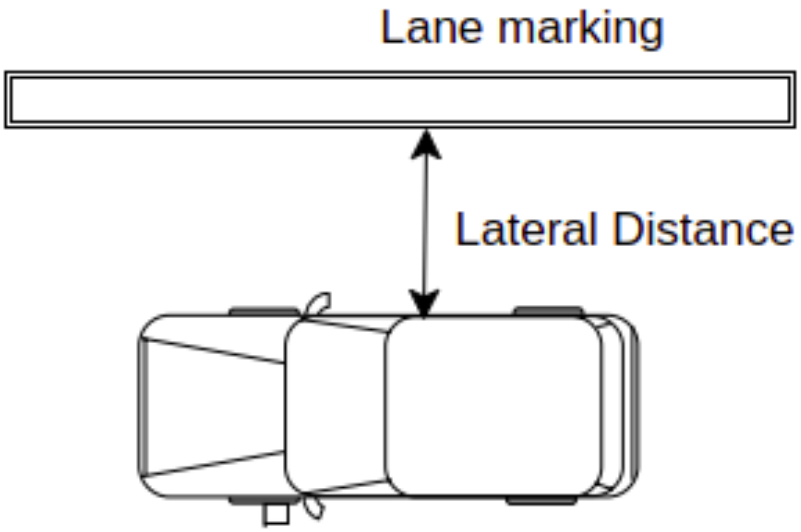


Inference Of Lateral Distance to Vehicle from Road Lane Marking using Data from Side Mounted Camera

Master's thesis in Systems Control and Mechatronics

Supreeth Ramachandra
Dikshit Venkatesh

DEPARTMENT OF ELECTRICAL ENGINEERING

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2022
www.chalmers.se

Master's thesis 2022

Inference Of Lateral Distance to Vehicle from Road Lane Marking using Data from Side Mounted Camera

Supreeth Ramachandra
Dikshit Venkatesh

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Chalmers University of Technology
Gothenburg, Sweden 2022

Inference Of Lateral Distance to Vehicle from Road Lane Marking using Data from Side Mounted Camera and Machine Learning

Supreeth Ramachandra,
Dikshit Venkatesh

Supervisor: Hasith Karunasekera , Electrical Engineering
Examiner: Jonas Sjöberg, Electrical Engineering

Master's Thesis 2022
Department of Electrical Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Lateral distance of the car to road lane marking

Typeset in LaTeX
Gothenburg, Sweden 2022

Inference Of Lateral Distance to Vehicle from Road Lane Marking using Data from Side Mounted Camera and Machine Learning

Supreeth Ramachandra
Dikshit Venkatesh

Department of Electrical Engineering
Division of Automatic Control
Chalmers University of Technology

Abstract

Pilot Assist is an *Advanced Driver Assistance Systems*(ADAS) functionality used in Volvo cars, that helps the driver to keep the vehicle inside a road lane. In this thesis, we propose a method to estimate the lateral distance between the ego vehicle and the road lane, which can be used to evaluate the *Pilot Assist* functionality in test cars. Images from side-mounted vision cameras are used for this purpose. Our approach involves three steps: (a) Detecting the lane marking on the image, (b) Tracking the lane markings and (c) Estimating the distance between the ego vehicle and the detected lane marking.

Three alternative approaches are used to locate the lane markings: to locate the lane as an object, the object detection algorithm *You Only Look Once*(yolo) or semantically segment the lane using an approach based on *Conditional Generative Adversarial Network*(cGAN) and a Semantic Segmentation network based on *UNet* architecture. As the second step, positions of the lane markings are tracked using a Linear Kalman filter in order to estimate their position when the lane markings are missing in certain images in the sequence. In the final step, a look-up table and a neural network approach is used to estimate the distance between the ego vehicle and the lane marking by relating image coordinates to real world distances.

A data set is created to train and evaluate the proposed method by labelling images, which is then extended using various data augmentation strategies. The performance of lane detection algorithms are evaluated using *Dice coefficient*(F1-score), where it was 0.96 for segmentation approach and 0.88 for detection approach on the validation data set. The performance of the Distance estimation software pipeline is evaluated on the test data set, and the average lateral distance error relative to ground truth is found to be approximately 28mm and 32mm when using the segmentation and detection approaches, respectively.

Keywords: Computer Vision, Deep Learning, Generative modelling, Kalman Filter, Advanced Driver Assistance Systems.

Acknowledgements

We would like to extend our appreciation to our supervisor at Volvo Cars, Willy Tunbratt, as well as the manager of the Pilot Assist department, Maja Diho, for allowing us to conduct our thesis at Volvo Cars, providing us with guidance throughout the thesis, and providing us with continuous support throughout the thesis process.

Our deepest gratitude goes out to our advisor at Chalmers, Hasith Karunasekara, for always providing us with valuable information and feedback and helping us reach our goals on this thesis. Your assistance was essential to the completion of this thesis.

Jonas Sjöberg, our Examiner at Chalmers, was extremely helpful in this process. His academic guidance was very helpful.

Lastly, we want to give a great thank you to our families and friends, for supporting us throughout this thesis and the rest of our academic journey

Supreeth Ramachandra, Dikshit Venkatesh, Gothenburg, August 2022

Preface

The work on this thesis was conducted in collaboration between the two authors

During the course of this thesis, Supreeth Ramachandra was the main responsible for implementing lane segmentation node, lane tracking node, and distance estimation node. Linear Kalman filter was designed, tuned and implemented for lane tracking. Lateral distance estimation was done using the in-house tool developed by Volvo cars, to determine lateral distance of vehicle to road lane from the distorted images, multilayer perceptron was trained for distance prediction using pixel coordinates as input.

Dikshit Venkatesh was main responsible for carrying out Lane detection node using pretrained Yolov4-tiny and creating test dataset by annotating the test videos. Simple polynomial function was developed for lateral distance estimation. Final results for comparison were generated using the lane tracking algorithm and lateral distance estimation developed by first author.

Acronyms

.PNG Portable Network Graphics. 16

AD Autonomous Driving. 5
ADAS Advanced Driver-Assistance Systems. 1, 5
ANNs Artificial Neural Networks. 6
AP Average Precision. 2, 15, 21, 22, 33–35

BCE Binary crossentropy. 2, 21
BDD Berkeley Deep Drive. 16

CBL Convolutional Base Layer. 21
cGAN Conditional Generative adversarial network. xv, 2, 10–12, 19, 22, 41–43, 46, 47
CIoU Complete IoU Loss. 21
CNN Convolutional Neural Network. 6, 10, 21
CSP Cross Stage Partial. 21
CV Computer Vision. 5, 6, 9
CVAT Computer Vision Annotation Tool. 16

DCGAN Deep Convolution GAN. 25
DGNSS Differential Global Navigation Satellite System. 1
DGPS Differential Global Positioning System. 1
DL Deep Learning. 5, 6

ED Edge Drawing. 5

FC Fully connected Layers. 7
FN False Negatives. 14
FP False Positive. 14, 27, 47
FPN Feature Pyramid Network. 21

GAN Generative adversarial network. xv, 6, 10–12
GPS Global Positioning System. 1

HSV Hue, saturation and value. 5

IMU Inertial Measurement Unit. 1
IOU Intersection Over Union. xv, 2, 8, 15, 21, 34, 35, 37

JSON JavaScript Object Notation. 16

mAE mean Absolute Error. 15, 42, 43, 46

NLP Natural Language Processing. 6

R-CNN Region-based Convolutional Neural Network. 8, 47
RANSAC Randon Sample Consensus. 6
ReLU Rectified Linear Unit. 7, 21, 23, 31
RGB Red, Green and Blue. 5, 16, 22
ROTAC Real World Objective Testing of Autonomous Cars. 30, 47

SLAM Simultaneous Localization and Mapping. 6
SSD Single Shot Detector. 2, 8, 47

TP True Negatives. 14
TP True Positives. 14

YOLO You Only Look Once. xiii–xv, xvii, 2, 8, 9, 19–23, 33–35, 40–43, 46, 47
YOLOv4 You Only Look Once version4. 9
YOLOv7 You Only Look Once version7. 9, 19
YUV Luminance, blue and red. 5

Contents

List of Figures

List of Tables

1
Introduction

Pilot Assist functionality in Volvo Cars aids the driver in maintaining the vehicle within the lane, as long as the lane markings are visible and vehicle speed is less than the recommended speed. Volvo cars uses an in-house tool to measure and grade the performance of the Pilot Assist functionality[1]. In its current form, in-house Advanced Driver-Assistance Systems (ADAS) grading tool is equipped with four RGB cameras and Differential Global Navigation Satellite System (DGNSS) aided Inertial Measurement Unit (IMU).

Differential Global Positioning System (DGPS) based ADAS validation tool is affected by various errors common to the Global Positioning System (GPS) technology, where GPS positioning accuracy is greatly impacted by the delay in signal reception. The delay is due to the multipath effect where the signal is reflected multiple times before reaching the GPS receiver. Furthermore, satellite signal may be blocked due to high rise buildings, tunnels and trees.

In contrast to DGPS, Cameras offer advantages such as rich contextual information and uninterrupted data availability. Furthermore, since camera is a passive sensor, it is not affected by external signals that may be interrupted by the environment. As a result, image data obtained from the camera is used to estimate the ego vehicle's lateral distance from the road lane as shown in the Figure 1.1. However, when the scene is impacted by factors such as the absence of clear lane markings, poor visibility due to adverse weather conditions, lane marking occlusions, and poor illumination, the quality of the information captured by the camera is adversely affected.

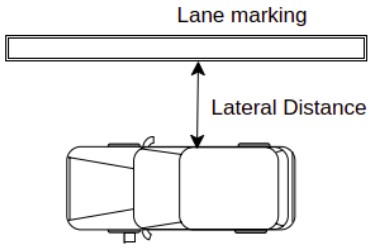


Figure 1.1: Lateral Distance from ego vehicle to road lane

[1]https://www.volvocars.com/se/support/topics/anvanda-din-bil/bra-att-veta/pilot-assist-och-korfaltsassistans

1.1 Objective

Primary goal in this thesis is to calculate the lateral distance from the ego vehicle to the lane lines on the road as shown in the Figure 1.1. The data from side-mounted camera as shown in Figure 1.2 is utilized to accomplish this. The main goal is divided to three objectives;

Figure 1.2: Volvo Cars Setup for data collection. Image courtesy: Volvo Cars

Objective 1: Lane detection The objective is to detect road lane line in the visual data obtained from the side mounted camera. Two strategies are implemented in order to achieve this objective. One method using YOLO object detection network, by treating road lane marking as an object, while the other method explores Semantic Segmentation using generative methods like cGAN[1].

i. Detecting Lane Marking as Object Single stage object detectors like YOLO[2] and Single Shot Detector (SSD) [3] detect multiple objects in an image by predicting confidence scores as well as bounding boxes in single evaluation. Such models can be used to locate the lane as a bounding box covering the lane segment. These boxes contain the location of the lane in image co-ordinates. The accuracy of these detectors can be evaluated by Average Precision (AP), IOU and *F1-score*.

ii. Lane Segmentation: Given an image sequence as input, the lane segmentation network will output a segmentation mask corresponding to the lane markers in the image. A semantic segmentation network based on U-NET [4] architecture is designed and trained in Adversarial manner[5]. Network is trained using Binary crossentropy (BCE) WithLogitsLoss function. *Pixel loss*, *F1-score*, and IOU are used to evaluate the Lane segmentation network on validation data for every epoch during training.

Objective 2: Lane Tracking The dataset contains images of continuous, discontinuous, and blurry lanes. A tracking method is implemented in order to determine the location of lanes in scenarios where the lane marking is occluded or has completely worn away. The linear Kalman filter is modeled and tuned to track the lanes. The lane tracking node is evaluated qualitatively in relation to ground truth.

Objective 3: Lateral Distance Estimation The final objective is to determine the distance between the vehicle and the lane in 3D world coordinates. The output from objective 2 will contain the estimated x and y coordinates of the road lane lines in 2D coordinates. The look-up table developed by Volvo Cars is used to estimate the distance from the lane to the vehicle. The purpose of this table is to convert the image/pixel coordinates into the corresponding world coordinates in millimeters. Vanilla Neural network is trained to fit the polynomial to the parameters represented in look-up table. Lateral distance is estimated using interpolation or extrapolation from the polynomial fitted to the data. The model is evaluated based on the average lateral distance error between the ground truth and the model prediction.

1.2 Limitations

The following factors restrict the scope of the master's thesis:

- Data from the Volvo dataset is limited to images taken during the day. It covers scenarios such as illumination and shadow. Real-world lane images taken in rain or snow are not taken into account.

- Aside from the lane lines captured from the side-mounted camera, lane selection arrows, change of lane arrows, and other stops or parking markings are not considered during this study.

- Estimation of lateral distance to road lane on the snow covered roads will not be explored under this thesis.

- Real-time inference from the proposed software pipeline is not considered, so the collected data should be processed off-line.

1.3 Outline

The thesis has been divided into following chapters:

- In chapter 2, the literature on lane detection is reviewed and then the theoretical aspects relevant to the thesis are presented.
- Chapter 3 describes the setup for data collection, labelling methods and process for creation of synthetic data.
- In chapter 4, the hardware and software tools used to develop and test the model are described in detail. Also the network architectures used for lane detection, algorithm for tracking of missing lanes and distance estimation in world co-ordinates are explained.
- Chapter 5 examines the thesis outcome based on the objectives in section 1.1 and evaluates the implementation results.
- In chapter 6, the summary and areas for further research are presented.

2
Theory

The theory used in this thesis is described in this section. In Section 2.1 related works pertaining to Lane detection using Classical Computer vision and Deep learning is explained. Theory behind Lane tracking and Evaluation criteria is explained in Sections 2.2 and 2.3.

2.1 Related work

Robust Lane detection is very important for maintaining the integrity of Autonomous Driving (AD) and ADAS features. The camera is the widely used sensor for lane detection [6]. Many methods have been proposed for lane detection based on Computer Vision (CV) and Deep Learning (DL) approaches.

2.1.1 Traditional Computer Vision based approach

CV-based methods are classified into two categories model-based and feature-based. Model-based methods[7], [8] use mathematical models to describe lane structure. As compared to feature-based methods, model-based methods are more robust; they are difficult to model and will always involve model-related uncertainties, as well as being computationally intensive. [7], [8].

Features-based approach of Lane detection, uses color information in the images [9], [10]. In preprocessing step of features-based lane detection, the input image in Red, Green and Blue (RGB) colorspace is converted to grey space, Hue, saturation and value (HSV) or Luminance, blue and red (YUV) colorspace. Major reason for this is because RGB colorspace is affected by the intensity of the ambient lighting conditions[11]. Color of the road lanes is not always consistent, sometimes we encounter yellow lane lines, converting the image [12] from RGB to HSV colorspace is found to improve the accuracy of detection of yellow lane lines. Edge detectors like Canny edge detector[13] is then applied to compute edge map. For computing edge maps, Edge Drawing (ED)lines [14] are found to be more efficient in terms of computation as well as more accurate than the Canny Edge Detector. Then Hough transform is applied to extract the lines that contain certain number of edge lines. Hough transform can only be applied in case of straight lines. Spline fitting[15] is used in case of curved lines. A curvier road will require more control points to fit the spline, increasing the computational complexity of the problem.

2.1.2 Deep Learning based approach

In contrast to traditional CV-based techniques, DL approach outperform them in terms of accuracy[16] and generalization in applications such as semantic segmentation, image classification, object detection, and Simultaneous Localization and Mapping (SLAM).

Authors in [17] have tried to combine benefits of both the DL and CV in lane detection problem. When the road scene is clear, image is denoised using Gaussian Blur to reduce the effect of illumination variations on the lane detection accuracy of the algorithm. Edge image is calculated using convolution of denoised image with a hat shaped kernel. Randon Sample Consensus (RANSAC) is used to detect straight line candidates for lanes. When road scene is complicated with road lanes occluded, shadows from the trees, intersections, Convolutional Neural Network (CNN) is used before RANSAC to improve edge detection output. This method has found to increase the lane detection accuracy of the algorithm, despite achieving high accuracy, since these methods rely heavily on semantics of the scene and pixel based classification, performance of the model is found to be highly variable in the situations where there is high variations of illumination.

In [18] authors have used Generator-Discriminator based architecture for generating semantic segmentation masks of the road lanes. Given the implicit data distribution is already modeled by Generator, Generative models[1] are able to accurately regenerate lane pixels from the highly noisy background. Another interesting work [19] leverages GAN[20] ability to make the semantic segmentation network output to be more realistic or better structure-preserving. Issues of illumination, lane line occlusions because of road wear and tear, bears minimum effect on generalizing ability of the Generative models[18]. .

