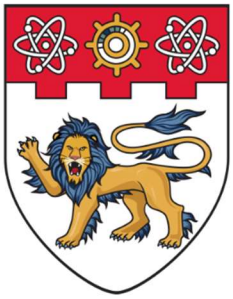
Domain-adaptive lane detection for transitioning weather conditions

Gibson, Gideon Thomas

2024

Gibson, G. T. (2024). Domain-adaptive lane detection for transitioning weather conditions. Final Year Project (FYP), Nanyang Technological University, Singapore. https://hdl.handle.net/10356/177111



NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

Domain-adaptive Lane Detection for Transitioning Weather Conditions

Submitted by

Gideon Thomas Gibson

(Project Number: A3165-231)

Supervisor: Assoc. Prof. Soong Boon Hee

Co-supervisor: Dr. Wang Jian-Gang

SCHOOL OF ELECTRICAL AND ELECTRONIC ENGINEERING

A Final Year Project report presented to the

Nanyang Technological University

In partial fulfilment of the

Requirements for the

Bachelor's Degree of Engineering

April 2024

Abstract

Lane detection is a critical task handled by autonomous vehicles to perceive their environment accurately. Ensuring robust and reliable lane detection capabilities is essential for the safety of the vehicle's operations. However, adverse weather conditions like rain can significantly impede the performance of lane detectors. They introduce rain artifacts in images, which can distort the lane features, making them more difficult to distinguish. Rain also causes road surfaces to be more reflective to light, distorting much of the markings on the road, further causing a greater hindrance to lane detection. When it rains, the road surface also becomes more slippery, making correct and accurate lane detection even more crucial in autonomous vehicles for their safety. Therefore, it is crucial to adapt lane detectors to perform reliably in both clear and rainy conditions.

Currently, state-of-the-art approaches rely on advanced deep learning networks and modules due to their success in various applications. Techniques such as vanishing point guided network (VPGNet) [1] and an efficient neural network known as ENet [22] have all been successfully employed to enhance the detector's ability to generalize to new and unseen road conditions. An addition module has also been developed to be added to ENet to improve its performance, the module uses Self Attention Distillation to accomplish this, and the complete model is known as ENet-SAD [23]. However, only VPGNet has been employed and studied in adverse weather conditions, the other technique has not although it has showed much promise with clear weather datasets. Hence, investigating such a technique in adverse weather conditions as well and combining it with VPGNet holds the potential to achieve adaptive lane detection capabilities in transitioning weather conditions.

For this project, I will be focusing on the above two techniques, ENet-SAD and VPGNet, tailored for the lane detection application in transitioning weather. The investigation will encompass both rainy and clear weather scenarios. By conducting this research, I aim to enhance the safety and reliability of autonomous vehicles, making them better equipped to handle changing weather conditions during lane detection.

I acknowledge that in every model there are both advantages and flaws, in this paper, I will also be expounding on them.

Acknowledgement

I would first like to thank my project supervisor, Associate Professor Soong Boon Hee, and my project co-supervisor, Dr. Wang Jian-Gang, who have both shown continuous concern in my progress and provided words of encouragement towards me in this journey.

Next, I would also like to take this opportunity to express my sincere gratitude towards Dr. Teoh Eam Khwang and Dr. Prabhu Shankar Mahendran, for their kind support, timely assistance and being willing to give up their free time to meet and guide me in the last four months.

Lastly, I would like to thank my family for their unwavering support towards me during my academic journey and their words of affirmation during the low points that keep me going in life.

Gideon Thomas Gibson
2024

Table of Contents

List of Figures

List of Tables

Chapter 1: Introduction

1.1 Background

According to Smart Nation Singapore, a Singapore government agency, autonomous vehicle technology implemented within the public transport system in Singapore could be exceedingly useful in aiding Singapore to overcome its land and manpower constraints. It could also provide the disabled and elderly with more mobility and independence. This technology could also have many applications in the freight transportation and utility services sectors, allowing cargo to be transported in the night without a human driver, hence reducing the load on Singapore's road network during the day and peak hours [3]. With so many applications and benefits, the popularity of autonomous vehicles is skyrocketing, with the market projected to grow by around 750% by the year 2030 [4].

Lane detection is a crucial part of the autonomous vehicle system. With the trends and applications stated in the paragraph above, it is imperative that a reliable and robust lane detection system is developed for the safety of all. Without such a system, autonomous vehicles will go to undesirable locations, causing the general public harm and danger.

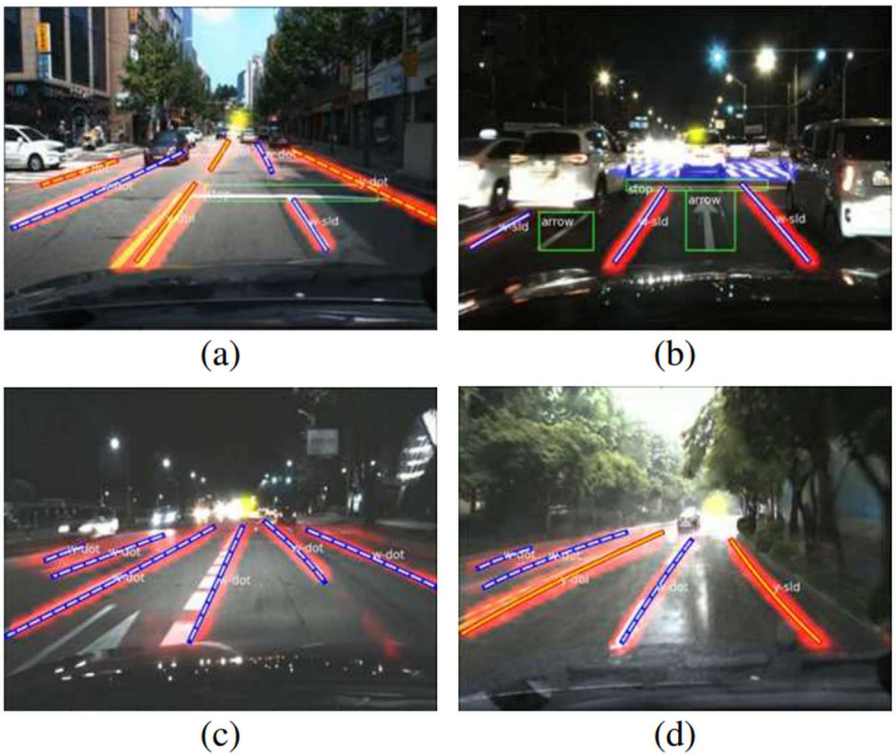


Figure 1-1: Examples of lane detection [1]

Figure 1 shows examples of lane detection. Autonomous vehicles use these images to decide the path they should take and allow them to stay within the lanes of the road. Autonomous vehicles also use these images to identify traffic symbols such as lane and road markings that help the vehicle to understand what traffic regulations to abide by.

1.2 Motivation

For general use of autonomous vehicles, it is evident that lane detection systems should work in all kinds of conditions and not just in ideal conditions, such as clear skies and high illumination conditions. There have been many algorithms based on Convolutional Neural Network (CNN) that have been written for the purpose of lane detection, however, most of them have only been tested on datasets of clear weather conditions. Very little research has been done on lane detection in adverse weather conditions. Hence, this project aims to rectify this and implement some of these models to databases of adverse weather conditions and improve them for such applications.

The models that I have chosen are vanishing point guided network (VPGNet) [1] and an efficient neural network called ENet with a Self Attention Distillation (SAD) module, the complete model is known as ENet-SAD. [23].

1.3 Objectives

The aim of this study is to develop a technique or method that can effectively execute lane detection even in the mist of adverse weather conditions where lane detection becomes a lot more challenging. This aim

will be realized in three different stages. The first stage will be to investigate the performance of the chosen two techniques mentioned in the above paragraph in both clear and rainy weather conditions. Next, the second stage entails the improvement and combination of the techniques mentioned above to create a new method that is more efficient at lane detection in adverse weather conditions. Lastly, I will evaluate my adjusted method with adverse weather conditions dataset and make any required finetuning.

1.4 Scope

In this project, it is assumed that only the frontal view is of interest. Clear weather is defined as weather conditions that have no rain and relatively few clouds in the sky, hence a higher light intensity. Whereas adverse weather is defined as weather conditions that is raining, hence rain drops obscure the image of the road and there are relatively more clouds in the sky, resulting in lower light conditions. The datasets I will be using are a dataset captured in Korea by the research team of the VPGNet research paper [1], and CULane [24].

1.5 Report Organization

Chapter 2: Literature Review

In this chapter, I will summarize some of the models that I have looked at that have been used for lane detection purposes. I will highlight some of their strengths and flaws, and lastly conclude on why I have chosen to focus and implement the few that I did.

Chapter 3: Survey of Existing Approaches

In this chapter, I will delve into some of the most popular models and approaches that have been employed for lane detection to examine their advantages and disadvantages. I will also see how they are lacking and how they can potentially be improved.

Chapter 4: Implementation of ENet-SAD

In this chapter, I will expound on how I have implemented ENet-SAD for lane detection. Training, validation, and testing it using the CULane dataset, followed by concluding on the implementation.

Chapter 5: Conclusion and Recommendation for Future Work

In this chapter, the results of this project will be reviewed to draw conclusions from them. I will also give some recommendations for future works that can be carried out to improve lane detection in adverse weather conditions.

Chapter 2:
Literature Review

2.1 Overview of Lane Detection Techniques

There are three primary ways to execute lane detection[36]. The first is a segmentation approach. This is the most common approach and involves classifying each individual pixel into classes, hence segmenting them into induvial lanes. They typically use an encoder-decoder architecture. Although they are relatively straight forward to engineer, pixel-wise accurate, and can use the learning in one model on other models, they are relatively more time intensive and apply the same technique to each pixel, without drawing the relationship between pixels of a same line.

The second approach is an anchor based one. In this approach, a regression of some values is done. The anchors in this case are predefined lines that are produced over the input, pictures of traffic situations, that indicate likely road lane locations and shapes. For this approach, Intersection over Union (IoU) is used to lineup these anchors with the ground-truth lines. The model then makes predictions as to how to change these anchors to increase their accuracy. Lastly, these models then use non-maxima suppression to keep only the anchors that have a high probability of being accurate.

Lastly, the third approach is a parameter based one. In this approach, polynomial coefficients are regressed to obtain the equation of the traffic lanes. Models that use this technique usually assume that the equations of lanes are in the third order and regression is used to obtain the four coefficients of these equations. Hence, the number of neurons required in these models are four neutrons per line. For example, if there are 3 lanes, it will take 4 lines to define them, hence a total of 16 neurons will be required.

2.2 Domain Adaptation for Semantic Segmentation with Maximum Squares Loss

2.2.1 Overview

The problem with using deep neural networks for semantic segmentation has always been that an extremely large number of samples with pixel-level labels are required if it was to work properly. This is unfeasible and not practical for real-world applications as to create properly pixel-wise semantic labelled datasets would be too time demanding. For example, to correctly annotate one Cityscapes image, it would take ninety minutes [6]. Hence, real-world semantic segmentation datasets all contain very few samples. In order to overcome this shortcoming, synthetic datasets, which can be easily labelled and generated with many different situations have been used. However, this too does not solve the problem as models trained with these synthetic datasets do not perform well when tested on real-world images due to the differences in appearances between the two datasets.

In order to bridge this gap, unsupervised domain adaptation (UDA) techniques and methods have been proposed. For UDA, the source domain is the synthetic dataset and target domain is the datasets of unlabeled real-world images. UDA works by using the data from the target domain to increase the efficiency of the model created from the source domain.

In more recent research, some semi-supervised learning techniques [10][11] have been used to support UDA to great effect. Entropy minimization [10] is one such semi-supervised learning technique that has been very popular in recent times. However, this too has its downside. As the gradient of the entropy minimization method is studied, it can be concluded that a higher prediction probability causes the magnitude of the gradient of the target sample to be larger, as seen in figure 4. If we are to assume that target samples with higher prediction probability are more accurate in self-training to be true [12], samples that are more accurate will be trained more thoroughly than those of lower accuracy. Hence, when using the method described above (entropy minimization on UDA), it works very well only on samples that are easy to transfer as the gradient of the entropy is biased towards such samples. This problem has been termed by the researchers of this paper as probability imbalance.

