Lane detection algorithm for autonomous vehicle using machine learning

Wu, Guan Jie

2023

Wu, G. J. (2023). Lane detection algorithm for autonomous vehicle using machine learning. Final Year Project (FYP), Nanyang Technological University, Singapore.
https://hdl.handle.net/10356/167291

https://hdl.handle.net/10356/167291

Acad Year
22/23

PROJECT NO.
C072

LANE DETECTION ALGORITHM FOR AUTONOMOUS VEHICLE USING MACHINE LEARNING

LANE DETECTION ALGORITHM FOR AUTONOMOUS VEHICLE USING MACHINE LEARNING

Wu Guan Jie

SCHOOL OF MECHANICAL AND AEROSPACE ENGINEERING
NANYANG TECHNOLOGICAL UNIVERSITY

Year 2022/2023

LANE DETECTION ALGORITHM FOR AUTONOMOUS VEHICLE USING MACHINE LEARNING

SUBMITTED

BY

WU GUAN JIE

SCHOOL OF MECHANICAL AND AEROSPACE ENGINEERING

A final year project report
presented to
Nanyang Technological University
in partial fulfilment of the
requirements for the
Degree of Bachelor of Engineering (Mechanical Engineering)
Nanyang Technological University

Year 2022/2023

Table of Contents

Abstract

Lane detection has been one of research areas in computer vision for decades. Traditional computer vision algorithms such as Canny edge detector and Hough Transform are commonly used in lane detection model. However, these models only work well under well-conditioned roads with clear lane marks and no occlusion. Hence, there is a need to shift from traditional computer vision algorithms to deep learning method in feature extraction. Convolution neural network (CNN) has been the de facto feature extraction module in computer vision. Spatial CNN (SCNN), the winner of 2017 TuSimple lane detection challenge, propose a special message passing mechanism for long and thin structure. Nonetheless, it can only run at about 20 frames per second (FPS) on NVIDIA GeForce RTX 2080 Ti GPU due to the heavy computational load of the special message passing mechanism.

This project explores the possibility of using transformer architecture as the feature extraction module in a lane detection model. This is because the self-attention layers with sufficient heads can express any convolutional layer. The performance of lane detection model based on SegFormer was compared with SCNN in terms of accuracy, F1-measure and FPS. The PyTorch model was converted to TensorRT engine through intermediary ONNX which further boost the FPS of the model during inference.

From the experiments, accuracy of SegFormer is higher than SCNN by 0.35% on TuSimple dataset while being 4 times faster than SCNN in inference speed. For CULane dataset, the F1-measure of SegFormer is lower than SCNN by 1.45% but the inference speed of SegFormer is 2 times faster than SCNN in this case. This justifies the trade-off between accuracy and computational load of the model and proven the strong feature representation power of transformer without any incorporation of lane prior.

Acknowledgement

I would like to express my sincere gratitude to Asst Prof Lyu Chen for giving me the opportunity to work on this project. This project had deepened my knowledge in deep learning and semantic segmentation. Besides, I was allocated powerful computing resources, i.e. NVIDIA GeForce RTX 2080 Ti GPU which allowed me to get my hands dirty by experimenting different deep learning models in lane detection. Furthermore, I appreciated the constructive and detailed advice provided by Prof Lyu in my project and presentations.

Also, I would like to express my appreciation to Dr Zhongxu for meeting up with me biweekly to discuss on the problems in this project. His encouragement had pushed me to keep moving forward when I failed to improve the model performance by tuning hyperparameters, changing loss function etc. Besides, I would like to thank Dr Zhongxu for being always available on Microsoft Teams when I faced some challenges.

Finally, I appreciate the opportunity provided by School of Mechanical and Aerospace Engineering for the opportunity to work on this project. Indeed, I learnt and reflected a lot from this meaningful project.

List of Figures

List of Tables

Chapter 1: Introduction

Driving is a mundane task which demands the full attention of driver and strict adherence to rule to guarantee the safety of the journey. According to National Highway Traffic Safety Administration (NHTSA), 32% (13,347) of the total passenger vehicle involved in fatal crashes in 2020 because of speeding and alcohol-impaired driving while 8.1% (3,142) of total fatalities in 2020 were due to distracted driving.[1] Also, according to the survey by American Automobile Association, American drivers spent a total of 91 billion hours driving in 2021.[2] These alarming statistics have highlighted the role of autonomous driving in enhancing road safety and enabling humans to focus on creative tasks. Hence, autonomous driving remains as one of the most active research areas with total global investment exceeding $106 billion since 2010.[3]

A typical autonomous driving system consists of perception, decision making, planning and control. Lane detection is one of the key components in perception and vision-based sensor such as camera is considered as the most promising modality for this task.[4] This is because lane marking is one of the visual signals designed for human vision in our road network. It carries crucial information which will be used in the latter stages. For example, in Singapore, white zigzag line will be labelled on the road if there is zebra crossing ahead.[5] This gives a clue to driver to slow down the vehicle and be alert if there is any road users crossing road.

There are 2 categories in vision-based lane detection. The first category is about using traditional computer vision techniques to extract handcrafted features of the lanes in an image such as line width, edge and etc.[6] Typically, Hough transform is applied after the edge detector for curve fitting. However, lane detection algorithm based on these traditional techniques often suffers under challenging scenarios such as weak lane marks, shaded roads, various lightning conditions etc. The second category is about using machine learning model, especially deep learning model such as convolutional neural network (CNN)-based and transformer-based model in lane detection.[7], [8] CNN has been the de-facto neural network in computer vision since AlexNet was introduced and it serves as the backbone of the whole model in generating feature map of the raw image.[9] In 2017, transformer architecture has been introduced in Natural Language Processing and its application to visual data has gained traction in recent years. These transformer-based lane detection models were commonly tested on dataset such as TuSimple dataset which focus on real highway scenarios. Nonetheless, there is a lack of study of the performance of transformer-based lane detection model under different challenging road conditions.

The aim of this project was to develop a robust and lightweight lane detection model based on transformer architecture which can perform well under various challenging road conditions. The model

was segmentation-based which generates pixel wise probability map of lanes in an image. Thorough analysis and comparison had been done on the performance of the model based on traditional computer vision techniques, CNN and transformer under various challenging road conditions. Insight from the analysis will serve as catalyst in the adoption of transformer as the backbone in computer vision model and shifting the research focus on more diverse and challenging scenarios.

Chapter 2: Literature Review

Lane detection remains to be an imperative part in advanced driver assistance systems (ADAS) or autonomous driving system and it has been solved with different methods across the last two decades. In this section, traditional computer vision techniques and deep learning models in lane detection will be reviewed. Convolutional neural network and transformer architecture will be discussed in the section of deep learning.

2.1 Traditional Computer Vision

Traditional computer vision technique requires least data and computing resources as compared to deep learning models. There are three main procedures in traditional image-based lane detection algorithm, which are image pre-processing, feature extraction and model fitting and lane tracking.

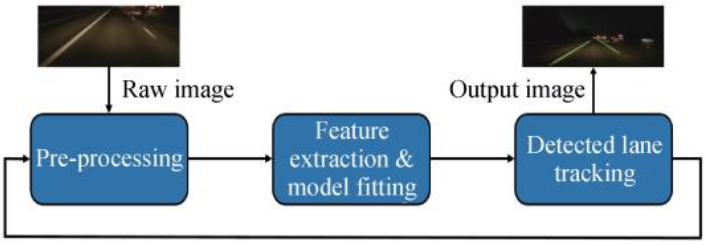

Figure 1: Architecture of a typical lane detection system. The feedback loop represents the lane tracking module and could be implemented by Kalman filter.[10]

2.1.1 Image pre-processing

Different image pre-processing techniques are used in different studies with specific purposes. For example, image filtering is applied by using mean, median or Gaussian filters to remove noise in the image. Images in red, green and blue (RGB) format is converted into grayscale or other colour format to increase the contrast of lane markers with the surroundings. Also, region of interest (ROI) can be used to reduce computational load and increase the accuracy. This is because the pixels needed to be processed are reduced and the objects outside of ROI with similar features as the lane markers are also eliminated. In [11], front-facing image is transformed into bird's-eye-view by using inverse perspective mapping (IPM) algorithm. This simplified the lane detection problem into detecting a pair of parallel lines. In [12], the image is converted into YCbCr format for the preparation of thresholding. The YCbCr colour space separates the luminance component (Y) from the chrominance components (Cb and Cr) of an image or video signal.

2.1.2 Feature extraction and model fitting

Feature extraction is done by using features such as edges, colours and textures. In [13], Sobel edge operator is used to extract edges from the pre-processed input and generates edge map as output. In [14], adaptive thresholding is applied to extract the lane markers from the grayscale image. In adaptive

thresholding, the threshold varies depending on the characteristics of neighbourhood which allows it to perform better than global thresholding in handling different lightning conditions in an image as shown in Figure 2.

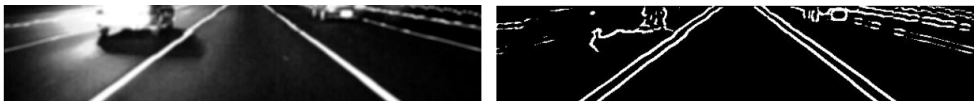

Figure 2: Left: grayscale image. Right: Binary image by adaptive thresholding. Adapted from [14].

In [12], one pixel is considered as white lane marker if the intensity value of the pixel is in the top 3% of the cumulative histogram for Y-component while one pixel is considered as yellow lane marker if its intensity value is in the lowest 1% of the cumulative histogram for Cb-component. The final binary map is obtained from the OR operation of white and yellow lane marker binary map. In [15], lane markings are extracted by exploiting their intensity, i.e. dark-light-dark characteristics. This method has been tested on different road scenarios such as highways and urban roadways under different weather conditions for day and night. However, its performance on road conditions with weak or no lane markings is not validated or discussed in the paper.

2.1.3 Model fitting

After visual feature of the lane markers is extracted from the image, model fitting is applied to generate the compact high-level representation of the path. Generally, model fitting module takes a set of points along the lane markers detected as input. In [14], assuming that lanes are straight, Hough Transform is applied to a series of estimated lane marker points to obtain the parameter of the detected lane lines. Hough Transform is a common feature extraction technique applied to the edge map and generate the parameters of the straight line detected. Although Hough Transformation can be generalised to other shapes such as circle or curve, it is not efficient to do so and a spline model applied together with RANSAC algorithm would be a better choice. In [16], detected lane-marking pixels are grouped into line segment. 100 hypotheses are then generated using RANSAC algorithm based on these line segments.

Parameters of RANSAC algorithm:

1. N – minimum number of random samples
2. K – maximum number of iterations
3. T – residual threshold to determine inliers and outliers

Steps of RANSAC algorithm:

1. N points are sampled randomly from the original data.
2. A model is fitted to N points.

3. All the points which are not sampled are classified as inlier or outlier by calculating the residual with the estimated model. A point is considered as inlier if the residual is smaller or equal to T.
4. The estimated model is saved if the number of inliers is maximal.
5. Step 1 to 4 is repeated until K iterations is reached or when there is no outlier for the estimated model.

However, such computer vision technique only works well for clear lane line with good contrast with background. It fails to detect lane lines under varied lightning conditions and not robust to interferences such as guardrails, railroad which will be analysed more in section 4. Also, there are a lot of parameters that are needed to be finetuned by humans which is time-consuming and the model may not generalise well to different conditions. Hence, there is a need for applying machine learning model in feature extraction.

2.2 Machine Learning

There are 4 categories in machine learning, which are supervised learning, unsupervised learning, semi-supervised learning and reinforcement learning.[17] The supervised learning model is trained on a labelled dataset and the ground truth data provided is used to evaluate model's performance during training. It discovers the rules underlie the labelled dataset and minimize the error between the output and the ground truth label. Most of the lane detection model are based on supervised learning due to the ease of preparation of ground truth for lane detection dataset.

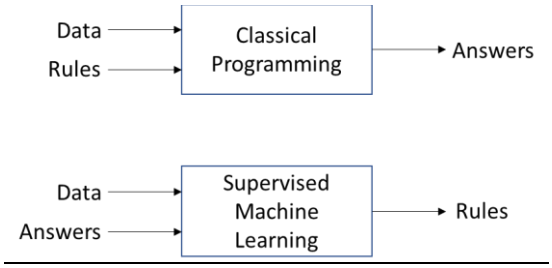

Figure 3: Difference between classical programming and supervised machine learning.

2.3 Deep Learning

Deep learning is a smaller and specialised field within machine learning. It is built by artificial neural networks (ANNs). The word "deep" means that the model is built with very deep neural network. For example, ResNet-50 is a convolutional neural network with 50 layers deep as compared to ANNs in traditional machine learning which are only at most 3 layers. In each layer, matrix multiplication is performed followed by non-linearity functions. The multi-layer non-linearity functions transform the input into high level abstract representation which is then used to perform classification or regression tasks. For example, convolutional neural network (CNN) extracts edges, textures, contours and the objects layer by layer (low level to high level features) as shown in Figure 4.