For Lane Detection, Deep Learning based algorithms are chosen over feature-based algorithms due to their performance in conditions such as, varying illumination and lane marking occlusions.

2.1.3 Convolutional Neural Networks

CNN are a class of Artificial Neural Networks (ANNs) which are mainly used in analyzing visual information. They are also used in image classification, image segmentation, signal processing, Natural Language Processing (NLP), speech recognition and auto correlation of data.

The LeNet architecture, which Yann LeCun designed in 1998, is the most well-known CNN implementation. It was mainly designed for the handwritten digit recognition problem [21]. Figure 2.1 illustrates the LeNet structure.

[1] https://openai.com/blog/generative-models/

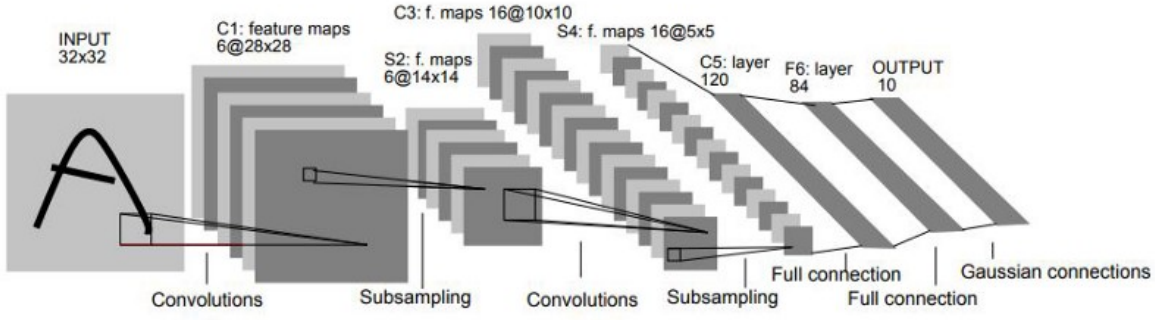


Figure 2.1: LeNet architecture

The general CNN architecture consists of convolutional layers, pooling layers and fully-connected layers. Convolutional layers: This layer extracts the important features from the input image. It uses specialized mathematical operation called convolution also called kernel or filter instead of normal matrix multiplication. Y=x*w. here, Y is the output, input x and weight w. The kernel of specific size M x M slides over the complete image. The output obtained is called feature map which is then passed on to next layer to learn all the other features.

Pooling Layers: This layer is used to decrease the dimensionality of each feature map. Computational costs are reduced by doing this. This in turn reduces the noise and variations while applying filters on the image. There are different pooling methods namely, max pooling, average pooling, Sum pooling. In max polling, the maximum element is selected from the feature map. In Average pooling, the average of elements in an image section is taken. In Sum pooling, the sum of all the elements in a section is considered.

Fully connected Layers (FC): These are the last layer in CNN which connects the neurons of previous layers with neurons of next layers. The high-level features obtained from the convolutional and pooling layers are flattened and provided as input to FC. The FC contains weights and biases of the neurons which uses features and classifies the image into different classes.

Dropout: This layer prevents the neural network from overfitting and improve the performance of the model. Overfitting happens when a model learns in detail and also the noise in the training data and does not generalize well. This is avoided by neglecting some neurons during the training process.

Activation functions: It is one of the important parameters in the model. It controls the flow of information from one layer to another. The network gains non-linearity through activation functions. Commonly used activation functions are sigmoid, Rectified Linear Unit (ReLU). Softmax, tanH. The decision to activate or not activate a neuron is made by activation functions.

2.1.3.1 Detecting Lane as an Object

Object detection is a computer vision approach for detecting and localizing the objects of interest in images or videos. The initial step of the task is to determine the lane's coordinates in the image if present. Object detection can be divided into two types: single stage and two-stage methods. Single stage methods such as YOLO and SSD object detection as a regression problem by taking image as input and outputs the class probabilities and bounding box co-ordinates. On the other hand, two stage methods like Faster Region-based Convolutional Neural Network (R-CNN) and Mask R-CNN, first produce region of interest (ROI) using region proposal network and then perform classification and bounding box regression. [22]

YOLO is one of the single stage object detectors which is extensively used in real-time applications. It was first proposed by Joseph Redmon and his team [2]. In the paper, object detection is formulated as regression problem to spatially separated bounding boxes and associated class probabilities. Prior to YOLO, detection was framed as classification problem.

The architecture of YOLO is shown in the figure: The network has a total of 26 layers with two fully connected layers and the remaining are convolutional layers. The features from the image are extracted from the initial convolutional layers. The bounding boxes and class probabilities are the output from the final fully connected layers. For example: In the first part, lane region in the image is detected, and the second part is classification of the detected lane.

The input to YOLO can be single image or video (series of images). In single evaluation, it predicts bounding boxes and class probabilities for each region in an image. YOLO detector is very fast compared to other object detectors like R-CNN. R-CNN utilizes selective search to create many region proposals for each image. But YOLO scans the entire image at once and gives the predictions. It also creates less background errors which is seen in Fast R-CNN [2]. There are two parts to the model evaluation process.

YOLO generates n number of SxS grid cells from the input image as shown in the Figure 2.3. The data from each grid cell size is extracted. The bounding boxes (B) and confidence scores are then predicted which is obtained after detecting the object of interest. The confidence score is defined as intersection of union between the prediction and ground truth. If the value is zero, it signifies that the object of interest is not present in that cell. If the IOU is equal to 1, then predicted bounding box is the same as the ground truth box. Therefore, each cell in an image outputs a score and corresponding bounding boxes. The final predictions are encoded as an S x S x (B * 5 + C) tensor where C represent class probabilities. C value is just one for present dataset i.e. lane object. Large data set like PASCAL VOC has 20 labelled classes [23], then C = 20. The bounding box consists of five values: [x_{center}, y_{center}, w, h] and confidence score. The x_{center} and y_{center} are the centre coordinates of the rectangular box. The w and h are width and height of the box relative to the whole image. Over the years, several improvements are made on YOLO, to increase

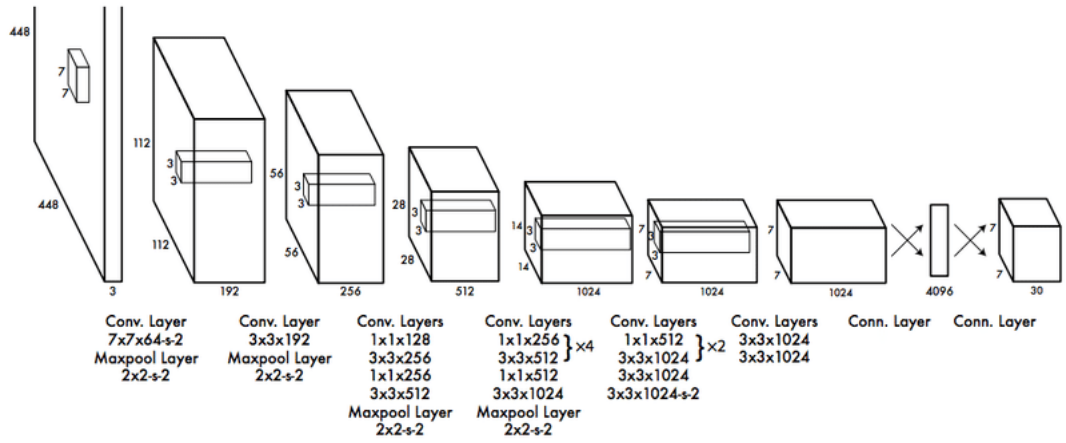


Figure 2.2: Architecture of YOLO

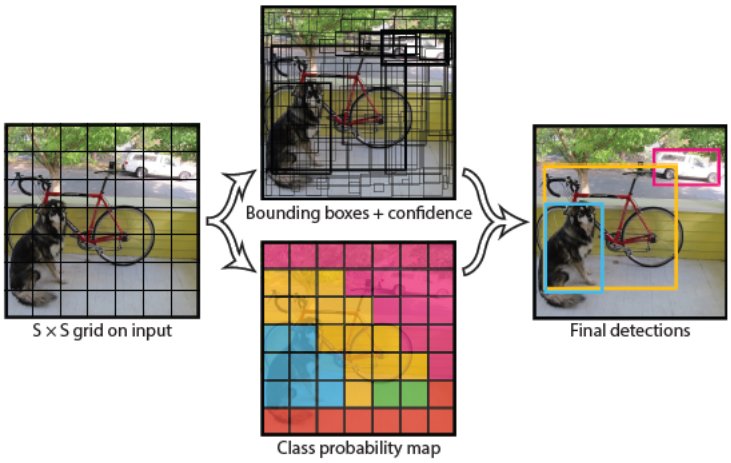


Figure 2.3: Output from YOLO

the accuracy of detection and inference speed. YOLO has been officially released in five versions. You Only Look Once version4 (YOLOv4) [24] was regarded as an optimal and accurate object detector in 2020. But in 2022, You Only Look Once version7 (YOLOv7) [25] surpasses all the other object detectors in both speed and accuracy.

2.1.3.2 Semantic Segmentation

The purpose of the Semantic Segmentation network in CV is to classify every pixel in an image into a predefined class. As a result of semantic segmentation, multiple objects of the same class are treated as single instance [26]. In the scope of the lane detection node, the region of interest is the pixel area corresponding to the road lane marking that should be classified as a single entity by the network. Semantic segmentation networks such as Fully Convolution Network [6], Pyramid Scene Parsing Network [27], DeepLab [28], and U-Net [29] are used for lane segmentation. But, these approaches are purely discriminative and relies on pixel intensity variations

and location semantics to segment lanes. This results in a high level of susceptibility to illumination and occlusions [18]. To address the issue of network inaccuracies when, the scene in the image is contaminated with varying illumination, occulsions resulted from road wear and tear and sensor degradation. It has been been found that Generating the lane markings instead of segmenting the pixels will help address these issues [18].

Generative network, models the distribution of the training data. It does so by mapping the input data to an unknown data distribution, and aim of the network is to generate data for any given sample of the distribution. That is they are capable of learning implicit probability density function from the given set of data. GAN was first introduced by Goodfellow et al.[20]. General architecture of GAN is as shown in the Figure 2.4 A GAN is a type of deep learning network that can generate data with similar characteristics as the input training data.

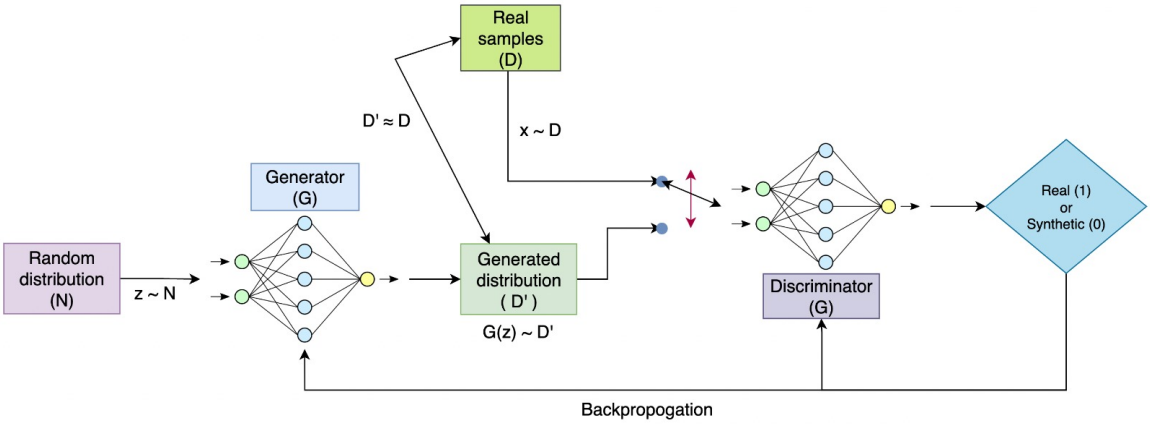


Figure 2.4: Architecture of GAN

GAN consists of two networks, Generator CNN and a Discriminator CNN as shown in Figure 2.4. With random noise as input, generator outputs data similar to the probability distribution of the training samples. Discriminator takes both the data generated from generator and the training data and classifies the observation as real or generated.

The cGAN is also a type of generative network, which unlike GAN uses labels during the training process. cGAN is also comprised of Generator and Discriminator similar to GAN. Typical architecture of cGAN is as shown in the Figure 2.5. The difference is, while training cGAN along with the images, labels are also provided as an input.

GAN outputs random image which is not available in the training data set. But, for lane segmentation, rather than generating samples randomly, we would need to generate samples from a class in a given context. Since the model needs to generate lane markings, cGAN model is best suited for this purpose.

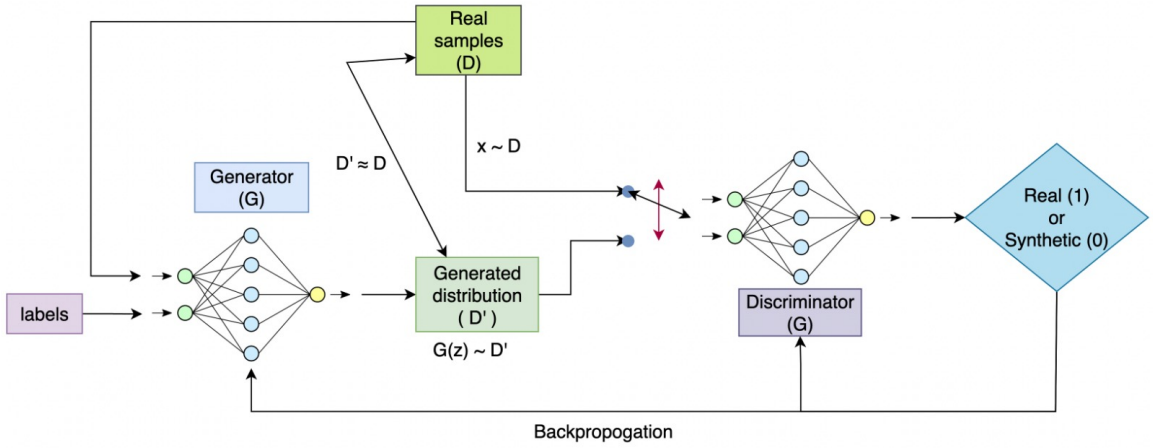


Figure 2.5: Architecture of cGAN

Loss Function:
The discriminator determines whether data sampled from the training data set, true data sample is "real" or "1," and whether data generated by the generator, generated data sample is "fake" or "0". Each loss corresponding to the missclassification of real samples, the Discriminator loss, and the loss corresponding to generated samples, the Generator loss, must be calculated to determine the discriminator's loss. Losses of the discriminator are calculated by adding the two losses above. The objective of the Loss function is to minimize the missclassification of data that has been sampled from the training data set and data that has been generated by the generator. Loss function of the GAN is also known as min-max loss [5] as described in equation 2.1.

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}\,(\boldsymbol{x})}[\log D(\boldsymbol{x}) + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))] \tag{2.1}$$

For cGAN The loss function in the equation 2.1 is modified as described in the equation 2.2, where samples(x) in the data generated by the Generator is conditioned on the samples(y) from training data set.

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}\,(\boldsymbol{x})}[\log D(\boldsymbol{x} \mid \boldsymbol{y})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z} \mid \boldsymbol{y})))] \tag{2.2}$$

i.Discriminator Loss
During training of the GAN, discriminator classifies both the true data sample and the generated data sample. Discriminator is penalized for misclassification of true samples(1) as generated sample(0) and vice versa, by maximizing the function described in equation 2.3.

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right)\right] \tag{2.3}$$

Where, log D(x) refers to probability that the Discriminator is rightly classifying the data sampled from training dataset as real image(1). log (1- D(G(z))) refers to probability that the Discriminator is rightly classifying the sample generated by the Generator as fake image(0).

ii. Generator Loss:
During training of the GAN, generator produces the sample data set whose distribution will be similar to the training dataset with the random noise as input to the Generator network. The generated samples then propogates through Discriminator, which in turn classifies the input as either “real” (1) or “fake” (0) based on the training progress of the Discriminator to distinguish between generated sample and the data sampled from the training data set. Generator then gets rewarded if Discriminator misclassifies the Generator output as "Real(1)" and gets penalized otherwise.

