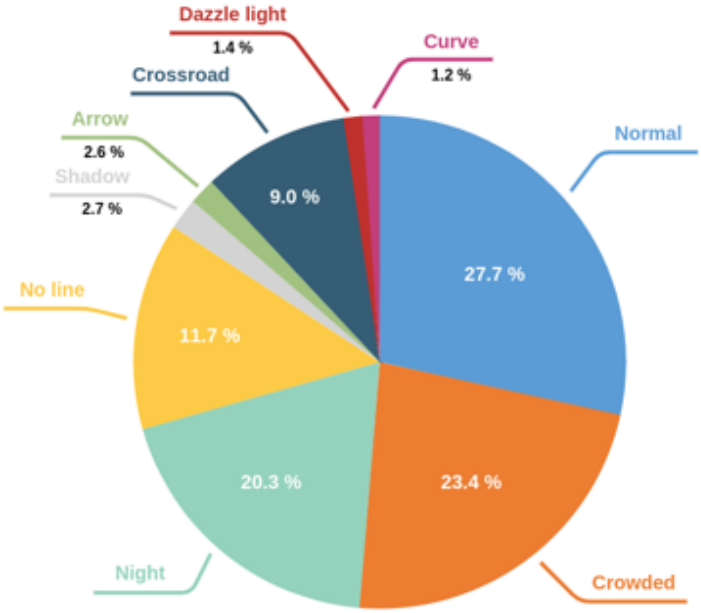


Figure 1: CULane categories distribution

such as occlusions, and night scenes. CULane [11] contains frames with *night scenes, curved lanes, urban traffic, shadow, no lane marking and other scenarios.* Some examples are provided in Figure 2.

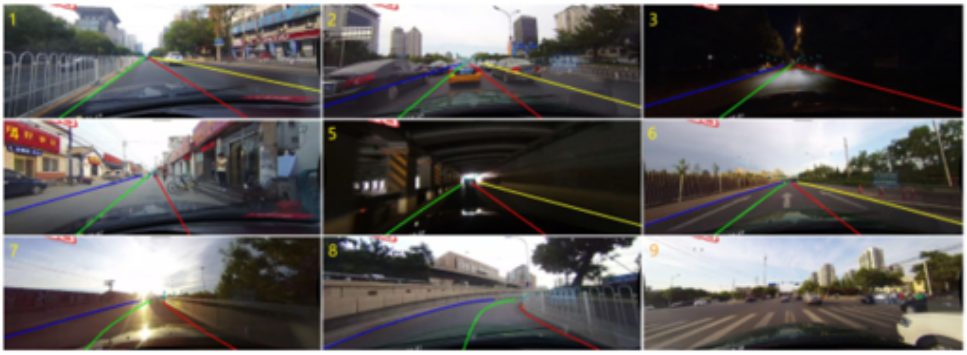


Figure 2: CULane examples

The annotations of each frame are simply stored as a separate text file in which each line is represented by a sequence of points (x_i, y_i).

2.2 BOSCH Dataset

A dataset of 5,740 front camera images was provided by BOSCH. 1,926 frames out of them were annotated by the team members. The images were taken from all over the world, introducing a new challenge: similar lane markings have different meanings in different countries. For example, in Japan, dashed lines are often used as deceleration lanes, or in the UK, zigzag lane markings are used to indicate that a pedestrian crossing is close by.

2.2.1 Distribution

In this section, some statistical evaluation is presented on the annotated portion of the BOSCH dataset. In the Table 1 and the Figure 3, the distribution of Frames and Lane markings for each Country is presented. The following countries are grouped under the EU: *Germany, UK, Ireland*.

Country	Frames	Lanes	Curved Frames
United States	326	1210	11
Japan	530	2074	47
EU	570	1678	65
China	500	1842	33
TOTAL	1926	6804	156

Table 1: Frames, Lanes & Curved Frames distribution

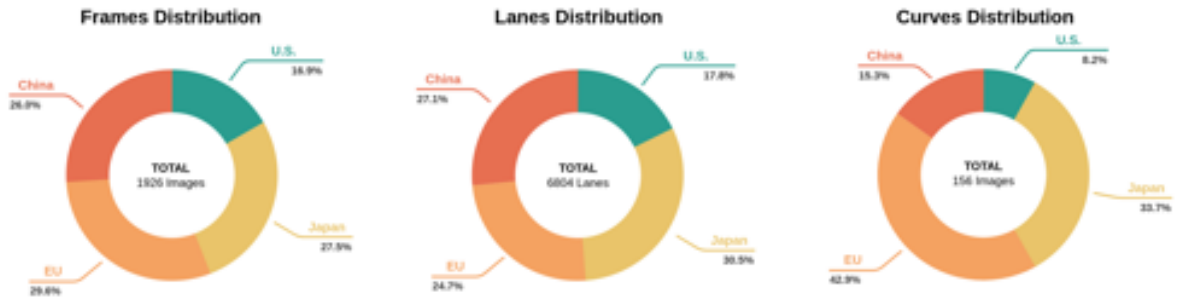


Figure 3: Frames,Lanes & Curved Frames distribution

All the distribution looks more or less balanced. It is worth to point out that for the curved split, the *curvature* has been implemented based on [9]. Then, an image belongs to the curved split if exceeds a certain threshold. An interesting finding was that most of the curved scenarios belong to either Japan or EU sequences.

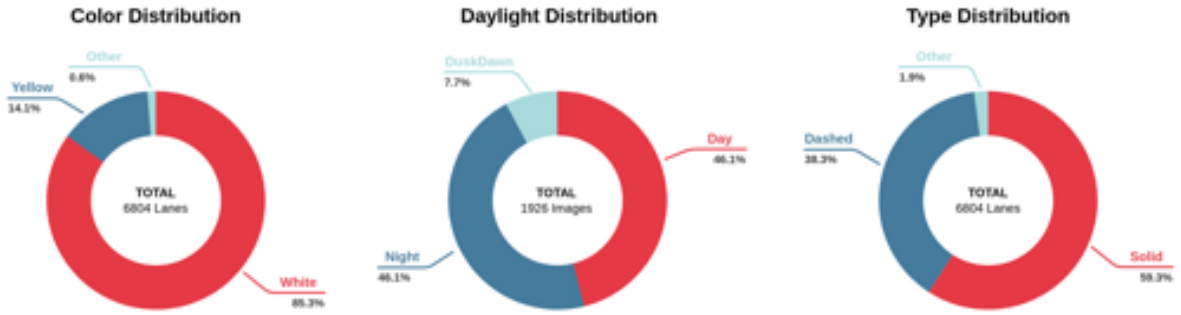


Figure 4: Color, Daylight & Type distribution

The BOSCH dataset has way more lane markings compared to the CULane dataset[11]. Furthermore, the goal is to also detect, and classify the type of the lane marking. A brief analysis of the distribution is presented in Figure[4] along with the Table[2].

Country	white	yellow	red	unsure	solid	dashed	zigzag	unsure
United States	816	377	0	17	775	393	0	42
Japan	1933	139	0	2	1385	656	0	33
EU	1522	118	18	20	766	849	28	35
China	1534	308	0	0	1107	710	4	21
TOTAL	5805	942	18	38	4033	2608	32	131

Table 2: Type & Color distribution per Country

2.2.2 Manual Annotations

The first task was to understand, and annotate the dataset. CVAT [2], an open-source computer vision annotation tool developed by Intel, was used to generate the label. There are different information to annotate and can be divided in the following categories:

- **Scene Tag:** Used to specify additional information about the scene, for example bad weather conditions if there are any.
- **Lane Key Point:** Used to mark down with a single point an intersection, like merging lanes or splitting lanes.
- **Lane Marking:** This is the principal type of marking used to mark down a driving line. The annotation of the lane follows the middle of it and we can specify additional information such as:
  - **details**: *badly visible, reflectors, interruption, multimarking, occlusion, roundabout*
  - **direction**: *ego, ongoing, both, unsure*
  - **function**: *lane separator, bicycle lane, deceleration lane, unsure*
  - **type**: *solid, dashed, zigzag, unsure*

  - **color**: *white, yellow, red, blue, unsure*

- **Ego Lane:** This marking is very relevant for driving purposes. The difference between lane markings and ego lane markings is that the latter represents the inner side of the driving lines.