The formula for entropy and the formula to calculate the gradient function of the entropy are as follows:

$$H(p|x_t) = -p\log p - (1-p)\log(1-p), \quad (1)$$

$$|\frac{dH}{dp}| = |\log p - \log(1-p)|. \quad (2)$$

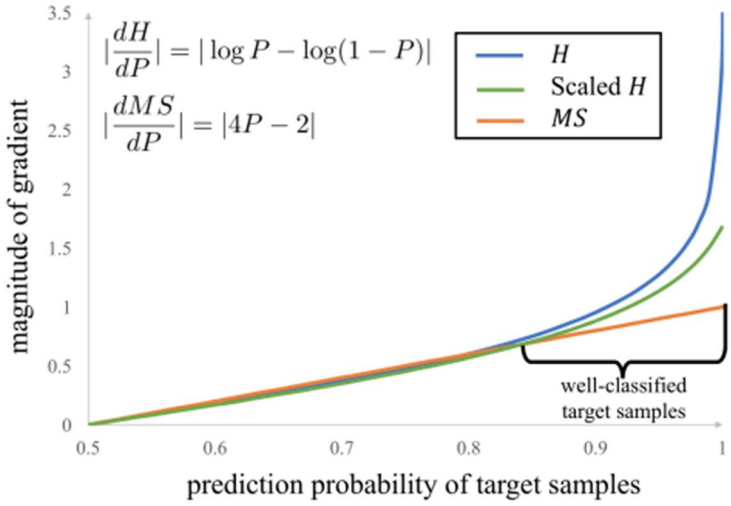


Figure 2-1: A graph of the magnitude of gradient of the entropy minimization method (H) and the maximum squares loss (MS) against the prediction probability of target samples [2]

In order to mitigate this and balance the gradient of accurately classified target samples, this paper proposes a new loss: the maximum squares loss. This loss reduces the above-mentioned problem of the gradient of the entropy being biased towards samples that are easy to transfer, hence stopping them from dominating the training process. Lastly, this paper also introduces the image-wise weighting ratio to alleviate the class imbalance in the unlabeled target domain and Multi-level Self-produced Guidance for UDA. The datasets used in this paper are Cityscapes [5], GTA5 [6], SYNTHIA [7], Office-31[8] and NTHU[9].

2.2.2 Maximum Squares Loss

Maximum squares loss can help with the probability imbalance problem. Different classes such as cars, trucks, road, sky, and bus all have very different probabilities. It has been discovered that classes that can be identified with a high level of accuracy always have higher prediction probabilities as seen in figure 5.

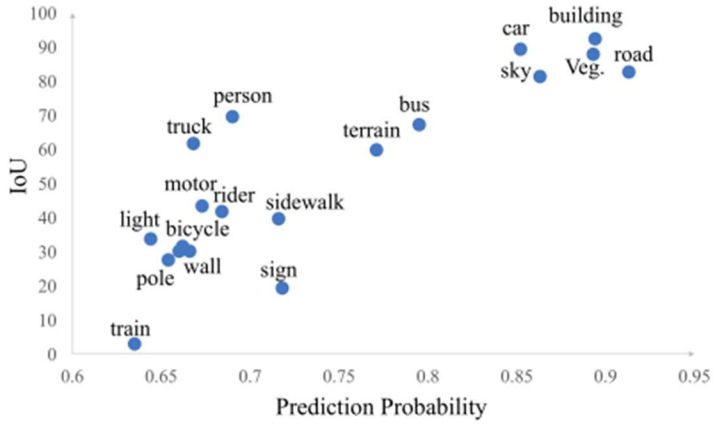


Figure 2-2: A graph of the mean of prediction probability against the Intersection over Union(IoU) for each target class

However, based on equation 2, the rate at which the magnitude of gradient increases of the high probability point is approximated as $|\log p|(p \rightarrow 0)$, which would indicate that it will grow to infinity. This

would mean that a hard to identify class will produce a much smaller gradient on each pixel than a class that is simple to identify, which will result in the aforementioned probability imbalance problem.

In order to solve the probability imbalance problem, this paper has defined maximum squares loss as:

$$\mathcal{L}_T(x_t) = -\frac{1}{2N}\sum_{n=1}^{N}\sum_{c=1}^{C}(p_t^{n,c})^2. \quad (3)$$

The formulation of the maximum squares loss and its gradient function are as follows:

$$MS(p|x_t) = -p^2 - (1-p)^2, \quad (4)$$

$$|\frac{dMS}{dp}| = |4p - 2|. \quad (5)$$

As seen in figure 4 and equations 4 and 5, the maximum squares loss has a magnitude of gradients that is increasing linearly. Therefore, maximum square loss can stop samples with higher prediction probability from producing excessive gradients.

2.2.3 Image-wise Class-balanced Weighting Factor

As we can see from figure 5, classes that have higher prediction probability such as road, building, vegetation, sky, and car always have more pixels on the label map, hence, this leads to an imbalance in the in the label map, with more pixels coming from these classes. Normally, the usual method to rectify this would be to introduce a weighting factor α_c, which is usually set as the inverse class frequency [13]. Unfortunately, for this case, which utilizes UDA, it does not include any class labels for the calculation of the class frequency to be done. Also, because it cannot be confirmed for certain that the class frequency of the source and target domains are the same, it is not possible to replace the target class statistics with the class statistics on the source domain dataset. So instead of using the method above, the researchers in the paper calculated the class frequency on each target image. They did so by using the following equations:

$$m^{n,c^*} = \begin{cases} 1 & \text{if } c^* = \arg\max_c p^{n,c} \\ 0 & \text{otherwise}, \end{cases} \quad (6)$$

$$N^c = \sum_n m^{n,c}. \quad (7)$$

In equation 3, in order to average the loss on the target image, we divided the total sum by *N*. However, instead of doing that, for the new equations the researchers used the number of classes N^c to average the loss. The researchers also used the interpolation between these two numbers to make the equation more stable, as the predictions were inaccurate:

$$\mathcal{L}_T(x_t) = -\sum_{n=1}^{N}\sum_{c=1}^{C}\frac{1}{2(N^c)^{\alpha}\times N^{(1-\alpha)}}(p_t^{n,c})^2, \quad (8)$$

where α is treated as a hyper-parameter to be selected by cross-validation.

2.2.4 Performance Matrices

For this research paper, the two performance matrices being used were the Intersection-over-Union (IoU) of each class and the mean-Intersection-over-Union (mIoU). IoU is a performance matrix that is used to evaluate the accuracy of an object recognition or annotation algorithm. It is calculated by taking the number of true-positives and dividing it by true-positive + true-negative + false-negative, lastly, the result is multiplied by 100 to get a percentage. mIoU is the mean value across all the classes in the dataset.

2.3 Vanishing Point Guided Network for Lane and Road Marking Detection and Recognition

2.3.1 Overview

The VPGNet network was inspired by the work of [14] and [15]. VPGNet introduces a specialized architecture to address the unique characteristics of lane and road markings. The network's competitive advantage lies in its ability to simultaneously detect and recognize these markings while also localizing vanishing points.

To handle the task of detecting and recognizing lane and road markings, VPGNet uses a data layer that induces grid-level annotation. Unlike previous approaches that use a single box regression task, which is suitable for blob-shaped objects like traffic signs or vehicles, VPGNet utilizes a grid-level mask regression. This approach allows points on the grid to be regressed to the nearest grid cell, and the results are combined through a multilabel classification task to represent the object. This enables the integration of two distinct targets—lane and road markings—each with different characteristics and shapes.

The overall design of the network is outlined below in table 1 and figure 2, the VPGNet network performs the following four tasks: grid regression, object detection, multi-label classification, and vanishing point prediction. All the four functions of this algorithm work together to allow the network to detect and classify lane and road markings while predicting the vanishing region in a single forward pass. Notably, the vanishing point detection task contributes to inferring global geometric context during the training of lane and road marking patterns. The structure of the network facilitates efficient and simultaneous processing of multiple tasks related to lane detection and recognition.

Table 1: The VPGNet network structure [1]

Layer	Conv 1	Conv 2	Conv 3	Conv 4	Conv 5	Conv 6	Conv 7	Conv 8
Kernel size, stride, pad	11, 4, 0	5, 1, 2	3, 1, 1	3, 1, 1	3, 1, 1	6, 1, 3	1, 1, 0	1, 1, 0
Pooling size, stride	3, 2	3, 2			3, 2			
Addition	LRN	LRN				Dropout	Dropout, branched	Branched
Receptive field	11	51	99	131	163	355	355	355

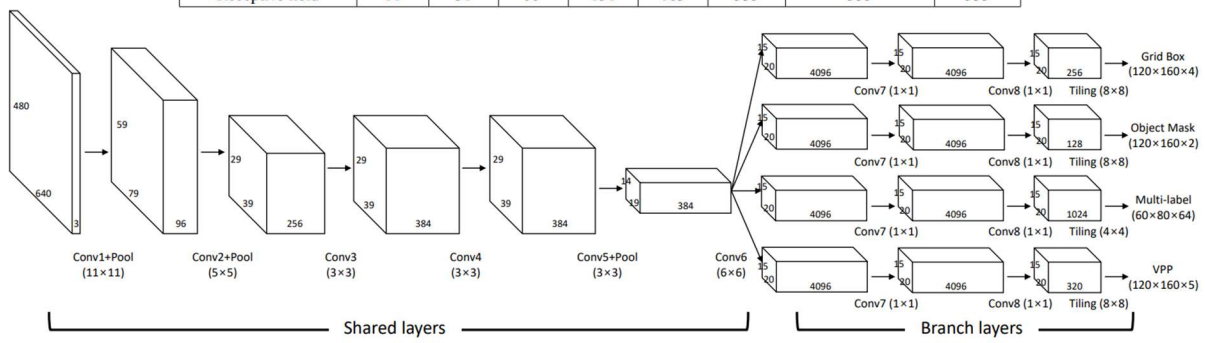


Figure 2-3: A visual representation of the VPGNet network structure [1]

2.3.2 Vanishing Point Prediction Task

Adverse weather conditions, poor illumination, and occlusion can reduce the visibility of lanes. Despite these challenges, humans possess an innate ability to anticipate lane locations based on global information such as nearby road structures and traffic flow [16] [17] [18]. Drawing inspiration from this

research, the developers of the VPGNet network developed a Vanishing Point Prediction (VPP) module to make the VPGNet network more efficient and reliable at lane detection in adverse weather conditions. The VPP copies the predictive ability of humans based on the research previously mentioned.

The vanishing point is defined as the point on the horizon where parallel lines in three-dimensional space converge to a two-dimensional plane, typically observed in driving scenarios where lane and road markings converge to a single point. This concept is crucial for providing a global geometric context, aiding in the inference of lane and road marking locations. The VPP module is integrated into a multi-task network to train the geometric patterns associated with lane convergence to a single point.

Previous work by Borji [19] demonstrated that a Convolutional Neural Network (CNN) can localize the vanishing point. This involved vectorizing the spatial output of the network to predict the exact location of the vanishing point using a SoftMax classifier.

2.3.3 Post-Processing

All the different components that the algorithm finds such as lanes, road markings and VPs have to be used in real-world applications, hence they must be represented in an applicable manner for this. Therefore, post-processing techniques are used to obtain images that are visually useful for such a purpose. For the instance of lane classes, point sampling, clustering, and lane regression are methods that are used. Whereas, for the road marking classes, we apply grid sampling and box clustering.

Lastly, in order to post-process our VP, the 4 quadrants are used. The point at which the four quadrants intersect is the VP. In other words, a point is required to be found where the confidences of the quadrant channels become very close to one another. Equations 9 and 10 formulate the boundary intersection of each quadrant:

$$P_{avg} = \frac{1 - (\sum p_0(x,y))/(m \times n)}{4} \quad (9)$$

$$loc_{vp} = \arg\min_{(x,y)} \sum_{n=1}^{4} |P_{avg} - p_n(x,y)|^2 \quad (10)$$

where P_{avg} is the probability that a VP exists in the image, p_n(x, y) is the confidence of (x, y) on n_{th} channel (n = 0: absence channel), $m \times n$ is the confidence map size, and loc_{vp} is the location of the VP.

2.3.4 Performance Matrices

In order to evaluate the accuracy of lane detection, the authors of this paper computed the minimum distance from the center of each cell to the sampled lane point for every cell. A threshold of R was set, and if the distance above fell within the threshold R, the sample point would be labelled as true positive, and the corresponding grid cell will be labelled as correctly detected. The F1 score for this was also measured for comparison.

In order to evaluate the accuracy of road marking detection, the authors of this paper computed every predicted cell that overlapped with the ground truth grid cells. These overlapped cells were labeled as true positive cells. If more than 50% of the predicted cells were true positive cells over a clustered blob, the road marking would be labelled as correctly identified. The recall score for this was also measured for comparison.

In order to evaluate the accuracy of vanishing point, the authors of this paper computed the Euclidean distance between the predicted vanishing point and a ground truth point. The recall score is also evaluated by varying the threshold distance R from the ground truth vanishing point.

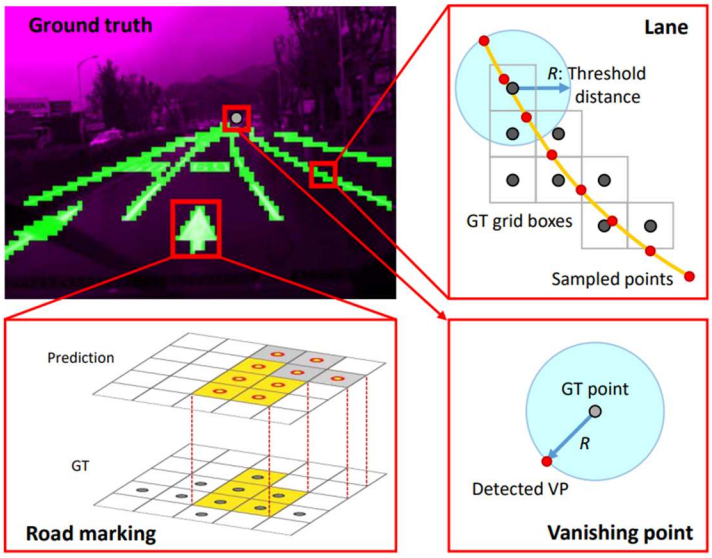


Figure 2-4: A visual representation of the performance matrices mentioned above [1]

2.4 ENet: A Deep Neural Network Architecture for Real-Time Semantic Segmentation

2.4.1 Overview

For real-time lane detection in real world autonomous vehicle applications, the ability to execute pixel-wise semantic segmentation, which is to categorize each pixel in an image to a certain class, in real-time is both critical and essential. This operation should be able to be performed with little delay and requiring little power on a mobile device. With many large datasets being created and more powerful computers as well, neural networks are becoming increasingly popular. These factors are the motivation for this paper. This paper proposes a neutral network architecture, known as ENet, that was crafted to be very fast and accurate for real world lane detection in autonomous vehicles.