Training:
Training of the GAN and cGAN involves the following steps:

- Discriminator is trained using the data sampled from training data set to correctly classify them as "Real(1)".
- Generated samples from the Generator is used to train the Discriminator to correctly classify them as "Fake(0)".
- Once Discriminator is trained on true samples and generated sample, using the predictions from the Discriminator, Generator is trained with an objective to fool the Discriminator into classifying generated samples as "Real(1)".
- Generator and Discriminator network is trained alternatively, when the Generator network is being trained, weights of the Discriminator network is frozen and vice-versa.

2.2 Lane Tracking

Upon robust lane detection, tracking the position of lane markers is the next crucial step where road lane markers may be missing in some image frames. In this thesis, the Kalman filter is utilized to track the lanes when road lane markers are missing from the image frames.

2.2.1 Kalman Filter

The Kalman filter is one of the most widely used recursive Bayes filtering techniques[30]. The Kalman filter estimates the joint probability distribution of the unknown variables from the noisy measurements obtained by the sensor over time, based on the measurement and state estimation through the current time instance k. Essentially, there are two approaches to using the Kalman filter. The Filtering approach uses measurements of the unknown variable taken from the sensors until the current time k to estimate its current state. In the prediction approach measurements until the current time instance k is considered to predict the state of the unknown variable in the next time instance $k + 1$.

A Kalman filter is composed of two steps as shown in Figure 2.6, these steps are executed one after the other in a recursive manner, the prediction step and the measurement step. As part of the prediction step, the Kalman filter estimates the current state of the unknown variable using the measurements and state estimates from the previous time interval $k-1$. At the update step, when the measurement of the unknown variable arrives at the current time instance k, state estimations from the prediction step are updated to include the information regarding the state of the unknown variable. When the prediction step is performed at the next time instance $k+1$, the output from the update step at time k will be the new input.

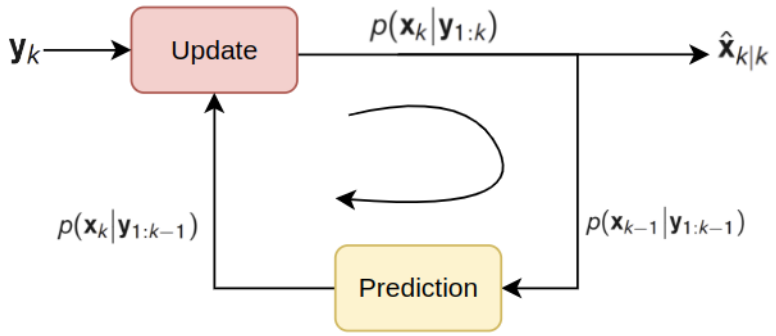


Figure 2.6: Steps involved in Kalman Filter

Initialization: The Linear Kalman filter assumes that unknown variables whose state transformation is being tracked are random and uniformly distributed. The state of the unknown variable and its uncertainty are set to a random value before the first measurement is taken. In the absence of any information regarding the state of an unknown variable, the initialized state uncertainty will be large, and Kalman filter will take more iterations to converge to the true state of the variable.

Prediction: In the prediction step, posterior density of the unknown variable estimated during update step from the previous time instance given measurements until the previous time instance $p\left(\mathbf{x}_{k-1} \mid \mathbf{y}_{1:k-1}\right)$ is combined with the motion model $p\left(\mathbf{x}_k \mid \mathbf{x}_{k-1}\right)$ is used to predict the state of the unknown variable at the current time instance $p\left(\mathbf{x}_k \mid \mathbf{Y}_{1:k-1}\right) = \mathcal{N}\left(\mathbf{x}_k; \hat{\mathbf{x}}_{k|k-1}, \mathbf{P}_{k|k-1}\right)$ as shown in equation 2.5. Motion model is a mathematical model representing state transformation of the unknown variable as shown in equation 2.4,

$$\mathbf{x}_k = \mathbf{A}_{k-1}\mathbf{x}_{k-1} + \mathbf{q}_{k-1}, \tag{2.4}$$

Where, A_{k-1} is the state transition matrix, and q_{k-1} is the process noise in vector form which has the same dimension as the state vector. It has zero mean and covariance Q_{k-1}.

$$p\left(x_k \mid y_{1:k-1}\right) = \int p\left(x_k \mid x_{k-1}\right) p\left(x_{k-1} \mid y_{1:k-1}\right) dx_{k-1} \tag{2.5}$$

And, the state covariance $P_{k|k-1}$ is calculated using the equation 2.6, where $P_{k-1|k-1}$ is the state covariance calculated during the update step at the previous time in-

stance

$$\mathbf{P}_{k|k-1} = \mathbf{A}_{k-1}\mathbf{P}_{k-1|k-1}\mathbf{A}_{k-1}^{T} + \mathbf{Q}_{k-1} \tag{2.6}$$

Update: In the update step, a posterior estimate of the state of the unknown variable $p(\mathbf{x}_k \mid \mathbf{Y}_{1:k}) = \mathcal{N}\left(\mathbf{x}_k; \hat{\mathbf{x}}_{k|k}, \mathbf{P}_{k|k}\right)$ is calculated by combining the outcome of the prediction step with the measurements Y_k obtained at the current time instance. Where, posterior state estimate $\hat{\mathbf{x}}_{k|k}$ is calculated using the equation 2.7

$$\hat{\mathbf{x}}_{k|k} = \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k\mathbf{v}_k \tag{2.7}$$

Where, the Kalman gain K_k based on the previous error estimates decides, which of either the estimate or measurements to give more weightage to, when calculating posterior distribution of the unknown variable at each time instance.

v_k called as innovation factor is calculated using the equation 2.8. It captures difference in actual measurement and estimated observation from the measurement model $p(\mathbf{x}_k \mid \mathbf{y}_k)$ as shown in the equation 2.9

$$\mathbf{v}_k = \mathbf{y}_k - \mathbf{H}_k\hat{\mathbf{x}}_{k|k-1} \tag{2.8}$$

$$\mathbf{y}_k = \mathbf{H}_k\mathbf{x}_k + \mathbf{r}_k, \tag{2.9}$$

Where H_k is the matrix relating state of the unknown variable with the observation from the sensor, and r_k represents normally distributes sensor noise $\mathbf{r}_k \sim \mathcal{N}(0, \mathbf{R}_k)$ with covariance R_k. When both the motion model represented in equation 2.4 and measurement model represented in equation 2.9 used in the Kalman filter are linear, then the corresponding variant of the Kalman filter is called as Linear Kalman filter.

2.3 Evaluation of model

The various evaluation metrics used to assess model performance are briefly described in this section. The most popular classification metric is **Accuracy**. It determines the number of correct predictions made by the model. The formula is given by:

$$Accuracy = \frac{Number\ of\ correct\ predictions}{Total\ number\ of\ predictions} \tag{2.10}$$

Confusion matrix is a matrix which comprises the number of True Positives (TP), True Negatives (TP), False Positive (FP), False Negatives (FN) for each class. It does not evaluate the model's performance. The performance metrics like Precision and Recall is calculated from confusion matrix. **Precision** evaluates the model performance based on number predictions which are true.[31]

$$Precision = \frac{TP}{TP\ +\ FP} \tag{2.11}$$

Recall is another important metric which determines the number of actual positives which are identified correctly.[31]

$$Recall = \frac{TP}{TP \ + \ FN} \tag{2.12}$$

Another metric which determines the model performance is the ***F1-score***. It is harmonic mean of precision and recall ranging from 0 to 1. *F1-score* 1 represents that the model perfectly classifies the prediction to the correct class and 0 represents that model is unable to classify to correct class. Higher the *F1-score*, the better is the performance of the model.

$$F1 = \frac{2 * Precision * Recall}{Precision \ + \ Recall} \tag{2.13}$$

IOU is another important evaluation metric used to measure the accuracy of an object detector on a particular data set. It is calculated by dividing the area of overlap by the area of union between the ground truth bounding box and predicted bounding box. AP is average value of the points on Precision-Recall curve for each possible threshold for the same class. AP at fixed IOU, such as IOU=0.5 and IOU=0.75 is represented as AP50 and AP75 respectively.

mean Absolute Error (mAE), popular regression metric is the mean difference between the Ground truth and the Predicted Values. It signifies the deviation spread of prediction values from the real values. It is always a positive value.

$$Mean\ Absolute\ Error = \frac{1}{N}\sum_{j=1}^{N}|y_j - \hat{y_j}| \tag{2.14}$$

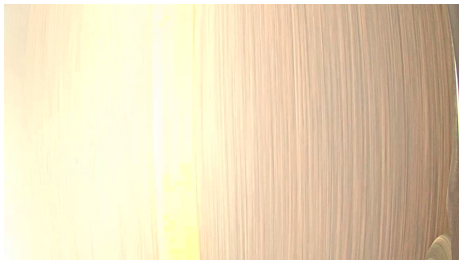
3
Dataset

Section 3.1 of this chapter starts with a brief description on various publicly available dataset and experimental setup of the Volvo Cars to collect the data, data annotation format and the tool used to annotate the data. Section 3.2 is used to explain the importance of data augmentation and augmentation policies used in this research work.

3.1 Data Collection setup

Lane detection is one of the most active research topics in autonomous driving. In order to assess and train state-of-the-art lane detection algorithms, datasets are continuously updated. Datasets like TuSimple [32] , CULane [33], Berkeley Deep Drive (BDD)100K [34], and Apolloscape [35] are available as open-source. The datasets, however, all make use of front-facing cameras to collect the data. As a result, they serve as a means of estimating the location of the lane ahead of the vehicle. Since, the purpose is to estimate the distance between the vehicle and the lane marking at the current time instance, Volvo Cars had proposed mounting the cameras laterally and facing downward as represented in Figure 1.2 since the side-mounted camera will not be affected by dynamic objects acting as a noise source, when lane markings are the only object of interest. Also lane position can be predicted more accurately due to its proximity to the camera. By excluding most of the visual clutter that is normally collected by front-facing cameras, we are able to focus exclusively on the lane markings.

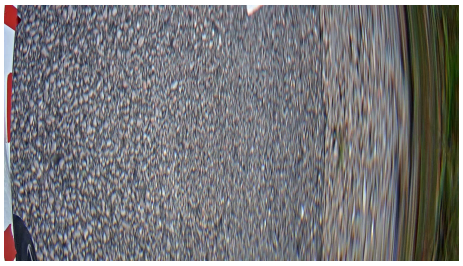
Data is collected in the form of videos, each video is approximately five minutes long. Dataset consisted of around 3,100 images, annotated for object detection and semantic segmentation tasks. Images are in RGB colorspace and they are of the size 1280x720 and Portable Network Graphics (.PNG) format. Dataset is covering the following scenarios, illuminated lane marker 3.1a, shadow on lane marker 3.1b, worn-out lane markers 3.1c and missing lane markers 3.1d. Dataset annotations is extracted in COCO format [36] using the open-source tool Computer Vision Annotation Tool (CVAT) [37]. The JavaScript Object Notation (JSON) file is generated by CVAT, contained pixel co-ordinates corresponding to the road lane line marking, in the format useful for both Object detection network and Semantic Segmentation network.



(a) Illuminated lane

(b) Shadow over lane

(c) Worn out lane



(d) Missing lane

Figure 3.1: Dataset with different scenarios

3.2 Synthetic Data Augmentation:

It was necessary to augment the dataset to add scenarios such as a light snow-covered road, sensor blur, night, and overexposed image frame due to reflections from the road surface. To achieve this, open-source image augmentation libraries [1] and method in [38] are used.

To simulate night and illumination conditions, we have used *Add*, and *Multiply* functions from *Vidaug*1 open source data augmentation library. In *Add* and *Multiply* function, the intensity of the pixels in an input image is multiplied or added by a constant value to whiten or blacken the image while keeping the sum below 255 and above 0, per pixel per color channel. This is to simulate night and very bright ambient lighting conditions. *Elastic Transformation* is used to simulate sensor blurring during heavy rain. Elastic transformation, a form of affine transformation to randomly move the pixels around inside the fixed grid, this simulates sensor blurring effect [39].

[1] https://github.com/okankop/vidaug

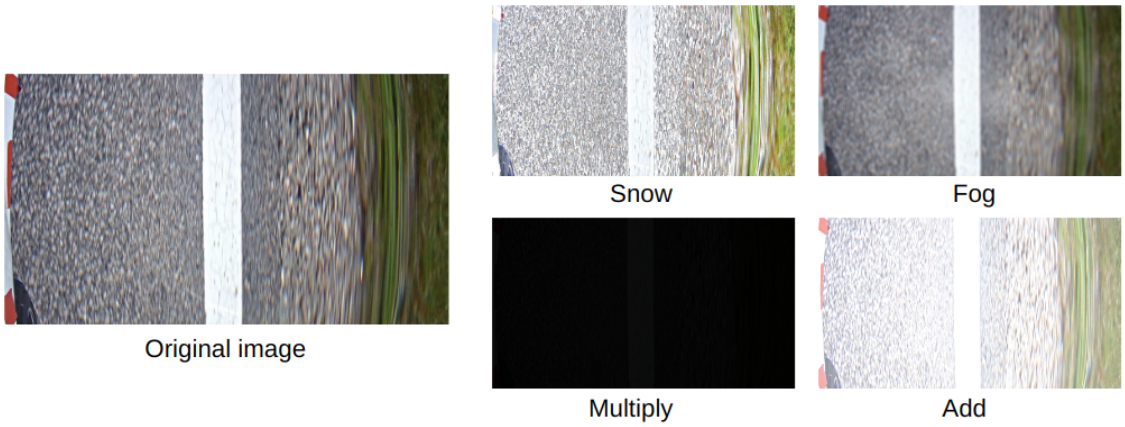


Figure 3.2: Different data augmentation policies performed on the original image

After data augmentation, the data set size increased from around 3100 images to 15,717 images covering rain, sensor blur, snow, sunny, and shadow conditions, of which 11,592 are synthetic images. Augmented data set is divided into Training and Validation sets. The training set and Validation set are split in the ratio 85:15. Another 442 images are annotated to construct the test data set. Images in the test set are not augmented.

4

Methods

The pipeline developed in this thesis for estimating lateral distance is composed of four nodes. This section starts with block diagram of the proposed software pipeline 4.1. To provide details about each node in the software pipeline, this chapter is structured as follows. Section 4.1 discusses various algorithms implemented for lane detection. In section 4.2, implementation of lane tracking node is discussed, and in section 4.3, the implementation of lateral distance estimation node is discussed.

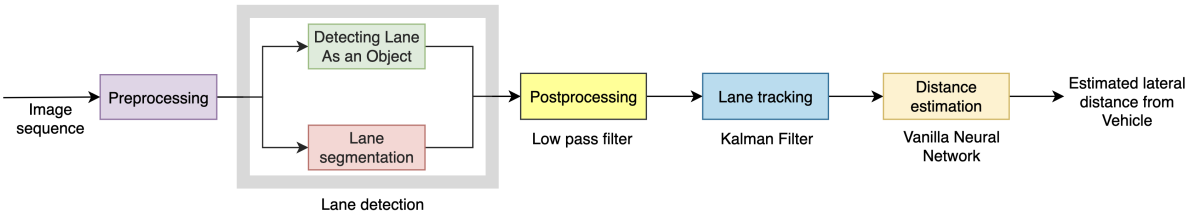


Figure 4.1: Proposed Software Pipeline

The figure 4.1 shows the outline of the software pipeline implemented in this thesis. The Lane detection node, Post-processing node, Lane Tracking node, and Lane estimation node make up the pipeline. Two methods are investigated in the lane detection node. The result in both methods is the lane's location in image coordinates.

4.1 Lane Detection

In this section, two approaches implemented for lane detection is explained. Lane Detection using YOLOv4 network is explained in sub-section 4.1.1 and lane detection using Semantic segmentation based on cGAN is explained in sub-section 4.1.2.