- **Horizontal Lane:** With this lane marking, stopping lanes are annotated.

In Figure 5, there are some examples of what the annotation look like on the CVAT [2] interface.

Figure 5: BOSCH Dataset example images

CVAT [2] generates an xml file containing all the points and all additional information as described before, from this, the labels are parsed and formatted according to the CULane format [11].

3 Literature Review

3.1 PINet

3.1.1 Architecture

The Point Instance Network (PINet) [8] method to detect lanes is based on the idea of detecting lane key points that characterize different lanes instead of predicting whether each pixel of the image belongs to a lane marking or not. By doing this, one does not need to generate binary masks as in the LaneAF [12] architecture.
Figure 6 shows the main architecture of PINet. The network takes an image as an input and passes it through several hourglass modules which are encoder-decoder architectures as shown in Figure 7. One of the advantages of PINet is that the user is able to select how many such hourglass modules one wants to stack together. Therefore, one can also control the model size. These hourglass modules produce a confidence field, an offset field and a feature field as an output. The confidence values are used to predict whether a pixel is a key point or not. The offset value localizes the position of the line. Last but not least, the feature values are used to cluster the predicted pixel line key points.

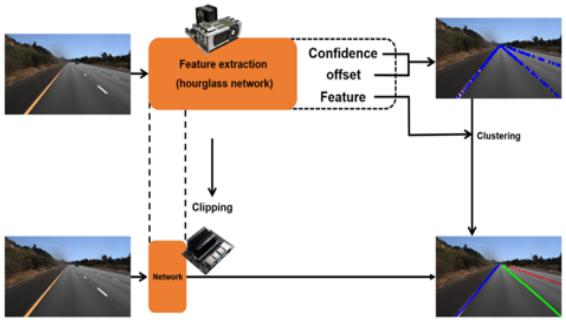


Figure 6: The architecture of PINet [8]

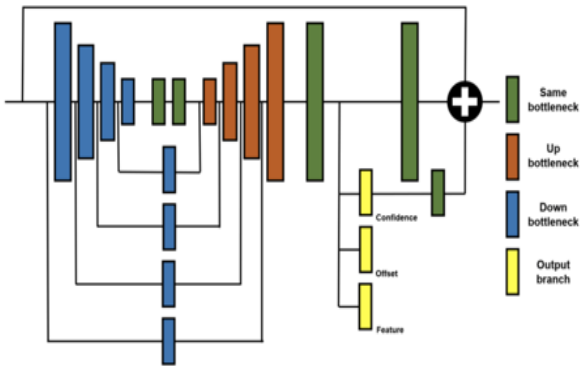


Figure 7: Hourglass module encoder-decoder architecture [8]

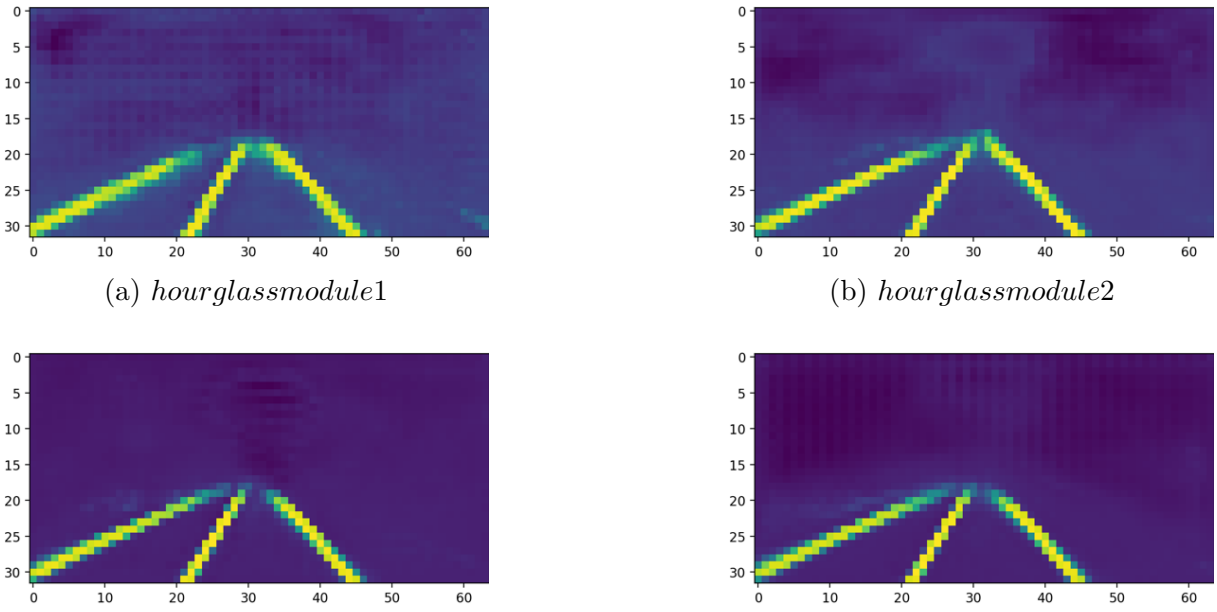
3.1.2 Confidence fields

As explained before, the key points prediction mainly depends on the confidence field. Next, in order to better understand what is happening inside the neural network, an example from the BOSCH dataset is demonstrated. The next predictions are produced by the network in Figure 10b and trained on 886 BOSCH images from scratch i.e it is not using any weights of a pre-trained model on CULane [11]. Specific details about the training can be found 3.1.3.

Figure 8: Image from BOSCH dataset

The shown image [8] is passed through four hourglass modules which. produce the next

confidence fields.



(a) *hourglassmodule*1 (b) *hourglassmodule*2

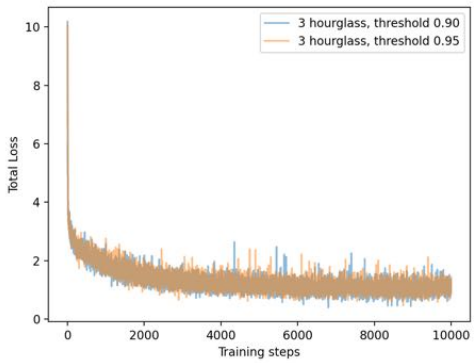
(c) *hourglassmodule*3 (d) *hourglassmodule*4

Figure 9: Confidence fields of 4 hourglass modules

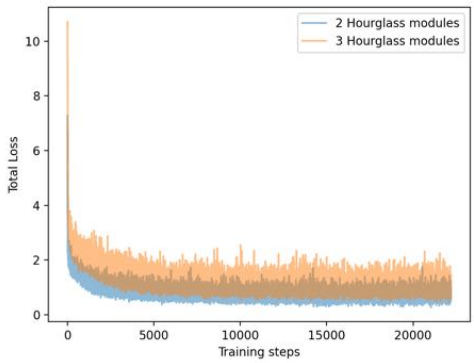
Figure 9 shows that after each hourglass module the network becomes more confident where possible key points might be. However, it does not detect the horizontal line that is present in Figure 8

3.1.3 Training from scratch

The PINet architecture shown in Figure 6 was trained from scratch using only images from the BOSCH dataset. This was performed in May when the labeled images count was 1,500. The architecture was trained on 800 images and tested on 200. Several experiments were conducted. In the first case, 3 hourglass modules were used and the confidence threshold was set to 0.90 and 0.95 respectively. In the second case, 3 and 2 hourglass modules were used with a 0.95 confidence threshold in both cases. The main goal is to see how the choice of hourglass modules and confidence threshold affects the training. Moreover, the learning rate was set to 0.001, he weight decay to $1e-5$ and several data augmentation techniques were used. Specifically, random flipping, random translation, random rotation, random noise, random intensity, and random shadows were applied. For all these data augmentation techniques the ratio was set to 0.6.



(a) *Case*1



(b) *Case*2

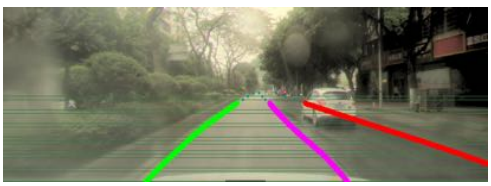
Figure 10: PINet training from scratch train losses

Figure 10a shows that changing the confidence threshold from 0.95 to 0.90 does not greatly affect training performance. However, as it is visible from Figure 10b, choosing a smaller network using two hourglass modules instead of three leads to a lower loss during training.

(a) Night scene

(b) Urban occluded scene

(c) Urban occluded scene

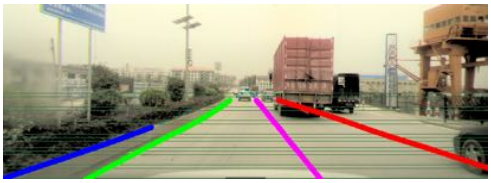


(d) Bad weather scene

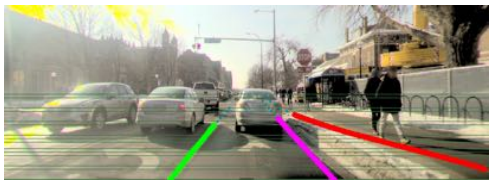
Figure 11: Good predictions on test data

Figure 11 shows four good predictions of the trained from scratch model, as illustrated in Figure 10b, using 2 hourglass modules on four images from the test set.It is visible that the model performs well in difficult conditions such as occluded or rainy scenarios. Nevertheless, there are also scenarios where the model fails to detect lanes or where it detects false positive lanes.

(a) Curved lanes scene



(b) False positive scene



(c) False positive snow scene

(d) Multiple lanes scene

Figure 12: Failed predictions on test data

3.2 LaneATT

Before delving into the details of LaneATT [16], it is worth mentioning another architecture that motivated the idea of anchor-based lane-detection. It is the architecture of the well-known two-stage 2D object detector "*Faster R-CNN*" [12]. As depicted in Figure 13, the network consists of two main stages. The first stage comprises a backbone CNN architecture to learn the semantic information presented in the image. The second stage is the Region Proposal Network (RPN) which proposes the regions of interest that might contain objects of interest in the original image.

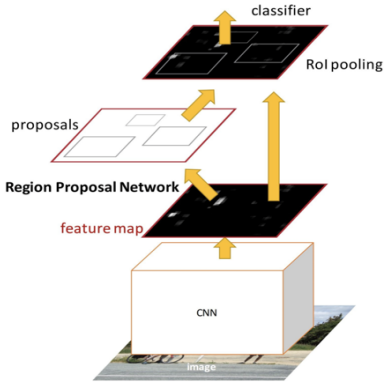


Figure 13: The architecture of Faster R-CNN [12]

3.2.1 Line-CNN

Faster R-CNN [12] was the main inspiration for "Line-CNN" [10] which is a lane-detection network. As illustrated in Figure 14, the network looks very similar to Figure 13. However, instead of having an RPN in Figure 13 there is a Line Proposal Unit (LPU) in Figure 14. The LPU does not process the entire feature map, it just processes the three boundaries of it (left, right, and bottom). It does not consider the pixels at the top

boundaries to reduce the computations and it is never the case to find lane markings in any scene where they are extended to the top border of the image, given that the camera is mounted on top of the vehicle.
It does not consider the pixels at the top boundaries to reduce the computations. Because it is not possible to find lane markings on the top border of the image, given the fact that the camera is mounted on top of the vehicle.

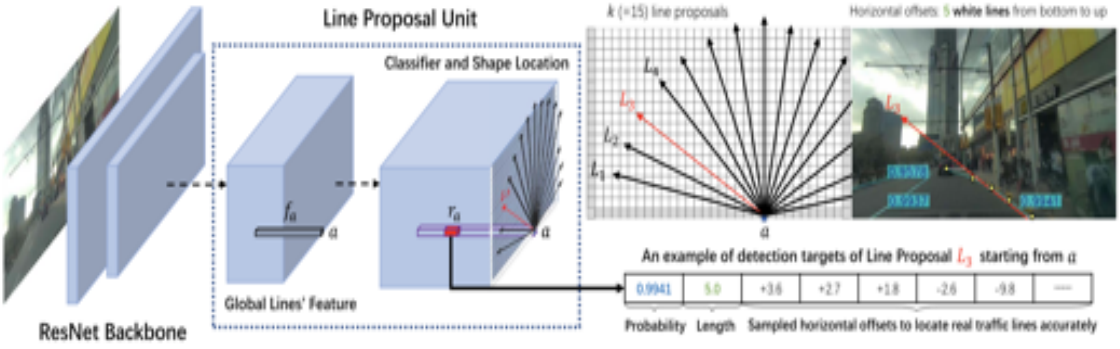


Figure 14: The architecture of Line-CNN [10]

The LPU applies 1x1 convolution-kernels on the boundary pixels of the feature map, so that each pixel can be encoded in 1024-dimensional vector. Then, each vector is passed through a classifier and a regressor. The classifier outputs $2k$ scores for each vector, where k is the number of anchors to be generated for each pixel. An anchor represents a line proposal according to some angle as depicted in Figure 14. The regressor outputs k x $(S+1)$ outcomes, where S is the number of evenly distributed horizontal slicing lines on the original image. Also, k is different for each boundary, thus, the number of anchors generated for a pixel in the bottom boundary is different from those generated for a pixel in the right boundary.

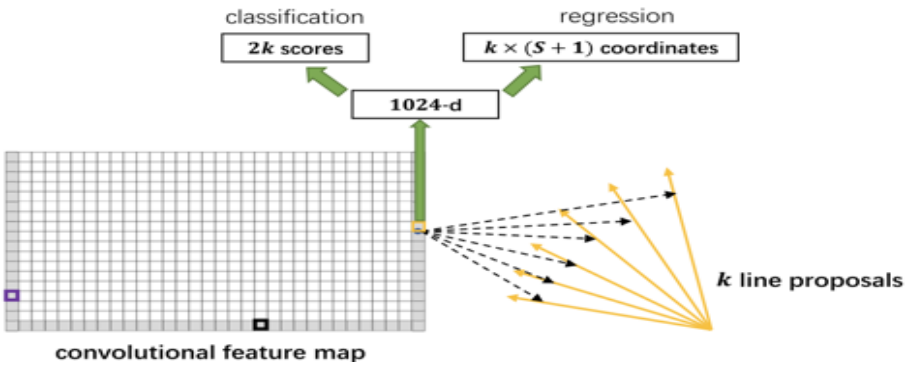


Figure 15: Illustration of how the LPU generates proposals

As illustrated in the Figure 14, the outcome vector for the L_3 anchor has the probability score of being a real-lane marking, a regressed value called "Length", and S regressed outcomes. The "Length" outcome determines how many points to be considered from the S next outcomes to add them as horizontal shifts to the original orientation of the anchor to produce a proposal of a lane marking.

The configuration of Line-CNN [10] is simple to grasp, fast in terms of the inference speed, and easy to implement. However, it suffers from some drawbacks that can be summarized into the following points:

- The vector encoding produced in the LPU relies on the local features only. Because the feature map is generated by applying standard convolutions, thus, small and regular receptive fields impose restrictions in the encoding. In other words, each lane marking does not have enough global information about other lane markings in the scene. This limitation is affecting the performance of the network in the scenes that contain heavy occlusions.
- The anchors proposed for each boundary rely on a relatively steep anchor angles. Also, the regressor outputs only horizontal shifts. Both limitations make it very difficult for the network to detect horizontal lane markings.
- The representation of the anchors as straight lines, lines of degree 1, hinders the performance of detecting very curved lane markings. As the horizontal shifts will not be enough to add the curvature to the straight lines.

3.2.2 Introducing LaneATT

The architecture of LaneATT [16] is mainly inspired from Line-CNN [10]. The architecture is meant to solve the first short-coming faced by Line-CNN [10], which is the locality representation of the anchors. Thus, "ATT" in the name indicates using the attention mechanism to make the anchor encodings contain global information about other anchors in the image.

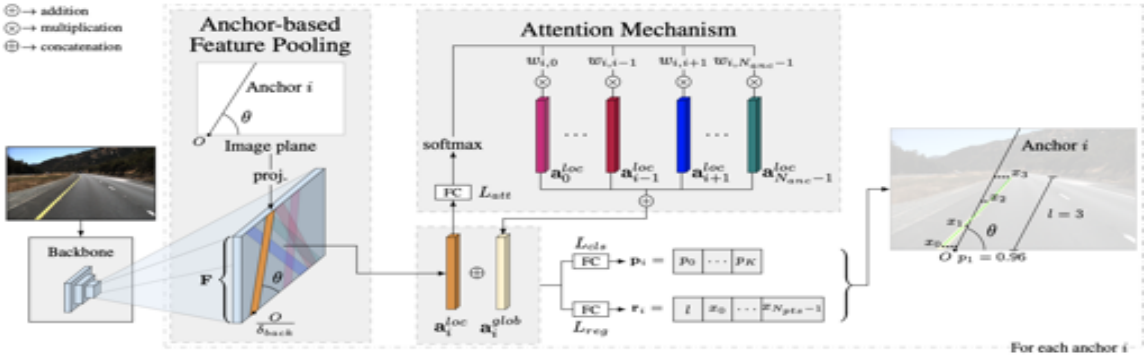


Figure 16: The architecture of LaneATT [16]

The same anchor angles, and the anchor setup used by Line-CNN [10] to generate proposals are used in LaneATT [16]. The anchor-based feature pooling step uses the anchor angles to define the anchors' local encoding instead of just applying 1x1 convolutions like in Line-CNN [10]. The novel idea of the architecture is introducing the attention mechanism before passing the encodings to the classifier and the regressor. The attention mechanism consists of a fully-connected layer L_{att} that processes the local encodings of an anchor i and every other anchor j. Then, it produces a probability output $w_{i,j}$ for every anchor j. Afterwards, the main representation for an anchor i becomes

$a_i^{global} = a_i^{local} \oplus \sum_j w_{i,j} a_j^{local}$. The attention mechanism improved the performance of the network, especially in the heavily occluded scenes, as depicted in Figure 17.

Figure 17: Performance of LaneATT on an image from the BOSCH dataset. (Left): unannotated image, (Right): annotated image. Green lines are network predictions, and blue ones are ground truth annotations.

3.3 LaneAF

LaneAF [1] employs an off-the-shelf convolutional neural network backbone, DLA34 [17], that aggregates and refines multi-scale features. This resulted in superior performance when compared to other architectures and losses previously proposed for lane detection. LaneAF also proposes two affinity fields, the horizontal affinity field (HAF) and vertical affinity field (VAF) that are suitable for clustering and associating pixels belonging to amorphous entities like lane markings. The network takes in the image as an input and produces a binary segmentation mask, "HAF" and "VAF", which are used to cluster lane markings and give predictions in the affinity fields decoder, as shown in Figure 18.

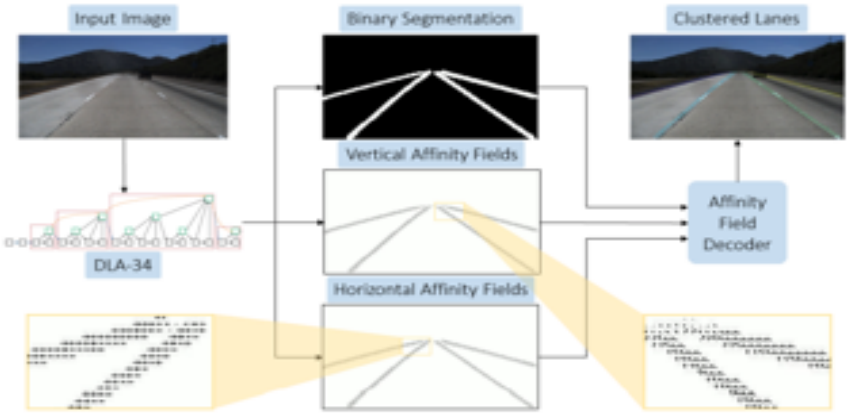


Figure 18: The architecture of LaneAF [1]

3.3.1 Introducing DLA34

Aggregation is defined as the combination of different layers throughout a network. Deep Layer Aggregation(DLA) [17] focuses on effective aggregation of depth, resolutions and scales. As networks can contain many layers and connections, modular design is used to help counter complexity by grouping and repetition. Layers are grouped into blocks and then grouped into stages by their feature resolution.
The DLA family is introduced with the goal of aggregating layers to better fuse semantic and spatial information for recognition and localization. Two structures are proposed: the iterative deep aggregation(IDA) and hierarchical deep aggregation(HDA). The DLA models extend densely connected networks and feature pyramid networks with iterative and hierarchical skip connections that deepen the representations and refine resolutions.

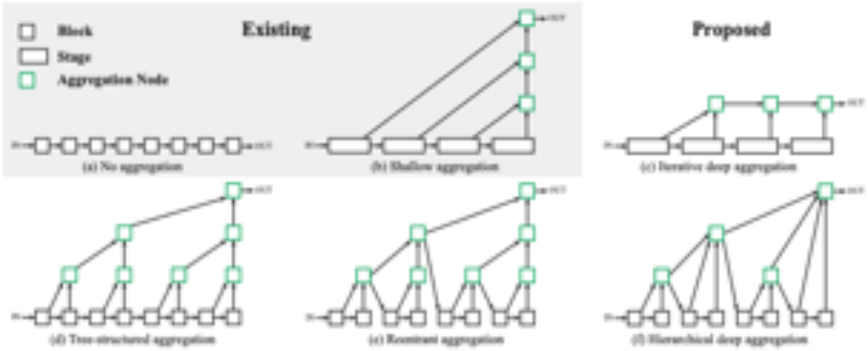


Figure 19: Different approaches to aggregation [17]

In conventional classification and regression networks, blocks are composed without aggregation, as shown in Figure 19 (a), like the VGG net [15]. Figure 19 (b) shows the network structure that is commonly used for tasks like segmentation and detection, where parts of the network are combined by skip connections like U-net [13]. However, this aggregation is only done in a shallow way by merging the earlier parts with the later parts in a single step each. Therefore, IDA, shown in Figure 19 (c), is introduced. By reordering the skip connections, the network aggregates iteratively such that the shallowest parts are aggregated the most. While IDA effectively merges stages, it is still only sequential. Thus, a tree-structured aggregation, as shown in Figure 19 (d) is also proposed to aggregate hierarchically. Through the tree structure of blocks, feature channels are preserved and combined. Shallower and deeper layers are used to learn richer combinations that span the feature hierarchy. Figure 19 (e) and (f) are refinements of (d) that deepen aggregation by routing the intermediate results back into the the network and improve efficiency by merging successive aggregations at the same depth.
DLA is a general architecture family that is compatible with different backbones. The internal structure of the blocks and stages is not required. DLA34 makes use of one typical type of residual blocks [5]: Basic Blocks, which combine stacked convolutions with an identity skip connection, as the one used in ResNet-34 [4] and has similar number of layers.
To reach the necessary resolution of a segmentation task, IDA is again used in the upsampling procedure. Outputs of stages 3-6, red boxes shown in Figure 20, are first projected

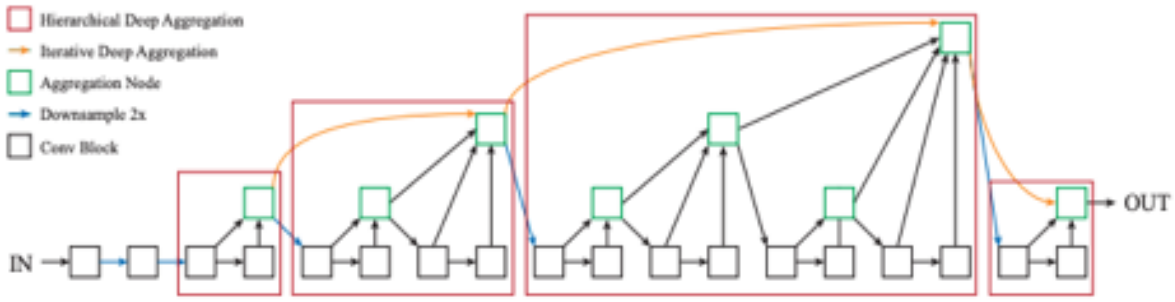


Figure 20: DLA architecture with HDA and IDA [17]

to 32 channels and then interpolated to the same resolution as stage 2. Finally, these stages are aggregated iteratively to learn a deep fusion of low and high level features. Deformable convolution operations [3] that can adapt the spatial sampling grid for convolutions based on their inputs are also incorporated in these architectures.

3.3.2 Deformable Convolution

Deformable ConvNets were first introduced in [3] with the goal to enhance CNN's capability of modeling geometric transformations for sophisticated vision tasks, such as object detection and semantic segmentation. Two new modules are proposed, one of which is the deformable convolution. It adds 2D offsets to the regular grid sampling locations in the standard convolution, which enables free form deformation of the sampling grid. Figure 21 (a) shows the sampling locations of standard convolution and (b) illustrates deformable sampling locations(dark blue points) with augmented offsets(light blue arrows). With learned 2D offsets, deformable convolution can generalize various transformations for scale, aspect ratio and rotation.

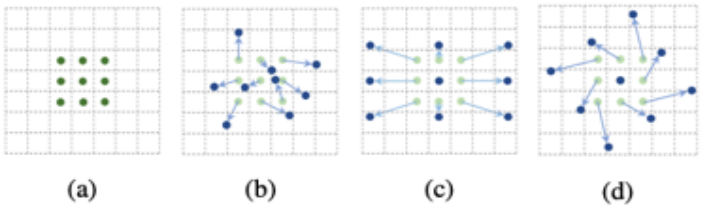


Figure 21: Illustration of sampling locations

3.3.3 Affinity Fields

For a given image, the "HAF" and "VAF" can be thought of as vector fields that assign a unit vector to each (x, y) location in the image. The "HAF" enables pixel clustering horizontally and the "VAF" vertically. With the predicted affinity fields and binary mask, clustering line pixels is achieved through a row-by-row decoding process from bottom to top. The affinity fields are created using the ground truth as follows:

- First the "HAF" is calculated. Starting from the bottom row of the segmentation map, find pixels with positive values, indicating each line in the ground truth. Assign corresponding values to these pixels according to their relative positions to the center of their belonging line, as illustrated in Figure 22 (a).
- Then the "VAF" of current row is generated. Compute the normalized vectors pointing to the center of the row above which belongs to the same line cluster and assign them to the corresponding pixels of the current row, as shown in Figure 22 (c).

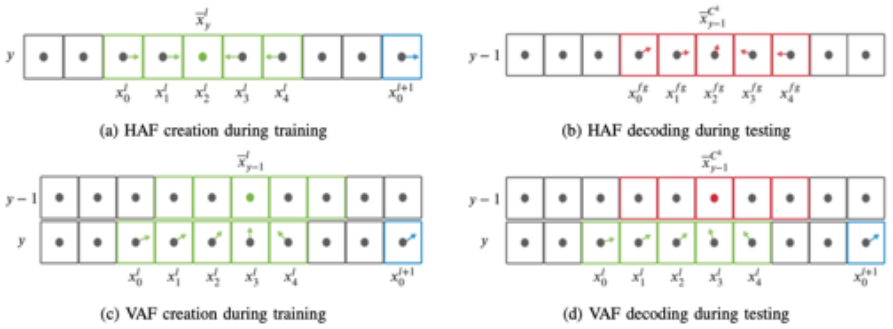


Figure 22: Illustration of "HAF" and "VAF" creation and decoding processes

Similarly, the affinity fields are decoded to cluster foreground pixels into lines. Foreground pixels are first assigned to clusters based on the predicted "HAF" and the center of the cluster is found based on the predicted value. This process is illustrated in Figure 22 (b). Next, the horizontal clusters are assigned to existing lines using the "VAF". The optimal assignment is achieved when the error of associating cluster to an existing line is the minimum. The error is calculated as the mean difference between the predicted "VAF" of current row and the "VAF" generated using the ground truth.
The proposed per-pixel affinity fields enables LaneAF to successfully cluster line pixels regardless of the shape and the width of the line and to predict a variable number of lanes without assuming a maximum number of lines.

4 Architecture Comparisons

This section is meant to demonstrate the differences between the 3 architectures mentioned in the previous section. Several experiments were conducted on the 3 architectures to choose one of them to improve its performance.
First, to compare the performance of these architectures, 3 metrics are used: precision, recall and the F1 score. Precision is defined as the number of true positives over the number of true positives plus the number of false positives:

$$Precision = \frac{TP}{TP + FP}, \tag{1}$$

Recall is defined as the number of true positives over the number of true positives plus the number of false negatives:

$$Recall = \frac{TP}{TP + FN}, \tag{2}$$

	PINet			LaneATT (ResNet-34)			LaneAF		
	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
DuskDawn	0.429	0.457	0.442	0.457	0.251	0.324	0.614	0.348	0.444
Night	0.508	0.548	0.527	0.663	0.356	0.464	0.758	0.474	0.583
Day	0.616	0.65	0.632	0.731	0.527	0.613	0.772	0.594	0.671
Lane-key-points	0.54	0.529	0.534	0.687	0.329	0.444	0.858	0.463	0.601
Multi-marking	0.482	0.442	0.461	0.705	0.319	0.439	0.732	0.362	0.485
DS-JP	0.697	0.616	0.654	0.751	0.435	0.551	0.887	0.542	0.673
DTN	0.456	0.479	0.467	0.652	0.367	0.47	0.687	0.404	0.509
LB-XO	0.384	0.491	0.431	0.495	0.289	0.365	0.621	0.439	0.515
JLR_CN	0.705	0.795	0.747	0.782	0.651	0.711	0.821	0.741	0.779
Overall (BOSCH Dataset)	0.651	0.421	0.512	0.686	0.431	0.529	0.763	0.529	0.625
Overall (CULane Dataset (test-split))	0.795	0.699	0.744	0.812	0.697	0.750	0.873	0.781	0.825

Figure 23: Architecture performance comparison

The F1 score is defined as a harmonic mean of precision and recall:

$$F1 = 2 \cdot \frac{Precision \times Recall}{Precision + Recall}. \tag{3}$$

The table in Figure 23 shows the performance of all three architectures both on the BOSCH dataset and on the CULane test set. These results refer only to the models that were already pre-trained by the authors of the respective papers using the CULane dataset. It is evident that the best F1 score is achieved by the LaneAF architecture, both for the BOSCH dataset yielding $F1 = 0.625$ and the CULane test set yielding $F1 = 0.825$. The anchor representation used in LaneATT leads to a poor performance on curved scenarios. On the other hand, LaneAF's architecture does not have any disadvantage for detecting curved scenarios. Since curved scenarios frequently appear during driving, it is crucial that the network is able to correctly perform on them. For this reason, LaneAF was prefered over LaneATT.
The PINet architecture does perform well on curved scenarios as LaneAF. Since PINet detects lane key points, it is able to recognize all types of lines. However, it only outputs information regarding the line's location. On the other hand, LaneAF produces affinity fields which yield more information then just the location of the line. These affinity fields enable to extract the next two line characteristics in a post-processing step:

- The horizontal affinity fields can be used to compute the line width. One can horizontally detect where the first and last affinity field vectors of a line are located and then compute the width.
- The inner edge of the ego lines can be predicted using the affinity fields. For this purpose, a post-processing code was developed which starts from the middle of the image and looks for the first left and right detected lines respectively. Then, it computes the left and right ego line using the affinity fields of these lines. An example is shown in Figure 24 .

Last but not least, to confirm that each architecture, especially LaneAF is able to learn from the BOSCH images, an overfitting experiment was carried out. In this case, the goal

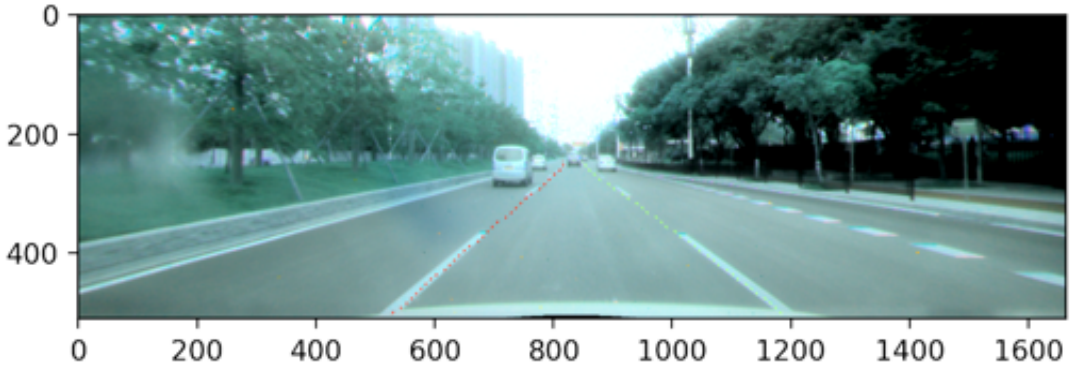


Figure 24: Ego line detection from affinity fields

is to overfit a complicated scenario from the BOSCH dataset and see the performance of each model.

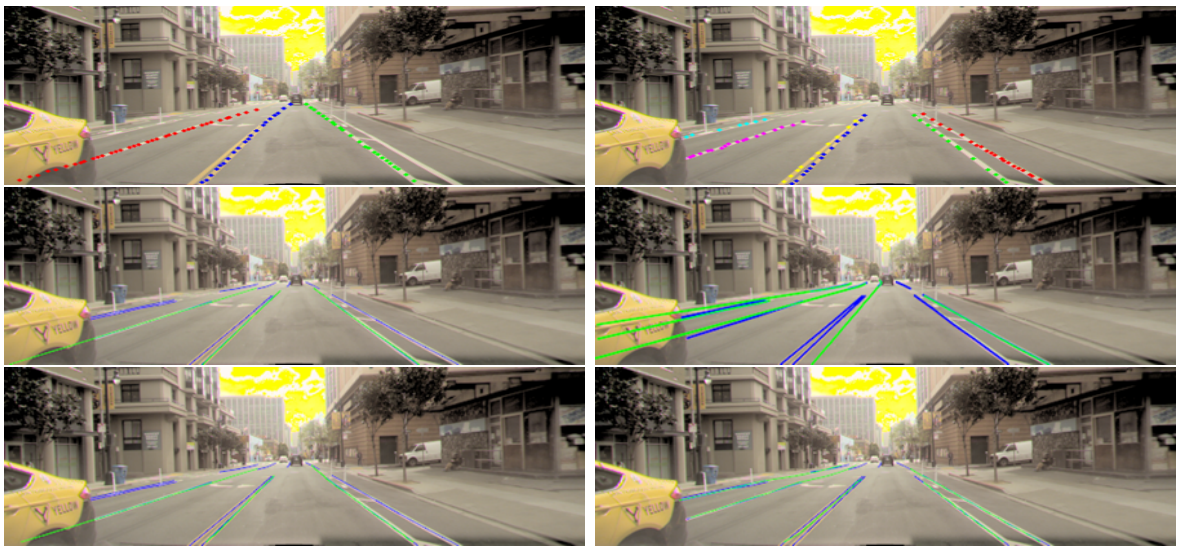


Figure 25: (Top to bottom) Overfitting experiments on PINet, LaneATT and LaneAF, (Left): Before overfitting, (Right): After Overfitting

From the top down, Figure 25 shows the prediction results of PINet, LaneATT and LaneAF respectively, before and after the overfitting experiments. In this scenario, there are occluded lines and multi-marking lines, which are difficult to predict. PINet recognizes all lines that are close to the car, but neglects short lines at the end of the road. LaneATT, on the other hand, extends the predictions to the end but has troubles predicting multi-marking lines. LaneAF shows good performance after overfitting but cannot distinguish multi-marking lines that are close to each other, which can be taken care of by decreasing the width of lines in the ground truth. In addition to this scenario, the overfitting experiments have been conducted on more difficult scenes such as scenes with multiple parallel curve lines and scenes with lane key points. In general, LaneAF outperforms the other 2 model in several difficult scenarios.
After carefully evaluating the performance and characteristics of all architectures, the decision was made to focus on LaneAF.

5 Lane Type Prediction

In this section, the discussion is about the methodology adopted to change the architecture of LaneAF [1] to predict the lane type as well as the lane coordinates. Two approaches were devised in order to tackle this problem.

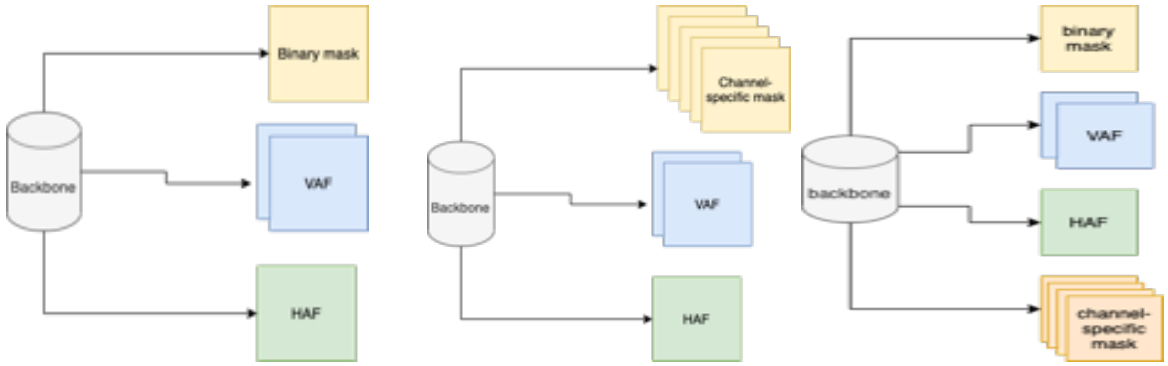


Figure 26: (Left): The conventional pipeline of LaneAF [1]. (Middle): First approach to predict lane type. (Right): Second approach to predict lane type.

As shown in Figure 26, the original implementation has 3 heads as explained in the literature review section. The first head meant to predict a binary mask to distinguish between foreground and background pixels. Thus, the original implementation does not take the lane type into account while producing the foreground pixels. In the first approach, everything is almost the same except the first head. The binary mask head is replaced by a 5-channel head. Each channel represents a pixel type. The values $(0, 1, 2, 3, 4)$ represent background, solid, dashed, zigzag, and unsure types, respectively. The second approach is very similar in spirit to the first approach. However, it keeps the old architecture as is with an extra head for the 4 types to be predicted excluding the background type.
In order to understand how the architecture is changed to accommodate the type in the output, there are 3 main parts to be explained in the following subsections.

5.1 New Ground-truth Maps

This step is a common one for both approaches adopted. The original implementation relied on the segmentation maps generated for each scene. In these segmentation maps each lane is assigned to a unique id value that is greater than 0. Thus, each 0-valued pixel in the segmentation map is a background pixel. Having this, a ground-truth binary mask is easily obtained from each segmentation map and compared against the binary mask produced by the first head of the network. The introduced implementation replaces the normal segmentation map with a segmentation map for the lane type. This type map assigns each pixel a value between 1 to 4 if the pixel is part of a lane, and 0 otherwise. The value obtained is dependent on the lane type, as explained in the introduction of this section. As shown in Figure 27, all dashed lanes in the image are colored yellow and the only solid lane has a different color.

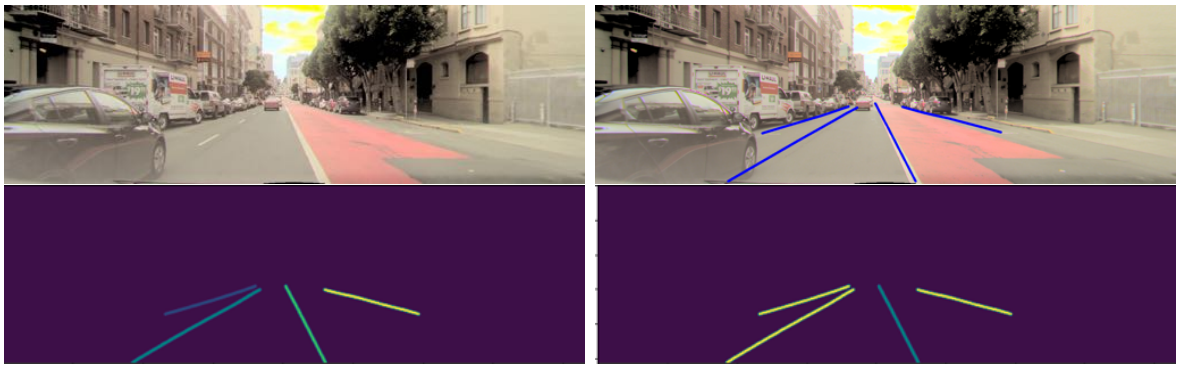


Figure 27: (Top-Left): The original image. (Top-Right): The image with lane annotations. (Bottom-Left): Original segmentation map. (Bottom-Right): Type segmentation map.

5.2 Training Modification

5.2.1 Approach 1

In order to allow for a similar training scheme to the original one, each channel in the 5-channel head, depending on the approach, is treated as a binary mask. In other words, a sigmoid activation is applied to the first channel so that it produces a binary mask, where each 1-valued pixel represents a background pixel and each 0-valued one represents a foreground pixel. Similarly, on the rest of the channels where each channel has its own type of pixels. Also, 5 ground-truth binary masks were generated from the type map using similar logic as the original one. For example, pixels with value 2 in the type map, dashed lane pixels, will be in a mask where they have a value of 1 and the rest are 0-valued pixels. Following this approach, the same type of losses for the original segmentation can be applied on each channel of the type segmentation maps. The "HAF" and the "VAF" heads along with their losses are unaffected by the new modifications.

5.2.2 Approach 2

In approach 2, besides the original implementation, the additional type head is established as the one mentioned in approach 1 except that it only contains 4 channels, each representing solid, dashed, zigzag and unsure, respectively.

5.3 Inference Modification

5.3.1 Approach 1

The inference is done in the following order:

1. The 5-channel head is passed through a softmax head that is applied over the channel dimension. In other words, each pixel location becomes a probability distribution over the 5 channels.

2. A new map is generated, where the pixels values are the indices of the channels with the maximum probability outputted from the previous step.

3. The decoding process in the original implementation required a map with values in the range $[0, 1]$, a single "HAF" channel, and a double-channel "VAF". The aim was to use the same decoding and clustering algorithm devised in the original implementation. Thus, another new map was generated from the output of the previous step. In the new map, each pixel with value in the range $(1, 2, 3, 4)$ is replaced with 0.7 and the 0 values are left as they are.

4. The output of the previous step is a list of clusters, where each cluster represents a unique line. The pixels of each cluster are obtained along with their corresponding pixels in the output of step 2. Then, the majority index from the pixels outputted from step 2 is the type of the predicted lane. In other words, if the majority index is 1, then the line type is solid, and so on.

Following the above steps, the network will be capable of inferring the line coordinates along with their types using almost the same post-processing procedure used for the original implementation.

5.3.2 Approach 2

The inference of approach 2 is done as the following:

- The "HAF"s and "VAF"s are decoded using the binary segmentation mask to get the line instances as the original implementation. This step produces the line clusters with line ids.

- The type head outputs 4 binary segmentation masks, after rounding the sigmoid outputs. The number of positive pixels in each mask is counted based on the indices of the foreground pixels predicted in the previous step. Then, the type of each line is decided using the majority vote.

6 Results

In this section, the results of the two type approaches are shown. Each model in this section is fine-tuned, validated, and tested on the same train, validation, and test splits. The train split contains 1,348 frames, and the validation, and test splits each contains 289 frames. Green lines presented in the images in this section indicate dashed lines, yellow is for solid ones, and red is for zigzag.

6.1 Approach 1

Several experiments have been conducted on approach 1. Different learning rates, freezing several parts of the network, trying different combination of loss functions for the type channels, and loading pre-trained weights on different parts of the network. The setup with the best F1-score (CULane metric) and best classification scores is obtained by following the scheme mentioned below.

- The network's backbone is frozen with pre-trained weights from ImageNet and CULane.
- The "VAF" and "HAF" heads are loaded with the CULane pre-trained weights and are not frozen while fine-tuning.
- For the type head, the weighted binary cross entropy loss, a standard loss for imbalanced binary segmentation tasks, is used. The loss is calculated as:

$$L_{BCE} = -\frac{1}{N}\sum_{i}\left[w \cdot t_i \cdot log(o_i) + (1 - t_i) \cdot log(1 - o_i)\right], \tag{4}$$

  where t_i is the target value for the pixel i and o_i is the sigmoid output. To further account for the imbalanced dataset, an additional intersection over union (IoU) loss [18] is employed:

$$L_{IoU} = \frac{1}{N}\sum_{i}\left[1 - \frac{t_i \cdot o_i}{t_i + o_i - t_i \cdot o_i}\right], \tag{5}$$

  Each loss is applied on each channel separately.
- The learning rate = 0.004, and the batch size = 2. The fine-tuning lasted for 40 epochs. The best model's performance on the validation set was obtained in the 28^{th} epoch.