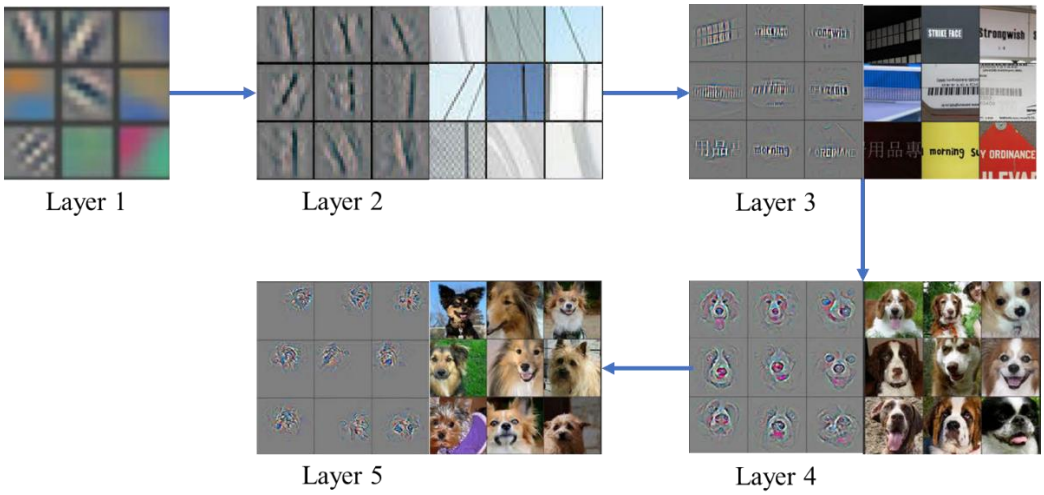

Figure 4: Visualisation of feature maps. Layer 2 extracts corners/edges. Layer 3 extracts textures and texts. Layer 4 extracts class-specific contours. Layer 5 extracts objects with pose variation. Adapted from [18].

Thanks to the advancement in the computing resource, image sensor resolution, memory capacity and the availability of large dataset, deep learning has progressed significantly in the computer vision domain. Deep learning models have achieved state of the art (SOTA) performance in lane detection as compared to traditional computer vision techniques.

There are 4 different approaches when deep learning is applied in lane detection, which are segmentation-based method, anchor-based method, parameter-based method and feature-based method. Among these methods, segmentation-based lane detection model is the mainstream method with highest upper bound for accuracy.

2.3.1 Different categories of deep learning lane detection model

2.3.1.1 Segmentation-based lane detection model

This type of lane detection model detects the position of lane lines by generating pixel-wise probability map via encoder-decoder architecture. In [7], Spatial CNN (SCNN) was proposed with special message passing mechanism. Instead of layer-by-layer convolution, slice-by-slice convolutions within feature maps was proposed. SCNN is applied as encoder in which VGG-16 is the backbone of feature extraction to generate feature map. Up-sampling module is then applied as decoder to upsample the feature map and perform pixel-wise prediction. SCNN has achieved better accuracy in segmentation of long and continuous shape structures such as traffic lanes and won 1st place in TuSimple Lane Detection Challenge in 2017.

2.3.1.2 Anchor-based lane detection model

This type of lane detection model outputs the coordinates of the lane by classification and regression from a set of lines proposed. In [19], line proposal unit (LPU) is proposed based on region proposal network from Faster R-CNN. Three parallel 1x1 convolutional layers are slide over left, right and bottom boundaries of the feature map respectively to obtain 1024-dimensional feature vector. Two

sibling 1x1 convolutional layers are then applied for line regression with fifteen line proposals and line classification (lane or not lane) respectively. During testing, non-maximum suppression (NMS) is applied because there may be more than one candidate being proposed for the same lane line.

2.3.1.3 Parameter-based lane detection model

This type of lane detection model outputs the parameter of lane line via encoder-decoder architecture. In [8], CNN is used as the backbone of encoder for feature extraction while fully-connected layer is used as the decoder which outputs the coefficients of polynomials representing the lane detected. Third order polynomial is used in modelling the lane line detected. Parameter-based model is less computationally intensive as compared to segmentation-based model.

2.3.1.4 Feature-based lane detection model

This type of lane detection model directly outputs the position of the lane lines by decoding the feature map generated by encoder. In [20], lane is represented with a series of x-coordinate for 72 equally spaced y-coordinates. The lane representations are directly outputted from the transformer decoder which takes vehicle bounding box embeddings, ROI embeddings and encoder embedding as inputs.

2.3.2 Convolutional neural network (CNN)

CNNs are designed to process data with grid-like topology.[24] In computer vision context, the input to the first layer is image and the output, feature map will be the input of the next layer.

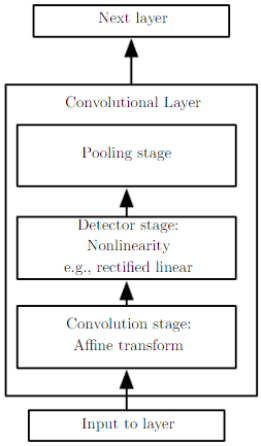

Figure 5: Components of Typical CNN.[24]

2.3.2.1 Convolution stage

Considering an image with size W x H x C as input, each filter with C kernels will be applied to every pixel from corresponding channel of the input volume. Kernel is the square matrix with the size typically smaller than the size of the input image and filter is formed by stacking multiple kernels in a row. In practice, a kernel with size 3 x 3 is used such as in ResNet.[25]

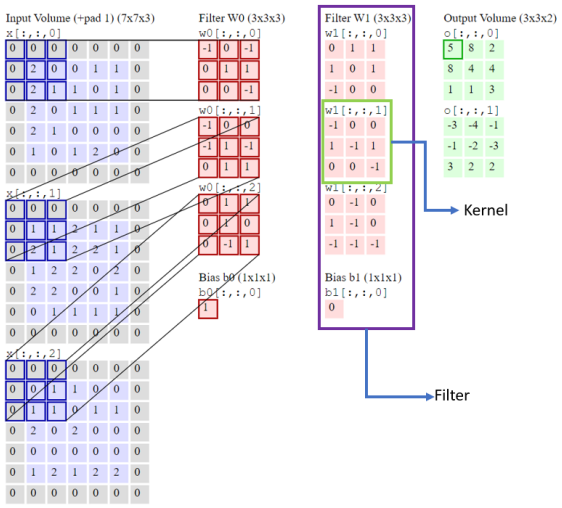

Figure 6: Illustration of 3-D convolutions. The input volumes, filters and output volumes are visualised with each depth slice stacked in rows.[26]

2.3.2.2 Detector stage

Convolution is a form of linear operation and most of the real-world data are not linearly separable. A multilayer CNN without activation function in the hidden layer will just transform the input linearly for multiple times. Hence, in detector stage, an activation function (non-linearity function) will be applied to the output from the convolution stage. Activation functions applied repeatedly in multilayer CNN will enable CNN to approximate any function due to its non-linear property. These non-linear functions will help the neural network describe the complicated and non-linear relationship between inputs and outputs.

One of the commonly used activation functions is rectified linear unit (ReLU) which zeroes out all the negative values of the input.

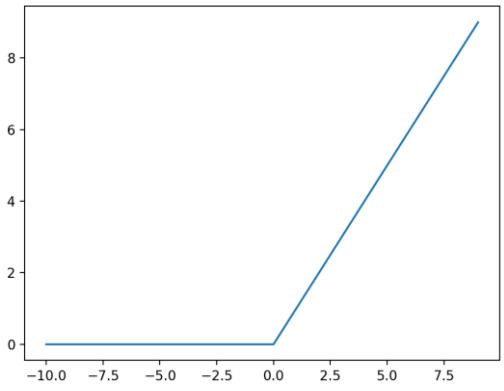

Figure 7: ReLU activation function.

2.3.2.3 Pooling stage

A pooling function replaces the output of non-overlapping subregions in the initial representation with a summary statistic of the region.[24] Pooling will downsample the initial feature map and this will reduce the parameter and computation of the next layer. The typical example is maximum pooling (max pooling) operation. In this operation, the maximum values from each non-overlapping patches will be calculated and put into the corresponding location of the downsampled feature map as shown in Figure 8.

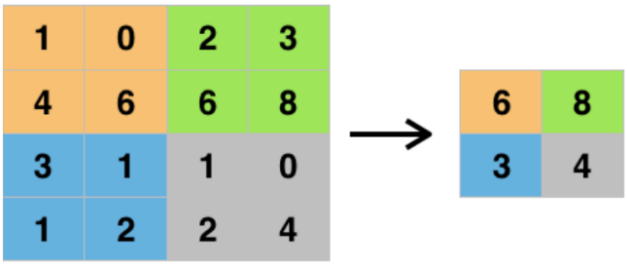

Figure 8: Illustration of Maximum Pooling.[27]

2.3.3 Transformer

Transformer[28] architecture is first introduced in 2017 with its application in natural language processing (NLP) and has become the SOTA method in solving NLP tasks. The main integral part of Transformer is the attention mechanism which draws global dependencies between input and output. Due to its highly parallel computation graph, transformer is highly efficient as compared to its predecessor, recurrent neural network with sequential nature. Residual connection and layer normalization are used in transformer to enhance the training stability and speed during training.

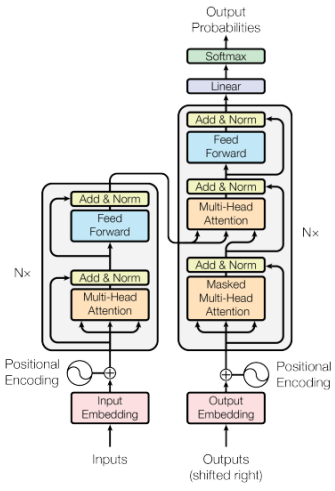

Figure 9: Model architecture of Transformer.[28]

Given its success in NLP domain, its application in computer vision has been explored and achieve SOTA result such as in image segmentation, object detection etc. Vision Transformer (ViT) is the first model which apply Transformer without modification and become the SOTA model on multiple image

recognition benchmarks. In [29], Transformer is applied as encoder-decoder with CNN as backbone in parameter-based lane detection model and achieve 96% accuracy on TuSimple benchmark.

2.3.3.1 Multi-Head Attention

In ViT, the image is divided into multiple blocks with 16 by 16 pixels. Each block will be flattened and form the input embedding. After addition of positional encoding and input embedding, the result is transformed linearly into three new matrices, which are Queries (Q), Keys (K) and Values (V). The channels of Q, K and V are divided by the number of head, h so that the output of multi-head attention will have the same number of channel as input after concatenation. Scaled Dot-Product Attention is then calculated according to equation 1 where d_k is the channel of the queries. The computation is repeated for h times and all the results are concatenated followed by linear transformation. The ultimate result will be the output of multi-head attention. As shown in Figure 10, multi-head attention allows the model to learn different information from different representation subspaces.

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V \quad (1)$$

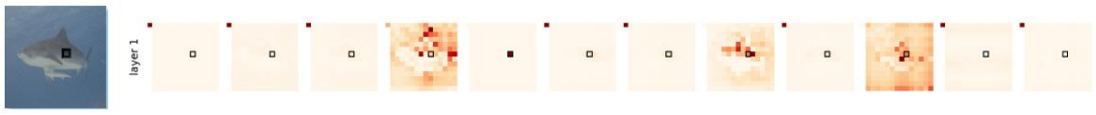

Figure 10: Attention map of every head in layer 1 of Vision Transformer (ViT). The small black box is the query 16x16 block.[30]

Chapter 3: Methodology

3.1 Implementation Environment

3.1.1 Software

This project was written in python 3.6.13 with Conda as the environment management system. Pip and Conda were used as package management system. Major libraries used were listed in the table below:

Libraries	Description
PyTorch	PyTorch is a machine learning library which provides different python application programming interface (APIs) for building, training and deploying machine learning models.
NumPy	NumPy is a Python library that is widely used for scientific computing and data analysis. It provides support for arrays, matrices and high-level mathematical functions.
JSON (JavaScript Object Notation)	JSON is a data-interchange format designed for easy reading and writing by both humans and machines. It was used for json string parsing in this project.
OpenCV-Python	OpenCV-Python is a Python wrapper around the popular computer vision library OpenCV. OpenCV is a comprehensive computer vision library written in C++ that includes a wide range of image and video processing algorithms.
MMsegmentation	APIs of MMsegmentation library was used in building SegFormer model.
Matplotlib	A comprehensive library for visualisations in Python.
tensorrt	TensorRT is a software development kit (SDK) and it was used to boost the inference performance of the model trained.
PyCUDA	PyCUDA is a Python library that provides access to the NVIDIA CUDA library. It was used in the

	deployment of model engine by allocating page-locked memory on host device.
Weights and Biases (WandB)	WandB provides tools which help developers track, visualise and reproduce machine learning experiments. Its APIs were used in this project to log and visualise the gradients in backpropagation, training and validation loss.

Table 1: Libraries used with descriptions.

3.1.2 Hardware

The hardware involved was the powerful computer in Continental-NTU Corporate Lab. It was equipped with Intel Core i9-10900X CPU @ 3.70GHz and 3 NVIDIA GeForce RTX 2080 Ti GPUs. The powerful CPU allowed fast data preparation and processing while GPU allowed fast model training and testing by performing computations in parallel.