4.1.1 Lane Marking as Object: using YOLO

4.1.1.1 Transfer Learning

The detector's accuracy is crucial for any task involving object detection. In real-time visual tracking systems, inference speed is just as crucial as accuracy. As mentioned in Section 2.1.3.1, YOLOv7 is one of the state-of-the-art detectors which is faster and comparable in accuracy compared to other detectors. It is computationally less expensive. It takes a significant amount of data to train a neural

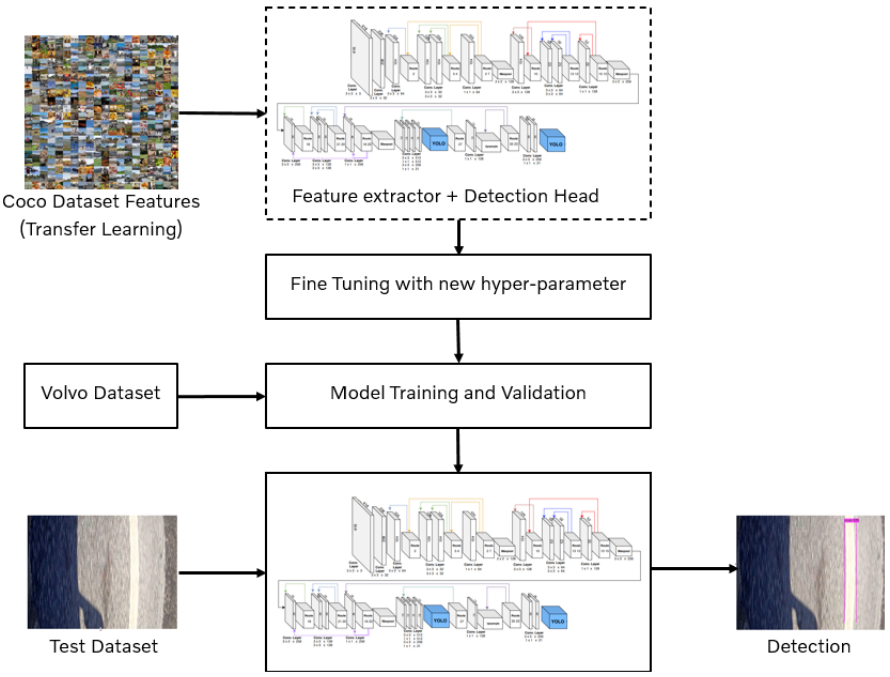


Figure 4.2: Blueprint of the proposed YOLO lane detection

network from scratch. There are only a few thousand images in the data set of Volvo Cars. Therefore, transfer learning using pre-trained weights from COCO [40] data set [24] is used in this thesis. The figure 4.2 provides the overview of the proposed YOLO model. The YOLO tiny model is trained using the pre-trained weights on the available Volvo Data set.

4.1.1.2 Network architecture

In this thesis, a lightweight YOLOversion 4-tiny [41] model is used rather than the original model, YOLO version 4 model [24]. This model is used since there is just one class to be detected in an entire image.

The model consists of only 38 layers compared to the original model's with approximately 100 layers. As a result, the training period is shortened without lowering detection precision. Figure 4.3 shows the structure of YOLOv4-tiny. It is composed of following parts:

- Input: Image Sequence
- Backbone network: CSPDarknet53-Tiny
- Neck network: Feature pyramid network
- Head: YOLOversion 3, one-stage Dense prediction head

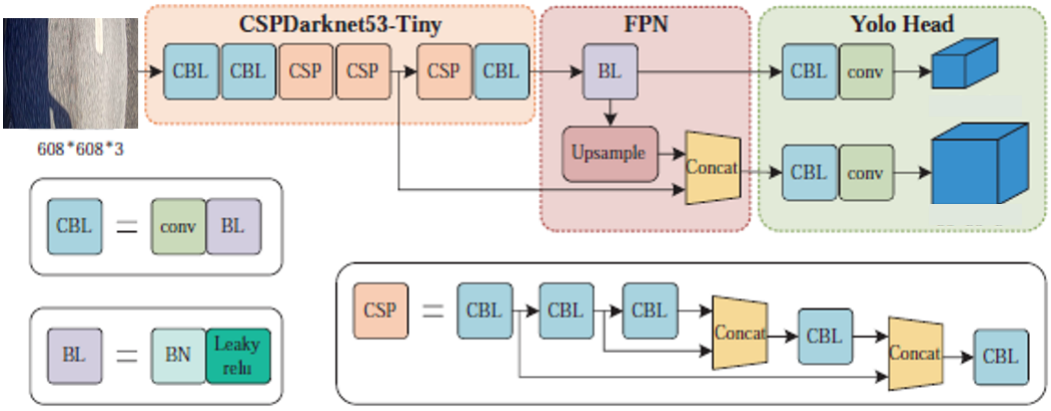


Figure 4.3: Architecture of Yolov4-tiny

As shown in the figure 4.3, the CSPDarknet53-Tiny network is made up of the Convolutional Base Layer (CBL) and Cross Stage Partial (CSP) block [42]. It contains 29 convolutional layers 3 x3 and 6.1 M parameters. As shown in the figure, CSP block consists of 'n' no. of CBL in series and also concatenated to each other. CBL consists of several layers like convolutional, batch normalization and activation function. This is followed by Feature Pyramid Network (FPN) which involves upsampling and concatenation of CSP and CBL. Two yolo heads are used for the final detection.

4.1.1.3 Training

The YOLO model was trained over 85% of the total data set. The data set included ground truth images and YOLO format labels [relative (x_{center}, y_{center}, w, h)]. For training, pre-trained weights from COCO data set is loaded into the network. The network is trained till the training loss converges to minima. The input image with 1920 x 1080 resolution is divided into grides by the convolutional layers and the features are extracted. During training, the CNN layers are divided at predetermined intervals and also concatenated. This allowed the network to reuse the features and pass extremely fine-grained features to the next layer. This reduced number of hyper-parameters required during the training. For instance, the layer 3 inherits the same tensor obtained in layer 2. The layer 6 concatenates the layers 4 and 5 channel-wise which increases the depth of the layer (104 x 104 x 32 + 104 x 104 x 32 = 104 x 104 x 64). These concatenation enhances the flow of gradient through the layers. The vanishing gradient problem is solved using leaky ReLU activation function [43]. YOLO outputs the object's class and bounding boxes, thus the loss in the output is optimized using a multi-loss algorithm [2]. It consists of classification, localization and confidence loss as represented in 4.1.

$$L_{total} = L_{classification} + L_{localization} + L_{confidence} \tag{4.1}$$

BCE loss is employed for classification loss. For localization loss, Complete IoU Loss (CIoU) loss is used. It has the best AP for IOU from 0.5 to 0.9 [44]. The anchor boxes size, shape, and quantity are calculated for the Volvo data set. The optimum

Layer	Type	Filters	Size/Stride	Input	Output
0	Convolutional	32	3 x 3/2	416 x 416 x 3	208 x 208 x 32
1	Convolutional	64	3 x 3/2	208 x 208 x 32	104 x 104 x 64
2	Convolutional	64	3 x 3/1	104 x 104 x 64	104 x 104 x 64
3	Route 2				
4	Convolutional	32	3 x 3/1	104 x 104 x 32	104 x 104 x 32
5	Convolutional	32	3 x 3/1	104 x 104 x 32	104 x 104 x 32
6	Route 5, 4				
7	Convolutional	64	1 x 1/1	104 x 104 x 64	104 x 104 x 64
8	Route 2, 7				
9	Maxpool		2 x 2/2	104 x 104 x 52	52 x 52 x 128
10	Convolutional	128	3 x 3/1	52 x 52 x 128	52 x 52 x 128
11	Route 10				
12	Convolutional	64	3 x 3/1	52 x 52 x 64	52 x 52 x 64
13	Convolutional	64	3 x 3/1	52 x 52 x 64	52 x 52 x 64
14	Route 13, 12				
15	Convolutional	128	1 x 1/1	52 x 52 x 128	52 x 52 x 128
16	Route 10, 15				
17	Maxpool		2 x 2/2	52 x 52 x 256	26 x 26 x 256
18	Convolutional	256	3 x 3/1	26 x 26 x 256	26 x 26 x 256
19	Route 18				
20	Convolutional	128	3 x 3/1	26 x 26 x 128	26 x 26 x 128
21	Convolutional	128	3 x 3/1	26 x 26 x 128	26 x 26 x 128
22	Route 21, 20				
23	Convolutional	256	3 x 3/1	26 x 26 x 256	26 x 26 x 256
24	Route 18, 23				
25	Maxpool		2 x 2/2	26 x 26 x 512	13 x 13 x 512
26	Convolutional	512	3 x 3/1	13 x 13 x 512	13 x 13 x 512
27	Convolutional	256	1 x 1/1	13 x 13 x 512	13 x 13 x 256
28	Convolutional	512	3 x 3/1	13 x 13 x 256	13 x 13 x 512
29	Convolutional	18	1 x 1/1	13 x 13 x 512	13 x 13 x 18
30	YOLO				
31	Route 27				
32	Convolutional	128	1 x 1/1	13 x 13 x 256	13 x 13 x 128
33	Upsample		2x		
34	Route 33, 23				
35	Convolutional	256	3 x 3/1	26 x 26 x 384	26 x 26 x 256
36	Convolutional	18	1 x 1/1	26 x 26 x 256	26 x 26 x 18
37	YOLO				

Figure 4.4: Input and output size at each layer of the used YOLO network

hyper-parameters listed in table 4.1 increased the network's accuracy and reduced the computational time by one-third compared to lane segmentation method 4.1.2

The AP on the validation data set is further increased with maxpooling layers of stride 2 instead of 1. During training, this helped in extracting less lane segments in the images. After training, the network's inference rate is increased with two detection heads of prediction scales 13x13 and 26x26.

4.1.2 Lane Segmentation

A Lane segmentation network based on cGAN, takes an RGB image of size 512x512 corresponding to height and width, as input and produces a segmentation map corresponding to the road lane marking in the form of binary image format of size 512x512. All pixels other than those corresponding to road lane markings are set to

Table 4.1: Hyper-parameters to fine tune the YOLO model

Parameters	Value
Batch Size	64
Subdivisions	16
Learning Rate	0.0025
Activation	Leaky ReLU
Epochs	50

zero while those corresponding to road lane markings are set to 255.

4.1.2.1 Network Architecture

The Lane segmentation network comprises two neural networks, namely the Generator network and the Discriminator network. The network composition is named as *Generative net* in this thesis. An overview of the Generative net architecture is as shown in Figure 4.5. Generative net is designed based on UNet architecture developed by Olaf Ronneberger et al.[4]. Further, the Generator architecture is divided into two parts: an encoder and a decoder.

An encoder is also known as a contraction path, which involves stacking successive layers of convolution layers with Max Pooling layers. The purpose of this setup is to capture finer details in the image, during this process resolution of the input image is reduced. The decoder is also called as expanding path where successive transposed convolution layers are stacked. Decoder network increases the resolution of the output. Common problem associated with the deep neural network is as the number of layers increases, model saturation inaccuracy creeps in, also known as model degradation problem [45]. Due to degradation problem , model fails to learn even identity mapping. In order to overcome this issue, skip connections is introduced between the corresponding layers of encoder and decoder between the corresponding layers of Encoder and Decoder as shown in the Figure 4.5.

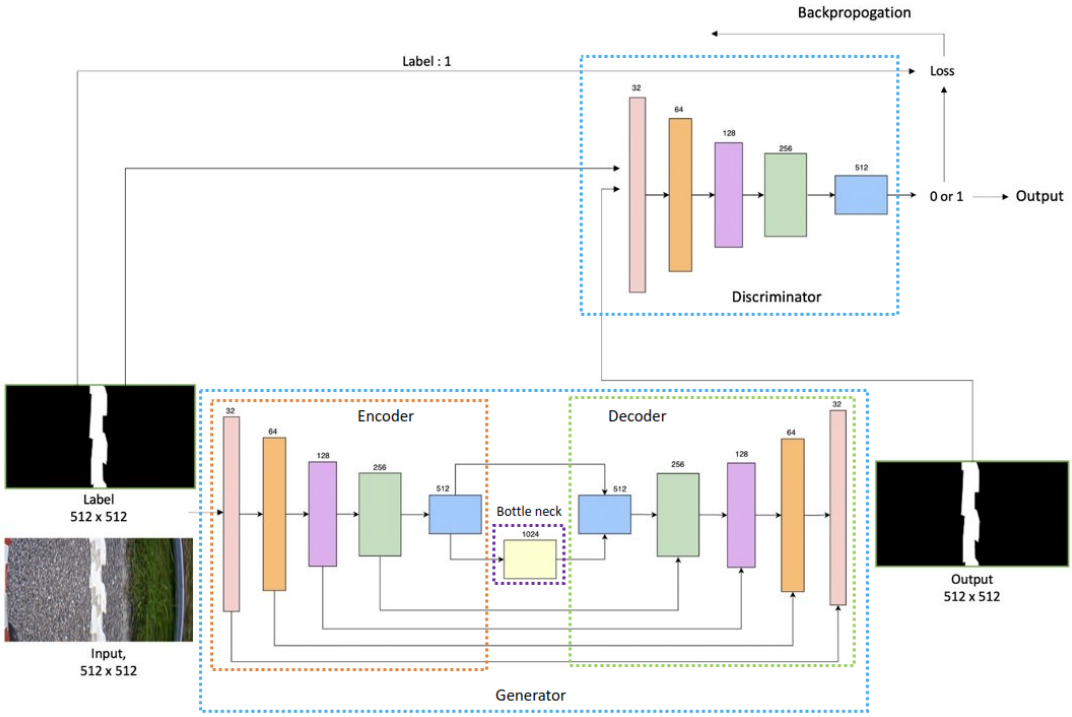


Figure 4.5: Generative Network Architecture

Encoder or contraction path has six blocks, each blocks has 3 layers, setup as shown in the Figure 4.6. Each layer within a block has the same number of filters, n. The filter sizes in each block are 32, 64, 128, 256, 512 and 1024, respectively.

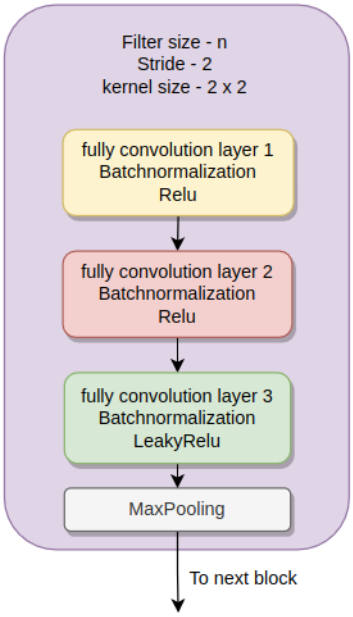


Figure 4.6: Representation of network block

Decoder or expanding path has five blocks, each blocks has 3 layers, setup as shown in the Figure 4.7. Each layer within a block has the same number of filters, n. The filter sizes in each block are 512, 256, 128, 64 and 32, respectively. Output image from the Generator has the size [512x512x1] .

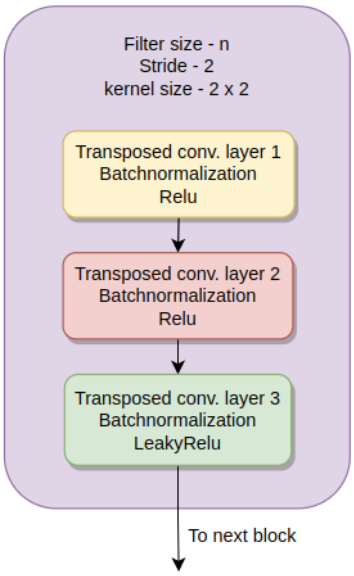


Figure 4.7: Representation of network block

The encoder in the Generator network has been made modular. The encoder is also used as a Discriminator network, except that the output of the Discriminator network is a scalar tensor.

Following the suggestion from the authors of Deep Convolution GAN (DCGAN) [46], all weights in the model are randomly initialized by sampling from a zero-centered Normal distribution with standard deviation = 0.02 [46], to speedup the convergence of the network. For the optimized utilization of the hardware resources, another technique called as Mixed Precision Training module native for PyTorch [1] is used.

4.1.2.2 Training

During training, the Generator network and Discriminator network are trained separately till convergence. The network architecture of the Generator is shown in the Figure 4.5. ADAM optimizer[47] is used during training of the networks.The ADAM optimization algorithm calculates an exponential moving average of the gradient and the squared gradient, and $\beta 1$ and $\beta 2$ parameters in ADAM optimizer algorithm control the decay rates of these moving averages. Generator network designed in this thesis is named as *LaneSeg net*.

[1] https://pytorch.org/blog/what-every-user-should-know-about-mixed-precision-training-in-pytorch/

Output from the Generator trained until convergence, is now used along with the samples from the training set to train only the Discriminator. Discriminator network architecture is as shown in the figure 4.5.

Once Discriminator is trained until convergence, fully trained Segmentation network and Discriminator network is trained in Adversarial manner [5] in which Generator network tries to increase the Discriminator training loss and vice-versa. This helped to overcome Generator model convergence problem. Several hyperparameters are tried during the training of the network. Hyperparameters listed in the Table 4.2 is found to be best suited for the network for quick convergence.