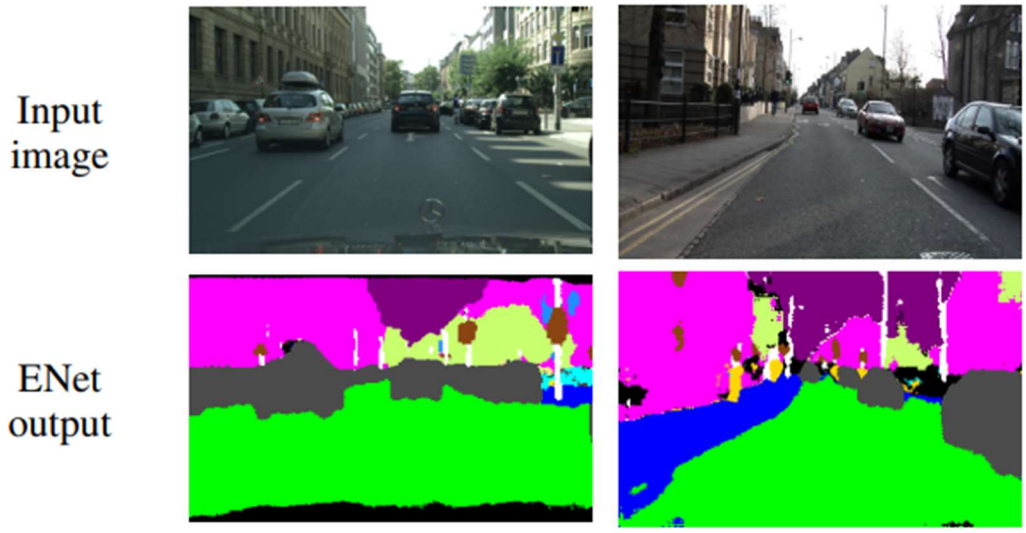


Figure 2-5: The predictions from the ENet model [22]

Deep neural networks are the most common architectures when it comes to state-of-the-art computer vision and semantic segmentation and is no different in this case. ENet is a CNN which combines an encoder and a decoder to achieve its results. The encoder being used is a standard CNN, similar to VGG16 [25]. The encoder is trained to categorize the inputs accordingly. The decoder is use to increase the sampling of the output of the encoder [26, 27, 28, 29, 30]. Current models that attempt this are slow and inaccurate, hence this model was created to rectify that.

2.4.2 Network Architecture

The network architecture is shown in the table, Table 2-1, and figure, Figure 2-5, below.

Table 2-1: The architecture of ENet with an example input of 512x512 [22]

Name	Type	Output size
initial		$16 \times 256 \times 256$
bottleneck1.0	downsampling	$64 \times 128 \times 128$
$4\times$ bottleneck1.x		$64 \times 128 \times 128$
bottleneck2.0	downsampling	$128 \times 64 \times 64$
bottleneck2.1		$128 \times 64 \times 64$
bottleneck2.2	dilated 2	$128 \times 64 \times 64$
bottleneck2.3	asymmetric 5	$128 \times 64 \times 64$
bottleneck2.4	dilated 4	$128 \times 64 \times 64$
bottleneck2.5		$128 \times 64 \times 64$
bottleneck2.6	dilated 8	$128 \times 64 \times 64$
bottleneck2.7	asymmetric 5	$128 \times 64 \times 64$
bottleneck2.8	dilated 16	$128 \times 64 \times 64$
Repeat section 2, without bottleneck2.0		
bottleneck4.0	upsampling	$64 \times 128 \times 128$
bottleneck4.1		$64 \times 128 \times 128$
bottleneck4.2		$64 \times 128 \times 128$
bottleneck5.0	upsampling	$16 \times 256 \times 256$
bottleneck5.1		$16 \times 256 \times 256$
fullconv		$C \times 512 \times 512$

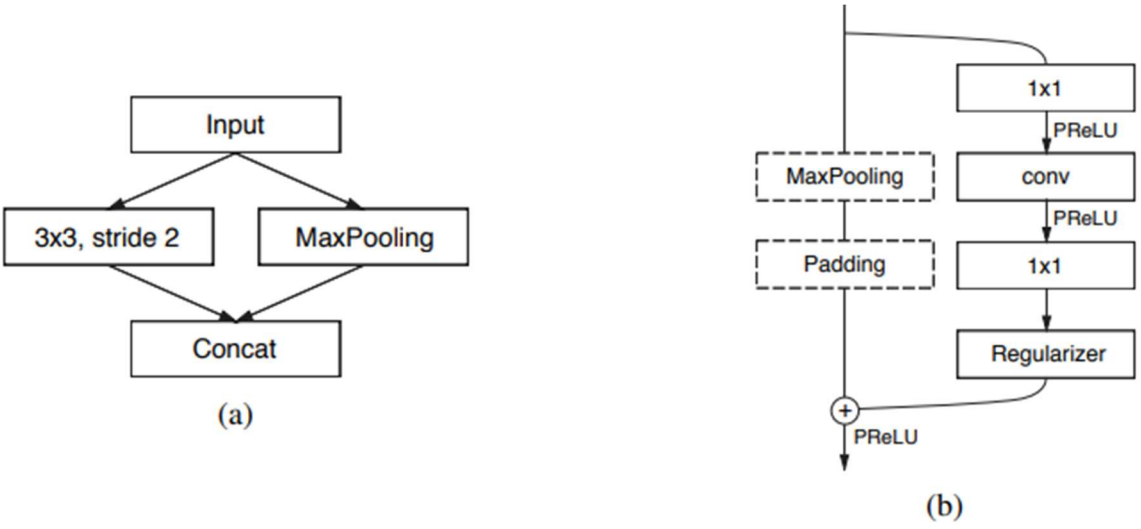


Figure 2-6: The network architecture of ENet [22]

As seen from Table 2-1, the network has serval stages. As shown in Figure 2-5b, there is a one primary route that has extensions that diverge from it that contain filters that perform convolution. All these

branches than merge back. This method has been largely inspired by ResNets [31]. Also, from Figure 2-5b, it can be seen that in between each convolution, batch normalization [32] and PReLU [33] have been placed. The initial stage is depicted in Figure 2-5a.

2.4.3 Performance

The main performance matric used to determine accuracy is instance-level intersection over union metric (iIoU). Which is IoU, which has been touched on above, weighed by the average object size. Using this performance matric, ENet performs exceedingly well when tested against the Cityscapes dataset [5]. The table below, Table 2-2, shows these results.

Table 2-2: The results of ENet when tested against Cityscapes [22]

Model	Class IoU	Class iIoU	Category IoU	Category iIoU
SegNet	56.1	34.2	79.8	**66.4**
ENet	**58.3**	**34.4**	**80.4**	64.0

2.5 Learning Lightweight Lane Detection CNNs by Self Attention Distillation

2.5.1 Overview

It is very difficult to train deep convolution neural network models for the purpose of lane detection as the datasets that are used in the training process has lane annotations that have very little and unclear supervisory signals. If these lane annotations are all these models have to train without anything with more context in them, these models will perform very badly when it comes to more difficult to interpret scenes. Scenes that have very bad lighting conditions and scenes with the lanes are not very obvious due to fading of lane markings for example. Hence, the motivation for this paper. In this paper, Self Attention Distillation (SAD) is used to allow the model that it is added to, ENet, to learn from itself to improve the accuracy without the need for more human effort, annotating datasets, and external supervision. In SAD, it takes advantage of the attention maps that are created in its own layers, that contains a lot of context, to train the lower layers. This is the first model that uses the model's own attention maps as the distillation targets.

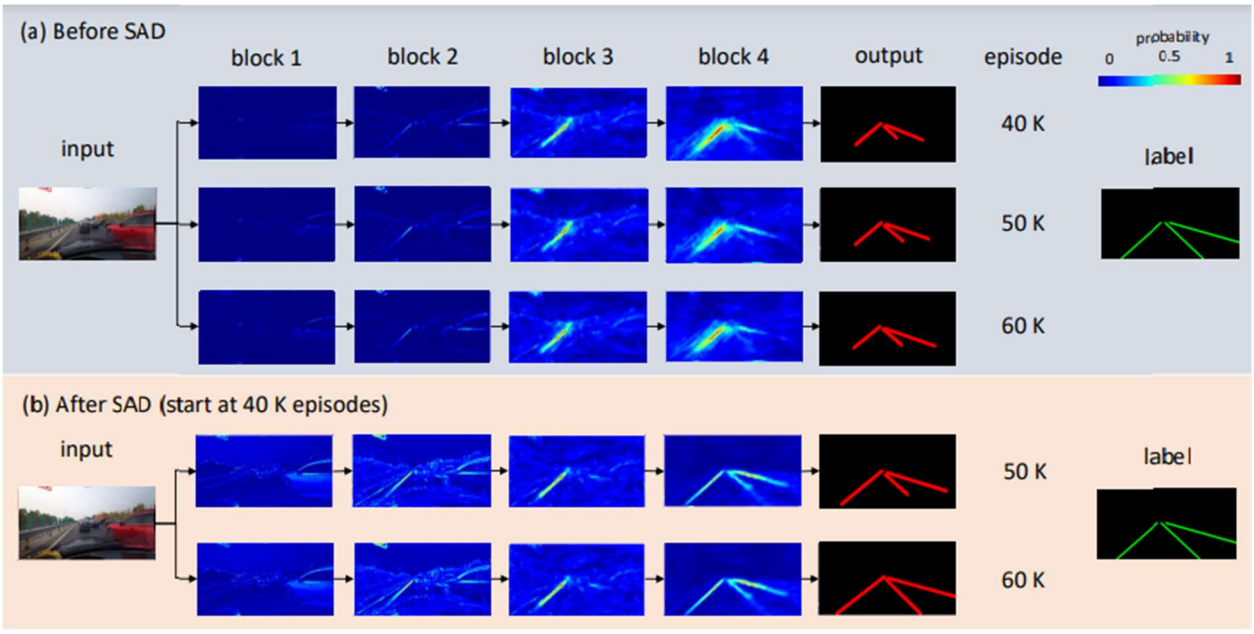


Figure 2-7: (a) ENet attention maps before SAD (b) ENet attention maps after SAD [23]

2.5.2 Self Attention Distillation

There are 2 categories of attention maps that have been discussed above. The first category is activation-based while the other category is gradient-based [34]. From experimentation, this paper has proved that gradient-based attention maps have produced results that indicated massive improvements, while the other category produces hardly any improvements. Hence, gradient-based attention maps will be used in for this model.

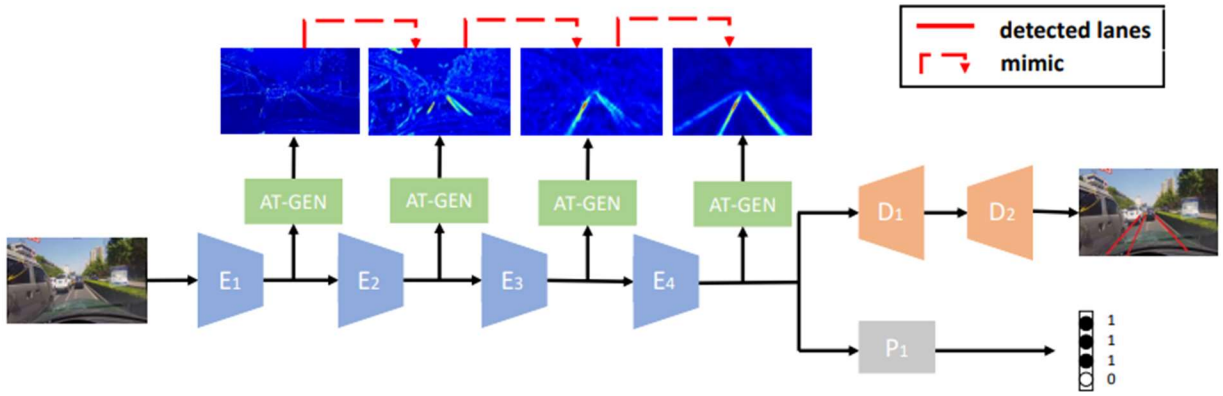


Figure 2-8: An example of how SAD works [23]

Figure 2-7 shows how SAD works. E1 to E4 are the encoders of ENet, while D1 and D2 are the decoders of ENet. Both of which have be touched on above when discussing ENet. P1 is the small network that was added when SAD was added to the model in order to predict the presence of lanes in the maps. Lastly, AT-GEN refers to attention generators.