6.2 Approach 2

The same experiments, including different learning rates, different batch sizes, different loss functions, and freezing parts of the network, have been carried out on approach 2. The setup with the best F1-score (CULane metric) and best classification scores is obtained by following the scheme mentioned below.

1. The original LaneAF model was fine-tuned over the BOSCH split for 30 epochs without freezing any layers.
2. The new type head is added to the output of the previous step. Then, the network's backbone along with the old 3 heads were frozen while fine-tuning.
3. For the type head, a combination of the weighted binary cross-entropy loss with a weight and IoU loss is utilized as loss for each channel.
4. The new type head is trained for 40 epochs. The best model performance on the validation set was achieved in the 32^{th} epoch.
5. The learning rate = 0.0005 and the batch size = 8.

As illustrated in Figures 28,29, both approaches produce very similar line predictions. However, the second approach coordinates are smoother and more accurate than the first

Figure 28: Left images are the original images. Middle images are the first approach predictions. Right images are the second approach predictions.

Figure 29: Left images are with ground truth annotations. Middle images are the first approach predictions. Right images are the second approach predictions.

approach ones. Both approaches failed to detect any type other than dashed or solid. Since the other types represent less than 2% of the dataset. Also, both approaches get confused between solid and dashed types in the curved scenarios as presented in Figure 29 in the first row. From the confusion matrices in Figure 30, it is clear that the second approach is capable of detecting more TPs compared to the first approach. This findings results in a higher recall value and slightly better classification scores per type, especially in the solid type. Both approaches were compared against a new version of LaneAF. This new version does not predict the lane type as the original implementation. However, it has just been fine-tuned on the same splits used for both approaches. Also, the backbone was freezed and the original 3 heads were fine-tuned. The fine-tuning happened on the best LaneAF weights mentioned in the official repository. The comparison is summarized in table 3.

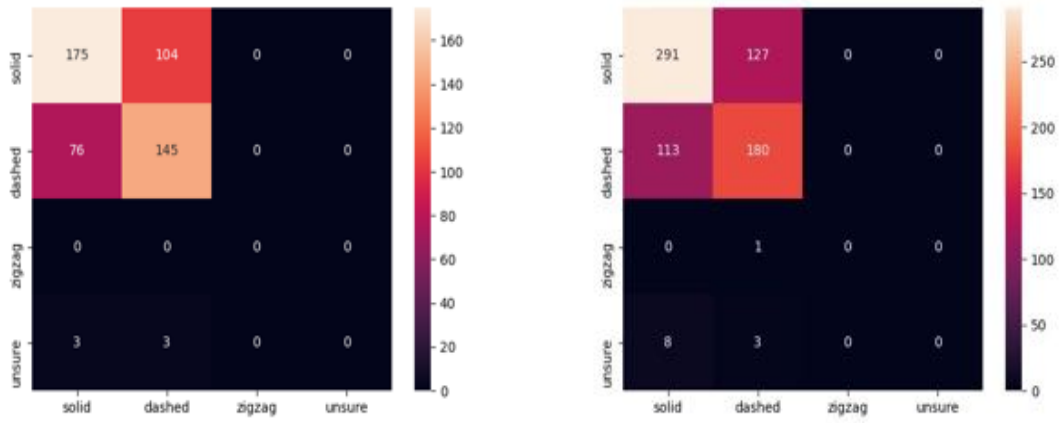


Figure 30: Left is the confusion matrix of the first approach. Right is the confusion matrix for the second approach. The targets are on the Y-axis, and the predictions are on the X-axis.

Model	TP	FP	FN	Precision	Recall	F1-score
LaneAF fine-tuned	497	190	529	0.72	0.48	0.58
Approach 1	506	208	520	0.71	0.49	0.582
Approach 2	723	158	303	0.82	0.7	0.76

Table 3: The original CULane metric evaluation on the two approaches and the fine-tuned version of LaneAF. These metrics account for lane lines coordinates and not for the type.

7 Conclusions

In the first part of this project, a large portion of the BOSCH dataset was annotated which helped the team to be familiar with the problem setup and the dataset corner-cases. Then, several state-of-the-art architectures have been evaluated, they were originally trained on CULane but fine-tuned on the BOSCH dataset. LaneAF resulted to be the most promising one, both in terms of performance, and in terms of the modularity of the code to integrate new features. From the results of the fine-tuned model on the BOSCH dataset, it is clear that the obtained line coordinate accuracy outperforms the original implementation accuracy.
In the second part of this project, several experiments to adapt LaneAF code to detect also additional features have been conducted, like the inclusion of line's type. The achieved results seem to be promising, considering that the model has been trained on roughly 1,500 images. Furthermore, one key feature of this algorithm is to be able to generate the affinity fields with respect to the lines. This enables further research and applications, for example, line width estimation and many more.
To conclude, an algorithm that detects lines along with their types has been developed. Furthermore, possible future developments concerned with the inclusion of additional line semantic features like color, the line's function, or intersection points' detection are possible due to the presence of the affinity fields in the current model.

References

[1] Hala Abualsaud et al. "LaneAF: Robust Multi-Lane Detection with Affinity Fields". In: *arXiv preprint arXiv:2103.12040* (2021).

[2] *Computer Vision Annotation Tool (CVAT)*. https://github.com/openvinotoolkit/cvat. Accessed: 2021-07-13.

[3] Jifeng Dai et al. "Deformable Convolutional Networks". In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. Oct. 2017.

[4] Kaiming He et al. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[5] Kaiming He et al. "Identity mappings in deep residual networks". In: *European conference on computer vision*. Springer. 2016, pp. 630–645.

[6] TuSimple. https://github.com/TuSimple/tusimple-benchmark. In: ().

[7] Paperswithcode. https://paperswithcode.com/task/lane-detection. In: ().

[8] Yeongmin Ko et al. *Key Points Estimation and Point Instance Segmentation Approach for Lane Detection*. 2020. arXiv: 2002.06604 [cs.CV].

[9] *KurvenkrÃ¼mmung in Python*. https://www.delftstack.com/de/howto/numpy/curvature-formula-numpy/. Accessed: 2021-07-13.

[10] Xiang Li et al. "Line-CNN: End-to-End Traffic Line Detection With Line Proposal Unit". In: *IEEE Transactions on Intelligent Transportation Systems* 21.1 (2020), pp. 248–258. DOI: 10.1109/TITS.2019.2890870.

[11] Xingang Pan et al. "Spatial as deep: Spatial cnn for traffic scene understanding". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 32. 1. 2018.

[12] Shaoqing Ren et al. *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*. 2016. arXiv: 1506.01497 [cs.CV].

[13] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-net: Convolutional networks for biomedical image segmentation". In: *International Conference on Medical image computing and computer-assisted intervention*. Springer. 2015, pp. 234–241.

[14] Alexandru Constantin Serban, Erik Poll, and Joost Visser. "A standard driven software architecture for fully autonomous vehicles". In: *2018 IEEE International Conference on Software Architecture Companion (ICSA-C)*. IEEE. 2018, pp. 120–127.

[15] Karen Simonyan and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition". In: *arXiv preprint arXiv:1409.1556* (2014).

[16] Lucas Tabelini et al. *Keep your Eyes on the Lane: Real-time Attention-guided Lane Detection*. 2020. arXiv: 2010.12035 [cs.CV].

[17] Fisher Yu, Dequan Wang, and Trevor Darrell. "Deep Layer Aggregation". In: *CoRR* abs/1707.06484 (2017). arXiv: 1707.06484. URL: http://arxiv.org/abs/1707.06484.

[18] Dingfu Zhou et al. "Iou loss for 2d/3d object detection". In: *2019 International Conference on 3D Vision (3DV)*. IEEE. 2019, pp. 85–94.

Appendix