Table 4.2: Hyper-parameters to fine tune the model

Parameters	Value
Epochs	150
Batch Size	16
Learning Rate	0.0002
Training Device	CUDA

4.1.2.3 Post-processing

The Semantic segmentation network produced a binary image in which pixels corresponding to lane markings are set to 255 and the other image pixels are set to 0. Morphological operations[2] , Erosion and Dilation are performed on the binary image to remove any discontinuities in the segmentation mask in the image.

The four corners of the lane marking are identified using findContours[3] and minAreaRect [4] functions from python opencv2, outcome of this process for one of the image frame is illustrated in the Figure 4.8. Ten points are marked along the length of the centerline of the Bounding Box, this is used as input to the lane tracking node.

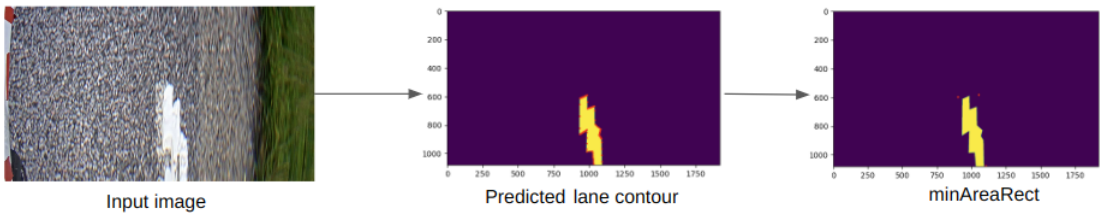


Figure 4.8: Illustration of Lane Marking corner Points Extraction

[2]https://docs.opencv.org/4.x/d9/d61/tutorial$_{p}y_{m}orphological_{o}ps.html$
[3]https://docs.opencv.org/3.4/d4/d73/tutorial$_{p}y_{c}ontours_{b}egin.html$
[4]https://docs.opencv.org/3.4/de/d62/tutorial$_{b}ounding_{r}otated_{e}llipses.html$

The segmentation mask with the maximum area is selected as the lane marking, in order to eliminate potential FP, as shown in Figure 4.9.

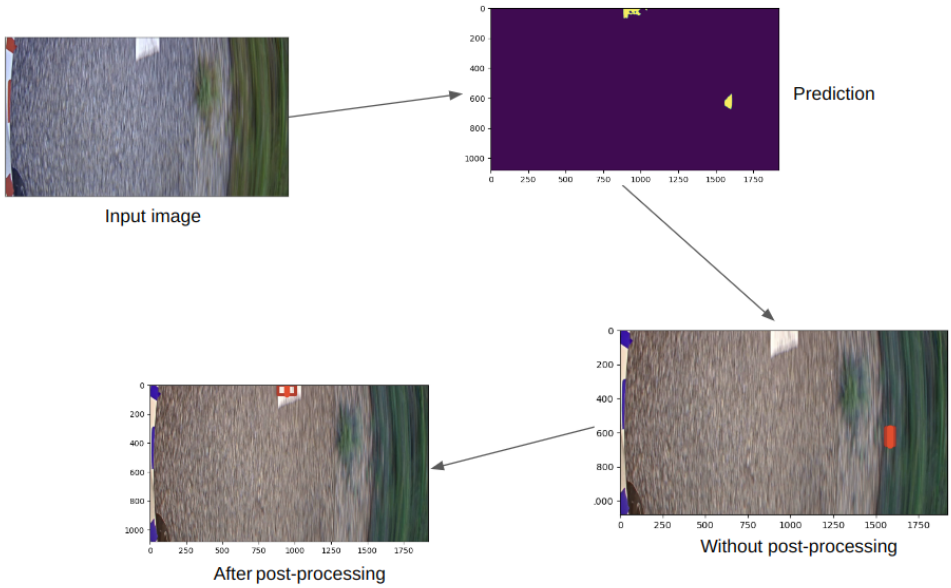


Figure 4.9: Filtering of False positives

4.2 Lane Tracking

When processing an image sequence, some images do not contain lane markings. Consequently, the lane detection node cannot produce a detection that corresponds to the lane marking. To overcome this problem, a linear Kalman filter-based lane tracking algorithm is developed, which tracks the lane marking in every frame by utilizing the lane marking detected from the previous frame at time $k-1$.

Due to the orientation of the camera used to collect data, lane markings always appear straight. Hence, Linear Kalman filter based lane tracking algorithm was more suitable.

4.2.1 Linear Kalman Filter

A lane marking is being tracked based on its position in the image, which is the unknown variable. The position of the lane marking in each image frame is the x and y coordinates of the pixels representing lane marking. Using the kinematic equations, the transformation of the lane marking in every image frame is mathematically modeled in state-space form. The state vector x_k consists of x and y coordinates of the image pixels representing lane markings and it's velocities as shown in equation 4.2. Where x coordinate represents distance of the lane marking along the width of the image in pixels and y coordinate represents distance of the calibration point along the height of the image in pixels.

$$\mathbf{x}_k = \begin{bmatrix} x_k \\ y_k \\ \dot{x}_k \\ \dot{y}_k \end{bmatrix} \tag{4.2}$$

Using kinematic equations, relationship between position and velocity of the X and Y coordinates of the pixel representing lane marking is modelled as shown in the equations 4.3, 4.4, 4.5 and 4.6

$$x_k = x_{k-1} + \dot{x}_{k-1}\Delta t + \frac{1}{2}\ddot{x}_{k-1}(\Delta t)^2 \tag{4.3}$$

$$y_k = y_{k-1} + \dot{y}_{k-1}\Delta t + \frac{1}{2}\ddot{y}_{k-1}(\Delta t)^2 \tag{4.4}$$

$$\dot{x}_k = \dot{x}_{k-1} + \ddot{x}_{k-1}\Delta t \tag{4.5}$$

$$\dot{y}_k = \dot{y}_{k-1} + \ddot{y}_{k-1}\Delta t \tag{4.6}$$

Since, State vector X_k is the combination of both position and velocities of the X and Y coordinates. State-space form is modelled as in equation 4.7. Continuous time motion model is discretized by sampling at regular time interval, Δt=0.01 seconds.

$$\begin{bmatrix} x_k \\ y_k \\ \dot{x}_k \\ \dot{y}_k \end{bmatrix} = \begin{bmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{k-1} \\ y_{k-1} \\ \dot{x}_{k-1} \\ \dot{y}_{k-1} \end{bmatrix} + \begin{bmatrix} 0.5\Delta t^2 & 0 \\ 0 & 0.5\Delta t^2 \\ \Delta t & 0 \\ 0 & \Delta t \end{bmatrix} \cdot \begin{bmatrix} \ddot{x}_{k-1} \\ \ddot{y}_{k-1} \end{bmatrix} + q_{k-1} \tag{4.7}$$

Where, q_{k-1} is the uniformly distributed process noise modelled as a stochastic variable having it's first moment, mean '0' and second moment, covariance Q_{k-1} represented in equation 4.8.

$$Q_{k-1} = \begin{bmatrix} \sigma_x^2 & 0 & \sigma_x\sigma_{\dot{x}} & 0 \\ 0 & \sigma_y^2 & 0 & \sigma_y\sigma_{\dot{y}} \\ \sigma_{\dot{x}}\sigma_x & 0 & \sigma_{\dot{x}}^2 & 0 \\ 0 & \sigma_{\dot{y}}\sigma_y & 0 & \sigma_y^2 \end{bmatrix} \tag{4.8}$$

Measurement model Y_k, which relates state vector with the observation, in the state-space form is represented in the equation 4.9.

$$y_k = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} x_k \\ y_k \\ x_{k-1} \\ y_{k-1} \end{bmatrix} + r_k \tag{4.9}$$

Where, r_k is the uniformly distributed measurement noise modelled as a stochastic variable having it's first moment, mean '0' and second moment, covariance R_k represented in the equation 4.10.

$$R_k = \begin{bmatrix} \sigma_x^2 & 0 \\ 0 & \sigma_y^2 \end{bmatrix} \tag{4.10}$$

Since in every frame, only position of the X and Y coordinates of the pixel corresponding to the lane marking is tracked, transformation matrix in equation 4.9 has one's in the location corresponding to the x_k and y_k in the state vector.

The framework of the Linear Kalman Filter based Lane tracking algorithm is illustrated as shown in the Figure 4.10.

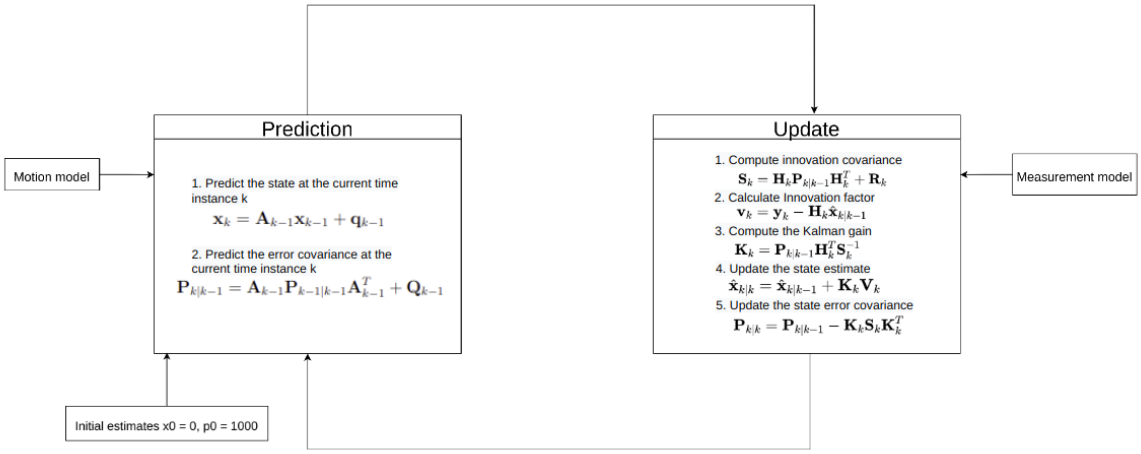


Figure 4.10: Lane Tracking framework

A representation of the output of the lane tracking node implemented in this thesis can be seen in Figure 4.11. A lane marking cannot be found in the image frame obtained at time k. Lane tracking node predicts the position of the lane marking at time instance k based on the position of the lane marking, computed at the previous time instance $k-1$.

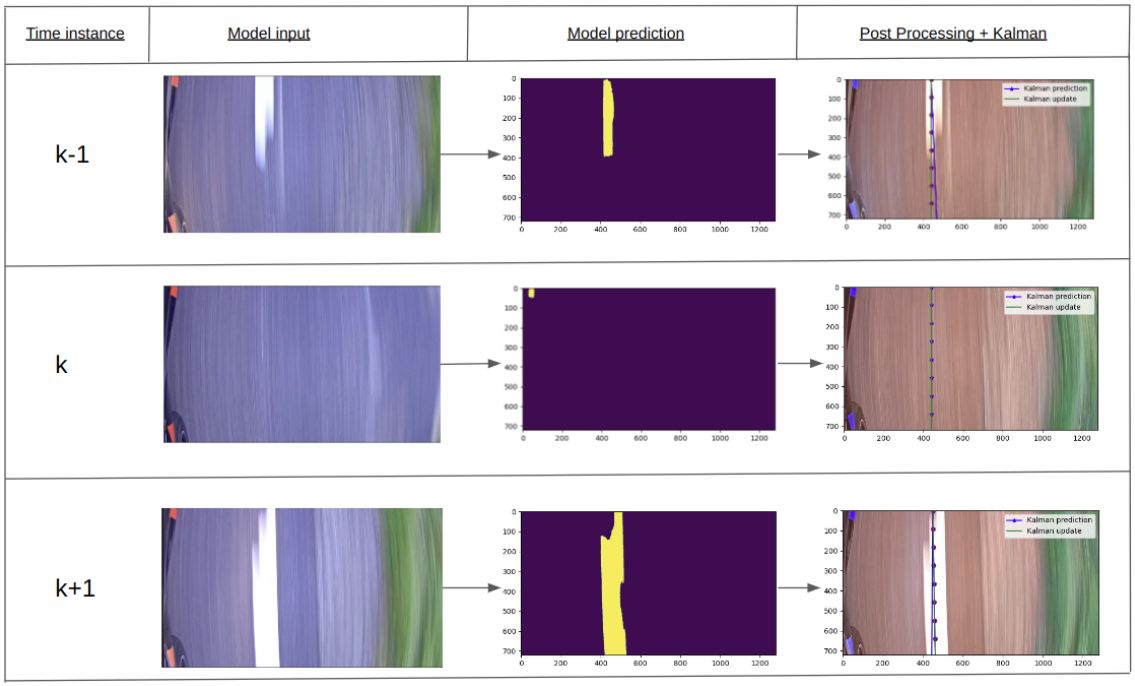


Figure 4.11: Illustration of output from Lane Tracking node

4.3 Lateral Distance Estimation

This section deals in computing the lateral distance between the vehicle and the lane marking from the tracking results. It involves the conversion of 2D image plane to 3D world coordinates. This process is done using the inbuilt Volvo cars Real World Objective Testing of Autonomous Cars (ROTAC) Matlab tool.

The ROTAC tool takes in an image sequences of calibration pattern such as checkerboard of known length and width. It detects 2D image points corresponding to the corners of the squares in the checkerboard. The centre of the image coordinate system is chosen as the origin point as marked in the Figure 4.12. The tool outputs the lateral distance of all the detected points along with corresponding image coordinates as shown in the figure 4.14a.

The look up table 4.14a contains only detected points and did not cover all x,y coordinates. A method is developed to extend the look up table to calculate the distance for any given x,y coordinate. This was done using curve fitting function. The data 4.14a is used to approximate a function that takes x, distance as input and gives lateral distance as output in millimeters. Polyfit is a python function that computes a least squares polynomial for a given set of data points. The types of functions used are linear, exponential, quadratic and cubic functions as well as a neural network to fit the data as shown in Figure 4.3

The best match with regards to low bias and low variance was found using cubic polynomial function 4.14e and neural networks 4.14f. A vanilla neural network 4.13 is used as a function approximator. It consists of four linear layers with ReLU activation function. Before training the network, the input data points were standardized which means subtracting the mean value or centering the data. It was found that the standardized data gave a better fit than the normalized data.