3.2 Pipeline

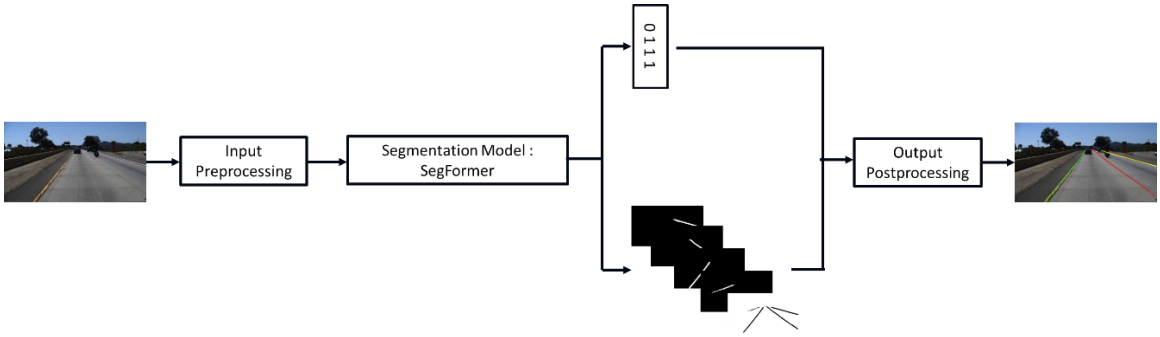

Figure 11: Pipeline of Lane Detection.

Figure 11 shows the pipeline of lane detection proposed in this paper. In the following sections, details on the generation of ground truth segmentation map, input preprocessing, model building and output postprocessing will be explained.

3.2.1 Generation of Ground Truth Segmentation Map


```
{"lanes": [[-2, -2, -2, -2, 632, 625, 617, 609, 601, 594, 586, 578, 570, 563, 555, 547, 539, 532, 524, 516, 508, 501, 493, 485, 477,
469, 462, 454, 446, 438, 431, 423, 415, 407, 400, 392, 384, 376, 369, 361, 353, 345, 338, 330, 322, 314, 307, 299], [-2, -2, -2, -2,
719, 734, 748, 762, 777, 791, 805, 820, 834, 848, 863, 877, 891, 906, 920, 934, 949, 963, 978, 992, 1006, 1021, 1035, 1049, 1064, 1078,
1092, 1107, 1121, 1135, 1150, 1164, 1178, 1193, 1207, 1221, 1236, 1250, 1265, -2, -2, -2, -2, -2], [-2, -2, -2, -2, -2, 532, 503, 474,
445, 416, 387, 358, 329, 300, 271, 241, 212, 183, 154, 125, 96, 67, 38, 9, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2,
-2, -2, -2, -2, -2, -2, -2, -2, -2], [-2, -2, -2, 781, 822, 862, 903, 944, 984, 1025, 1066, 1107, 1147, 1188, 1229, 1269, -2, -2, -2,
-2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2, -2]], "h_samples": [240,
250, 260, 270, 280, 290, 300, 310, 320, 330, 340, 350, 360, 370, 380, 390, 400, 410, 420, 430, 440, 450, 460, 470, 480, 490, 500, 510,
520, 530, 540, 550, 560, 570, 580, 590, 600, 610, 620, 630, 640, 650, 660, 670, 680, 690, 700, 710], "raw_file": "clips/0313-1/6040/20.
jpg"}
```


Figure 12: Example of ground truth label of TuSimple dataset.

For TuSimple dataset, lanes were labelled using x coordinates ("lanes") and y coordinates ("h_samples") as shown in Figure 12. Since the lane detection problem was framed as segmentation problem, ground truth segmentation maps were generated as shown in Figure 13. The width of each lane line was configured to be 15 pixels wide. In the ground truth segmentation map, numbers 2 and 3 represent the left and right ego lane lines while number 1 and 4 represent immediate left and right lane lines beside the ego lane. For visualization, numbers 1 to 4 were represented by blue, green, red and yellow respectively.

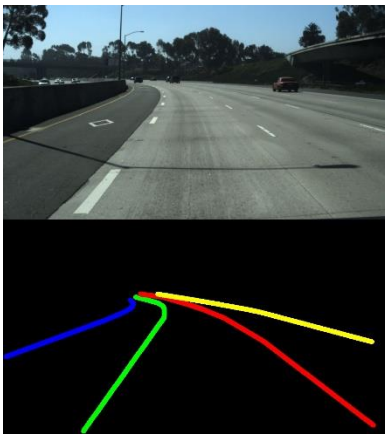

Figure 13: Training Image and Its Corresponding Segmentation Map.

Since there was no label given on ego lane and immediate side lane, lanes were classified as follow:

1. For each image, coordinates of each lane were read.
2. Tip and tail coordinates were assigned as (x_1, y_1) and (x_2, y_2).
3. The angle formed by the straight line connecting tip and tail were calculated:

$$\theta = \tan^{-1}\left(\frac{y_1 - y_2}{x_2 - x_1}\right)$$

4. Ego lane and immediate side lane were determined by the following rules:
   i. $\theta_{egoleft}$ and θ_{left} are smaller than 90°. $\theta_{egoleft}$ is smaller than θ_{left}.
   ii. $\theta_{egoright}$ and θ_{right} are larger than 90°. $\theta_{egoright}$ is smaller than θ_{right}.

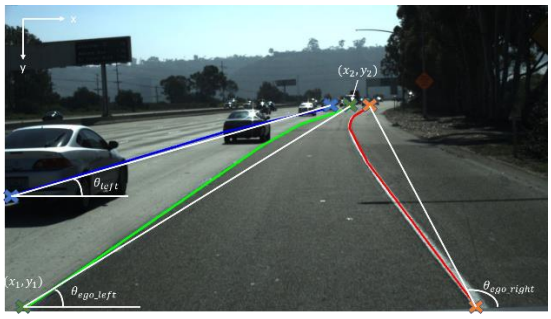

Figure 14: Calculation of angle for each lane.

Generation of ground truth segmentation map was skipped for CULane dataset because the ground truth segmentation maps were provided for every image in the dataset.

3.2.2 Input Preprocessing

Before feeding the image and ground truth segmentation map into the model, TuSimple images were resized to (288,512) while CULane images were resized to (288,800). Color_Jitter() function was then applied to the image to change the brightness, contrast, hue and saturation of the image randomly as shown in Figure 15. Random flipping with a probability of 0.5 is then applied to the image and ground truth segmentation map as shown in Figure 16. After augmentation, TuSimple images were normalized with ImageNet's mean and standard deviation while CULane images were normalized with mean and standard deviation provided officially.

Figure 15: Augmentation applied by Color_Jitter() function. The leftmost is the original image while the rest are the image augmented.

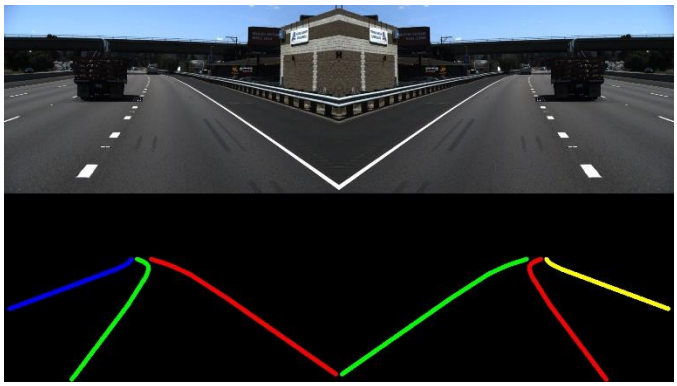

Figure 16: Augmentation applied by RandomFlip() function. The left image is the original image while the right image is the image flipped horizontally.

3.2.3 Model Building

PyTorch was used in the training and evaluation of the model. SegFormer was used as the backbone of the model by calling APIs provided by MMsegmentation toolbox.

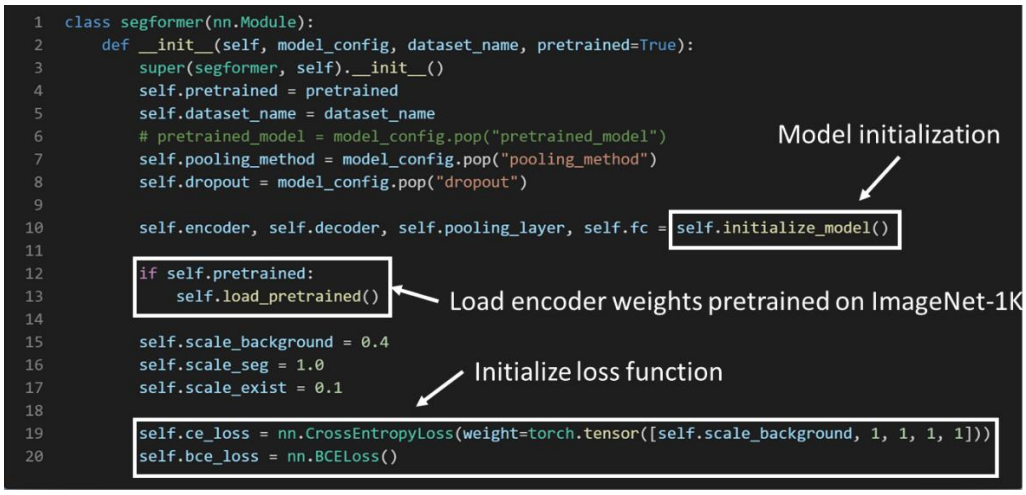

Figure 17: Init function of class segformer.

The model was initialized and the pretrained weights of encoder was loaded by calling load_pretrained() function and initialize_model() function.

```
def load_pretrained(self):
    encoder_weight = torch.load("/home/automan/wuguanjie/SegFormer/pretrained/mit_b0.pth")
    # encoder pretrained on ImageNet 1K, remove the FCL weight and bias, decoder is randomly initialized
    encoder_weight.pop("head.weight")
    encoder_weight.pop("head.bias")
    # remove pretrained weights of spatial reduction for the first 2 blocks since the sr_ratio has been reduced
    encoder_weight.pop("block1.0.attn.sr.weight")
    encoder_weight.pop("block1.1.attn.sr.weight")
    encoder_weight.pop("block2.0.attn.sr.weight")
    encoder_weight.pop("block2.1.attn.sr.weight")
    self.encoder.load_state_dict(encoder_weight,strict=False)
```

Figure 18: load_pretrained() function under the class segformer.

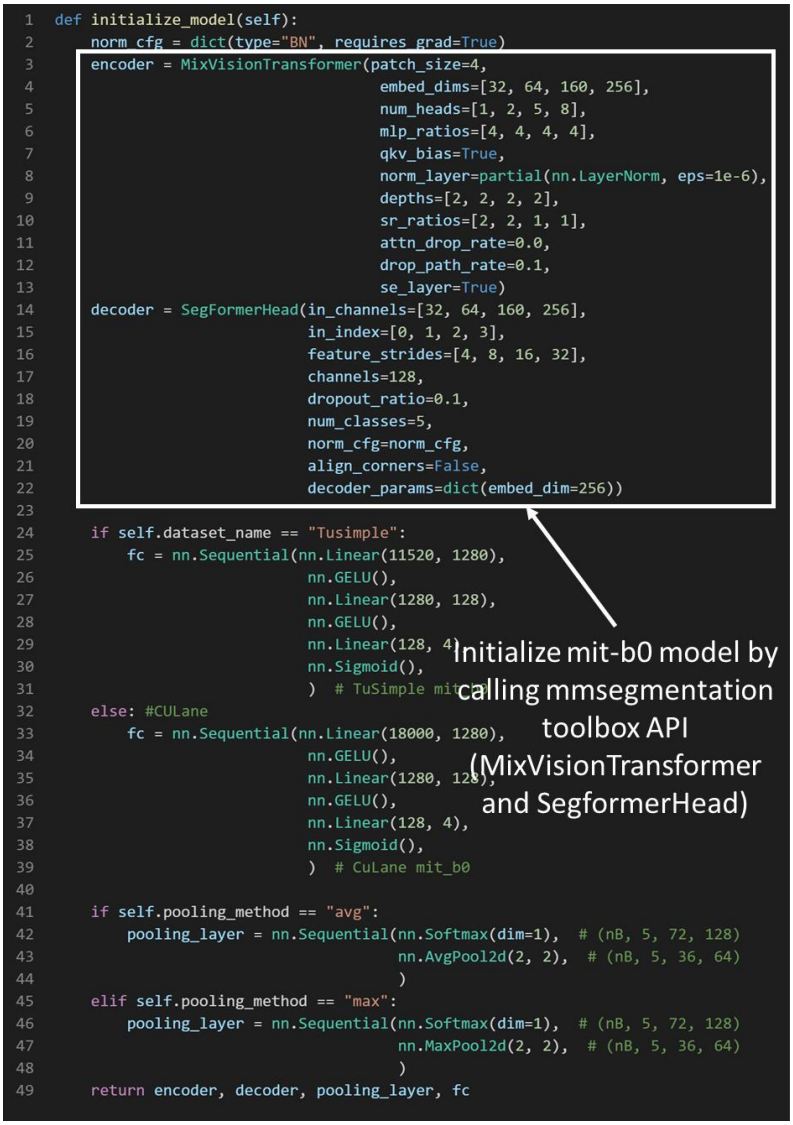

Figure 19: initialize_model() function under class segformer.

For TuSimple and CULane dataset, the model was trained using AdamW optimizer with batch size of 8 and 4 respectively for 30 epochs. The learning rate scheduler was ReduceLROnPlateau with a multiplicative factor of 0.1 and patience of 5 epochs.

The forward pass and backward pass of the model were executed as shown in Figure 20.

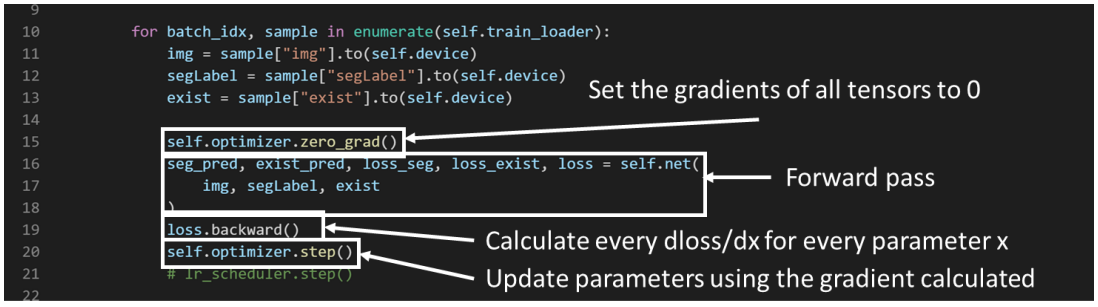

Figure 20: Training loop of the model.

3.2.4 Output postprocessing

The output from the model would be 5 segmentation maps and four binary numbers which indicate the existence of lanes (0: the lane does not exist, 1: the lane exists).

Figure 21: Segmentation Output. (1): Background pixels. (2): Immediate left lane line beside ego lane. (3): left ego lane line. (4): right ego lane line. (5): immediate right lane line beside ego lane.

The first segmentation map was discarded since it segmented out the background pixels. The next four segmentation maps would only be postprocessed if and only if its corresponding predicted lane existence was 1. For each segmentation map, the x coordinates with the maximum probability of lane line were sampled at a list of y coordinates with fixed interval. If the maximum probability of the x coordinate at the corresponding y coordinates ("h_samples") was greater than 0.3, it would be considered as positive. If the maximum probability was smaller than 0.3, it would be considered as negative and -1 will be used to indicate the non-existence of lane point. The list of y coordinates for TuSimple dataset starts from 160 to 710 with interval of 10 pixels while that of CULane dataset starts from 99 to 589 with interval of 20 pixels.

After all the points had been sampled, RANSAC algorithm was applied to fit a fourth polynomial function with the coordinates given. The hyperparameters of RANSAC algorithm were as shown in Figure 22.

```
if num_samples > 10:
    ransac = RANSACRegressor(
        PolynomialRegression(degree=poly_degree),
        residual_threshold=0.1 * np.std(x_val),
        min_samples=int(0.4 * num_samples),
        max_trials=100,
        random_state=0,
    )
```

Figure 22: Hyperparameters of RANSAC Algorithm.

3.3 Model Construction

In this project, lane detection was modelled as a semantic segmentation problem. It could be modelled as binary semantic segmentation problem with only one single segmentation output, which classified every pixel as lane or non-lane. However, its application would be limited to the system with localization due to the lack of information on ego lane. Hence, 4-lane detection model was constructed in this project. 4 was chosen because the information of ego lane, immediate left lane and right lane beside ego lane were critical in controlling the car.

Figure 23: Difference between binary semantic segmentation model with one segmentation map and 4-lane semantic segmentation model. left: segmentation of lane line with only one output map. right: segmentation of n lane line with (n+1) output map.

CNNs model have been the de-facto for computer vision task. However, CNNs have strong prior as compared to transformer. For example, considering convolution layer as fully connected layer, all the weights are assumed to be zero, except the spatially defined neighbourhood of the input. This means that the function should only learn the local interaction of the input.

There have been a lot of modifications applied to CNNs to achieve better lane detection performance. For example, in SCNN, the special message passing mechanism was applied to better capture the long and thin structure. However, this was computationally intensive and could not be applied in real time setting.

According to paper 'on the relationship between self-attention and convolutional layers', the self-attention layers with sufficient heads are able to express any convolutional layer when they are applied to images. Also, the fully attentional models are the generalization of CNNs which learn the kernel pattern and the filters' weights at the same time. Hence, transformer is chosen for feature extraction in this project.

3.3.1 SegFormer

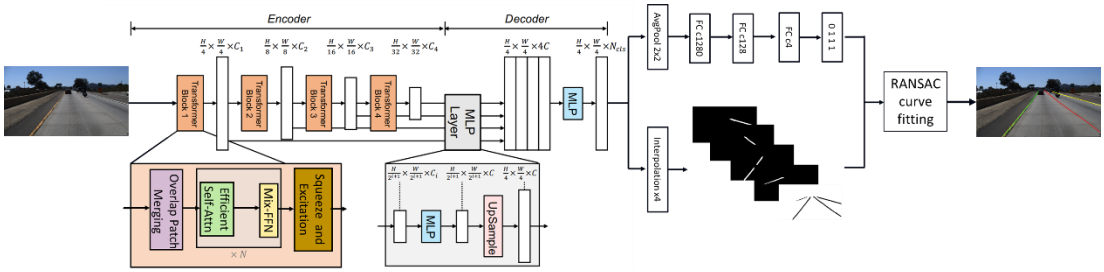

Figure 24:Implementation of lane detection model with SegFormer. Best view with zooming in.

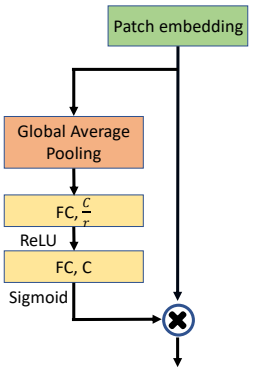

Figure 25: Squeeze and Excitation Network.

The model used for feature extraction in this project was SegFormer. SegFormer is a simple and efficient sematic segmentation framework combining both transformer architecture with multilayer perceptron (MLP) decoders. The most lightweight model, SegFormer-B0 was used in this paper because it only has 8.4G floating-point operations per second (FLOPs) when it was applied on image classification task (ADE20K) given an image of 512 x 512 pixels.

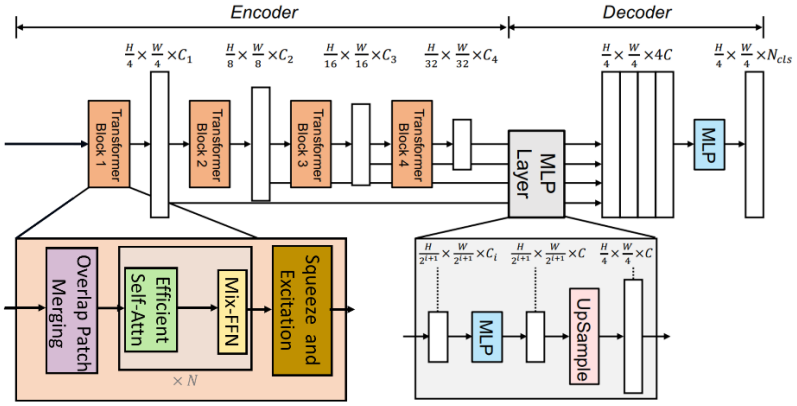

Figure 26: SegFormer model architecture. Modified from [31].

An image of H x W x 3 is divided into small patches of 4 x 4. Smaller size patches is chosen because semantic segmentation is a dense prediction task. These patches are projected into embeddings and used as the input to the hierarchical Transformer encoder and output CNN-like multi-level feature maps.

These features maps are then fed into multilayer perceptron decoder and output N_{class} of semantic segmentation maps. N_{class} is 5 in this project.

3.3.1.1 Hierarchical Transformer Encoder

3.3.1.1.1 Hierarchical Feature Representation

In a typical CNN, the input image is processed by multiple convolutional layers followed by max pooling layer, each of which generates a feature map that encodes a different abstraction of the input image. In the encoder of CNN, the resolution of the feature map will decrease but its channel will increase from layer to layer as shown in Figure 27.

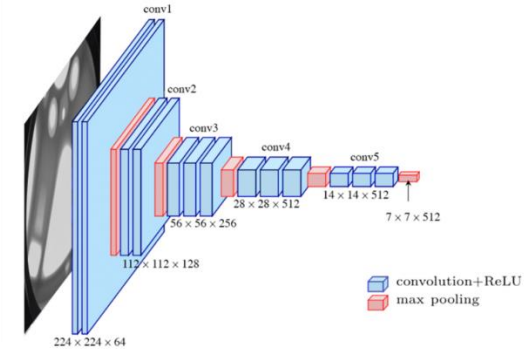

Figure 27: VGG-16 Net.[32]

The transformer encoder in SegFormer is designed to mimic this behaviour and generate feature maps at different resolution to boost its performance on semantic segmentation. Given an image of H x W x 3, overlapped patch merging is performed to obtain feature map with resolution of $\frac{H}{2^{i+1}} \times \frac{W}{2^{i+1}} \times C_i$, where $i \in \{1,2,3,4\}$ and C_{i+1} is larger than C_i.

3.3.1.1.2 Overlapped Patch Merging

Overlapped patch merging is performed by using 2D convolution. Convolution layer with K =7, S = 4 and P = 3 is used to prepare the overlapped patch embedding for the first block in encoder, where K is the size of kernel, S is the stride and P is padding. Convolution layer with K = 3, S = 2 and P = 1 is applied to the feature map to downside its resolution by half before it is fed into the next transformer encoder block.

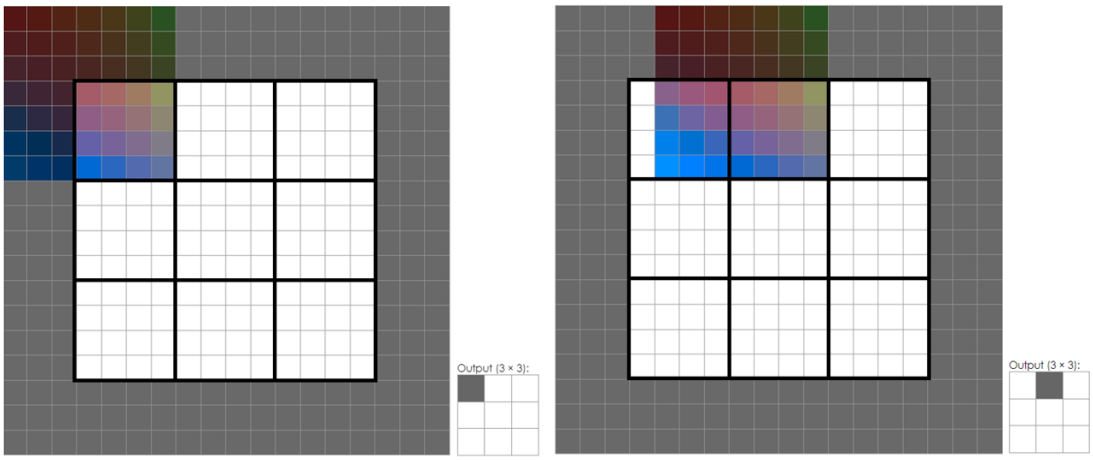

Figure 28 Example of overlapped patch merging. The input tensor size is 12 × 12, which is divided into 9 4 × 4 patches. Convolution is performed with kernel size = 7, stride = 4 and padding = 3.[33]

3.3.1.1.3 Efficient Self-Attention

The inputs of the multi-head self-attention mechanism are queries(Q), keys(K) and values(V) and they have the same dimensions, N × C, where N = H × W is the length of the sequence. The time complexity of this mechanism will be $O(N^2)$ and this is not ideal for large image resolution.

Reduction ratio R is introduced to decrease computational load by reducing N, the length of the sequence as follow:

$$\widehat{M} = Reshape\left(\frac{N}{R}\right)(M) \quad (2)$$

$$M = Linear(C, C)\left(\widehat{M}\right) \quad (3)$$

where M is the sequence to be reduced.

$Reshape\left(\frac{N}{R}\right)(M)$ operation will apply 2D convolution to reshape M from size N × C to $\frac{N}{R} \times C$ followed by linear transformation on $\widehat{M}$. Hence, the shape of M is $\frac{N}{R} \times C$ and the time complexity is reduced to $O(\frac{N^2}{R})$.

3.3.1.1.4 Mix-FFN

No positional encoding is used to represent the location of the patches because the zero-padding operation in overlapped patch merging will leak the absolute position information of patches. A depth-wise convolution layer of 3 × 3 is used in the feedforward network (FFN) to consider the effect of zero padding. The formulation of Mix-FFN is as follow:

$$x_{out} = MLP\left(GELU\left(Conv_{3\times3}\left(MLP(x_{in})\right)\right)\right) + x_{in} \quad (4)$$

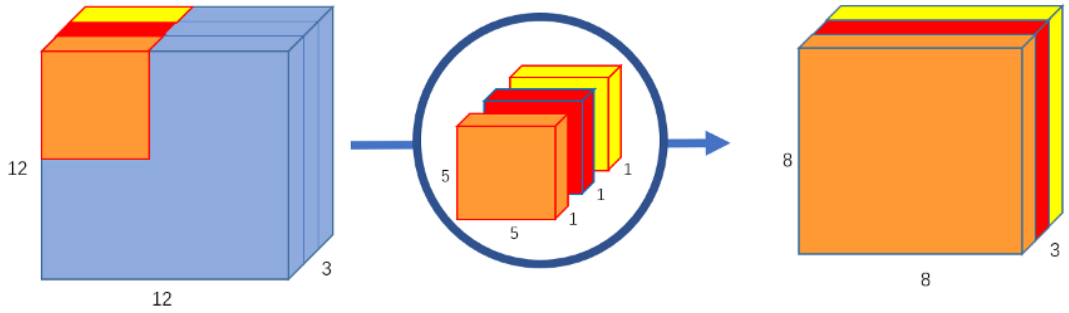

Figure 29: Depth-wise convolution

3.3.1.1.5 All-MLP Decoder

The decoder is formulated as follow:

$$\hat{F}_i = Linear(C_i, C)(F_i), i \in [1,4] \quad (5)$$

$$\hat{F}_i = Upsample\left(\frac{H}{4} \times \frac{W}{4}\right)(\hat{F}_i), i \in [1,4] \quad (6)$$

$$F = Linear(4C, C)\left(Concat(\hat{F}_i)\right), i \in [1,4] \quad (7)$$

$$M = Linear(C, N_{cls})(F) \quad (8)$$

Where F is the feature map from transformer block, M is the predicted mask, H and W is the original image height and width, $Linear(C_{in}, C_{out})(\cdot)$ refers to a linear layer with C_{in} dimensions input and C_{out} dimension output, $Upsample(height, width)(\cdot)$ refers to upsample input to the specified height and width.

3.3.2 Squeeze and Excitation Network

Squeeze-and-Excitation network[34] was introduced into the SegFormer model to model the interdependencies between channels of feature maps. Global average pooling is used to squeeze the spatial information of each channel into a channel descriptor. A bottleneck formed by two fully connected layer with ReLU function in between was used to learn the nonlinear relationship between channels. Sigmoid function was then used to map the values into range 0 to 1, considering the non-mutually exclusive relationship between channels. The feature map was then rescaled by multiplying with the weight generated from this network.

Chapter 4: Experiments & Discussions

4.1 Datasets

Initially, the widely used dataset, TuSimple was used for evaluating the model.[35] There were 6408 annotated frames which were extracted from the 20th frames of every clips. This dataset was captured on United States (US) highway roads by using forward facing camera at different daytime with good or medium weather. It was split into training set with 3626 frames, validation set with 358 frames and testing set with 2782 frames. All frames have the resolution of 1280 x 720.

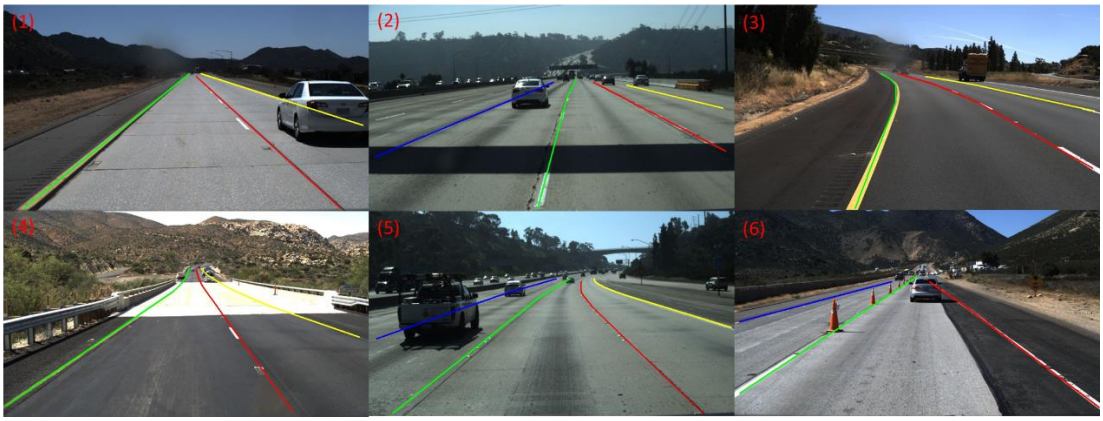

Figure 30 : Different scenarios in TuSimple dataset. (1): normal, (2): shadow, (3): curve, (4): change in illumination, (5): reflective lane marker, (6): cone

Besides, CULane dataset was used to evaluate the performance of the model under different challenging scenarios. The size of CULane dataset is about 20 times of the size of TuSimple dataset. It contains urban, rural and highway scenes which can be further divided into 9 categories as shown in Figure 31. It was split into training set with 88880 frames, validation set with 9675 frames and testing set with 36480 frames. All frames have the resolution of 1640 x 590.

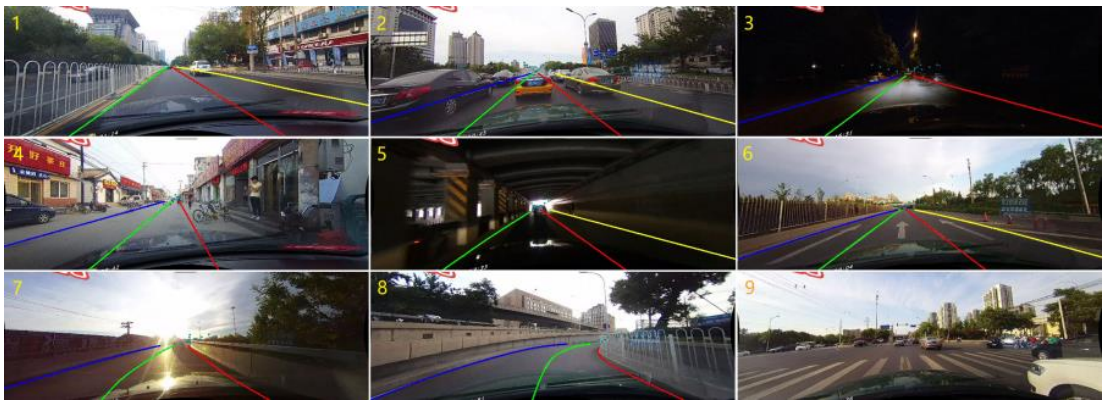

Figure 31 : Different scenarios in CULane dataset. (1): normal, (2): crowded, (3): night, (4): no line, (5): shadow, (6): arrow, (7): dazzle light, (8): curve, (9): crossroad [7]

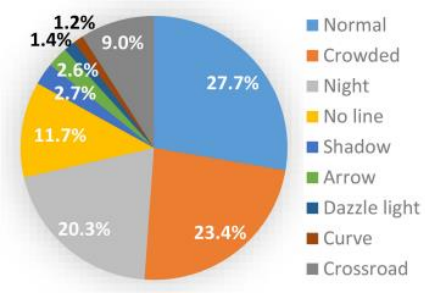

Figure 32 : Proportion of each scenario in test set. [7]

4.2 Evaluation Metrics

4.2.1 TuSimple Metrics

Since TuSimple dataset was used to benchmark the model, the metrics used in TuSimple lane detection challenge were followed. They were accuracy, false positive (FP) and false negative (FN). The accuracy was calculated by equation 9, where C_i and S_i was the count of correct detected points and ground truth points respectively. A point was considered as detected correctly if the width between ground truth point and detected point was less than 20 pixels. A lane was considered as positive detection if its accuracy was higher than 85%. False positive (FP) was calculated by equation 10 where F_{det} was the count of wrong detected lanes and N_{det} was the total count of detected lanes. FP implied that there was no match between the lane detected and any ground truth lane. False negative was calculated by equation 11 where M_{det} was the number of ground-truth lanes which was missed in the predictions and N_{gt} was the total count of ground-truth lanes. FN implied that the lane in ground-truth does not match with any detected lane.

$$accuracy = \frac{\sum_i C_i}{\sum_i S_i} \quad (9)$$

$$FP = \frac{F_{det}}{N_{det}} \quad (10)$$

$$FN = \frac{M_{det}}{N_{gt}} \quad (11)$$

4.2.2 CULane Metric

During testing, the width of lane lines was increased from 1 pixel to 30 pixels and intersection over union (IOU) was calculated between ground truth and detection. Detected lanes with IOUs greater than 0.5 was considered as TP. FP is the total number of wrong detected lane while FN is the total count of ground-truth lanes which is missed in the predictions. Precision and recall were calculated using equation 12 and 13. After these metrics had been obtained, F1-measure was calculated using equation 14.

$$precision = \frac{TP}{TP + FP} \quad (12)$$

$$recall = \frac{TP}{TP + FN} \quad (13)$$

$$F1\text{–}measure = 2 * \frac{precision * recall}{precision + recall} \quad (14)$$

4.3 Results

4.3.1 TuSimple dataset

Highway road scenarios are considered less complicated as compared to urban driving. There are fewer obstacles on highway and most of the lane lines are well defined with clear lane marks. Hence, the model was trained on TuSimple dataset first to assess its ability in lane detection. The results are shown below:

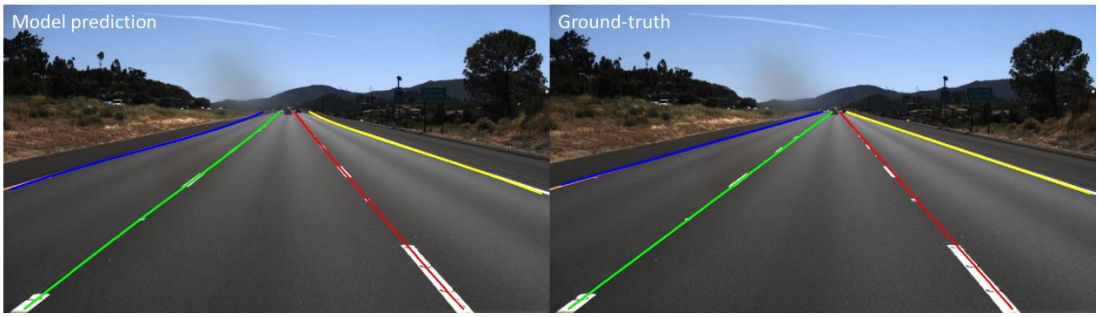

Figure 33: Lane detection under normal road condition.

Figure 34: Lane detection of road with shadow.

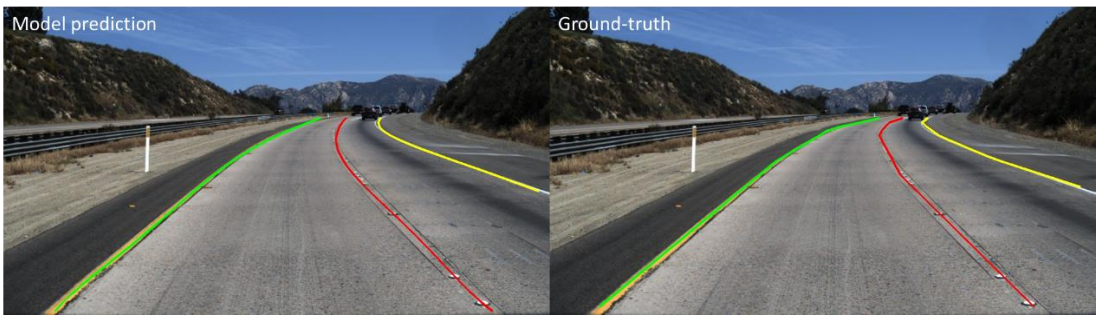

Figure 35:Lane detection of curved road.

Figure 36: Lane detection of road with cones.

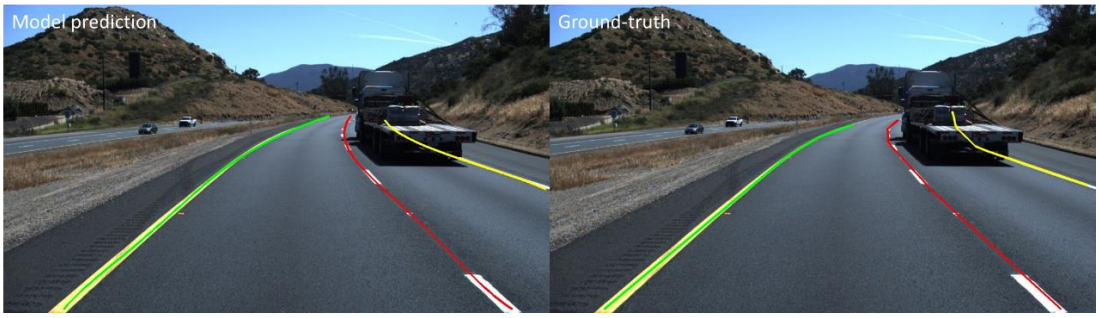

Figure 37: Lane detection of road with occlusion.

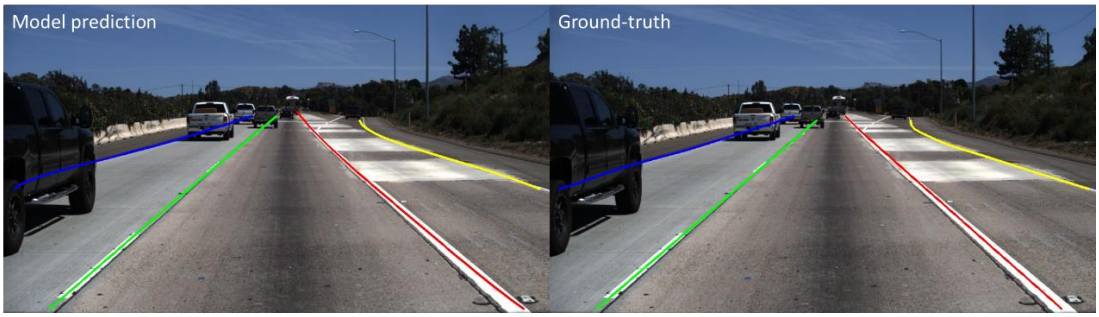

Figure 38: Lane detection of road with patches.

Figure 39: Lane detection of road (predicted lane line longer than ground truth lane line).

4.3.2 CULane dataset

The performance of the model on TuSimple dataset was satisfactory. Hence, the model was trained and tested on CULane dataset subsequently with even more complicated and challenging urban roadways.

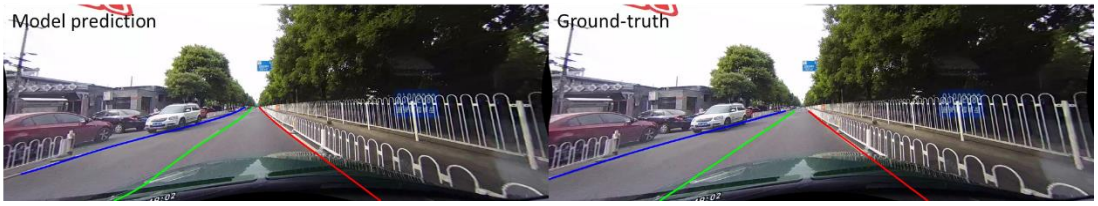

Figure 40: Lane detection under normal road condition.

Figure 41: Lane detection under crowded road condition.

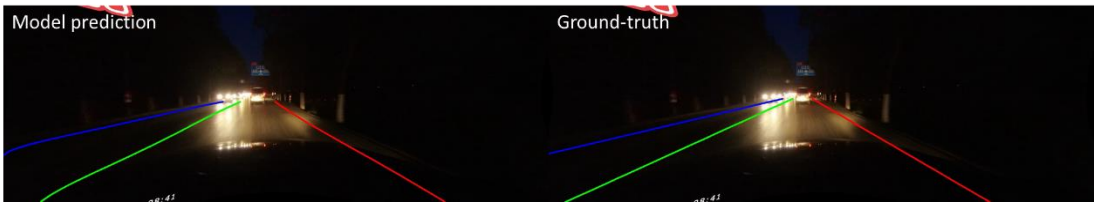

Figure 42: Lane detection at night.

Figure 43: Lane detection of road with no lines.

Figure 44: Lane detection of road with shadow.

Figure 45: Lane detection of road with arrows.

Figure 46: Lane detection of road with dazzle light.

Figure 47:Lane detection of curved road.

Figure 48:Lane detection at crossroad.

4.5 Discussions

In this section, comparison was made between deep learning model and traditional computer vision algorithm. After that, the deep learning model proposed was compared to SCNN, which is the winner of 2017 TuSimple lane detection challenge.

4.5.1 Robustness of traditional lane detection model

Most of the lane detection models using traditional computer vision techniques are not computationally intensive. Edge device with CPU only would be sufficed in handling the computational load of these model. Although these models could be implemented on low-cost devices, they only work well under very limited condition such as highway with good contrast between lane markers and the roads. Often, they suffer greatly when there are illumination changes, occlusions, weak lane marks etc.

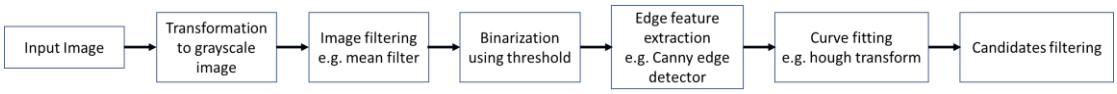

Figure 49: Typical flow of lane detection using traditional computer vision.

Among all the steps, binarization is considered as the most important step because it has direct impact on the output of model fitting. Thresholding is commonly used in binarization but it is not robust to illumination changes as shown in Figure 50.

Besides, there are a lot of false positive sample in lane lines detected when these models are deployed in urban roadways as shown in Figure 51. Although ROI can be used to eliminate these false detections, it will limit the lane detection model application to ego lane detection only, which is not sufficient for autonomous driving system.

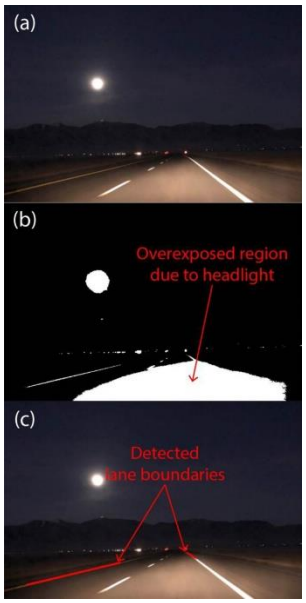

Figure 50: Failure of thresholding based on Otsu's method. (a) is the raw image. (b) is the binary image. (c) is the lane detection result.[6]

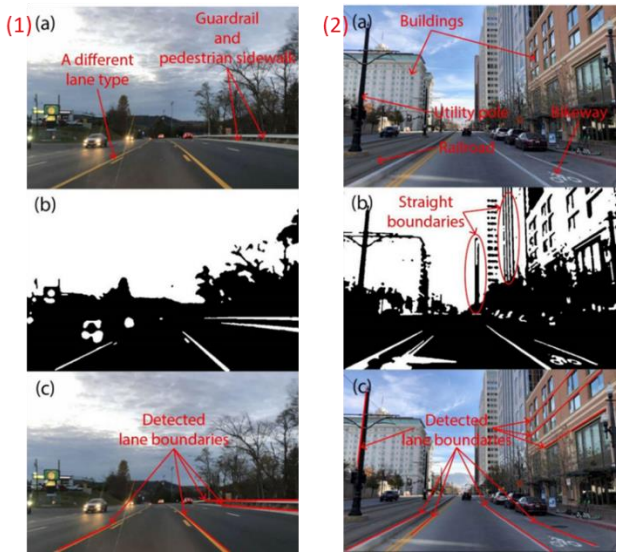

Figure 51: Illustration of false positive lane lines on (1) highways and (2) urban roadways.[6]

4.5.2 Deep learning lane detection model

Deep learning lane detection model are much more robust to different road scenarios as compared to traditional lane detection model.

The models were assessed in terms of accuracy, FLOPs, number of parameters (params) and frame per second (FPS) to explore the real time application of the models. All the tests were done on GeForce RTX 2080Ti.

Model	Input Resolution	Accuracy	False Positive	False Negative	FLOPs† (G)	Params† (M)	FPS
SegFormer (mit-b0)	(288,512)	**95.52%**	**0.0349**	**0.0451**	**3.86**	**18.22**	**86.58**
SCNN‡ (VGG-16)	(360,640)	95.17%	0.0637	0.0622	218.64	20.96	21.18

Table 2: Model performance on TuSimple dataset

For model testing with TuSimple test set, as shown in Table 2, SegFormer outperformed SCNN by 0.35% in terms of accuracy while being 4 times faster than SCNN in terms of FPS. Also, SegFormer could be deploy in real time autonomous driving system which typically has a minimum requirement of 30 FPS.

† Flops and params of SegFormer are calculated based on thop (https://github.com/Lyken17/pytorch-OpCounter)

‡ Performance of SCNN was obtained from https://github.com/voldemortX/pytorch-auto-drive/blob/master/docs/MODEL_ZOO.md

The performance of transformer architecture is more superior than CNN due to its ability in expressing any convolutional layers and draw global dependencies between pixels starting from the early layers. For example, in the first block of self-attention in SegFormer, some attention heads were focusing on part of the image locally such as lane markers while the other attention heads started drawing global dependencies between pixels such as identifying background pixels in an image. Visualization of attention maps can be found in appendix. One interesting observation from the attention maps visualised was that the sky pixels had a high dependency not only with other sky pixels, but also the road surface pixels in one of the self-attention head as shown in Figure 52. If ROI is used and the sky pixels are excluded, the model performance may be impacted negatively.

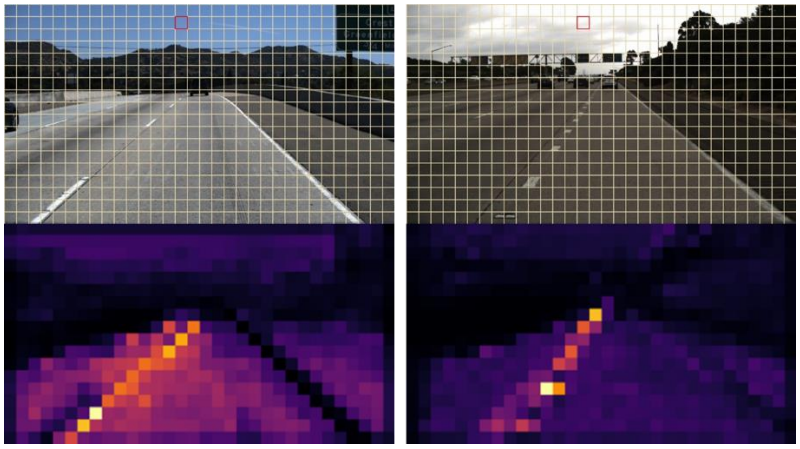

Figure 52: Attention map from the fourth head of first self-attention block from transformer block 3.

For model testing with CULane test set, as shown in Table 4, the F1-measure of SegFormer is 1.45% lower than SCNN averagely. The possible reason for SegFormer's lower performance on the complex and diverse CULane dataset could be attributed to its lack of inductive bias, which requires a larger amount of data for effective training and to surpass the performance of SCNN. However, the FPS of SegFormer was about 2 times faster than SCNN. The much higher FPS justified the trade-off between F1-measure and the computational load of the model.

Model	Input Resolution	FLOPs (G)†	Params (M)†	FPS
SegFormer (mit-b0)	(288,800)	**6.03**	26.52	**54.26**
SCNN (VGG-16) ‡	(288,800)	218.62	**20.74**	21.40

Table 3: Model FLOPs, parameter size and FPS on CULane dataset

Conditions	F1-measure of SegFormer (%)	F1-meaure of SCNN (%)
Normal	89.04	**90.6**
Crowded	67.64	**69.7**
Dazzle light	**61.65**	58.5
Shadow	66.01	**66.9**
No line	40.67	**43.4**
Arrow	80.98	**84.1**
Curve	62.43	**64.4**
Crossroad	2439(FP)	**1990 (FP)**
Night	63.72	**66.1**

Table 4: Model performance on CULane dataset (IOU = 0.5).

To further boost the FPS of the model, SegFormer-based lane detection model was converted from PyTorch to TensorRT engine through intermediary ONNX. After conversion, the FPS of TensorRT engine and PyTorch model were tested on GeForce RTX 2080Ti without postprocessing steps. The FPS of the TensorRT engine was boosted by at least 20% as compared to PyTorch model. Since the real time application of lane detection model only requires at most 30 FPS, the lane detection model proposed could possibly be deployed on less powerful GPU such as GeForce GTX 1080Ti.

Dataset	FPS (PyTorch)	FPS (TensorRT)	Difference
TuSimple	86.58	106.08	Increase by 22%
CULane	54.26	69.07	Increase by 27%

Table 5: FPS of SegFormer-based lane detection model deployed in PyTorch and TensorRT.

Based on the analysis above, it could be inferred that the objective of this project had been achieved. Without any doubt, SegFormer-based lane detection model was able to surpass the performance of the lane detection model using traditional computer vision techniques. Besides, it was able to handle different challenging road conditions robustly while being able to deploy in real time application on a mid-range GPU. However, this was still not a perfect model which can handle every road scenario in the real world as shown in Figure 53. Therefore, this leads to the chapter 5 which will discuss the possible ways in further improving the accuracy and robustness of the model.

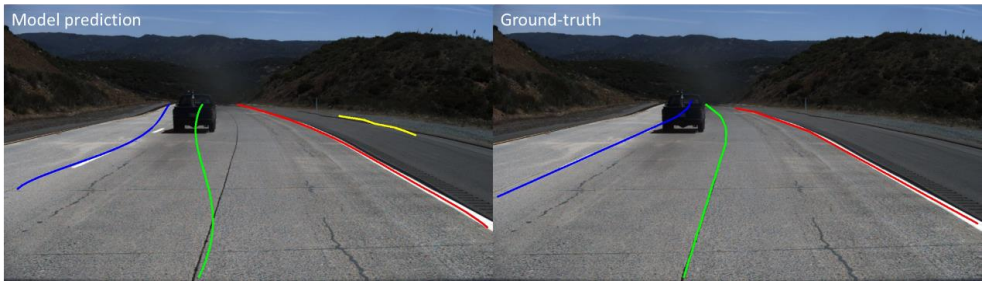

Figure 53: Failure Case.

4.5.3 Ablation study

Ablation study was done to understand the contribution of different components of the model. Due to the time constraint, ablation study was only done on TuSimple dataset. Baseline model is SegFormer model with reduction ratio [8,4,2,1] trained with color jitter augmentation. The introduction of random flipping augmentation indicated that the model performance was not saturated and the accuracy of the model could be further increased by collecting more data. By using a smaller reduction ratio, less information was lost which further increased the accuracy by 0.17%. Lastly, by introducing squeeze and excitation layer (se_layer), all the feature maps were weighted which result in a better fusion of different feature maps.

	Accuracy (%)	Increase in accuracy (%)	False Positive	False Negative
Baseline	94.97	-	0.0485	0.0624
Baseline + randomflip	95.14	0.17	0.0389	0.0549
Baseline with reduction ratio [2,2,1,1] + randomflip	95.31	0.17	0.0371	0.0489
Baseline with reduction ratio [2,2,1,1] + randomflip + se_layer	95.51	0.2	0.0349	0.0451

Table 6: Ablation Studies on TuSimple Dataset.

Chapter 5: Future Work

Although the lane detection model based on SegFormer had a reasonable robustness and accuracy, it is important to keep improving the model until it can be incorporated into a Level 5 autonomous driving system. The future works proposed are as follow:

1. Exploitation of spatial-temporal relationship in lane detection

   In the real world, considering the images are captured at 30 FPS, there are no abrupt changes in the lane lines within a few frames. Hence, multiple continuous frames can be used as input to the feature extraction module. Feature maps of these continuous frames can be passed to a sequence-to-one module such as recurrent neural network or another transformer block to exploit the spatial-temporal relationship between frames and further boost the accuracy and robustness of the lane detection model.

2. Exploitation of lane prior

   Lane markers can always be connected and form a continuous line. Hence, this property can be exploited by calculating the area loss between the predicted lane line after RANSAC fitting and the ground truth lane line. In this case, a supervision signal could be provided by the loss function to the model and reduce or eliminate the discontinuous segmentation of lane line as shown in Figure 54.

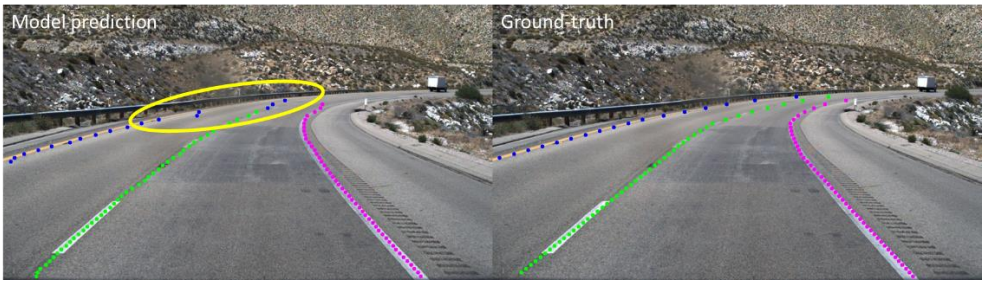

Figure 54: Discontinuity in lanes detected.

3. More high-quality data on challenging road scenarios

   More data should be collected on challenging road scenarios such as dazzle lightning, crowded roads etc. because these road scenarios are more difficult to learn as compared to normal road conditions. This is reflected in the F1-measure of the model as shown in Table 4. Also, a dataset with around 2 to 3 million images from diverse road scenarios should be created to improve the generalisation of the lane detection model. The author has conducted an experiment to evaluate the performance of the model trained on the TuSimple dataset using a separate dataset collected from bus 179. However, the result, as depicted in Figure 55, clearly indicates that the model has simply memorized the data patterns from the TuSimple dataset.

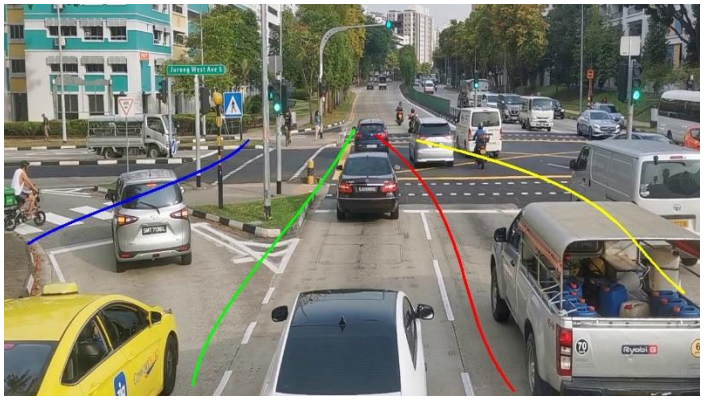

Figure 55: Lane detection result from the model trained on TuSimple dataset.

4. Extension of 4-lane detection model to multi-lane detection model

   Multi-lane detection model should detect all the lane lines in the image given. It is not wise to scale up the number of segmentation map output from the current 4-lane detection model due to the heavy computational load. Also, there are many different types of lane line such as a L-shape lane line at the junction which would require a specific segmentation mask for each of them. Hence, the model should be converted to binary semantic segmentation lane detection model with only 2 segmentation maps as output. One segmentation map learns to segment out the background pixel while another one segments out the lane lines. Integration with localization module would be required to identify the ego lane.

5. Classification of lane markings on the road

   Besides dividing road into multiple lanes, different lane markings also convey different signals to the drivers on the road. The next step of lane detection model would be to classify the lane markings on the road. Classification of lane markings are crucial in path planning. For example, the vehicles should not cross the unbroken double white lines. This would prevent wrong path being planned by the autonomous driving system.

Chapter 6: Conclusion

Lane markers on the road give crucial visual cues to the drivers about their driving path. It helps to keep vehicles on their respective path and maintaining the traffic order on road. To achieve autonomy in driving, lane detection has to be solved because it plays a pivotal role in scene understanding. There have been some applications of lane detection model in real world such as in ADAS system, which is non critical because of the human supervision in the loop. However, when the lane detection model is applied in Level 5 autonomous driving system, it should handle various challenging road conditions robustly and gracefully because human supervision is no longer in the loop. This remains as the bottleneck of the application of autonomous driving in the real world. One mistake of the lane detection model may cost the human lives in the vehicles.

In this project, the comparison between lane detection model based on traditional computer vision techniques and deep learning model had been made. Deep learning model had shown its edge over traditional computer vision techniques in handling illumination changes, occlusions, weak lane marks etc. Also, the possibility of using transformer as feature extraction module in lane detection model was explored. It is important to note that the self-attention layers with sufficient heads can express any convolutional layer. The performance of SegFormer as shown in Table 2 and Table 4 had proven this. It was able to achieve better accuracy than SCNN, the winner of 2017 TuSimple lane detection challenge on TuSimple dataset. Beside accuracy, SegFormer-based lane detection model was able to be deployed in real time application at FPS which is 4 times faster than SCNN. Although the F1-measure of SegFormer was slightly lower than that of SCNN on CULane dataset, SegFormer had doubled inference speed and this justified the trade-off between computational load and F1-measure of the model. Hence, the objective of this project has been achieved, which is to develop a lightweight transformer-based lane detection model that can handle various challenging road scenarios robustly.

Despite the decent performance of the model, the current model performance is still far from being able to be deployed in a Level 5 autonomous driving system. Future works especially exploitation of lane prior should be implemented to further boost the accuracy and robustness of the model under different road conditions. Also, a dataset with around 2 to 3 million images from diverse road scenarios should be created to improve the generalisation of the lane detection model.

Reference

[1] T. Stewart, 'Overview of Motor Vehicle Crashes in 2020', p. 43.

[2] 'AAAFTS_Research-Brief_2022-American-Driving-Survey-FINAL.pdf'. Accessed: Dec. 08, 2022. [Online]. Available: https://newsroom.aaa.com/wp-content/uploads/2022/10/AAAFTS_Research-Brief_2022-American-Driving-Survey-FINAL.pdf

[3] D. Holland-Letz, M. Kässer, B. Kloss, and T. Müller, 'Mobility's future: An investment reality check', Apr. 2021.

[4] A. Bar Hillel, R. Lerner, D. Levi, and G. Raz, 'Recent progress in road and lane detection: a survey', *Machine Vision and Applications*, vol. 25, no. 3, pp. 727–745, Apr. 2014, doi: 10.1007/s00138-011-0404-2.

[5] ONEMOTORING, 'LTA | Driving Rules and Information', *Driving Rules and Information*. https://onemotoring.lta.gov.sg/content/onemotoring/home/driving/road_safety_and_vehicle_rules/driving-rules.html (accessed Dec. 08, 2022).

[6] Q. Huang and J. Liu, 'Practical limitations of lane detection algorithm based on Hough transform in challenging scenarios', *International Journal of Advanced Robotic Systems*, vol. 18, no. 2, p. 17298814211008752, Mar. 2021, doi: 10.1177/17298814211008752.

[7] X. Pan, J. Shi, P. Luo, X. Wang, and X. Tang, 'Spatial As Deep: Spatial CNN for Traffic Scene Understanding'. arXiv, Dec. 17, 2017. Accessed: Dec. 24, 2022. [Online]. Available: http://arxiv.org/abs/1712.06080

[8] L. Tabelini, R. Berriel, T. M. Paixão, C. Badue, A. F. De Souza, and T. Oliveira-Santos, 'PolyLaneNet: Lane Estimation via Deep Polynomial Regression'. arXiv, Jul. 14, 2020. Accessed: Dec. 24, 2022. [Online]. Available: http://arxiv.org/abs/2004.10924

[9] A. Khan, A. Sohail, U. Zahoora, and A. S. Qureshi, 'A Survey of the Recent Architectures of Deep Convolutional Neural Networks', *Artif Intell Rev*, vol. 53, no. 8, pp. 5455–5516, Dec. 2020, doi: 10.1007/s10462-020-09825-6.

[10] Y. Xing *et al.*, 'Advances in Vision-Based Lane Detection: Algorithms, Integration, Assessment, and Perspectives on ACP-Based Parallel Vision', *IEEE/CAA Journal of Automatica Sinica*, vol. 5, no. 3, pp. 645–661, 2018, doi: 10.1109/JAS.2018.7511063.

[11] A. Borkar, M. Hayes, and M. T. Smith, 'Robust lane detection and tracking with ransac and Kalman filter', in *2009 16th IEEE International Conference on Image Processing (ICIP)*, 2009, pp. 3261–3264. doi: 10.1109/ICIP.2009.5413980.

[12] J. Son, H. Yoo, S. Kim, and K. Sohn, 'Real-time illumination invariant lane detection for lane departure warning system', *Expert Systems with Applications*, vol. 42, no. 4, pp. 1816–1824, 2015, doi: https://doi.org/10.1016/j.eswa.2014.10.024.

[13] H. Xu and H. Li, 'Study on a robust approach of lane departure warning algrithm', in *2010 2nd International Conference on Signal Processing Systems*, 2010, pp. V2-201-V2-204. doi: 10.1109/ICSPS.2010.5555281.

[14] A. Borkar, M. Hayes, M. T. Smith, and S. Pankanti, 'A layered approach to robust lane detection at night', in *2009 IEEE Workshop on Computational Intelligence in Vehicles and Vehicular Systems*, 2009, pp. 51–57. doi: 10.1109/CIVVS.2009.4938723.

[15] C.-J. Chen, B.-F. Wu, W.-H. Lin, C.-C. Kao, and Y.-H. Chen, 'Mobile lane departure warning systems', in *2009 IEEE 13th International Symposium on Consumer Electronics*, 2009, pp. 90–93. doi: 10.1109/ISCE.2009.5156928.

[16] Z. Kim, 'Robust Lane Detection and Tracking in Challenging Scenarios', *IEEE Transactions on Intelligent Transportation Systems*, vol. 9, no. 1, pp. 16–26, 2008, doi: 10.1109/TITS.2007.908582.

[17] I. Salian, 'NVIDIA Blog: Supervised Vs. Unsupervised Learning', *NVIDIA Blog*, Aug. 02, 2018. https://blogs.nvidia.com/blog/2018/08/02/supervised-unsupervised-learning/ (accessed Dec. 27, 2022).

[18] M. D. Zeiler and R. Fergus, 'Visualizing and Understanding Convolutional Networks'. arXiv, Nov. 28, 2013. Accessed: Dec. 27, 2022. [Online]. Available: http://arxiv.org/abs/1311.2901

[19] X. Li, J. Li, X. Hu, and J. Yang, 'Line-CNN: End-to-End Traffic Line Detection With Line Proposal Unit', *IEEE Transactions on Intelligent Transportation Systems*, vol. 21, no. 1, pp. 248–258, 2020, doi: 10.1109/TITS.2019.2890870.

[20] J. Han *et al.*, 'Laneformer: Object-aware Row-Column Transformers for Lane Detection'. arXiv, Mar. 18, 2022. Accessed: Dec. 24, 2022. [Online]. Available: http://arxiv.org/abs/2203.09830

[21] Y. LeCun, L. Bottou, Y. Bengio, and P. Ha, ‘Gradient-Based Learning Applied to Document Recognition’, 1998.
[22] Z. Li, F. Liu, W. Yang, S. Peng, and J. Zhou, ‘A Survey of Convolutional Neural Networks: Analysis, Applications, and Prospects’, *IEEE Trans. Neural Netw. Learning Syst.*, vol. 33, no. 12, pp. 6999–7019, Dec. 2022, doi: 10.1109/TNNLS.2021.3084827.
[23] ‘ImageNet Large Scale Visual Recognition Competition 2012 (ILSVRC2012)’. https://image-net.org/challenges/LSVRC/2012/results.html (accessed Dec. 27, 2022).
[24] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. [Online]. Available: http://www.deeplearningbook.org
[25] K. He, X. Zhang, S. Ren, and J. Sun, ‘Deep Residual Learning for Image Recognition’. arXiv, Dec. 10, 2015. Accessed: Aug. 17, 2022. [Online]. Available: http://arxiv.org/abs/1512.03385
[26] ‘CS231n Convolutional Neural Networks for Visual Recognition’. https://cs231n.github.io/convolutional-networks/ (accessed Aug. 17, 2022).
[27] ‘Max Pooling’, *DeepAI*, May 17, 2019. https://deepai.org/machine-learning-glossary-and-terms/max-pooling (accessed Aug. 17, 2022).
[28] A. Vaswani *et al.*, ‘Attention Is All You Need’. arXiv, 2017. doi: 10.48550/ARXIV.1706.03762.
[29] R. Liu, Z. Yuan, T. Liu, and Z. Xiong, ‘End-to-end Lane Shape Prediction with Transformers’, in *2021 IEEE Winter Conference on Applications of Computer Vision (WACV)*, Waikoloa, HI, USA: IEEE, Jan. 2021, pp. 3693–3701. doi: 10.1109/WACV48630.2021.00374.
[30] Jean-Baptiste Cordonnier, Andreas Loukas, and Martin Jaggi, ‘Visualization of Self-Attention Maps’. https://epfml.github.io/attention-cnn/ (accessed Jan. 04, 2023).
[31] E. Xie, W. Wang, Z. Yu, A. Anandkumar, J. M. Alvarez, and P. Luo, ‘SegFormer: Simple and Efficient Design for Semantic Segmentation with Transformers’. arXiv, Oct. 28, 2021. Accessed: Mar. 04, 2023. [Online]. Available: http://arxiv.org/abs/2105.15203
[32] M. Ferguson, R. ak, Y.-T. Lee, and K. Law, ‘Automatic localization of casting defects with convolutional neural networks’, Dec. 2017, pp. 1726–1735. doi: 10.1109/BigData.2017.8258115.
[33] Edward Z. Yang, ‘Convolution Visualizer’. https://ezyang.github.io/convolution-visualizer/ (accessed Mar. 04, 2023).
[34] J. Hu, L. Shen, S. Albanie, G. Sun, and E. Wu, ‘Squeeze-and-Excitation Networks’. arXiv, May 16, 2019. Accessed: Mar. 28, 2023. [Online]. Available: http://arxiv.org/abs/1709.01507
[35] ‘tusimple-benchmark’. https://github.com/TuSimple/tusimple-benchmark (accessed Dec. 25, 2022).

Appendix

Github Repository

https://github.com/Jayden9912/lane_detection.git

Attention map from first self-attention block of transformer block 1 (single head)

The image was divided into multiple blocks of 4 by 4 pixels. There were 9216 such small blocks being transformed into queries, keys and values (Q, K, V) in the first self-attention block. The block with red edges is the query block and zooming in is required for best view.

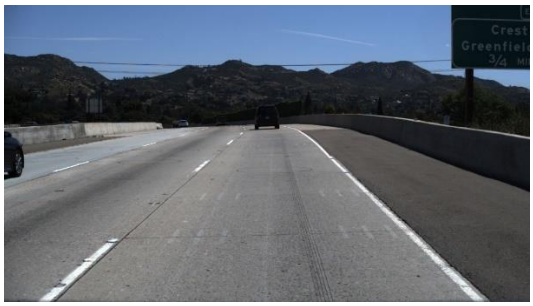

Figure 56: Original image.

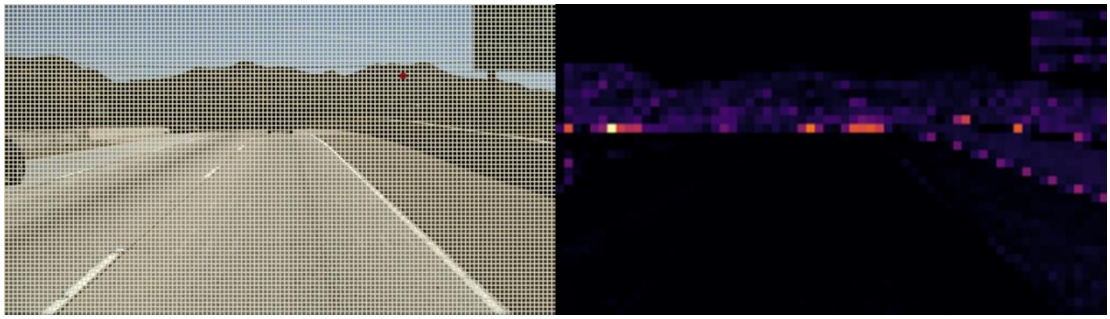

Figure 57: Attention map generated with mountain as query.

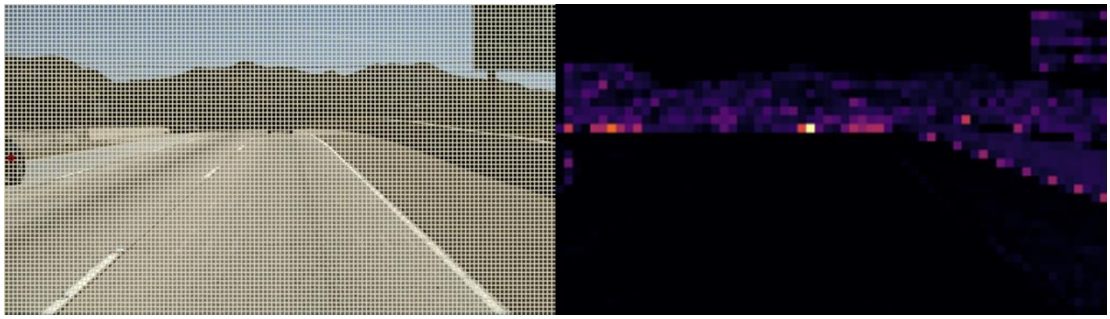

Figure 58: Attention map generated with car as query.

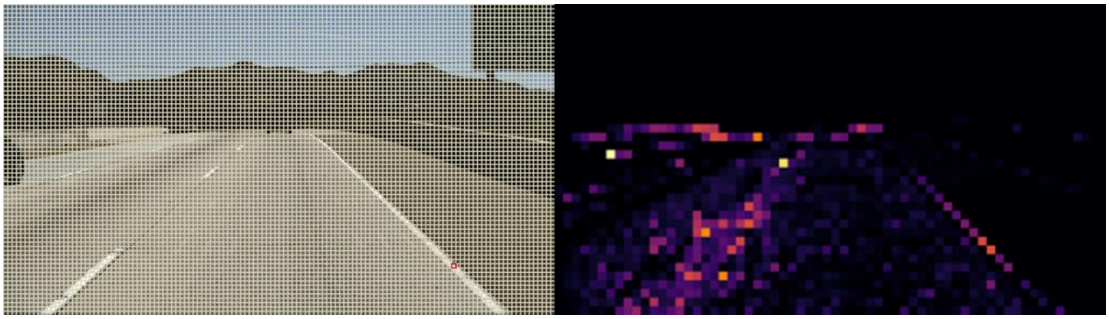

Figure 59: Attention map generated with lane mark as query.

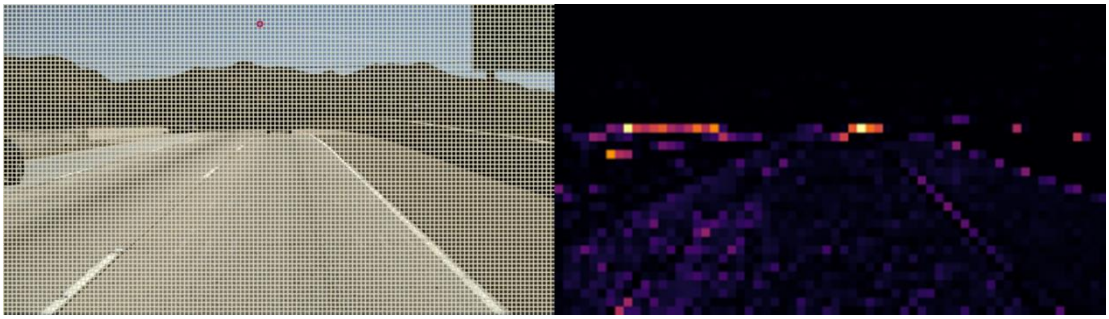

Figure 60: Attention map generated with sky as query.

Attention map from first self-attention block of transformer block 3 (five head)

The image was divided into multiple blocks of 16 by 16 pixels. There were 576 such small blocks being transformed into queries, keys and values (Q, K, V) in the first self-attention block. The block with red edges is the query block and attention maps are arranged in ascending order from the first head to the last head.

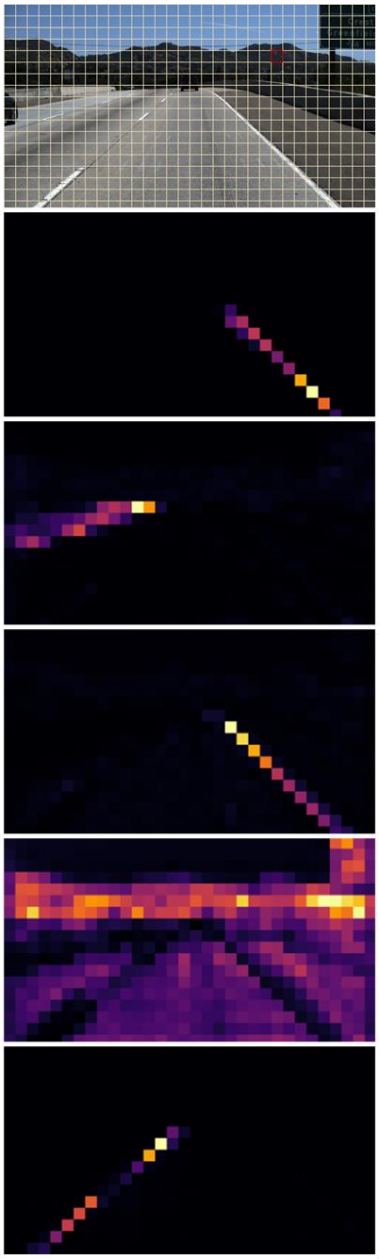

Figure 61: Attention maps from multiple attention heads with mountain as query.

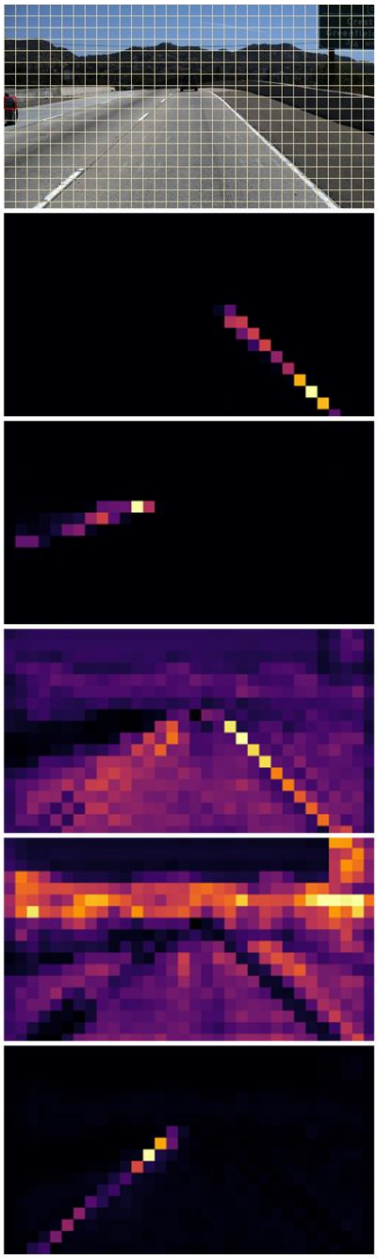

Figure 62: Attention maps from multiple attention heads with car as query.

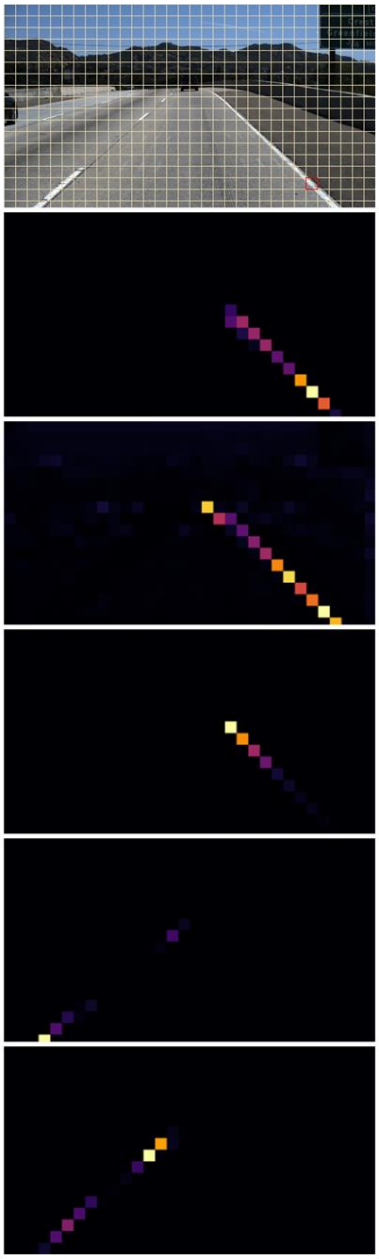

Figure 63: Attention maps from multiple attention heads with lane markers as query.

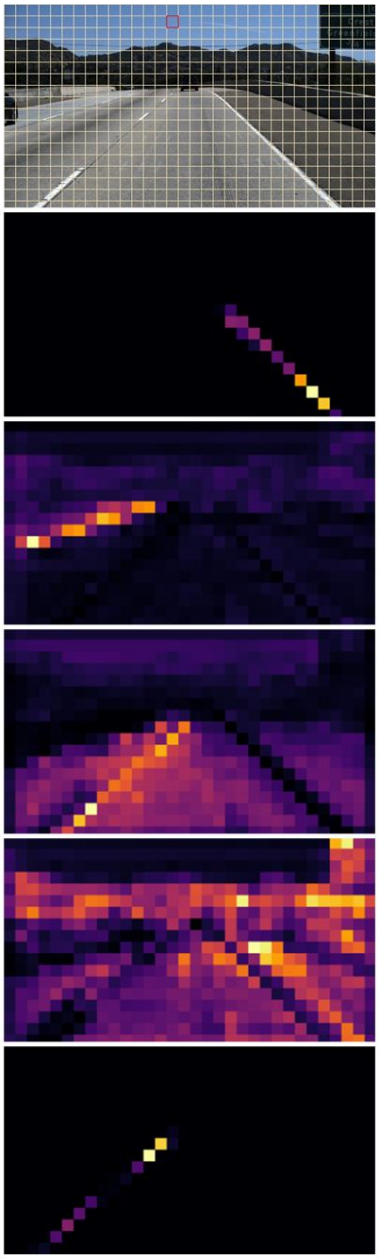

Figure 64: Attention maps from multiple attention heads with sky as query.