2.5.3 Performance

I will be focusing on the performance matric used for the CULane dataset as that was the dataset that I Concentrated on in the implementation of this model. Using IoU, which has been touched on above, prediction with IoUs greater than 0.5 will be treated as true positive (TP). Using TP, the performance matric used, F1, will be calculated. In order to calculate f1:

$$F_1 = \frac{2\times Precision\times Recall}{Precision+Recall} \text{ where } Precision = \frac{TP}{TP+FP} \text{ and } Recall = \frac{TP}{TP+FN}$$

Table 2-3: The performance (F1-measure) of various model on CULane testing set [23]

Category	Proportion	ENet-SAD	R-18-SAD	R-34-SAD	R-101-SAD	ResNet-101	SCNN
Normal	27.7%	90.1	89.8	89.9	**90.7**	90.2	90.6
Crowded	23.4%	68.8	68.1	68.5	**70.0**	68.2	69.7
Night	20.3%	66.0	64.2	64.6	**66.3**	65.9	66.1
No line	11.7%	41.6	42.5	42.2	**43.5**	41.7	43.4
Shadow	2.7%	65.9	67.5	**67.7**	67.0	64.6	66.9
Arrow	2.6%	84.0	83.9	83.8	**84.4**	84.0	84.1
Dazzle light	1.4%	**60.2**	59.8	59.9	59.9	59.8	58.5
Curve	1.2%	65.7	65.5	**66.0**	65.7	65.5	64.4
Crossroad	9.0%	1998	1995	**1960**	2052	2183	1990
Total	–	70.8	70.5	70.7	**71.8**	70.8	71.6
Runtime (ms)	–	**13.4**	25.3	50.5	171.2	171.2	133.5
Parameter (M)	–	**0.98**	12.41	22.72	52.53	52.53	20.72

As seen in the table above, Table 2-3, ENet-SAD performs the best when it comes to speed. Its accuracy also performs very well, only slightly behind R-101-SAD. The proportion column in the above table, Table 2-3, refers to the proportion of the CULane test set that falls under that category.

2.6 Spatial As Deep: Spatial CNN for Traffic Scene Understanding

2.6.1 Overview

The use of convolution neural networks (CNNs), a network of convolutional operations layered on top of one another, is very common in object recognition and this includes lane detection too. It has been proven to be effective at drawing semantics from raw pixels, however, its ability to connect pixels across rows and columns in a traffic picture with spatial relations has not been looked into as thoroughly. For real world applications of lane detection, the model has to be able to detect lanes even if they are partially obstructed by other objects or have faded lane markings. This is an aspect in which CNN has performed particularly poorly on. It is not able to detect long structure regions that might not be so clear either due to obstructions or fading of lane markings. In order to tackle this problem, this paper put forward the Spatial CNN (SCNN) model.

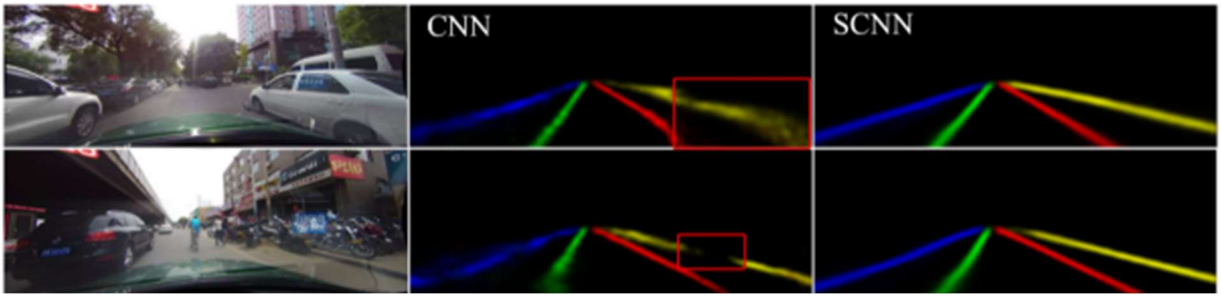


Figure 2-9: CNN and SCNN being compared in lane detection when there are obstructions to the lane or the lane marking is faded [21]

2.6.2 Spatial CNN

In SCNN, a deep CNN network is generalized to a rich spatial layer. For traditional CNNs, each convolutional layer will receive an image from the preceding layer, execute a convolutional and a nonlinear activation operation on it, and passes it on to the subsequent layer. In similar fashion, SCNN sees rows and columns in an image as a layer and applies the above sequence.

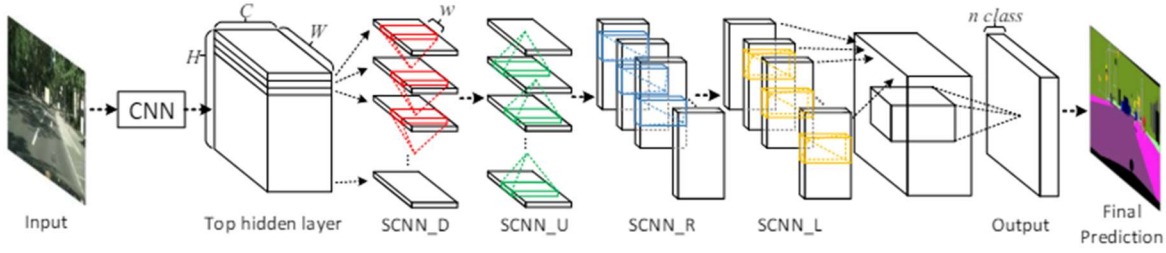


Figure 2-10: The implementation of SCNN in this paper [21]

As seen in Figure 2-10, the SCNN technique is implemented on a three-dimensional tensor where C is the number of channels, H is the number of rows and W is the number of columns. The tensor is separated

into H different parts, with the first part being feed into a convolutional layer that consists of C numbers of kernels with the size of C x w, with w being the width of the kernel. The resulting output will be input into the next part of the tensor to provide a new part which will be fed to the subsequent convolution layer. This sequence will continue until the last part of the tensor has been updated.

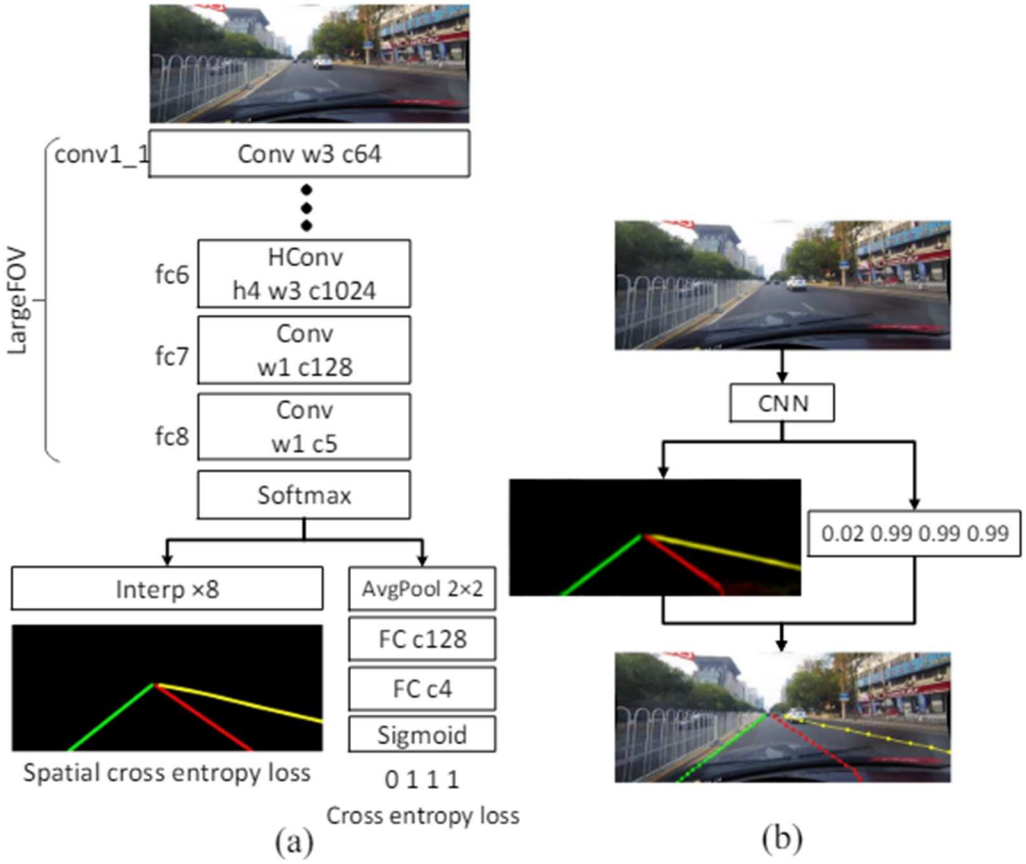


Figure 2-11: (a) The SCNN training model, (b) The process of predicting the lanes [21]

As seen in Figure 2-11, (a) shows the SCNN training model with 'Conv' representing the convolution layer, 'HConv' representing the atrous convolution layer and 'FC' standing for fully connected layer. 'c' represents the number of output channels, 'w' represents the width of the kernel and 'h' represents the rate for the atrous convolution.

2.6.3 Performance

The model was trained and tested using the Cityscapes dataset [5]. The performance matric used is F-measure, with F-measure being calculated with:

$(1+\beta^2)\frac{\text{Precision Recall}}{\beta^2\text{Precision}+\text{Recall}}$,where $\text{Precision} = \frac{TP}{TP+FP}$ and $\text{Recall} = \frac{TP}{TP+FN}$. β is set to 1.

True positives (TPs) are predictions that have IoUs above 0.3 for loose evaluations and 0.5 for strict evaluations.

Figure 2-12: Evaluation of the predictions which is based on IoU [21]

As seen in Figure 2-12, the ground truth is represented by the green lines, while the blue lines represent the true positives (TPs) and the red lines represent the false positives (FPs).

Table 2-4: The results of SCNN compared to other methods. Only FPs are shown for crossroad [21]

Category	Baseline	ReNet	DenseCRF	MRFNet	ResNet-50	ResNet-101	Baseline+SCNN
Normal	83.1	83.3	81.3	86.3	87.4	90.2	**90.6**
Crowded	61.0	60.5	58.8	65.2	64.1	68.2	**69.7**
Night	56.9	56.3	54.2	61.3	60.6	65.9	**66.1**
No line	34.0	34.5	31.9	37.2	38.1	41.7	**43.4**
Shadow	54.7	55.0	56.3	59.3	60.7	64.6	**66.9**
Arrow	74.0	74.1	71.2	76.9	79.0	84.0	**84.1**
Dazzle light	49.9	48.2	46.2	53.7	54.1	**59.8**	58.5
Curve	61.0	59.9	57.8	62.3	59.8	**65.5**	64.4
Crossroad	2060	2296	2253	**1837**	2505	2183	1990
Total	63.2	62.9	61.0	67.0	66.7	70.8	**71.6**

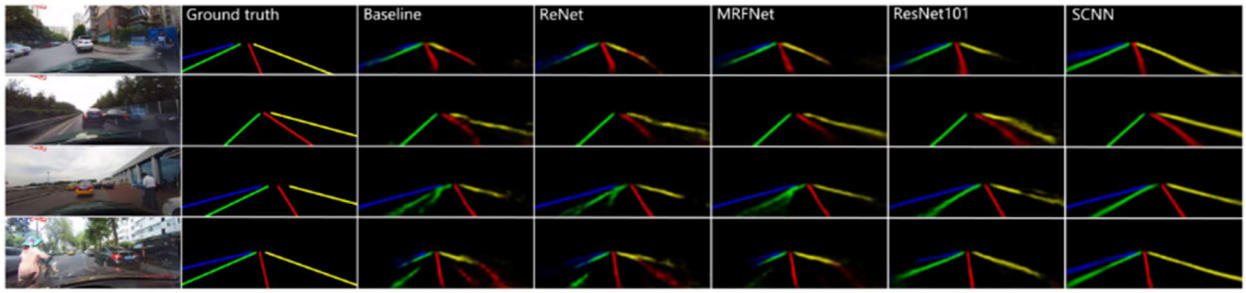


Figure 2-13: The generated probmaps of SCNN compared to other models [21]

As seen from Table 2-4 and Figure 2-13, the results of SCNN are very good and better as compared to other modern models.

2.7 Vision-Based Robust Lane Detection and Tracking in Challenging Conditions

2.7.1 Overview

There are four main categories of difficulties that techniques of lane detection experience. The first is the different lighting effects there are in a traffic situation created by the sun or other human made light sources. This causes glare and shadows that obscure lane markings, making them difficult to detect by lane detection techniques. This is made worst in wet weather conditions, where light is reflected. The second category is lane markings can be more difficult to identify due to cracked lanes, erosion, blur, and obstruction. This is usually caused by natural wear and tear, natural calamities and bad weather like snow and rain. The third category is when lane markings are blocked from view by various objects in frame of the camera. This can be the wiper of the vehicle, other vehicles, or traffic signs. The last category is when there are other lines that are captured in frame by the camera that can easily be mistaken for a lane but is not a lane. This includes road dividers, guardrails, and snow buildup alongside the road. In order to counter these challenges, the paper has proposed a straightforward, robust, and real-time lane detection and tracking technique.

2.7.2 Network Architecture

The first main contribution of the paper is to introduce a comprehensive intensity threshold range (CITR) at the stage where the edge is detected. This will increase the accuracy of the canny edge detector, especially for the edges of lanes that are obstructed or very difficult to see. The second contribution of this paper is to introduce a region of interest (ROI) that is shaped as an isosceles trapezoid so as to get rid of edges that are not wanted and tend to be very noisy. Within this ROI, Hough Transform is utilized in order to identify lines that might be lanes. These lines are known as candidate lane lines (CLL). In order to prevent the left and right lines from being dependent on the location of one another, the CLLs are split into two categories by means of slope-based constraint. The first category is candidate left lane lines (CLLLs) and the second category is candidate right lane lines (CRLLs). The third contribution of this paper is to introduce a method to better verify the validity of the lanes that have been detected. This method consists of two parts, and both parts use geometric constraints to check the characteristics of the detected lane as that wrongly detected lanes are avoided. The first part of this method is called angle based geometric constraint (AGC). Based on geometry, it can be easily spotted that lanes on the left form an acute angle with the x-axis while lanes on the right form an obtuse angle with the x-axis. With this knowledge, a constraint can be introduced to define the range left and right lanes have to fall into. Lines outside this range have a lower likelihood of being a lane. The range for the left lane is $[45°-c,\ 45°+c]$ while the range for the right lane is $[135°-c,\ 135°+c]$, where c is the variable for position of camera. Lines that were detected and fall within this range are called filtered candidate left lane lines (FCLLLs) for the lines detected on the left and filtered candidate right lane lines (FCRLLs) for the lines detected on the right. The second part to this method is called length based geometric constraint (LGC). This constraint leverages on the fact that it is most common that the camera will be placed in the middle of the car, and the this causes the right and left lanes to be found at the immediate front of the vehicle. This causes the left and right lanes to have the longest length out of all the detected lines. Hence, the line that is the longest

among the FCLLLs is taken to be the left lane marking and the line that is the longest among the FCRLLs is taken to be the right lane marking. The last contribution of the paper is to introduce a novel technique that aids in the tracking of the lanes. This is done by trying to predict the lane location of the subsequent frame by defining a range of horizontal lane positions (RHLPs) that are located along the horizontal axis and will be defined with relation to the lane position of the preceding frame. This allows it to continue tracking the lanes even if it is obstructed.

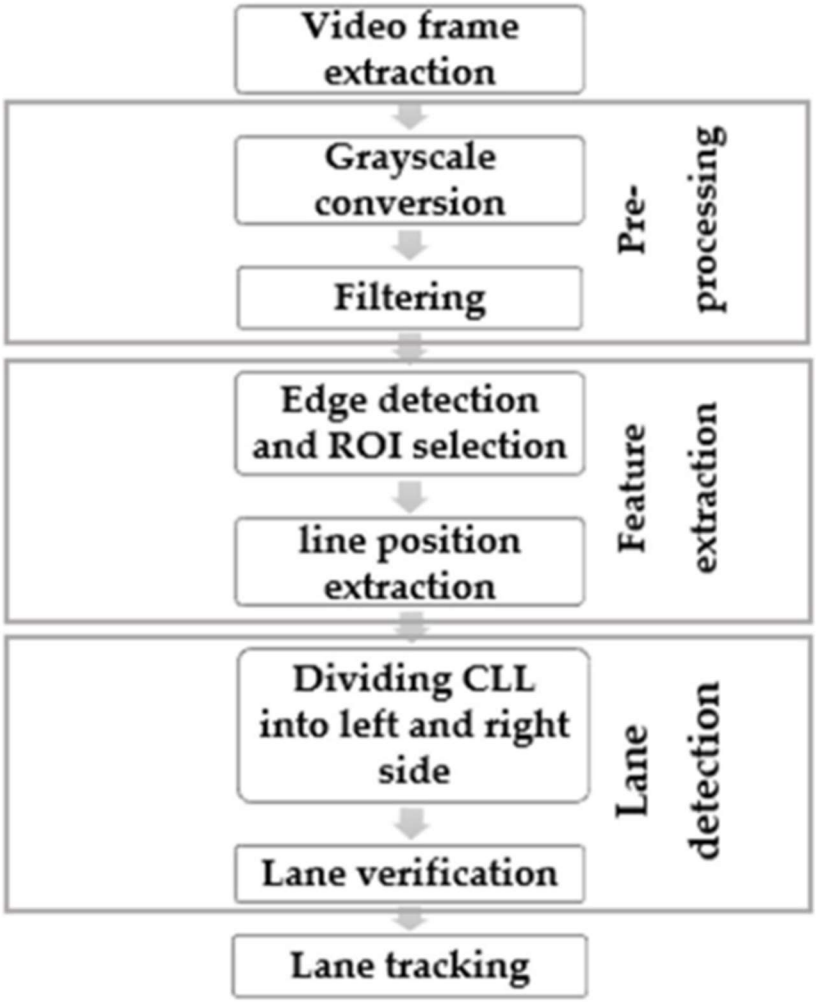


Figure 2-14: The network architecture of this technique [37]

As seen from Figure 2-14, the model starts with pre-processing. This is to get rid of the various noises that can be present in the image as so to increase the likelihood that the lanes can be detected accurately in the subsequent steps. The pre-processing involves the smoothing of the picture by using conventional filters such as Bilateral and Gaussian filters.

2.7.3 Performance

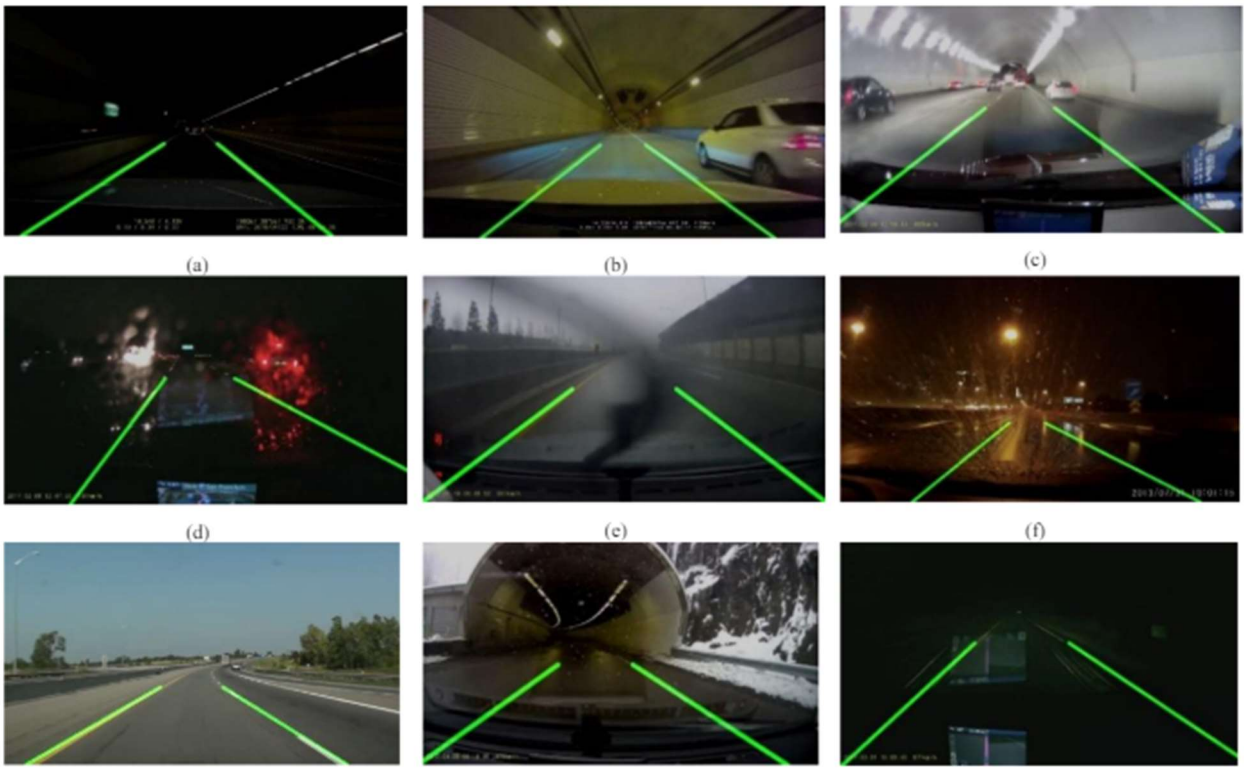


Figure 2-15: Using the proposed model to accurately detect lanes [37]

There are two performance metrics that are used. The first is detection rate (DR) which is calculated by:

$DR = \frac{CDF}{\mathrm{TF}} \times 100\%$, where CDF represents the number of frames that have been correctly detected and TF represents the total number of frames. The other matric is the processing time (ms/frame).

Table 2-5: A table of the performance of the proposed model compared to other models in different conditions [37]

	Detection rate (%)								Processing time (ms/frame)
	Day				Night				
	Clear	rain	Tunnel	Snow	Clear	Rain	Tunnel	Snow	
Son [9]	93.1(4656)*	89 (1300)*	93.2 (1000)*	-	94.3 (3000)*	93 (450)*	98.1(790)*	-	33
Jung[13]	88~98	-	-	-	87.7	-	-	-	39
Lee[1]	**99**	96.9	-	93	98	93.6	-	92.2	35.4
Marzougui [6]	90~93.5 (2218)*	90.20 (847)*	-	-	96.4(1440)*	-	98.85 (1489)*	-	21.54
Proposed	98.3 (6009)*	**96.9 (5836)***	**98.4 (1200)***	**95.3 (4400)***	**99.03 (5455)***	**96.5 (8987)***	**98.34 (3000)***	**94.2 (1464)***	22.32

*Number of frames used to evaluate the performance

As seen from Table 2-5, the performance of the proposed model is relatively good, outperforming many other advanced models. The proposed model is also very fast, with the second lowest processing time when compared to the other models. With high accuracy and speed, this model can be view as a one that can be applied to real world applications where real time implementation and reliability are of the utmost importance to ensure safety.

2.8 Concluding Remarks

In conclusion, there are a whole variety of methods and techniques that can be employed, each with their various pros and cons. Techniques can also be combined to cover the flaws of a certain method. The papers were also written at different times and the shortcoming of one method might have been remedied in a more recent paper. Hence, it is important to do a survey on the methods and techniques in order to have a broad idea of the lane detection landscape so as to better identify where the gaps are and what can be done to make a improvements to the landscape in order to move towards a truly real time, adaptable and robust solution to lane detection that can be implemented in the real world safely.

Chapter 3:
Survey of the Existing Approaches

3.1 Overview

There are so many different techniques and method in the landscape that can be utilized for lane detection. In order to get a better understanding of them and know their intricacies, advantages and disadvantages, a survey was done. Another reason for the survey was to see how the methods could complement each other and see how they could be further improved.

3.2 Domain Adaptation for Semantic Segmentation with Maximum Squares Loss

3.2.1 Advantages

One advantage of this method is that it tackles one of the biggest problems of using semantic segmentation in lane detection. That problem is that executing this learning takes a large dataset that is labelled, and this takes a lot of time, effort, and money. This paper solves this problem by using unsupervised domain adaptation (UDA) techniques to use what has been learnt from labeled synthetic datasets, such as SYNTHIA, on real world datasets that are not labeled. These labeled synthetic datasets are easy to generate.

Another advantage is that this method has proven to tackle a problem associated with entropy minimization, that is essential when employing UAD. This problem is that there is probability imbalance when entropy minimization is done. The paper tackles this by introducing the maximum square loss

technique. This technique takes classes with very high confidences and balances their gradients. This greatly strengthens the training process, especially for samples that are very challenging to transfer. This has resulted in very effective outcomes as compared to other cutting-edge methods.

3.2.2 Disadvantages

The first disadvantage is that the model that this paper is introducing has additional parameters that would necessitate intricate fine tuning that will inevitably increase the complexity of the technique. These parameters include the scale ratio, γ, and the weighting factor in relation to the image, α.

Another disadvantage is that this model is only tested against a few select other models and not tested on various datasets that contain more challenging scenarios.

3.2.3 Potential Improvements

The first potential improvement is that techniques can be devised in order to automatically fine tune the parameters that have been talked about above, γ and α. This can improve the performance of the model even when different datasets and scenarios are fed into the model.

Another potential improvement is seeing how the model stands up to other datasets. Datasets of real world and varying scenarios, including adverse weather. This could show that this model is indeed very accurate and can be generalized to many situations.

Lastly, this model can be integrated with other models in related fields that could complement this model, one such complementary model could be semi-supervised learning or domain adaptation. This could help to increase the effectiveness and accuracy of the model.

3.3 Vanishing Point Guided Network for Lane and Road Marking Detection and Recognition

3.3.1 Advantages

The first advantage of VPGNet is that it is a very complete and robust network, able to handle a wide range of tasks that are having to do with lane and traffic marking detection, such as vanishing point prediction.

Another advantage that this model has is that it has been tested against very challenging datasets that include adverse weather conditions such as rain and low light. This paper also created its own adverse weather conditions dataset to test the model as there were none that were deemed satisfactory. Not only has it been tested against these datasets, but it has also been proven to be effective in such scenarios.

The last advantage of this model is that it is able to produce the outstanding accuracy above in real-time as the speed it takes to process the inputs is very fast. This makes it suitable for real world applications.

3.3.2 Disadvantages

One disadvantage of this model is that the task of identifying the lanes in the images and the task to predict the vanishing point are dependent on one another. This could result in the learning being biased and hence, affecting the accuracy of the model.

Another disadvantage is that the model is very complex, it consists of two parts. The first part is training, involving just the task to predict the vanishing point, while the second part is training, involving all the tasks mentioned in the paper. This could result in greater room for error and the need to handle all the parameters with greater care.

The last disadvantage is that there could be an imbalance in the created dataset as some classes are more common than others.

3.3.3 Potential Improvements

In order to reduce the problem of dependency addressed above, method that have been designed with executing multiple tasks in the same model in mind can be explored to reduce this problem and increase the accuracy of the model. Such methods include dynamic loss weighting or curriculum learning.

Another potential improvement is to further augment the created dataset so as to address the problem of imbalance classes. This could greatly help the model to generalize to a wide variety of situations.

3.4 ENet: A Deep Neural Network Architecture for Real-Time Semantic Segmentation

3.4.1 Advantages

The first advantage is that this model solves the problem of very high costs of processing big input frames, which many popular architectures do not address this problem. It does this by improving the initial stages as they are by far the costliest. ENet solves this problem by designing its first 2 stages to greatly decrease the input size and use a much smaller set of feature maps. Hence, this makes ENet a much cheaper and less computationally intensive model.

Other advantages of ENet include not demanding a lot of memory to be executed and being able to be executed very fast, much faster than existing models. As such this model has accomplished to be maximized on limited resources, being very compact, making it a great model for lane detection.

3.4.2 Disadvantages

One disadvantage is that ENet only uses a few layers while deeper networks make use of many layers. This could affect the network's ability to learn characteristics that are more complicated.

Another disadvantage is that although the model has used downsampling to great effect, excessive downsampling can result in the loss of information, affecting the effectiveness of the model.

The last disadvantage is that ENet requires a large dataset to train on for it to be accurate and reliable, without that large dataset, overfitting can occur.

3.4.3 Potential Improvements

The first potential improvement is to explore methods that can help the model to keep contextual information even in the mist of downsampling. This could greatly help with the effectiveness of the model.

Another potential improvement is to integrate the model with deeper networks that are compatible with it so as to resolve the problem of ENet being a very shallow network. This can help the model when it come to more complicated characteristics.

Lastly, in order to solve the problem of requiring large datasets. Combining this model with unsupervised domain adaptation (UDA) techniques could prove very effective. This is because easily generated labeled synthetic datasets can be used instead. Remedying the problem of having to go produce and label datasets themselves.

3.5 Learning Lightweight Lane Detection CNNs by Self Attention Distillation

3.5.1 Advantages

There are many advantages to SAD. SAD is a very powerful algorithm, allowing existing models to reinforce representation learning of itself, and all this can be done without the requirement of more annotated images, which can require large amounts of human effort, and does not require external supervision. SAD is also deployed only in the training stage of the model and does not require additional computational resources; hence it does not affect the inference time of the model at all.

3.5.2 Disadvantages

One disadvantage of SAD is that it has not yet been extensively tested. It has not been tested in very adverse weather conditions such as rain and snow.

3.5.3 Potential Improvements

One potential improvement is to test SAD against datasets of adverse weather conditions to see how it stands up to these situations.

3.6 Spatial As Deep: Spatial CNN for Traffic Scene Understanding

3.6.1 Advantages

The first advantage of the SCNN model is that it passes it messages in a sequential manner which is much more efficient computationally than more tradition models.

Another advantage is that the messages that are passed in this model are passed as residual. This makes SCNN much simpler to train and allows it to produce better results.

Lastly, SCNN is very versatile as it can be implemented to any stage of a deep neural network and not just its outputs.

3.6.2 Disadvantages

The first disadvantage of the model is that it increases the complexity of the network. This may lead to the model requiring more memory and computational power during training and inference.

The other disadvantage is that this model requires a large dataset to train on for it to be accurate and reliable, without that large dataset, overfitting can occur.

3.6.3 Potential Improvements

A potential improvement is to combine this model with other techniques. In order to solve the problem of requiring large datasets. Combining this model with unsupervised domain adaptation (UDA) techniques could prove very effective. This is because easily generated labeled synthetic datasets can be used instead. Remedying the problem of having to go produce and label datasets themselves.

3.7 Vision-Based Robust Lane Detection and Tracking in Challenging Conditions

3.7.1 Advantages

The first advantage of this model is that it can predict where the lane is most likely to lead and go next. This can great help with the accuracy of lane tracking.

Another advantage is that this method has been tested and was designed to be able to tackle very difficult weather conditions, like snow, and other difficult conditions like sudden changes to lighting. It was also designed to handle multiple difficult conditions in a single situation. This model does all this to a very accurate degree as well.

The last advantage is that this model does not require training. Thus, there are no problems associated with training, such as overfitting, needing very larger and wide varying datasets, is not complicated computationally, can be easily interpreted and is very fast, fast enough for real world application.

3.7.2 Disadvantages

One disadvantage of this model is that it is relatively complex which might not be suitable for more resource scarce environment, which might not make it very suitable to real world lane detection applications.

Another disadvantage is that this model has many parameters and ranges that need to be carefully tuned for different situations.

3.7.3 Potential Improvements

The first potential improvement is that techniques can be devised in order to automatically fine tune the parameters and ranges. This can improve the performance of the model even when different datasets and scenarios are tested on the model.

Another potential improvement is that machines learning can be integrated into the model so as to improve its accuracy in more complex situations and increase robustness.

3.8 Concluding Remarks

In conclusion, I have chosen to focus on ENet-SAD as it is suites the needs of lane detection in autonomous vehicles. ENet-SAD is extremely fast, appropriate for real-time applications. It is also not very computationally expensive and does not require a lot of power. Hence, the module can be very small in size, applicable for mobile use. ENet-SAD has also been tested on CULane, which has a wide variety of traffic situations, including situations with bad lighting, and has performed relatively well in all cases.

The other model that I have chosen to focus on is VPGNet as this model has been specifically designed to be effective in adverse weather conditions such as rain. Hence, by taking the learning points from this model, ENet-SAD could be adapted to perform better in adverse weather conditions.

Lastly, the reason that I have chosen these two models is because, the GitHub codes for these two models are freely accessible to the public to use.

Chapter 4:
Implementation of ENet-SAD

4.1 Overview

In this chapter, I will outline the how I tried to implement the code that we created by the authors of the of the paper, Learning Lightweight Lane Detection CNNs by Self Attention Distillation. The code can be found on GitHub using the following link: https://github.com/InhwanBae/ENet-SAD_Pytorch?tab=readme-ov-file. I did not manage to get any results as I implemented the code on Google Colab, which is very different from traditional environments, and much of the code used deprecated functions.

4.2 Dataset

The dataset that was used is the CULane dataset. It was used as it is a dataset with a large variety of situations such as images with dazzling lights, nighttime, unclear lane lines and crossroads for example. Hence, the model can be tested against many real-world examples. CULane is also a dataset that is very easily accessible and can be downloaded easily.

It is a very big dataset that contains many scenarios that is difficult to implement lane detection and was collated for the purpose of academic research on lane identification. The images and videos were captured by cameras mounted on six separate cars that drove around the city of Beijing. The dataset contains a train set, a validation set and a test set. The test set is also further separated into eight categories: normal, crowded, night, no line, shadow, arrow, dazzle light, curve, and crossroad.

The CULane dataset was downloaded from the official website, https://xingangpan.github.io/projects/CULane.html. The website contained a link to a Google Drive

folder with all the images. They were downloaded and unzipped before being uploaded to a Google Drive account that was mounted to Google Colab.

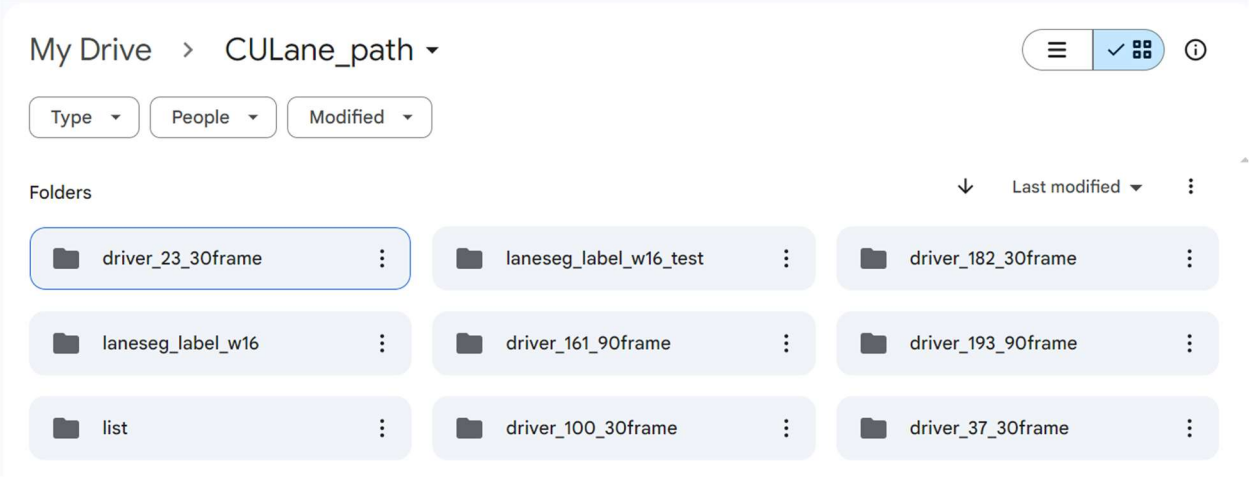


Figure 4-1: The CULane dataset uploaded on my Google Drive account

4.3 Environment

The environment I used was Google Colab. In Google Colab, I chose to use the NVIDIA T4 GPU for train, validation, and testing. The NVIDIA T4 GPU allows high-performance computing, which is suitable for machine learning [35].

4.4 Implementation

The first step was to clone the GitHub code into the environment.

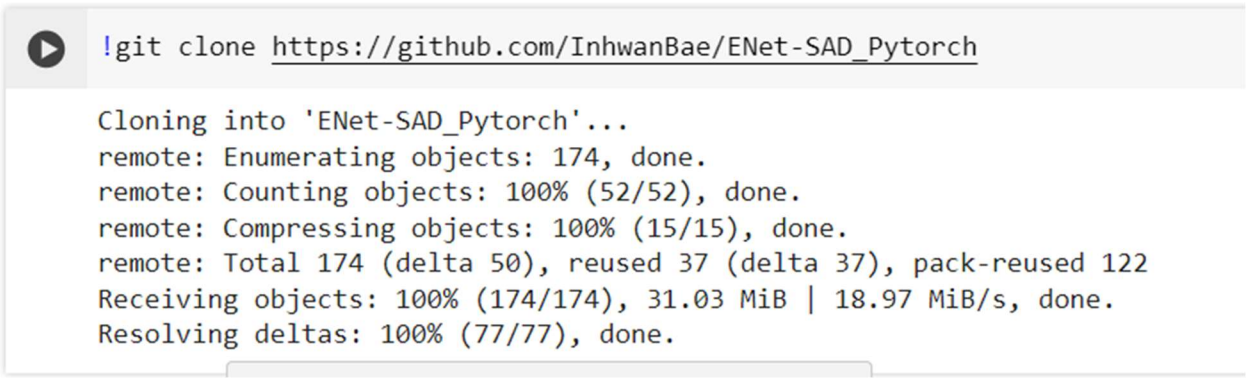


Figure 4-2: The code cell to clone the GitHub code

Before the model could be trained, there were some deprecated functions being used in the code that had to be edited.

a)

```
def image_summary(self, tag, images, step):
    """Log a list of images."""

    img_summaries = []
    for i, img in enumerate(images):
        # Write the image to a string
        try:
            s = StringIO()
        except:
            s = BytesIO()
        # scipy.misc.toimage(img).save(s, format="png")
        Image.fromarray(img).save(s, format='png')


        # Create an Image object
        img_sum = tf.Summary.Image(encoded_image_string=s.getvalue(),
                                   height=img.shape[0],
                                   width=img.shape[1])
        # Create a Summary value
        img_summaries.append(tf.Summary.Value(tag='%s/%d' % (tag, i), image=img_sum))

    # Create and write Summary
    summary = tf.Summary(value=img_summaries)
    self.writer.add_summary(summary, step)
```

b)

```
def image_summary(self, tag, images, step):
    """Log a list of images."""
    with self.writer.as_default():
        for i, img in enumerate(images):
            image = tf.image.encode_png(img)
            tf.summary.image(f"{tag}/{i}", [tf.io.decode_png(image)], step=step)

```

Figure 4-3: a) The function before editing which contains many deprecated functions b) The edited function, without any deprecated functions, that works the same as (a)

The model is then trained and validated. I trained the model until 8 epochs.

```
[ ] !python /content/ENet-SAD_Pytorch/train.py --exp_dir /content/ENet-SAD_Pytorch/experiments/exp1

2024-04-08 22:03:06.708815: E external/local_xla/xla/stream_executor/cuda/cuda_dnn.cc:9261] Unable to register cuDNN factory: Attempting to register factory for plugin c
2024-04-08 22:03:06.708868: E external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:607] Unable to register cuFFT factory: Attempting to register factory for plugin cu
2024-04-08 22:03:06.710518: E external/local_xla/xla/stream_executor/cuda/cuda_blas.cc:1515] Unable to register cuBLAS factory: Attempting to register factory for plugin
2024-04-08 22:03:07.829741: W tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT Warning: Could not find TensorRT
2024-04-08 22:03:09.015126: W tensorflow/core/common_runtime/gpu/gpu_bfc_allocator.cc:47] Overriding orig_value setting because the TF_FORCE_GPU_ALLOW_GROWTH environment
Train Epoch: 0
batch loss: 0.191: 100% 7407/7407 [50:24<00:00,  2.45it/s]
model is saved: /content/ENet-SAD_Pytorch/experiments/exp1/exp1.pth
-----------------------


Validation For Experiment:  /content/ENet-SAD_Pytorch/experiments/exp1
22:53:33
Val Epoch: 0
batch loss: 0.289: 100% 1210/1210 [04:13<00:00,  4.77it/s]
-----------------------

Train Epoch: 1
batch loss: 0.186: 100% 7407/7407 [50:10<00:00,  2.46it/s]
model is saved: /content/ENet-SAD_Pytorch/experiments/exp1/exp1.pth
-----------------------
```

Figure 4-4: The training and validation cell in Google Colab

After the training and validation was done, the next step was to the test the model.

Figure 4-5: Examples of lane detection using the model that was trained

The model was successfully tested and I managed to obatain the coordinate outputs of the detected lanes. As shown in Figure 3-5, a script was written to display the image and print the detected lanes over the image. From the images in Figure 3-5, it can be seen that the lane detection model is not as accurate as the one in the ENet-SAD paper, this could be due to the training only done to the eighth epoch, while training was done to the eightieth epoch in th paper. This was due to the a lack of time as training is an extremely time consuming process.

4.5 Conclusion

In conclusion, I successfully trained, validated and tested the model and was able to display some examples of the results that I have obatained. I ran into many problems trying to get the code to work as there were many deprecated functions and I had to replace many paths in the code as the environment is not a traditional one, like the one the code was written on.

Chapter 5:
Conclusion and Recommendation for Future Work

5.1 Conclusion

In conclusion, I have a good understanding of the landscape of lane detection techniques and models. I now have a better understanding of most of the pros and cons of the various methods and models that there are. I now know the strengths that each of the methods have and what work must be done in order to allow these models to be safely applied to real world situations.

I have also successfully implemented the ENet-SAD network, allowing me to know the how these models are to be implemented and understand on a deeper level how they work. Although the results were not as accurate and effective as the model was in the paper, that can be attributed to the number of rounds of training that was conducted. I only trained to the eighth epoch while the model was trained to the eightieth epoch in the paper.

Due to time constraints, I was also not able to implement the VPGNet model and test the Enet-SAD model on a dataset with adverse weather conditions, which I wanted to.

5.2 Recommendations for Future Work

5.2.1 Applying SAD to a model specializing in adverse weather conditions

Since SAD is a module that can be easily implemented with other feedforward CNNs. One area for further work is to apply SAD to a CNN that was built to specifically for adverse weather conditions and test the entire model on a dataset with a wide variety of adverse weather conditions situations. One such CNN that could be used for this is VPGNet.

5.2.2 Solving the problem of needing large datasets

In order to solve the problem of many models needing to have large datasets for training in order for it to be accurate and reliable, trying to implement unsupervised domain adaptation (UDA) techniques into these models could prove very effective.

5.2.3 Combining models and methods together to improve accuracy

One area that might be worth looking into is to combine compatible models and methods together to improve the results. For example, ENet could be combined with a deeper network to get better results for more complicated situations.

5.2.4 Automating the fine-tuning process of parameters

Automatically fine-tuning parameters and ranges can improve the performance of the model even when different datasets and scenarios are tested on the model. This is an area that can help models adjust to adverse weather conditions and increase their accuracy.

5.2.5 Improving datasets

Many models and techniques for lane detection utilize CNN which require large datasets for training and the quality and accuracy largely depends on the size and quality of these datasets. Hence, this is an area where further work might bear much fruit and improve the models greatly.

Bibliography

[1] S. Leev, J. Kim, J. S. Yoonv, S. Shinv, O. Bailo, N. Kim, T. Lee, H. S. Hong, S. Han, and I. S. Kweo, "VPGNet: Vanishing point guided network for lane and road marking detection and recognition," IEEE International Conference on Computer Vision, pages 1965-1973, 2017.

[2] M. Chen, H. Xue, and D. Cai, "Domain adaptation for semantic segmentation with maximum squares loss," IEEE International Conference on Computer Vision, pages 2090–2099, 2019.

[3] "Autonomous vehicles," Smart Nation Singapore, https://www.smartnation.gov.sg/initiatives/transport/autonomous-vehicles/ (accessed Nov. 17, 2023).

[4] M. Placek, "Projected number of autonomous vehicles worldwide 2030," Statista, https://www.statista.com/statistics/1230664/projected-number-autonomous-cars-worldwide/#:~:text=According%20to%20Next%20Move%20Strategy,some%20127%2C000%20units%20in%202030. (accessed Nov. 17, 2023).

[5] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, "The cityscapes dataset for semantic urban scene understanding," In CVPR, 2016.

[6] S. R. Richter, V. Vineet, S. Roth, and V. Koltun, "Playing for data: Ground truth from computer games," In ECCV, 2016.

[7] G. Ros, L. Sellart, J. Materzynska, D. Vazquez, and A. M. Lopez, "The SYNTHIA dataset: A large collection of synthetic images for semantic segmentation of urban scenes," In CVPR, 2016.

[8] K. Saenko and B. Kulis, "Adapting visual category models to new domains," In ECCV, 2010

[9] Y. Chen, W. Chen, Y. Chen, B. Tsai, Y. F. Wang, and M. Sun, "No more discrimination: Cross city adaptation of road scene segmenters," In ICCV, 2017.

[10] Y. Grandvalet and Y. Bengio, "Semi-supervised learning by entropy minimization," In NIPS, 2004.

[11] D. Lee, "Pseudo-label : The simple and efficient semi-supervised learning method for deep neural networks," ICML 2013 Workshop : Challenges in Representation Learning (WREPL).

[12] Y. Zou, Z. Yu, B. V. K. V. Kumar, and J. Wang, "Unsupervised domain adaptation for semantic segmentation via class-balanced self-training" In ECCV, 2018.

[13] T. Lin, P. Goyal, R. B. Girshick, K. He, and P. Dollar, "Focal loss for dense object detection," In ICCV, 2017.

[14] B. Huval, T. Wang, S. Tandon, J. Kiske, W. Song, J. Pazhayampallil, M. Andriluka, P. Rajpurkar, T. Migimatsu, R. Cheng-Yue, "An empirical evaluation of deep learning on highway driving," arXiv preprint arXiv:1504.01716, 2015.

[15] Z. Zhu, D. Liang, S. Zhang, X. Huang, B. Li, and S. Hu, "Traffic-sign detection and classification in the wild," In CVPR, 2016.

[16] M. Land, J. Horwood, "Which parts of the road guide steering?" Nature, 377(6547):339–340, 1995.

[17] M. F. Land and D. N. Lee, “Where do we look when we steer,” Nature, 369(6483):742–744, 1994.

[18] D. D. Salvucci and R. Gray, “A two-point visual control model of steering,” Perception, 33(10):1233–1248, 2004.

[19] A. Borji, “Vanishing point detection with convolutional neural networks,” *arXiv preprint arXiv:1609.00967*, 2016.

[20] C. Li, B. Zhang, J. Shi, and G. Cheng, “Multi-level Domain Adaptation for Lane Detection,” IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), pages 4379-4388, 2022.

[21] X. Pan, J. Shi, P. Luo, X. Wang, and X. Tang, “Spatial As Deep: Spatial CNN for Traffic Scene Understanding.” Available: https://arxiv.org/pdf/1712.06080.pdf

[22] A. Paszke, A. Chaurasia, S. Kim, and E. Culurciello, “ENet: A Deep Neural Network Architecture for Real-Time Semantic Segmentation.” Available: https://arxiv.org/pdf/1606.02147.pdf

[23] Y. Hou, Z. Ma, C. Liu, and C. Loy, “Learning Lightweight Lane Detection CNNs by Self Attention Distillation.” Accessed: Apr. 10, 2024. [Online]. Available: https://arxiv.org/pdf/1908.00821.pdf

[24] “CULane Dataset,” xingangpan.github.io. https://xingangpan.github.io/projects/CULane.html

[25] K. Simonyan and A. Zisserman, “Published as a conference paper at ICLR 2015 VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION,” 2015. Available: https://arxiv.org/pdf/1409.1556.pdf

[26] J. Long, E. Shelhamer, and T. Darrell, “Fully Convolutional Networks for Semantic Segmentation,” 2015. Available: https://arxiv.org/pdf/1411.4038.pdf

[27] H. Noh, S. Hong, and B. Han, “Learning Deconvolution Network for Semantic Segmentation.” Available: https://arxiv.org/pdf/1505.04366.pdf

[28] S. Zheng et al., “Conditional Random Fields as Recurrent Neural Networks.” Accessed: Apr. 11, 2024. [Online]. Available: https://arxiv.org/pdf/1502.03240.pdf

[29] D. Eigen and R. Fergus, “Predicting Depth, Surface Normals and Semantic Labels with a Common Multi-Scale Convolutional Architecture.” Available: https://arxiv.org/pdf/1411.4734.pdf

[30] S. Hong, H. Noh, and B. Han, “Decoupled Deep Neural Network for Semi-supervised Semantic Segmentation.” Accessed: Apr. 11, 2024. [Online]. Available: https://arxiv.org/pdf/1506.04924.pdf

[31] K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition,” Dec. 2015. Available: https://arxiv.org/pdf/1512.03385.pdf

[32] S. Ioffe, “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift,” 2015. Available: https://arxiv.org/pdf/1502.03167.pdf

[33] K. He, X. Zhang, S. Ren, and J. Sun, “Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification,” 2015. Available: https://arxiv.org/pdf/1502.01852.pdf

[34] S. Zagoruyko and N. Komodakis, “PAYING MORE ATTENTION TO ATTENTION: IMPROVING THE PERFORMANCE OF CONVOLUTIONAL NEURAL NETWORKS VIA ATTENTION TRANSFER.” Available: https://arxiv.org/pdf/1612.03928.pdf

[35] “NVIDIA T4 Tensor Core GPUs for Accelerating Inference,” NVIDIA. https://www.nvidia.com/en-sg/data-center/tesla-t4/ (accessed Apr. 14, 2024).

[36] “Lane Detection: The 3 types of Deep Learning (non-OpenCV) algorithms,” Welcome to The Library!, Oct. 09, 2023. https://www.thinkautonomous.ai/blog/lane-detection/#:~:text=For%20example%2C%20SCNN%20(Spatial%20Convolutional (accessed Apr. 20, 2024).

[37] S. Sultana, B. Ahmed, M. Paul, M. R. Islam, and S. Ahmad, “Vision-Based Robust Lane Detection and Tracking in Challenging Conditions,” 2017. Available: https://arxiv.org/ftp/arxiv/papers/2210/2210.10233.pdf

Appendix A: Configuration for Training

```
"device": "cuda:0",
"MAX_EPOCHES": 8,
"model": "enet_sad",

"dataset": {
    "dataset_name": "CULane",
    "batch_size": 12,
    "resize_shape": [800, 288]
},

"optim": {
    "lr": 1e-2,
    "momentum": 0.9,
    "weight_decay": 2e-5,
    "nesterov": true
},

"lr_scheduler": {
    "warmup": 50,
    "max_iter": 60000
},

"sad_start_iter": 40000
```

Appendix B: Code for Training and Validation

train.py

```python
import argparse
import json
import os
import shutil
import time

import torch.optim as optim
from torch.utils.data import DataLoader
from tqdm import tqdm

from config import *
import dataset
from model import SCNN
from model_ENET_SAD import ENet_SAD
from utils.tensorboard import TensorBoard
from utils.transforms import *
from utils.lr_scheduler import PolyLR


from multiprocessing import Process, JoinableQueue
from threading import Lock
import pickle
#from torch.multiprocessing import Process, SimpleQueue
def parse_args():
    parser = argparse.ArgumentParser()
    parser.add_argument("--exp_dir", type=str, default="./experiments/exp0")
    parser.add_argument("--resume", "-r", action="store_true")
    args = parser.parse_args()
    return args
args = parse_args()

# ------------ config ------------
exp_dir = args.exp_dir
while exp_dir[-1]=='/':
    exp_dir = exp_dir[:-1]
exp_name = exp_dir.split('/')[-1]

with open(os.path.join(exp_dir, "cfg.json")) as f:
    exp_cfg = json.load(f)
resize_shape = tuple(exp_cfg['dataset']['resize_shape'])

device = torch.device(exp_cfg['device'])
tensorboard = TensorBoard(exp_dir)

```

```
# ------------ train data ------------
# # CULane mean, std
mean=(0.3598, 0.3653, 0.3662)
std=(0.2573, 0.2663, 0.2756)
# Imagenet mean, std
# mean=(0.485, 0.456, 0.406)
# std=(0.229, 0.224, 0.225)
transform_train = Compose(Resize(resize_shape), Rotation(2), ToTensor(),
                          Normalize(mean=mean, std=std))
dataset_name = exp_cfg['dataset'].pop('dataset_name')
Dataset_Type = getattr(dataset, dataset_name)
train_dataset = Dataset_Type(Dataset_Path[dataset_name], "train", transform_train)
train_loader = DataLoader(train_dataset, batch_size=exp_cfg['dataset']['batch_size'],
 shuffle=True, collate_fn=train_dataset.collate, num_workers=6)

# ------------ val data ------------
transform_val_img = Resize(resize_shape)
transform_val_x = Compose(ToTensor(), Normalize(mean=mean, std=std))
transform_val = Compose(transform_val_img, transform_val_x)
val_dataset = Dataset_Type(Dataset_Path[dataset_name], "val", transform_val)
val_loader = DataLoader(val_dataset, batch_size=8, collate_fn=val_dataset.collate, num_workers=4)
```

```
# ------------ preparation ------------
if exp_cfg['model'] == "scnn":
    net = SCNN(resize_shape, pretrained=True)
elif exp_cfg['model'] == "enet_sad":
    net = ENet_SAD(resize_shape, sad=True, dataset=dataset_name)
else:
    raise Exception("Model not match. 'model' in 'cfg.json' should be 'scnn' or 'enet_sad'.")

net = net.to(device)
net = torch.nn.DataParallel(net)

optimizer = optim.SGD(net.parameters(), **exp_cfg['optim'])
lr_scheduler = PolyLR(optimizer, 0.9, **exp_cfg['lr_scheduler'])
best_val_loss = 1e6


def train(epoch):
    print("Train Epoch: {}".format(epoch))
    net.train()
    train_loss = 0
    train_loss_seg = 0
    train_loss_exist = 0
```

```
    progressbar = tqdm(range(len(train_loader)))

    for batch_idx, sample in enumerate(train_loader):
        img = sample['img'].to(device)
        segLabel = sample['segLabel'].to(device)
        exist = sample['exist'].to(device)

        optimizer.zero_grad()

        if exp_cfg['model'] == "scnn":
            seg_pred, exist_pred, loss_seg, loss_exist, loss = net(img, segLabel, exist)
        elif exp_cfg['model'] == "enet_sad":
            if (epoch * len(train_loader) + batch_idx) < exp_cfg['sad_start_iter']:
                seg_pred, exist_pred, loss_seg, loss_exist, loss = net(img, segLabel, exist, False)
            else:
                print("sad activated")
                seg_pred, exist_pred, loss_seg, loss_exist, loss = net(img, segLabel, exist, True)

        if isinstance(net, torch.nn.DataParallel):
            loss_seg = loss_seg.sum()
            loss_exist = loss_exist.sum()
            loss = loss.sum()
        loss.backward()
        optimizer.step()
        lr_scheduler.step()

        iter_idx = epoch * len(train_loader) + batch_idx
        train_loss = loss.item()
        train_loss_seg = loss_seg.item()
        train_loss_exist = loss_exist.item()
        progressbar.set_description("batch loss: {:.3f}".format(loss.item()))
        progressbar.update(1)

        lr = optimizer.param_groups[0]['lr']
        tensorboard.scalar_summary(exp_name + "/train_loss", train_loss, iter_idx)
        tensorboard.scalar_summary(exp_name + "/train_loss_seg", train_loss_seg, iter_idx)
        tensorboard.scalar_summary(exp_name + "/train_loss_exist", train_loss_exist, iter_idx)
        tensorboard.scalar_summary(exp_name + "/learning_rate", lr, iter_idx)

    progressbar.close()
    tensorboard.writer.flush()

    if epoch % 1 == 0:
        save_dict = {
            "epoch": epoch,
            "net": net.module.state_dict() if isinstance(net, torch.nn.DataParallel) else net.state_dict(),
```

```
            "optim": optimizer.state_dict(),
            "lr_scheduler": lr_scheduler.state_dict(),
            "best_val_loss": best_val_loss
        }
        save_name = os.path.join(exp_dir, exp_name + '.pth')
        torch.save(save_dict, save_name)
        print("model is saved: {}".format(save_name))

    print("------------------------\n")


def val(epoch):
    global best_val_loss

    print("Val Epoch: {}".format(epoch))

    net.eval()
    val_loss = 0
    val_loss_seg = 0
    val_loss_exist = 0
    progressbar = tqdm(range(len(val_loader)))

    with torch.no_grad():
        for batch_idx, sample in enumerate(val_loader):
            img = sample['img'].to(device)
            segLabel = sample['segLabel'].to(device)
            exist = sample['exist'].to(device)

            seg_pred, exist_pred, loss_seg, loss_exist, loss = net(img, segLabel, exist)
            if isinstance(net, torch.nn.DataParallel):
                loss_seg = loss_seg.sum()
                loss_exist = loss_exist.sum()
                loss = loss.sum()

            # visualize validation every 5 frame, 50 frames in all
            gap_num = 5
            if batch_idx%gap_num == 0 and batch_idx < 50 * gap_num:
                origin_imgs = []
                seg_pred = seg_pred.detach().cpu().numpy()
                exist_pred = exist_pred.detach().cpu().numpy()

                for b in range(len(img)):
                    img_name = sample['img_name'][b]
                    img = cv2.imread(img_name)
                    img = transform_val_img({'img': img})['img']
```

```
                    lane_img = np.zeros_like(img)
                    color = np.array([[255, 125, 0], [0, 255, 0], [0, 0, 255], [0, 255, 255]], dtype='uint8')

                    coord_mask = np.argmax(seg_pred[b], axis=0)
                    for i in range(0, 4):
                        if exist_pred[b, i] > 0.5:
                            lane_img[coord_mask==(i+1)] = color[i]
                    img = cv2.addWeighted(src1=lane_img, alpha=0.8, src2=img, beta=1., gamma=0.)
                    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
                    lane_img = cv2.cvtColor(lane_img, cv2.COLOR_BGR2RGB)
                    cv2.putText(lane_img, "{}".format([1 if exist_pred[b, i]>0.5 else 0 for i in range(4)]),
                    (20, 20), cv2.FONT_HERSHEY_SIMPLEX, 1.1, (255, 255, 255), 2)
                    origin_imgs.append(img)
                    origin_imgs.append(lane_img)
                tensorboard.image_summary("img_{}".format(batch_idx), origin_imgs, epoch)

            val_loss += loss.item()
            val_loss_seg += loss_seg.item()
            val_loss_exist += loss_exist.item()

            progressbar.set_description("batch loss: {:.3f}".format(loss.item()))
            progressbar.update(1)
    progressbar.close()
    iter_idx = (epoch + 1) * len(train_loader)  # keep align with training process iter_idx
    tensorboard.scalar_summary("val_loss", val_loss, iter_idx)
    tensorboard.scalar_summary("val_loss_seg", val_loss_seg, iter_idx)
    tensorboard.scalar_summary("val_loss_exist", val_loss_exist, iter_idx)
    tensorboard.writer.flush()

    print("------------------------\n")
    if val_loss < best_val_loss:
        best_val_loss = val_loss
        save_name = os.path.join(exp_dir, exp_name + '.pth')
        copy_name = os.path.join(exp_dir, exp_name + '_best.pth')
        shutil.copyfile(save_name, copy_name)


def main():
    global best_val_loss
    if args.resume:
        save_dict = torch.load(os.path.join(exp_dir, exp_name + '.pth'))
        if isinstance(net, torch.nn.DataParallel):
            net.module.load_state_dict(save_dict['net'])
        else:
            net.load_state_dict(save_dict['net'])
```

```
            optimizer.load_state_dict(save_dict['optim'])
            lr_scheduler.load_state_dict(save_dict['lr_scheduler'])
            start_epoch = save_dict['epoch'] + 1
            best_val_loss = save_dict.get("best_val_loss", 1e6)
        else:
            start_epoch = 0

        exp_cfg['MAX_EPOCHES'] = int(np.ceil(exp_cfg['lr_scheduler']['max_iter'] / len(train_loader)))
        for epoch in range(start_epoch, exp_cfg['MAX_EPOCHES']):
            train(epoch)
            if epoch % 1 == 0:
                print("\nValidation For Experiment: ", exp_dir)
                print(time.strftime('%H:%M:%S', time.localtime()))
                val(epoch)


if __name__ == "__main__":
    main()
```

Appendix C: Code for Testing

test_CULane.py

```python
import argparse
import json
import os

import torch.nn.functional as F
from torch.utils.data import DataLoader
from tqdm import tqdm

import dataset
from config import *
from model import SCNN
from model_ENET_SAD import ENet_SAD
from utils.prob2lines import getLane
from utils.transforms import *


def parse_args():
    parser = argparse.ArgumentParser()
    parser.add_argument("--exp_dir", type=str, default="./experiments/exp10")
    args = parser.parse_args()
    return args
# ------------ config ------------
args = parse_args()
exp_dir = args.exp_dir
exp_name = exp_dir.split('/')[-1]

with open(os.path.join(exp_dir, "cfg.json")) as f:
    exp_cfg = json.load(f)
resize_shape = tuple(exp_cfg['dataset']['resize_shape'])
device = torch.device('cuda')


def split_path(path):
    """split path tree into list"""
    folders = []
    while True:
        path, folder = os.path.split(path)
        if folder != "":
            folders.insert(0, folder)
        else:
            if path != "":
                folders.insert(0, path)
            break
    return folders
```

```
# ------------ data and model ------------
# # CULane mean, std
mean=(0.3598, 0.3653, 0.3662)
std=(0.2573, 0.2663, 0.2756)
# Imagenet mean, std
# mean = (0.485, 0.456, 0.406)
# std = (0.229, 0.224, 0.225)
dataset_name = exp_cfg['dataset'].pop('dataset_name')
Dataset_Type = getattr(dataset, dataset_name)
transform = Compose(Resize(resize_shape), ToTensor(),
                    Normalize(mean=mean, std=std))
test_dataset = Dataset_Type(Dataset_Path[dataset_name], "test", transform)
test_loader = DataLoader(test_dataset, batch_size=64, collate_fn=test_dataset.collate, num_workers=0)

if exp_cfg['model'] == "scnn":
    net = SCNN(input_size=resize_shape, pretrained=False)
elif exp_cfg['model'] == "enet_sad":
    net = ENet_SAD(resize_shape, sad=False, dataset=dataset_name)
else:
    raise Exception("Model not match. 'model' in 'cfg.json' should be 'scnn' or 'enet_sad'.")

save_name = os.path.join(exp_dir, exp_dir.split('/')[-1] + '_best.pth')
save_dict = torch.load(save_name, map_location='cpu')
print("\nloading", save_name, "...... From Epoch: ", save_dict['epoch'])
net.load_state_dict(save_dict['net'])
net = torch.nn.DataParallel(net.to(device))
net.eval()

# ------------ test ------------
out_path = os.path.join(exp_dir, "coord_output")
evaluation_path = os.path.join(exp_dir, "evaluate")
if not os.path.exists(out_path):
    os.mkdir(out_path)
if not os.path.exists(evaluation_path):
    os.mkdir(evaluation_path)

progressbar = tqdm(range(len(test_loader)))
with torch.no_grad():
    for batch_idx, sample in enumerate(test_loader):
        img = sample['img'].to(device)
        img_name = sample['img_name']

        seg_pred, exist_pred = net(img)[:2]
        seg_pred = F.softmax(seg_pred, dim=1)
        seg_pred = seg_pred.detach().cpu().numpy()
        exist_pred = exist_pred.detach().cpu().numpy()
```

```
        for b in range(len(seg_pred)):
            seg = seg_pred[b]
            exist = [1 if exist_pred[b, i] > 0.5 else 0 for i in range(4)]
            lane_coords = getLane.prob2lines_CULane(seg, exist, resize_shape=(590, 1640), y_px_gap=20, pts=18)

            path_tree = split_path(img_name[b])
            save_dir, save_name = path_tree[-3:-1], path_tree[-1]
            save_dir = os.path.join(out_path, *save_dir)
            save_name = save_name[:-3] + "lines.txt"
            save_name = os.path.join(save_dir, save_name)
            if not os.path.exists(save_dir):
                os.makedirs(save_dir)

            with open(save_name, "w") as f:
                for l in lane_coords:
                    for (x, y) in l:
                        print("{} {}".format(x, y), end=" ", file=f)
                    print(file=f)

        progressbar.update(1)
progressbar.close()
# ---- evaluate ----
os.system("sh utils/lane_evaluation/CULane/Run.sh " + exp_name)

```

Appendix D: Code for Drawing Detected Lanes on Images

```
from google.colab.patches import cv2_imshow
import cv2
import numpy as np

# Function to draw lanes on the image
def draw_lanes(image, lanes):
    # Convert coordinates to numpy array
    lanes = np.array(lanes, dtype=np.int32)
    # Reshape the lanes array into pairs of points
    lanes = lanes.reshape((-1, 1, 2))
    # Draw the lanes on the image
    cv2.polylines(image, [lanes], isClosed=False, color=(0, 255, 0), thickness=2)

# Path to the image
image_path = "/content/drive/MyDrive/CULane_path/driver_193_90frame/06060748_0790.MP4/00900.jpg"

# Load the image
image = cv2.imread(image_path)
```

```
# Detected lanes coordinates
lanes_coordinates = [
    [96, 549], [174, 529], [213, 509], [270, 489], [305, 469],
    [362, 449], [401, 429], [440, 409], [496, 389], [541, 369],
    [584, 349], [625, 329], [666, 309], [914, 489], [895, 469],
    [871, 449], [856, 429], [838, 409], [822, 389], [809, 369],
    [793, 349], [781, 329], [764, 309], [1527, 469], [1459, 449],
    [1385, 429], [1305, 409], [1217, 389], [1123, 369], [1033, 349],
    [959, 329], [885, 309]
]

# Draw lanes on the image
draw_lanes(image, lanes_coordinates)

# Display the image
cv2_imshow(image)
```