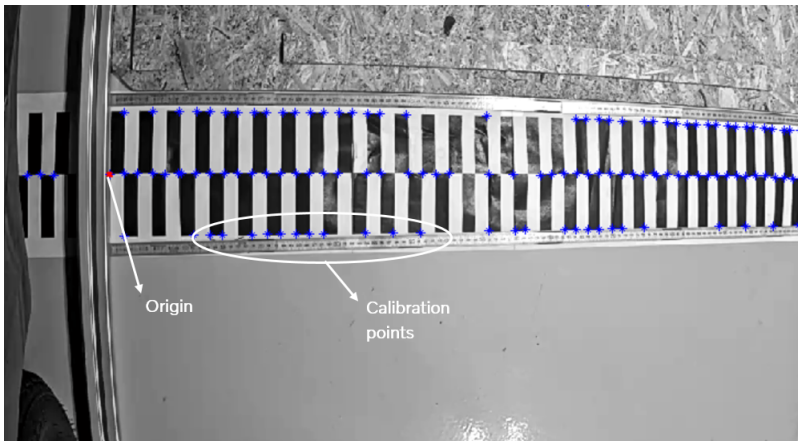


Figure 4.12: Plane pattern method with checkboard points

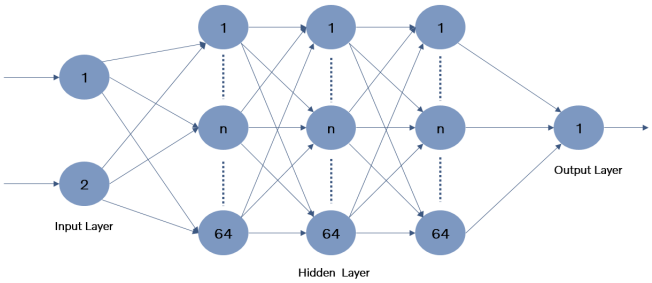
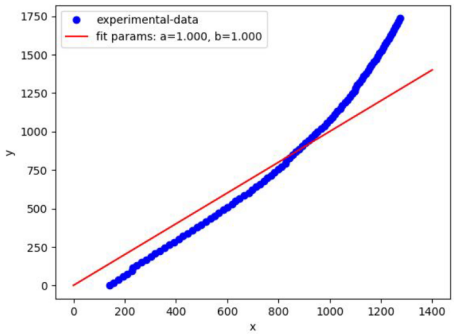


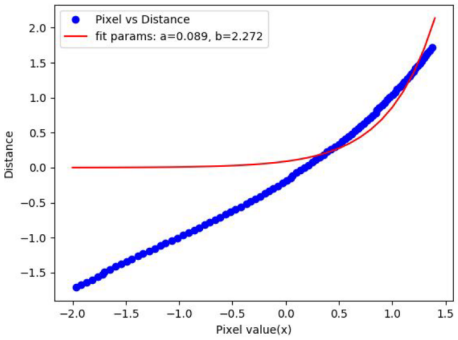
Figure 4.13: Vanilla Neural network

	Distance	x	y
	1	2	3
1	0	141	305
2	18.8679	158	304
3	37.7358	175	304
4	56.6038	194	304
5	75.4717	210	305
6	94.3396	227	304
7	113.2075	230	303
8	132.0755	248	303
9	150.9434	266	304
10	169.8113	284	303

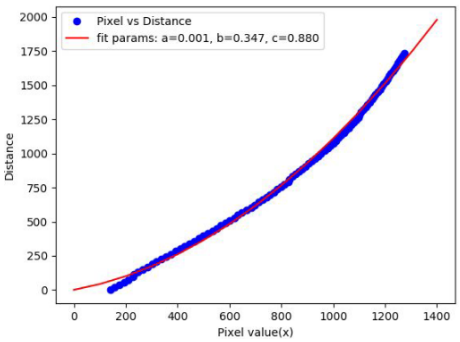
(a) Data from the tool



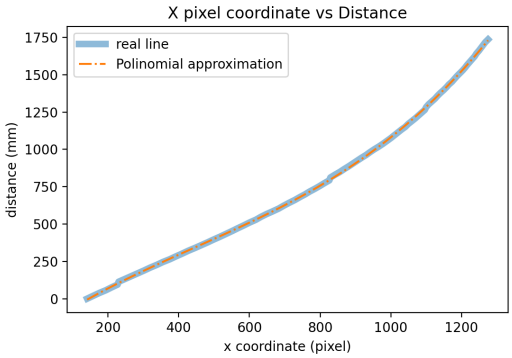
(b) Linear function



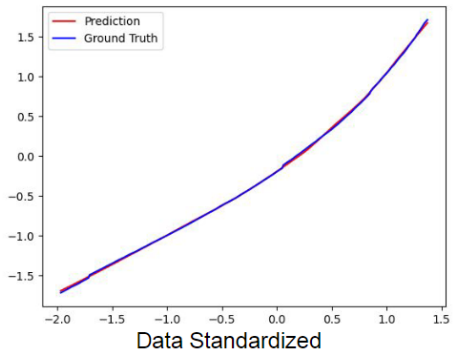
(c) Exponential function



(d) Quadratic function



(e) Cubic polynomial function



(f) Neural network function

Figure 4.14: Curve fitting functions

5
Results

In this chapter, the outcomes for the objectives in Section 1.1 are addressed. The results obtained from proposed methods are presented here. Chapter 5 is structured as follows: Sections 5.1 and 5.2 describes the results obtained from lane detection and segmentation methods, respectively. Section 5.3 compares the accuracy and performance of three lane detection models. In sections 5.4 and 5.5, the results obtained from Tracking algorithm and Distance estimation from lane are analyzed qualitatively. Section 5.6 introduces the hardware setup used to build and train the Deep learning models and discusses the basis for selecting the deep learning framework used. Both qualitative and quantitative evaluation metrics are used to measure the performance of the above proposed methods. All the evaluation metrics used in this chapter are explained in Section 2.3

5.1 Lane Detection

The performance analysis of the detecting lane as an object is presented in this section.

5.1.1 YOLO, average Loss and Convergence

The hyperparameters used to train the model on the Volvo Cars Dataset are listed in Table 4.1. Figure 5.1 displays AP for validation data set and training loss variation over number epochs the network is trained.
The blue line in the above figure indicates the training loss of the model. When the loss converges toward the minima, it shows that the training is complete. The red line displays the AP for validation data set. The precision of the line gradually increases as the number of training epochs increases.

Training is terminated when the loss maintains constant after several epochs. After every 1000 training iterations, the model weights are saved. Lastly, the best Average Precision weights are chosen to evaluate the model, on the test data set. The training loss of the model obtained is as low as 0.2111 and the best weights were obtained at 45 epochs.

After training for 40 number of epochs, the learning rate was dropped by a factor of 10, and after training for 45 number of epochs, it was again reduced by a factor of 10. The learning rate was initially set to 0.0025. Different initial learning rates

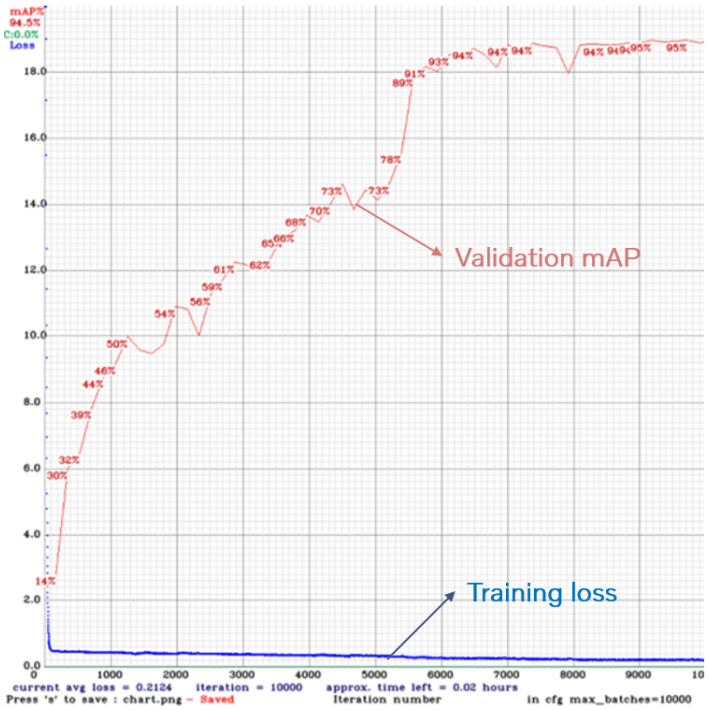


Figure 5.1: YOLO 608x608 performance curve

used are 0.5, 0.1, 0.01 and 0.002. Best results are achieved when using the learning rate of 0.002.

5.1.2 Overall performance

The precision of the object detector is improved by tuning the input layer resolution. The height and width of the network resolution is increased in the multiples of 32. Standard network resolution used is 416x416. The table shows the Comparison of the overall detection performance with various input dimensions. The proposed algorithm were compared at IOU of 0.5 and 0.75 from Table 5.1

Table 5.1: Comparison of the overall Lane detection performance of YOLOv4 tiny

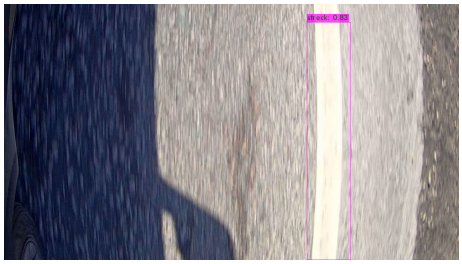
Model	$AP50_{val}$	$AP75_{val}$	Precision	Recall	F1 score
YOLOv4-Tiny (416 x 416)	91.0	81.0	0.84	0.9	0.86
YOLOv4-Tiny (512 x 512)	93.10	82.5	0.87	0.88	0.874
YOLOv4-Tiny (608 x 608)	94.75	84.0	0.92	0.84	0.88

Results in Table 5.1 are calculated based on the validation data set, which includes 2280 images. The table 5.1 clearly shows that the precision increases as input network resolution increases. The input network resolution 608x608 at IOU of 0.5 and 0.75 generates the highest AP of 94.75%. The increase in network input size will facilitate the network's ability to locate small lane segments in the image. However, this leads to higher training and inference durations. Lower resolution of 416x416, resulted in lower AP of 91%. The precision is higher for network resolution of

608x608, indicating a high number of positive predictions. However, it has lower recall of 0.84.

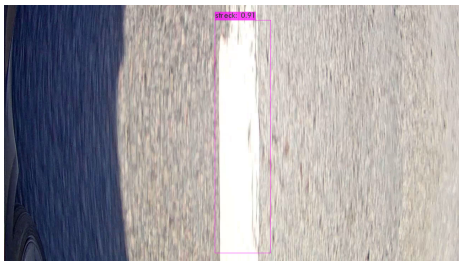
5.1.3 Detections

Figure 5.2 displays the network's detections on randomly chosen test data. As anticipated, input image size of 608x608 exhibits better lane detections than the 512x512 because the confidence score is high and the bounding box fit better. Despite having the maximum AP at 608x608, it was unable to detect some images at IOU 0.5 as shown in 5.2e.

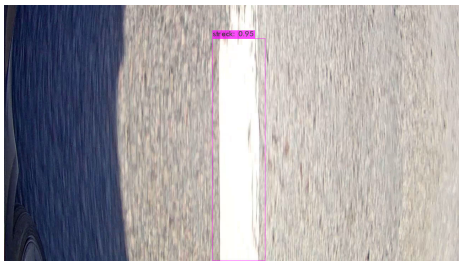


(a) 512 x 512

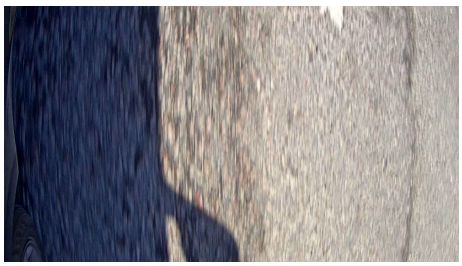
(b) 608 x 608



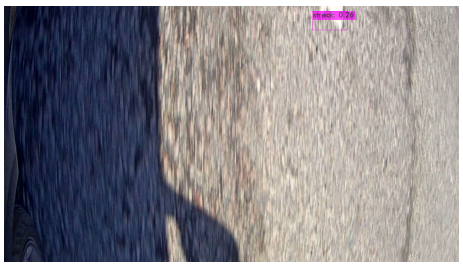
(c) 512 x 512



(d) 608 x 608



(e) Lane not detected



(f) Lane detected with IOU<0.5

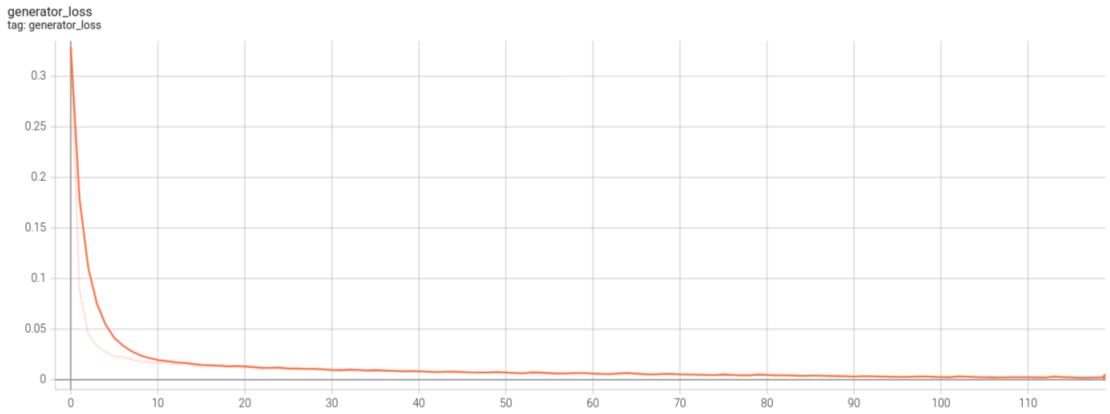
Figure 5.2: YOLO detection

5.2 Semantic Segmentation

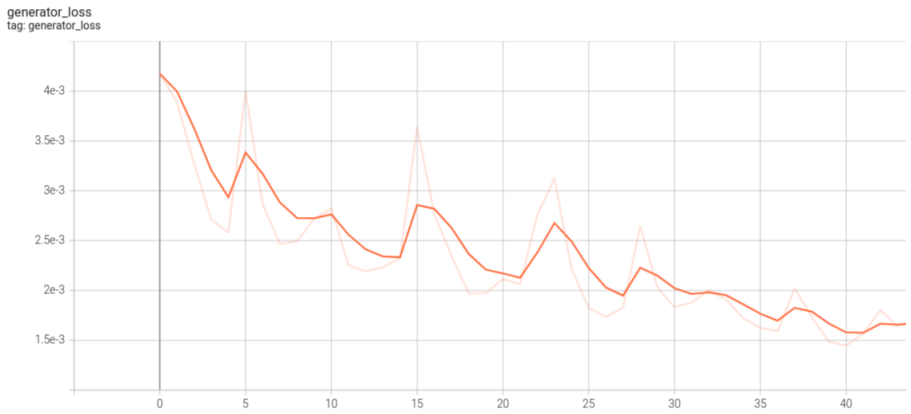
An analysis of the performance of the Semantic segmentation approach is presented in this section.

5.2.1 Training loss

The tuned hyper-parameters used to train the model are listed in the Table 4.2. Learning curve of the LaneSeg net and Generative net is shown in the Figure 5.3a and 5.3b. The horizontal axis of the curve corresponds to the number of epochs and loss calculated at each corresponding epoch is represented by vertical axis. The training is terminated when the loss tends to a constant value for a few epochs. For the set of tuning parameters selected, this indicates that network learning is saturated. Based on the trial-and-error method, 0.0002 is determined to be the optimal learning rate for LaneSeg net and Generative net. A variety of learning rates is tried during training, including 0.0002, 0.0001, 0.01 and 0.1. During training, the optimal batch size is found to be 16. Other batch sizes tried included 4,8 and 32. A further reduction in network loss is achieved through adversarial training of the Generative net, evident in the scale of the vertical axis in the Figure 5.3b.



(a) LaneSeg net training loss

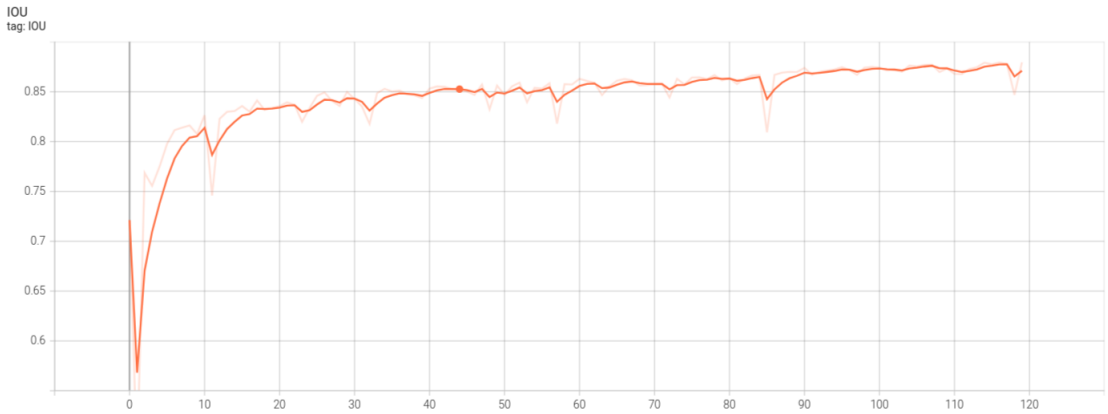


(b) Generative net training loss

Figure 5.3: Change in Lane Segmentation network loss during training

5.2.2 Performance Evaluation

At every epoch, the LaneSeg net's training progress is validated using the validation dataset which includes 2280 images and synthetic images are excluded in validation dataset. IOU, F1-score, and Pixel Accuracy is used as evaluation metrics. For computing IOU, a bounding box is drawn around the lane marking detected in the Generator network output, using the *minAreaRect* method from *openCV2* library. Accuracy is measured by computing the intersection of the bounding box area of the network's output with that of ground truth. Figure 5.4a and 5.4b represents the LaneSeg net and Generative net's improved prediction performance in terms of IOU. where horizontal axis represents the number of epochs and vertical axis represents IOU.

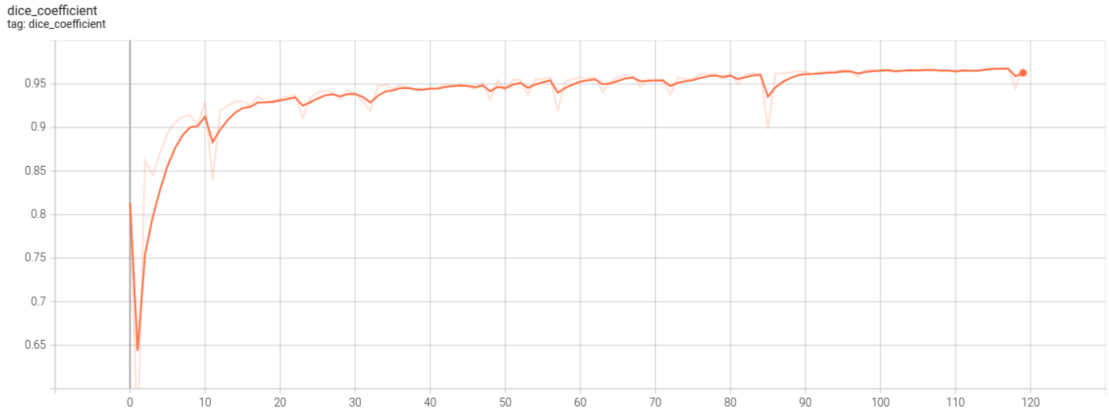


(a) Measurement of LaneSeg net Performance on Validation Dataset using IOU

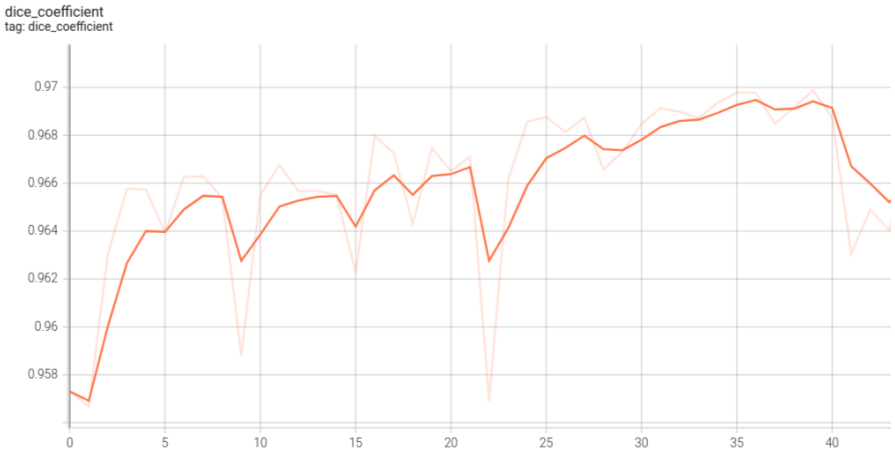
(b) Measurement of Generative Net Performance on Validation Dataset using IOU

Figure 5.4: Measurement of Lane Segmentation network Performance using IOU

Figure 5.5a and 5.5b represents the LaneSeg net and Generative net's improved prediction performance in terms of F1-score, where horizontal axis represents the number of epochs and vertical axis represents F1-score.



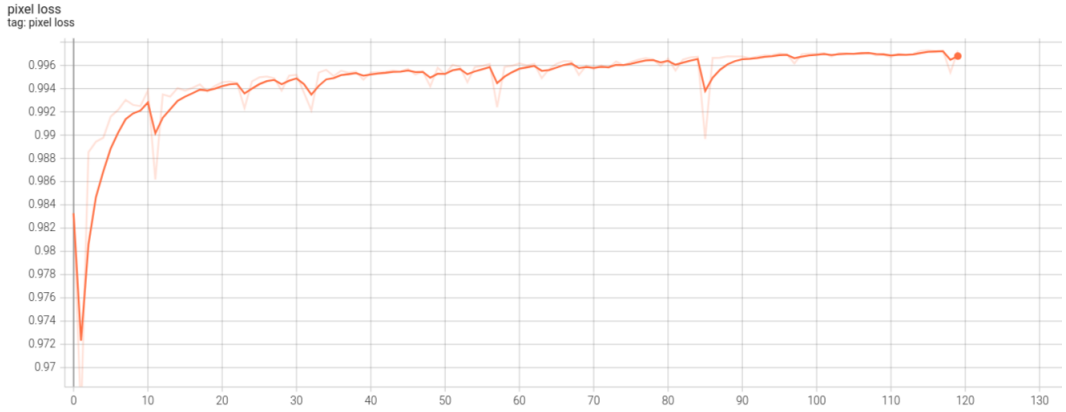
(a) Measurement of LaneSeg net Performance on Validation Dataset using F1-score



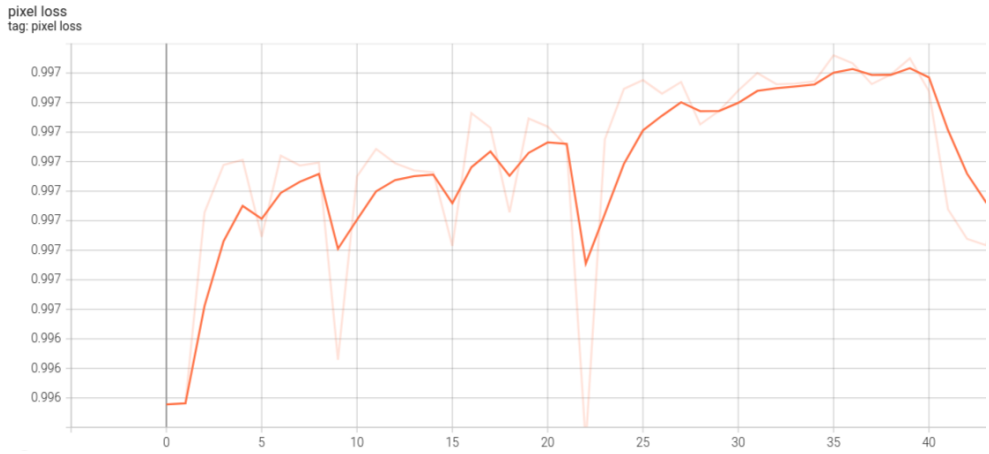
(b) Measurement of Generative Net Performance on Validation Dataset using F1-Score

Figure 5.5: Measurement of Lane Segmentation network Performance using F1-score

Figure 5.6a and 5.6b represents LaneSeg net and Generator network's improved learning progress in terms of *Pixel accuracy* where horizontal axis represents the number of epochs and vertical axis represents pixel accuracy in percentage.



(a) Measurement of LaneSeg net Performance on Validation Dataset using Pixel accuracy



(b) Measurement of Generative Net Performance on Validation Dataset using Pixel accuracy

Figure 5.6: Measurement of Lane Segmentation network Performance using Pixel Accuracy

Discriminator

Learning curve of the Discriminator network is as shown in the Figure 5.7. Horizontal axis in the curve represents number of epochs and vertical axis represents loss calculated at each corresponding epoch.

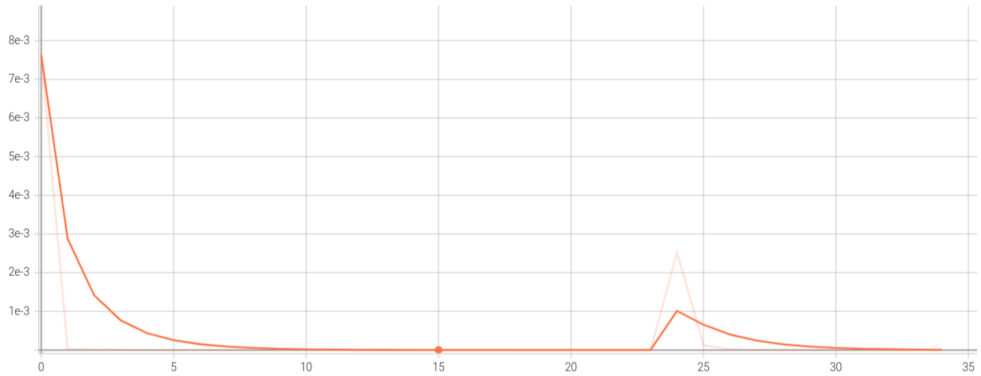


Figure 5.7: Learning curve of Discriminator network during training

5.3 Comparison between Lane Detection Models

Table 5.2: Performance comparison between lane detection models

Serial No.	**Method**	**Pixel Accuracy**	**F1 Score**	**Avg. fps**
1	LaneSeg net	0.97	0.95	45
2	Generative net	0.98	0.968	45
3	YOLOv4-Tiny	-	0.88	135

Table 5.2 shows the comparison of network performance between the three models. This performance is assessed on the validation data set.

Accuracy: F1 score for the models, LaneSeg net (0.97) and Generative net (0.98) are higher than YOLOv4-Tiny(0.88). Both the methods with higher F-1 score are image segmentation models so they are able to retain more information and to recognize the lanes better. Whereas YOLO model, examines the image at once instead of segmenting the image. Hence, it is unable to capture minute details in the image. Also out of 442 images in the test data set, YOLO was unable to recognize lanes in 54 images. LaneSeg net was unable to detect 42 images, and Generative net were unable to detect 38 images.

Performance: Since LaneSeg and Generative net are image segmentation models, these take more time for processing an image which results in avg fps of 45. Whereas YOLOv4-Tiny, which doesn't involve image segmentation has a higher avg fps 135 which is 3 times higher.

To summarize, LaneSeg net and Generative net provide better accuracy but YOLOv4-Tiny has higher inference speed which is useful for real time applications.

5.4 Lane Tracking

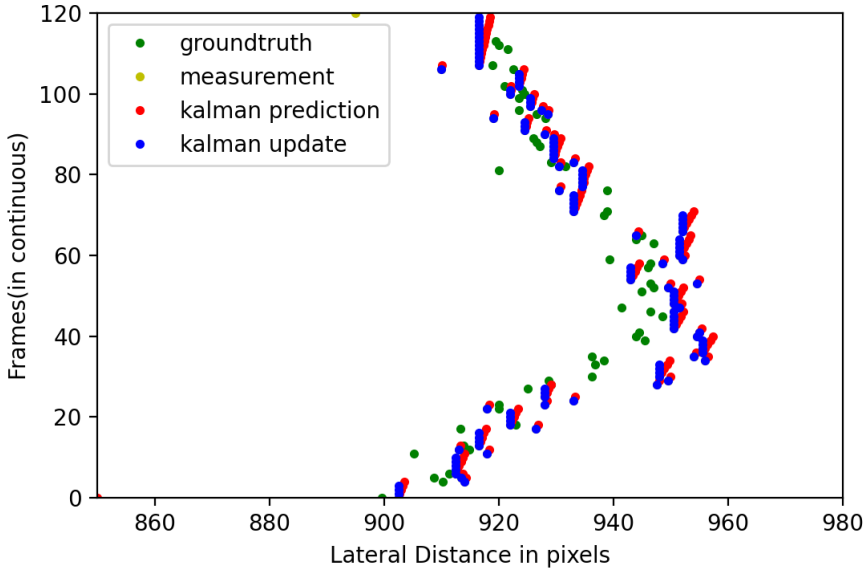


Figure 5.8: Kalman filter prediction and update trajectory comparison against groundtruth trajectory

The test data set contains image sequences with lanes that are not continuous. The figure 5.8 demonstrates how the kalman prediction model can predict the position of missing lanes. The update step corrects the difference between the prior prediction and the measurement, when there is lane information in the following image. The proposed methods shown in figure 5.9 predict similar lane trajectory across the series of frames. YOLO is out of trajectory in some frames as compared to cGAN because it failed to recognize lanes in few adjacent images. This will increase the uncertainty in the prediction step as measurement input is not available to reduce error in the update step.

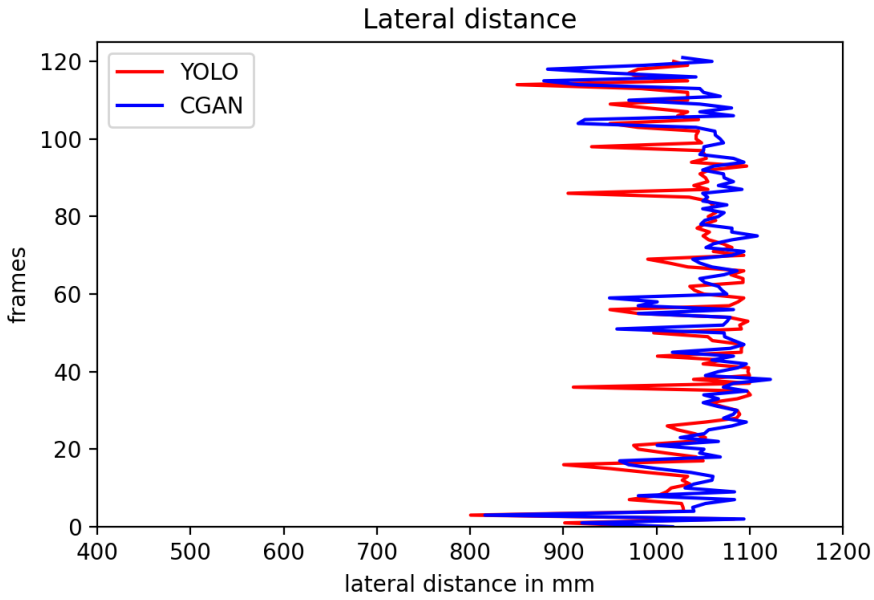


Figure 5.9: Filter Trajectory Comparison between the models

5.5 Lateral Distance Estimation

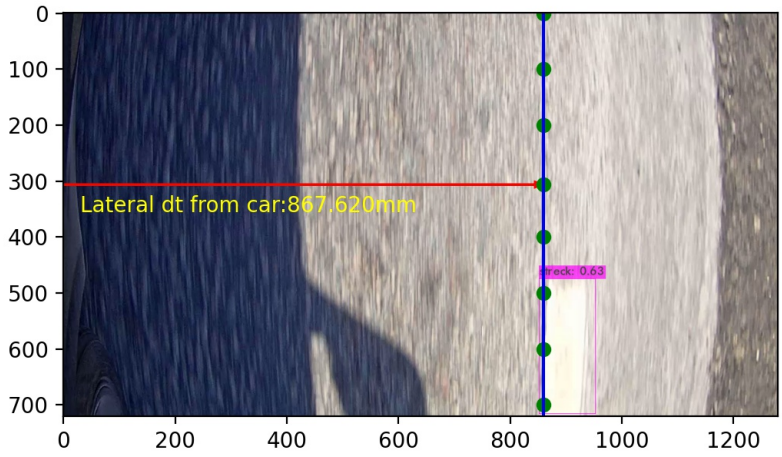


Figure 5.10: Final lateral distance output

The figure 5.10 shows the final output display from the proposed software pipeline. By calculating mAE between the ground truth and measured values, the accuracy of the lateral distance from the models was evaluated. The mAE is estimated to be approximately 28mm for cGAN and 32mm for YOLO when using a test data set of 442 sequential images. YOLO's mAE is high as it couldn't detect lane in few

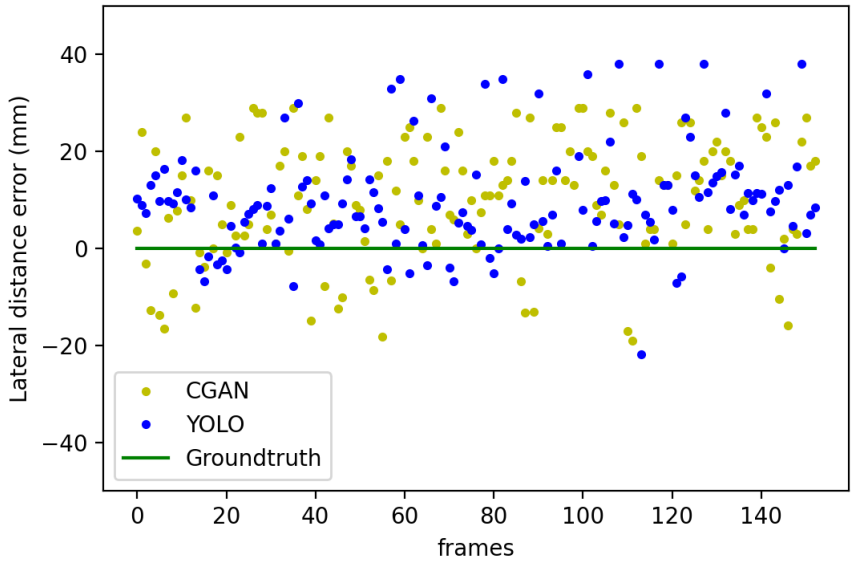


Figure 5.11: Lateral distance error in mm

images with confidence threshold of 0.5. Figures 5.11 shows that the lateral distance error between the detection models and ground truth has both positive and negative values. This is the reason for using mAE to evaluate the models. For example: In the frame1, the lateral distance from the center of the car to the lane is:

- Ground truth: 1020.20 mm
- Predicted cGAN: 1017.20 mm
- Predicted YOLO: 1025.62 mm

Similarly, in frame5:

- Ground truth: 1035.20 mm
- Predicted cGAN: 1038.98 mm
- Predicted YOLO: 1029.620 mm

5.6 Hardware and Software

Deep Learning models are developed using the Unix-based operating system Linux 18.04LTS and x86-64 architecture-based processor. Models are trained on a GPU-enabled server. Python 3.9.12 is used as the main programming language. And the various open-source libraries as shown in Figure 5.12 is used during this thesis.

Pytorch[48] is installed in the Anaconda3[49] environment. For parsing the image data during the preprocessing stage, the pillow python imaging library [1] is used. For image processing operations during postprocessing, the OpenCV2 library is used
A deep learning framework refers to an interface, library, or tool that facilitates the development of deep learning models quickly and efficiently. They do not delve into

[1] https://pillow.readthedocs.io/en/stable/

```
# pytorch deep learning libraries

import torch     # import torch has utils(has dataloader)
import torchvision   # import torchvision, has models, datasets
import torch.optim as optim  # import optim library
import torch.nn as nn   # library having nn.module class (ex. neural network layers)
import torch.nn.functional as F   # import functional library which has relu, softmax
import torchvision.transforms as transforms  # import transforms
import torch.optim as optim  # import optim library
from torch import autograd

# For mathematical computations

import numpy as np    # import numpy
import cv2      #Image processing library
from tqdm import tqdm   # To display status during training
from PIL import Image  #PILLOW image processing library

# For plotting the output from the models

import matplotlib.pyplot as plt  # import matplotlib pyplot
import seaborn as sns   # Data Visualization library

# For visualizing training progress of the network
▶ Launch TensorBoard Session
from torch.utils.tensorboard import SummaryWriter   # Visualize training parameters
```

Figure 5.12: Software packages used

details about underlying algorithms. Frameworks of this nature assist engineers in developing models from a collection of components that have already been built and optimized. The benefit of this is that we can avoid having to write hundreds of lines of code because frameworks will do most of the work for us. In the domain of deep learning, three frameworks, namely Keras [50], TensorFlow [51],and PyTorch, are widely used.

The TensorFlow machine learning framework is an open-source program developed by Google. Stable interfaces are available in TensorFlow for both Python and C++. The tensorBoard visualization library to visualize the training progress of the Neural Network is native to TensorFlow

Keras is an open-source machine-learning framework developed by Google. Keras provides a high-level user interface that can run as a wrapper over TensorFlow, CNTK, Theano, or PyTorch as the backend. Keras is known for its ease of use and syntactic simplicity. In addition, Keras offers a wide user base, a beginner-friendly user interface, error logs, and detailed documentation.

Pytorch is an open-source machine learning framework developed by the Facebook AI Research lab. The Pythonic interface of PyTorch makes it easy to build and train highly complex neural networks, so it is becoming the framework of choice for researchers over TensorFlow and Keras.

The framework to use is selected based on ease of use, library support, training duration, flexibility, debugging capabilities, and availability of the neural network training visualization tool.

Decision table					
#	Criteria	Weightage	TensorFlow	Keras	Pytorch
1	Flexibility, more pythonic way of coding	8	0	1	1
2	Ease of use	10	0	1	1
3	Function overhead	5	1	0	1
4	Userbase, mainly research community	8	1	0	1
5	Ease of debugging	8	0	1	1
Total score			13	26	39

On the basis of the total score from the Table 5.6, Pytorch is selected as a Deep learning framework in this thesis.

6
Conclusion and Future Work

This chapter summarizes and concludes this thesis work while introducing possible future research work.

6.1 Conclusion

The purpose of this thesis is to create a framework for lane distance estimation between the ego vehicle and the road lane. In order to achieve this, three methods YOLO, LaneSeg net and Generative net are implemented and compared. This thesis investigates the numerous factors (like model architectures, hyper-parameters, image resolution) that can assist in lane detection. The data set is collected with laterally side mounted camera facing downward. The objectives of this thesis are detecting lane and after detection tracking algorithm is implemented to compensate images with missing lanes in an image sequence, also later distance estimation method using a look up table and neural network

All the proposed methods mentioned above successfully detect the lane in the given image sequences. The network architectures LaneSeg net and Generative net, which have F1 scores of 0.95 and 0.968 respectively, analyse the image in pixel wise has better accuracy in detection. On the other hand, it takes longer time to train and more inference time. While developing Generative net, it was difficult to train generator and discriminator together. On other hand, YOLO model was easy to train and requires lower processing power. But it fail to perform when lane size is too small.

The implemented linear Kalman filter successfully tracks missing lanes. The covariance matrix of the measurement noise and covariance of the process noise was started with some reasonable initial guess and then tuned experimentally. The filter was tuned to trust the measurements (i.e. detection outputs from models) more than the motion model (reference). Linear Kalman filter was evaluated qualitatively.

The above proposed methods had an approximate mAE of 28mm with cGAN and 32mm with YOLO in test data set with 442 images. YOLO was not able to detect lanes which occupied just few pixels in the entire image at threshold 0.5. This directly affected the detection performance as well as tracking performance. It had more outliers than the other two methods. YOLO has inference speed of 135 fps which is three times higher than the other two models.

6.2 Future Work

In this thesis, three models were developed for lane detection, tracking and lateral estimation under various illumination, shadow, worn out and missing lane conditions. Future work could include developing and evaluation of models for realistic weather and lighting situations like snow, rain, fog.

The best way to improve an object detection model is to gather more representative data. In order to expand the data set size, the organization should use realistic images. The training data set should ideally include images of objects in a range of scales, rotations, lighting conditions, viewpoints, and backgrounds.

YOLO is a state-of-the-art real-time object detection algorithm. Many more updated versions that have faster inference and higher detection accuracy are currently being developed. Most recently 7th version of YOLO has been launched, and it is state-of-the-art detector. This algorithm's performance can be evaluated by comparing it to that of other detectors, such as SSD and R-CNN.

cGAN is a powerful deep learning method that may be applied to applications involving object detection. But when it comes to real-time applications, its range is constrained. The unsupervised learning method and unstable training make it more challenging to train and provide output. For lane detection in online mode, a teacher-student training strategy is therefore suggested to reduce the network size without affecting prediction performance of the network.

A robust method to detect and suppress FP, in the events when lane marking is underneath the ego vehicle.

For converting image coordinates to world coordinates, the ROTAC tool developed in-house is used. However, it is not recommended as a reliable approach to deploy. It is necessary to create an algorithm that uses both intrinsic and extrinsic parameters of the camera to calculate the camera matrix. These parameters will help in removing the lens distortion effects and reconstruct 3-D images from both cameras.

Two laterally downward-facing cameras are installed on the apparatus. Distance calculation from the lanes will be more robust if the image data from both cameras is integrated. One camera's failure to capture lane information can be made up for by the other. In this approach, each frame's lane distance will be determined with greater accuracy.

Bibliography

[1] M. Mirza and S. Osindero, "Conditional generative adversarial nets," *CoRR*, vol. abs/1411.1784, 2014. [Online]. Available: http://arxiv.org/abs/1411.1784

[2] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

[3] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *European conference on computer vision*. Springer, 2016, pp. 21–37.

[4] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," *CoRR*, vol. abs/1505.04597, 2015. [Online]. Available: http://arxiv.org/abs/1505.04597

[5] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," 6 2014.

[6] C. Lee and J.-H. Moon, "Robust lane detection and tracking for real-time applications," *IEEE Transactions on Intelligent Transportation Systems*, vol. 19, no. 12, pp. 4043–4048, 2018.

[7] M. Aly, "Real time detection of lane markers in urban streets," *CoRR*, vol. abs/1411.7113, 2014. [Online]. Available: http://arxiv.org/abs/1411.7113

[8] J. McCall and M. Trivedi, "Video-based lane estimation and tracking for driver assistance: survey, system, and evaluation," *IEEE Transactions on Intelligent Transportation Systems*, vol. 7, no. 1, pp. 20–37, 2006.

[9] T.-Y. Sun, S.-J. Tsai, and V. Chan, "Hsi color model based lane-marking detection," in *2006 IEEE Intelligent Transportation Systems Conference*, 2006, pp. 1168–1172.

[10] Z. Teng, J.-H. Kim, and D.-J. Kang, "Real-time lane detection by using multiple cues," in *ICCAS 2010*, 2010, pp. 2334–2337.

[11] Z. Ying, G. Li, S. Wen, and G. Tan, "ORGB: offset correction in RGB color space for illumination-robust image processing," *CoRR*, vol. abs/1708.00975, 2017. [Online]. Available: http://arxiv.org/abs/1708.00975

[12] Z. Sun, "Vision based lane detection for self-driving car," in *2020 IEEE International Conference on Advances in Electrical Engineering and Computer Applications(AEECA)*, 2020, pp. 635–638.

[13] J. Canny, "A computational approach to edge detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-8, no. 6, pp. 679–698, 1986.

[14] C. Akinlar and C. Topal, "Edlines: Real-time line segment detection by edge drawing (ed)," in *2011 18th IEEE International Conference on Image Processing*, 2011, pp. 2837–2840.

[15] A. Borkar, M. Hayes, and M. T. Smith, "A novel lane detection system with efficient ground truth generation," *IEEE Transactions on Intelligent Transportation Systems*, vol. 13, no. 1, pp. 365–374, 2012.

[16] M. A. Ponti, L. S. F. Ribeiro, T. S. Nazare, T. Bui, and J. Collomosse, "Everything you wanted to know about deep learning for computer vision but were afraid to ask," in *2017 30th SIBGRAPI Conference on Graphics, Patterns and Images Tutorials (SIBGRAPI-T)*, 2017, pp. 17–41.

[17] J. Kim and M. Lee, "Robust lane detection based on convolutional neural network and random sample consensus," in *Neural Information Processing*, C. K. Loo, K. S. Yap, K. W. Wong, A. Teoh, and K. Huang, Eds. Cham: Springer International Publishing, 2014, pp. 454–461.

[18] A. Soni, P. Padamwar, and K. R. Konda, "Contextual road lane and symbol generation for autonomous driving," *CoRR*, vol. abs/2201.07120, 2022. [Online]. Available: https://arxiv.org/abs/2201.07120

[19] M. Ghafoorian, C. Nugteren, N. Baka, O. Booij, and M. Hofmann, "EL-GAN: embedding loss driven generative adversarial networks for lane detection," *CoRR*, vol. abs/1806.05525, 2018. [Online]. Available: http://arxiv.org/abs/1806.05525

[20] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," 2014. [Online]. Available: https://arxiv.org/abs/1406.2661

[21] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[22] P. Soviany and R. T. Ionescu, "Optimizing the trade-off between single-stage and two-stage deep object detectors using image difficulty prediction," in *2018 20th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*. IEEE, 2018, pp. 209–214.

[23] M. Everingham, S. Eslami, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes challenge: A retrospective," *International journal of computer vision*, vol. 111, no. 1, pp. 98–136, 2015.

[24] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "Yolov4: Optimal speed and accuracy of object detection," *arXiv preprint arXiv:2004.10934*, 2020.

[25] C.-Y. Wang, A. Bochkovskiy, and H.-Y. M. Liao, "Yolov7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors," *arXiv preprint arXiv:2207.02696*, 2022.

[26] D. Neven, B. D. Brabandere, S. Georgoulis, M. Proesmans, and L. V. Gool, "Towards end-to-end lane detection: an instance segmentation approach," *CoRR*, vol. abs/1802.05591, 2018. [Online]. Available: http://arxiv.org/abs/1802.05591

[27] T.-P. Nguyen, V.-H. Tran, and C.-C. Huang, "Lane detection and tracking based on fully convolutional networks and probabilistic graphical models,"

in *2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, 2018, pp. 1282–1287.

[28] M. Li, "Lane detection based on deeplab," in *Proceedings of the 9th International Conference on Computer Engineering and Networks*, Q. Liu, X. Liu, L. Li, H. Zhou, and H.-H. Zhao, Eds. Singapore: Springer Singapore, 2021, pp. 481–487.

[29] S. Moujtahid, R. Benmokhtar, A. Breheret, and S.-E. Boukhdhir, "Spatial-unet: Deep learning-based lane detection using fisheye cameras for autonomous driving," in *Image Analysis and Processing – ICIAP 2022*, S. Sclaroff, C. Distante, M. Leo, G. M. Farinella, and F. Tombari, Eds. Cham: Springer International Publishing, 2022, pp. 576–586.

[30] S. Thrun, "Probabilistic robotics," *Communications of the ACM*, vol. 45, no. 3, pp. 52–57, 2002.

[31] D. M. Powers, "Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation," *arXiv preprint arXiv:2010.16061*, 2020.

[32] F. Pizzati, M. Allodi, A. Barrera, and F. García, "Lane detection and classification using cascaded cnns," 2019.

[33] P. L. X. W. Xingang Pan, Jianping Shi and X. Tang, "Spatial as deep: Spatial cnn for traffic scene understanding," in *AAAI Conference on Artificial Intelligence (AAAI)*, February 2018.

[34] F. Yu, H. Chen, X. Wang, W. Xian, Y. Chen, F. Liu, V. Madhavan, and T. Darrell, "Bdd100k: A diverse driving dataset for heterogeneous multitask learning," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 2636–2645.

[35] Y. Ma, X. Zhu, S. Zhang, R. Yang, W. Wang, and D. Manocha, "Trafficpredict: Trajectory prediction for heterogeneous traffic-agents," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 6120–6127.

[36] T. Lin, M. Maire, S. J. Belongie, L. D. Bourdev, R. B. Girshick, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft COCO: common objects in context," *CoRR*, vol. abs/1405.0312, 2014. [Online]. Available: http://arxiv.org/abs/1405.0312

[37] CVAT.ai Corporation, "Computer Vision Annotation Tool (CVAT)," 9 2022. [Online]. Available: https://github.com/opencv/cvat

[38] Z. Papakipos and J. Bitton, "Augly: Data augmentations for robustness," 2022.

[39] P. Simard, D. Steinkraus, and J. Platt, "Best practices for convolutional neural networks applied to visual document analysis," in *Seventh International Conference on Document Analysis and Recognition, 2003. Proceedings.*, 2003, pp. 958–963.

[40] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *European conference on computer vision*. Springer, 2014, pp. 740–755.

[41] A. Bochkovskiy, "Darknet: Open source neural networks in python 2020," in *https://github.com/AlexeyAB/darknet (accessed on 2 November 2020).*

[42] K. Li, L. Qin, Q. Li, F. Zhao, Z. Xu, and K. Liu, "Improved edge lightweight yolov4 and its application in on-site power system work," *Global Energy Interconnection*, vol. 5, no. 2, pp. 168–180, 2022.

[43] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Icml*, 2010.

[44] C.-Y. Wang, H.-Y. Mark Liao, Y.-H. Wu, P.-Y. Chen, J.-W. Hsieh, and I.-H. Yeh, "Cspnet: A new backbone that can enhance learning capability of cnn," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, 2020, pp. 390–391.

[45] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015. [Online]. Available: http://arxiv.org/abs/1512.03385

[46] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," 2015. [Online]. Available: https://arxiv.org/abs/1511.06434

[47] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014. [Online]. Available: https://arxiv.org/abs/1412.6980

[48] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Z. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," *CoRR*, vol. abs/1912.01703, 2019. [Online]. Available: http://arxiv.org/abs/1912.01703

[49] "Anaconda software distribution," 2020. [Online]. Available: https://docs.anaconda.com/

[50] J. Heaton, "Applications of deep neural networks," *CoRR*, vol. abs/2009.05673, 2020. [Online]. Available: https://arxiv.org/abs/2009.05673

[51] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. J. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Józefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. G. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. A. Tucker, V. Vanhoucke, V. Vasudevan, F. B. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *CoRR*, vol. abs/1603.04467, 2016. [Online]. Available: http://arxiv.org/abs/1603.04467

DEPARTMENT OF SOME SUBJECT OR TECHNOLOGY
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY