# Lane detection algorithm for autonomous vehicles using machine learning

Tee, Daniel Tean Ming

2021

# Lane Detection Algorithm for Autonomous Vehicles using Machine Learning (C092)


Tee Tean Ming, Daniel

**SCHOOL OF MECHANICAL AND AEROSPACE ENGINEERING**

**NANYANG TECHNOLOGICAL UNIVERSITY**

**YEAR 2020/2021**

# C092: Lane Detection Algorithm for Autonomous Vehicles using Machine Learning

SUBMITTED BY

**TEE TEAN MING, DANIEL**

SCHOOL OF MECHANICAL AND AEROSPACE ENGINEERING

A final year project report

presented to

**Nanyang Technological University**

In partial fulfilment of the

requirements for the

**Degree of Bachelor of Engineering (Mechanical Engineering)**

Nanyang Technological University

**Year 2020/2021**

# Abstract

Lane detection is an important component of any driver assistance or autopilot system. As such, it is pivotal to construct a robust lane detector capable of handling various on road conditions such as bad weather, faded lane markings, strong shadow, varied lighting and so on. Traditional technics involving image processing and model fitting showed a good ability to detect lane from its many distinctive features, however, oftentimes failed to do so in less than optimum conditions.

With the advent of machine learning and improvements in computing power, self-learning algorithms have been built to handle this complex task of extracting relevant characteristics and relating information to identify lanes. Nonetheless, such models are often computationally expensive, resulting in long training and prediction time. Also, a lot of training sample is required before models can achieve and outperform traditional methods.

This project seeks to improve upon previous machine learning techniques to create a model capable of detecting lanes through semantic segmentation. U-Net model was used as a benchmark and was revamped using VGG16 and MobileNetV2 as an encoder to create a 4-lane detector and classifier. Hough transform and guard zone were then introduced as post-processing operations to refine lines detected by model and allow for lane departure warning function, respectively.

# Acknowledgement

I hereby take this opportunity to express my sincere appreciation for all who have helped me make this project a success.

First and foremost, I would like to thank my supervisor, Asst. Professor Lyu Chen from the School of Mechanical and Aerospace Engineering for the opportunity to work on this project. Machine learning is one of the latest technology trends and this invaluable chance to get exposed to it has not just help me stay relevant in this dynamic technological landscape but has also taught me much about the utilization of big data to produce useful knowledge. Besides that, Professor Lyu has generously guided me throughout the entire journey of this project, meeting up every biweekly despite his busy schedule to discuss my project's latest development, improvements and provide constructive feedback.

Also, I would like to express my gratitude to Dr Zhong Xu for all the hours he has made available for me both in text and in consultation. His suggestions and feedback helped broaden my perspective to consider many different approaches to the problem and aided me much in various technical challenges encountered.

Lastly, I would like to thank the School of Mechanical and Aerospace Engineering for the opportunity to get exposed to research, to work on real-life problems and be guided in this journey of understanding out-of-curriculum advanced concepts through this Final Year Project. I am truly grateful for it and feel thoroughly blessed to have concluded my university journey in such a meaningful project.

# Table of Contents

# List of Figures

# List of Tables

# Chapter 1: Introduction

Worldwide vehicle ownership has been growing steadily year-on-year without signs of slowing down, this has led to more crowded roads and increased chance of being involved in a car accident. In a recent study, approximately 1.35 million people died from motor vehicle accidents with another 20 to 50 million people suffering non-fatal injuries, resulting in significant cost to both society and the individuals involved. [1] These accidents are primarily caused by human error such as inattention and distraction. [2] To reduce these preventable accidents and improve road safety, various Driver Assistance Systems (DAS) have been designed.

Driver assistance systems work by using the inputs from multiple sensor and detectors around the car to warn and provide aid to the driver. Among the type of help provided includes Adaptive Cruise Control (ACC), Lane Departure Warning (LDW), Lane Detection (LD), Blind Spot Monitoring (BSM) and many more. As lane detection has been proven to provide the greatest benefit to drivers and serves as the precursor for many complex driver assistance systems, this report would mainly focus on lane detection. [3]

As lane markings are a visual component, almost all recent study involving lane detection use visual sensor data from cameras or LiDARs. Presently, lane detection methods can be divided into two categories: the traditional image processing and machine learning methods. [4] The traditional image processing method involves feature extraction and then model fitting. Feature extraction depends on the identification of lane marking features for instance: its width, its distance from the next, its colours, edges, and textures to predict lane position. Model fitting then attempts to fit a specific mathematical line model on the lane positions detected to trace out possible lane curvature. For example, in a lane departure study by Jamel Baili [5], the camera image was first transformed into YCbCr format, smoothen using an average filter, had its edge features extracted using Sobel Filter and then modelled as straight lines using Hough Transform. This method achieved good lane detection however was

not robust enough to account for shadows and poor visibility conditions. On the other hand, the machine learning approach, specifically the Convolutional Neural Network (CNN), has shown to be significantly better than traditional image processing methods. [6] As machine learning techniques can learn from real-life data, they tend to be more robust and have shown to be capable of both lane detection plus lane classification. [7] However, this approach is more resource-intensive and requires a lot of training data. In fact, top autonomous vehicle companies such as Tesla, Google's Waymo and Einride use specialized chips, orders of magnitude faster than commercial graphical processing units (GPUs) and collect years of human labelled video footage to meet this demand.

This report aims to develop a low-cost, reliable plus robust lane classification and detection model using machine learning. A guard zone was also established in the algorithm to perform partial lane departure warning whenever the camera frame deviates from lane markings over a certain threshold value. The guard zone was calibrated based on the camera position in the world coordinate frame. By building a robust and less computationally expensive lane detection algorithm, this report also serves as a catalyst to enable commercially available dashboard cameras to be equipped with lane detection and lane departure warning capabilities.

# Chapter 2: Literature Review

Lane detection has been widely studied across the year and more so in recent years as the car industry gradually moves towards autonomy. In this literature review, we will explore all techniques involved in lane detection to integrate them into our final proposed lane detector.

## 2.1 Feature Extraction

Feature extraction is a technique used to shortlist potential candidates to be further processed into useful information. In the case of lane detection, lane markings constitute many features that could help identify lanes. Among them include lane boundary or edges, lane colour, lane characteristics and texture. [8]

### 2.1.1 Lane Boundary/Edges

Edges in images can be identified as small regions with an abrupt change in pixel intensity. As lane markings are distinctive from their background road surface, edges can be easily extracted using a whole range of gradient-based filters and their variants. The procedure involves:

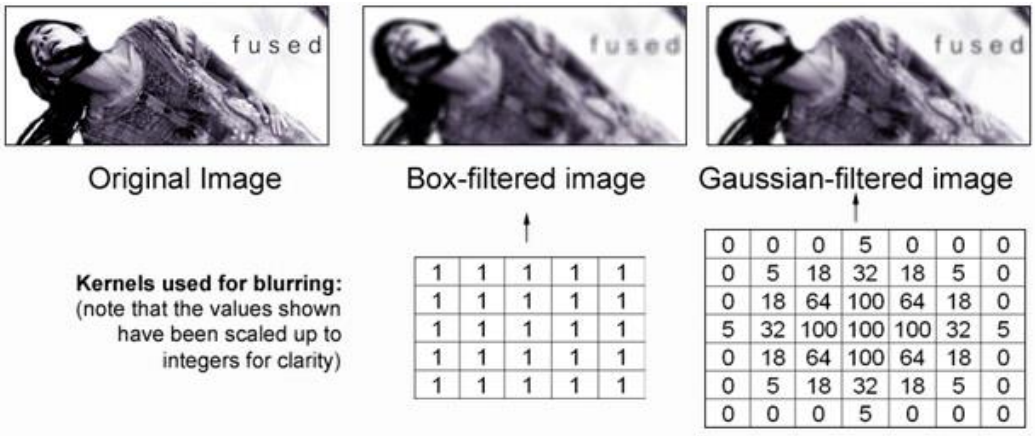


*Figure 1: Box Filter and Gaussian Filter [9]*

1. Smoothing of the image using box filter or Gaussian filter to reduce noise as shown in Figure 1.
2. Detecting edge points by performing a specific operation to extract potential edge candidates.
3. Selecting true members from edge point candidates that best represent lane.

Current edge detection operations include Sobel, Canny and steering filter. Sobel works by passing two 3x3 kernels through the original image to calculate approximations of derivatives in both the horizontal and vertical direction as shown in Figure 2. In a study by Q. Lin et al. [11], the input image was first smoothed using a 5x5 Gaussian filter before performing a Sobel operation to obtain an edge map containing edge direction and magnitude. True lane boundaries were then extracted by applying non-local maximum suppression (NLMS) on the edge map. The steering filter as shown in Figure 3 works like Sobel but utilizes three 3x3 kernels to obtain a directional response in any direction. This method was used to accurately extract many complex lane curvatures with robustness to different levels of illumination, road marking variations and shadowing. [13] Canny edge detection uses threshold on pixel magnitude and path to distinguish edges as shown in Figure 4. This technique was able to effectively merge and form a continuous edge line across the entire lane boundary. [14]

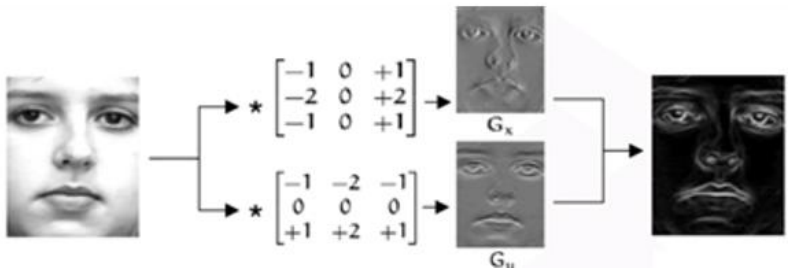


*Figure 2: Sobel Filter [10]*

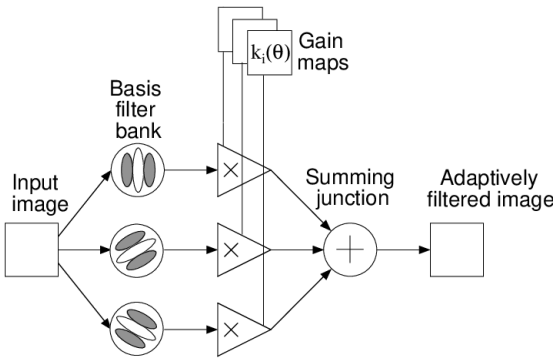


*Figure 3: Steerable Filter [12]*

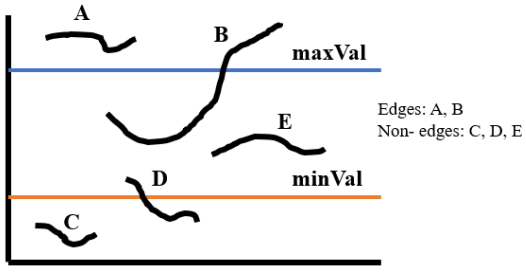


*Figure 4: Canny Edge Detector*

### 2.1.2 Lane Colour

Besides, edge feature extraction much research has also been done on colour-based feature detection. This feature utilizes the knowledge of lane markings being uniquely different from road. A histogram-based approach using grayscale pixel intensity was proposed in Chiu and Lin [15]. By converting the image into grayscale and plotting pixel magnitude on a histogram, an adaptive threshold could be computed to locate lane markings, such thresholding also proved to be more robust in detecting faded lanes. In a study by P.C. Wu et al. [16], YCbCr colour space transformation was proposed as most information is reserved in the Y(luminance) component. This greatly reduced computational load and storage space while maintaining its detection accuracy. Besides that, HSV colour space transformation was also shown to increase contrast and improve accuracy. [17] As lane colours are easily affected by environmental conditions, these detection strategies tend to be not as robust as edge detection methods.

### 2.1.3 Lane Characteristics and Texture

Research in this area for lane detection is rare as lane characteristics may vary from country to country, each adopting a different set of lane parameters. However, these features are often able to serve as an auxiliary input to improve prediction, remove false positives and so on. In a study by X. Shi et al. [18], lane-specific characteristics were able to be utilized to classify detected pixel blocks into lane marking or not. Three characteristics were used namely the gap between intensity peaks (lane marking width), distance between the two largest histogram bins (lane width) and the lane angle. Besides, Tseng and Lin [19] reinforce the viability of this technique by using lane width, lane angle and lane interception location on the x-axis to remove false positives obtained from their conjugated Gaussian model. Road texture also has been studied as such features are found to provide strong directional information. In a study by Rasmussen and Korah [20], Gabor filters were used to pick up directional components made by wheels of the preceding car. The focus of expansion was then estimated from directional information to finally identify lane markings.

## 2.2 Model Fitting

The main objective of this step is to find a model that can best represent the lane lines from the lane markings identified. In this section, we will be discussing different lane models and fitting methods studied over the years.

### 2.2.1 Lane Models

Lane models can be classified into three categories: parametric, semi-parametric and hybrid models.

Parametric models are finite-dimensional models that use simple geometric elements like straight lines, generic circumference arcs, parabolic curves, and hyperbolic polynomial curves to trace out lanes. As a car has a limited steering angle, these parametric models tend to provide a good approximation of lanes. For example, straight-line models were used in A. Borkar et al. [21] to accurately predict lanes close to the vehicle. In Y. Jiang et al. [22], Inverse Perspective Mapping (IPM) was also applied on image to remove perspective distortions before modelling lanes using generic circumference arc.

Semi-parametric models contain both finite- and infinite- dimensional components that are not limited by specific global geometry path, allowing them to have better lane fitting than parametric models. However, this also causes overfitting and unrealistic lane approximation if not performed cautiously. Among the most popular models would include spline plus its variations and active contours (snakes). Splines and their variations are smooth piecewise polynomial functions each with its unique properties. In ZuWhan Kim [23], lanes were modelled with cubic splines as curve produced would pass through all control points, allowing lane markings belonging to the same lane have their trajectory mapped. A lane modelled using cubic hermit spline was proposed in Huang et al. [24]. This type of spline utilizes tangent extracted from lane features to contribute to directionality and curvature of the curve as it passes through the other control points, resulting in a more realistic lane fit. Snakes are active models that dynamically contort their shape and position to achieve a state of minimal energy. The energy value of the snake depends on internal energy (continuity and smoothness), image energy (intensity, gradient, and termination) and constraint energy (user-defined

constrain to guide snake curvature). [25] A lane model using snakes was proposed in Sawano and Okada [26]. By minimizing snake energy, a smooth curve connecting control points that is robust to noise and broken boundaries was produced.

Hybrid models were also proposed in several studies to leverage the knowledge that near field lanes are roughly straight lines and curvature only take place at far-field. In K.H. Lim et al. [27], the image frame was divided into two regions with the closer region being modelled by straight lines and further region a river-flow method.

### 2.2.2 Fitting Methods

Modelling of lanes is highly dependent on the control points obtained from feature extraction, therefore it is important to make sure that noise from missing data and outliers are filtered out. Among the fitting methods, we will explore in this report include RANdom SAmpling Consensus (RANSAC), Least Square Optimization, Hough Transform and Density Based Spatial Clustering of Application with Noise (DBSCAN) algorithm.

#### 2.2.2.1 RANSAC

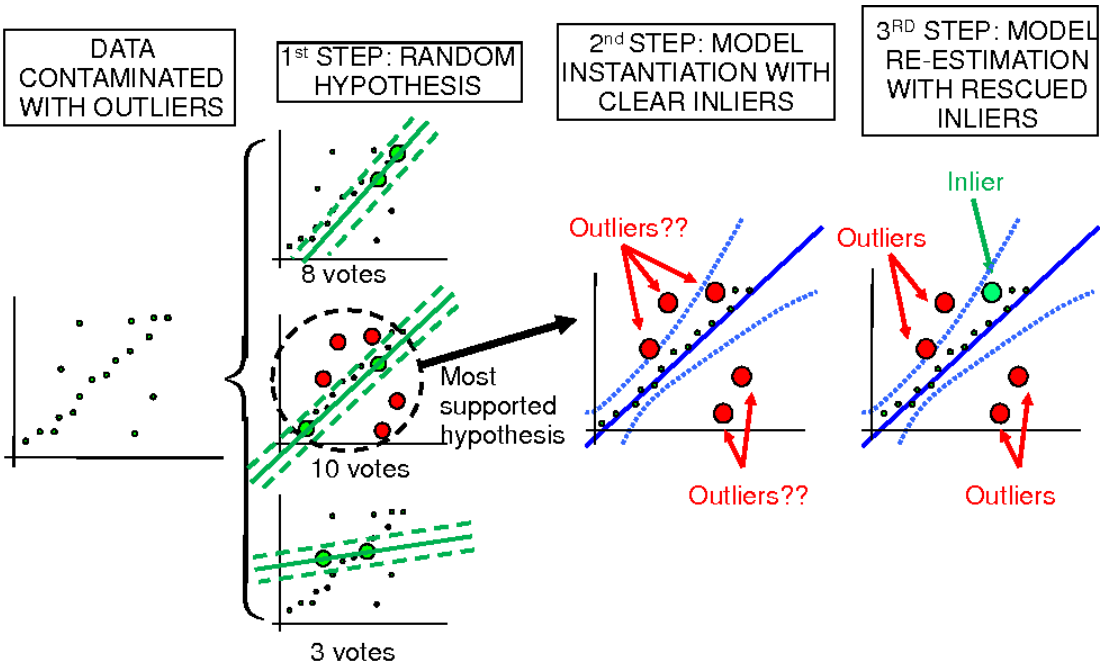


*Figure 5: RANSAC [28]*

RANSAC is an iterative method that uses a voting scheme to find the most representative fitting in observed data. It works based on the assumption that the bulk

of observations constitute inliers and that there are enough features to agree on a good model (few missing data). RANSAC algorithm procedure is as follows: [29]

1. Randomly select the minimum number of points required to determine model parameters (e.g., 2 for line model, 3 for cubic model etc.)
2. Solve the parameters of the model (e.g., form straight line between randomly selected points for line model)
3. Determine consensus set (number of points that fit model based on a predefined tolerance, ε)
4. If the ratio of a consensus set over the total number of points exceeds the predefined threshold, τ, model parameters are recalculated using identified inliers and operation is complete. Else, step 1 to 4 is repeated based on the predefined number of iterations, N.

A visual representation of the RANSAC algorithm is shown in Figure 5.

This fitting method is the most popular in lane detection applications for its robustness against noisy data. In A. Borkar et al. [21], linear modelled RANSAC was used to detect lanes. In another study by Yim and Oh [30], spline modelled RANSAC was iterated 100 times to fit lanes.

2.2.2.2 Least Square Optimization

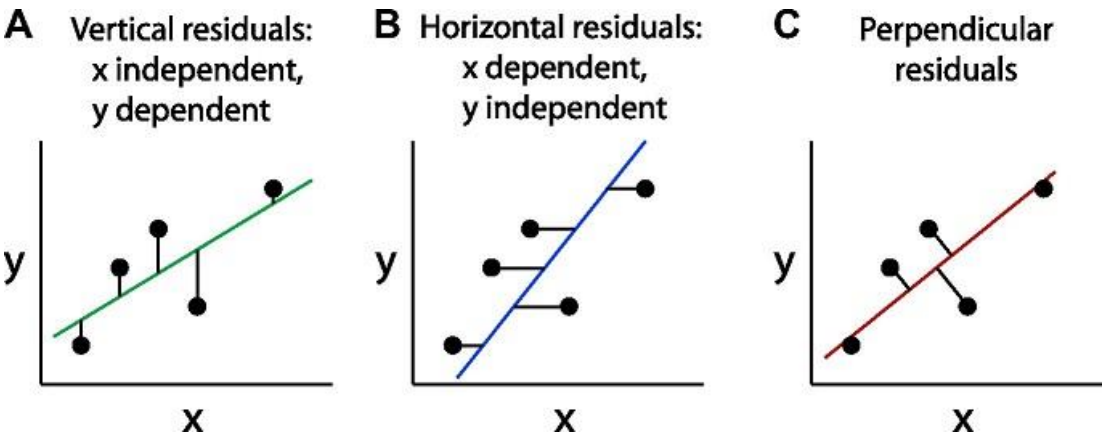


*Figure 6: Least Square Optimization [31]*

The least-square is a mathematical procedure used to find the optimum curve based on observed data by minimizing the sum of the squares of offset points from the curve. The offset could be calculated vertically, horizontally, or perpendicularly based on the

desired fitting parameter (e.g., vertical offset estimates best fitting y for given x) as shown in Figure 6. This fitting method involves several steps:

1. Randomly initialize a model to fit (e.g., linear, or non-linear)
2. Calculate offset between points and chosen model.
3. Calculate the sum of square offset.
4. Calculate and set the gradient of the sum of squares to zero to obtain the best fitting model.

The least-square optimization provides good model fitting in some cases but is susceptible to outlier interference, making it less favoured for lane detection applications. In F. Bounini et al. [32], lane line was fitted using the least-square method in a simulated environment. An integrated fitting method combining RANSAC, and least square optimization was proposed by M. Nieto et al. [33] to enhance robustness to outliers.

2.2.2.3 Hough Transform

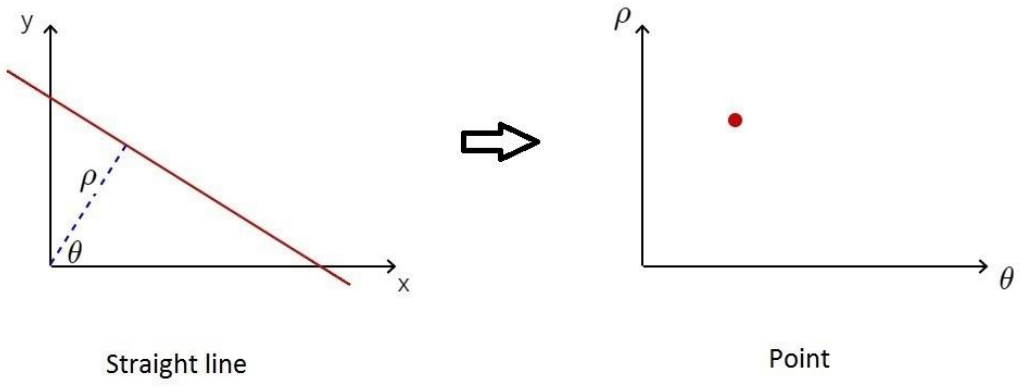


*Figure 7: Image Space and Hough Space [34]*

Hough transform is a technique that performs an explicit voting procedure to extract straight lines from isolated features within an image. It uses two parameters namely ρ, the orthogonal distance from the origin to line and θ, the angle between horizontal axis and distance line to represent lines in the hough space as shown in Figure 7. For hough transform to identify a set of points as belonging to the same straight line minimum number of intersections, minimum line length and minimum line gap are predefined.

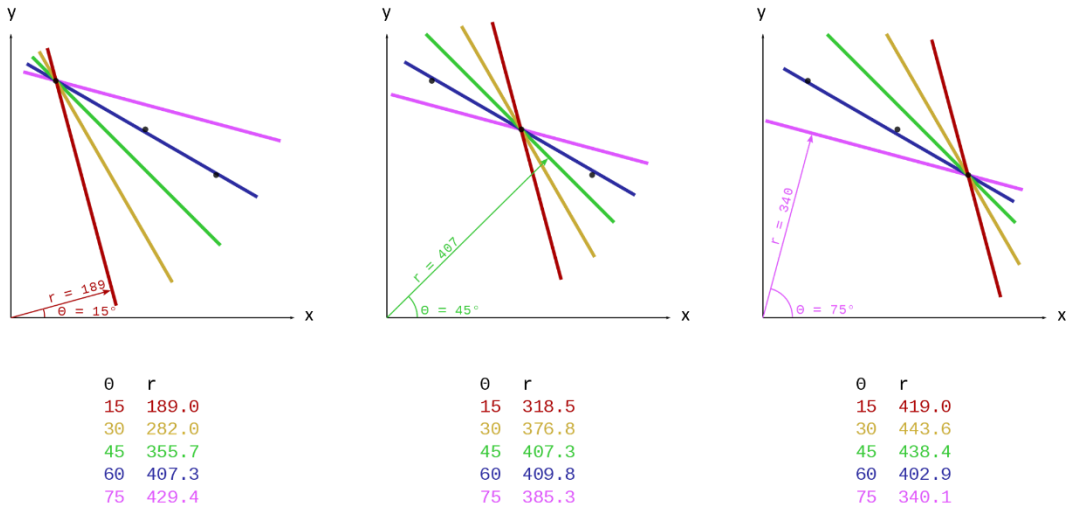


*Figure 8: Hough Transform [35]*

The algorithm then proceeds as follows to extract straight lines:

1. Several lines of different angles are plotted through each data point.
2. The hough parameters for each line are then calculated.
3. The number of intersections on lines are then accumulated based on hough parameters calculated. (i.e., All 3 blue line's $r$ and θ parameters are roughly similar, therefore the number of intersections is 3).
4. Lines that fulfil all three requirements defined above are then considered straight lines.

A graphical representation of these steps is shown in Figure 8.

Currently, hough transform has been used extensively for linear model fitting due to its robustness to noise and occlusion. In X. An et al. [36], hough transform was first used to detect the vanishing point under the assumption that the car is parallel to the road. This point was then used to support linear modelling of lanes. A novel approach integrating hough transform, RANSAC and B-spline model was proposed by M. Aly [37]. Such a fusion allowed for multiple lane fitting, with the near field being governed by hough transform and B-spline modelled RANSAC dealing with lanes in the far-field.

2.2.2.4 DBSCAN

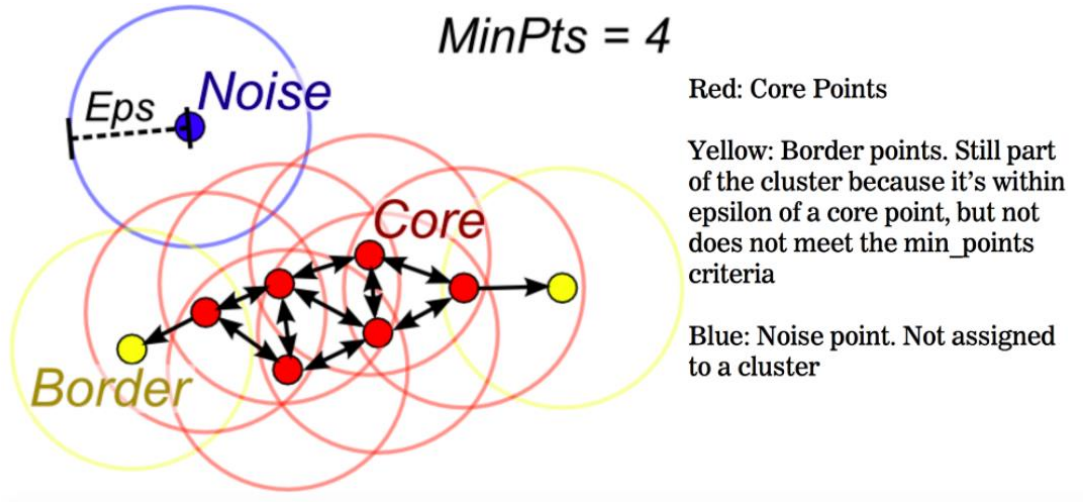


*Figure 9: DBSCAN [39]*

DBSCAN is a data clustering algorithm used to extract relevant data points and filter off outliers. It uses two predefined parameters namely ε, minimum distance to be considered part of cluster and minPts, minimum cluster size to separate areas of high density with the area of low density. These parameters are carefully chosen to obtain useful data with minPts value generally being the number of dimensions in the data set and ε value being chosen using a k-distance graph. DBSCAN algorithm works by [38]

1. First locating neighbours of every point (non-self-points within the ε of a point)
2. Identifying core points (points with more than minPts neighbours)
3. Locate points connected to core points while ignoring non-core points.
4. Assign all non-core points to a nearby cluster if it happens to be in ε range of it. Else, consider points as outliers.

Figure 9 shows these steps in a clear visual.

Despite being a wildly popular data clustering technique, its application in lane detection has been few. In J. Niu et al. [40], it was used in a two-stage feature extractor alongside hough transform to classify features obtain from hough transform into ego lane or not.

## 2.3 Machine Learning

Machine learning is the study of computer algorithms that can learn from past encounters and experiences. Such algorithms offer endless possibilities in applications as human developers are no longer required to identify important and relevant features or design elaborate decision trees to perform the desired task. Instead by provided lots of data and training samples, machine learning models can learn to extract relevant information, processes them, and engenders desired knowledge.

### 2.3.1 Types of Machine Learning

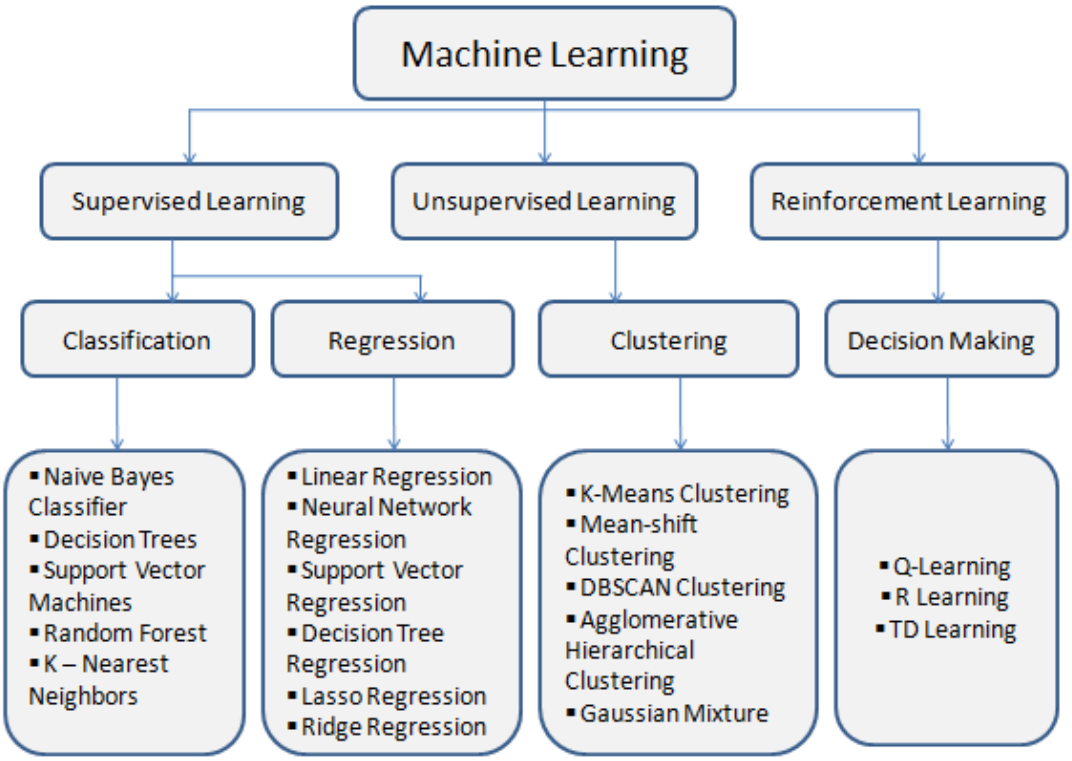


*Figure 10: Types of Machine Learning [41]*

Machine learning can be classified into 3 main types as shown in Figure 10:

1. Supervised learning

   The model learns from a large amount of labelled data and generalizes its understanding to respond correctly to new data. This learning method is especially useful when the desired target clearly understood (e.g., identifying different types of cars). Supervise learning can further be broken down into 2 problem types:

a) Classification

   For such a problem, the model is trained to classify input as discrete values (e.g., classifying animal images into dog, cats, and rats)

b) Regression

   In such cases, the model is trained to map input values to continuous output. Such outputs are not definitive and usually spread over a large spectrum. (e.g., predicting the future stock price of IBM)

2. Unsupervised learning

   The model learns from unlabelled data to identify patterns and similarities to split data into unique categories. Such a learning method useful to organise data and is widely used in customer segmentation (clustering customer into a different group of similar spending patterns, habits, etc) and genetics (grouping different DNA patterns according to its human feature contribution).

3. Reinforcement learning

   The model learns by interacting with the environment it is being placed in to maximize positive response or reward in the situation. Such a learning method is very useful when environmental variables can be quantified, there is a clear reward function and there is enough time for the model to learn through trial and error. The most notable example of its use case is AlphaGo.

As we have a clear understanding of lane and non-lane components, almost all machine learning lane detectors used a guided learning approach (i.e., supervised learning). In A. Gurghian et al. [42], an end-to-end lane detection method was proposed using a deep CNN trained on inputs from two front-facing cameras. A total of 120,000 images were used in the training with 80,000 of them being human labelled and 40,000 others computer generated. This sheer amount of labelled data available for training helped the model achieve a detection accuracy of 99%.

Despite data size being a major contributor to model accuracy, a well-structured model architecture is equally important for training to take place efficiently. Among the models commonly used for lane detection include CNN and Recurrent Neural Network (RNN) as seen in J. Li et al. [7]. These models have a different function but are built upon similar building block in which we will explore in the later section.

### 2.3.2 Layers of Machine Learning Models

Layers are high-level building blocks of a model. Each layer performs a specific operation and when organised into a network help machine learning models be capable of feature extraction, clustering of representative information and extracting relevant knowledge.

#### 2.3.2.1 Input Layer

This is the most fundamental layer as it is required to accept input. This layer comes in several configurations depending on input type (e.g., 2-D image, 3-D image, sequenced data input, etc.)

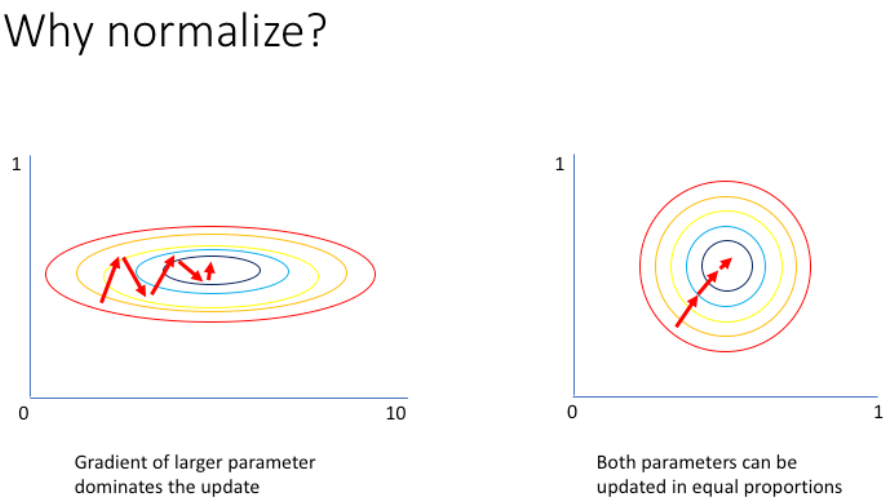


*Figure 11: Normalization [43]*

This layer also performs normalization to data, enabling data to be processed on a common scale. This is especially important for data with features of huge range disparities, as features of a larger scale would have greater influence over other features leading to biased results as seen in Figure 11. In J. Jin et al. [44], input normalization has also proven to increase training speed.

#### 2.3.2.2 Convolutional Layer

This layer works as a filter that passes over input to extract relevant features. It mainly consists of 3 components:

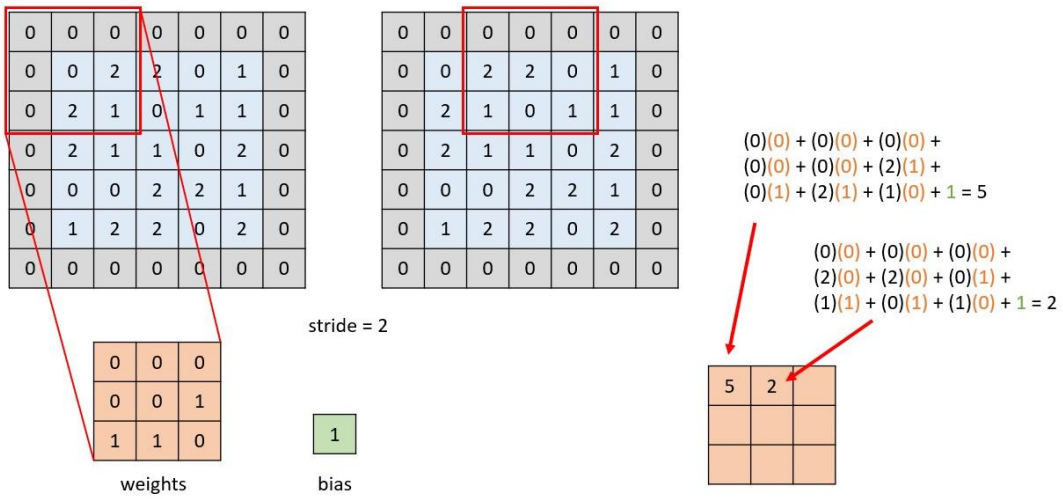


*Figure 12: Convolution and Stride [45]*

1. Kernel (Filter)

   This is the feature extractor capable of capturing certain input characteristics. It works by performing a linear operation, $z = wx + b$ , where w and b are trainable weights and biases over the entire kernel region. By using the same filter over different sections of an input, a map of activation can be obtained (feature map), indicating the strengths and location of a certain feature. Each convolution layer can have many feature extraction filters and as layers get deeper, weights and biases are manipulated to extract more complicated features.

2. Stride (Step)

   This governs the movement of filter over input. (e.g., stride = 1, would cause the filter to move one step at a time in the vertical or horizontal direction) By adjusting this parameter, the output size can be significantly reduced. Figure 12 shows the feature map obtained by a 3x3 kernel with stride 2.

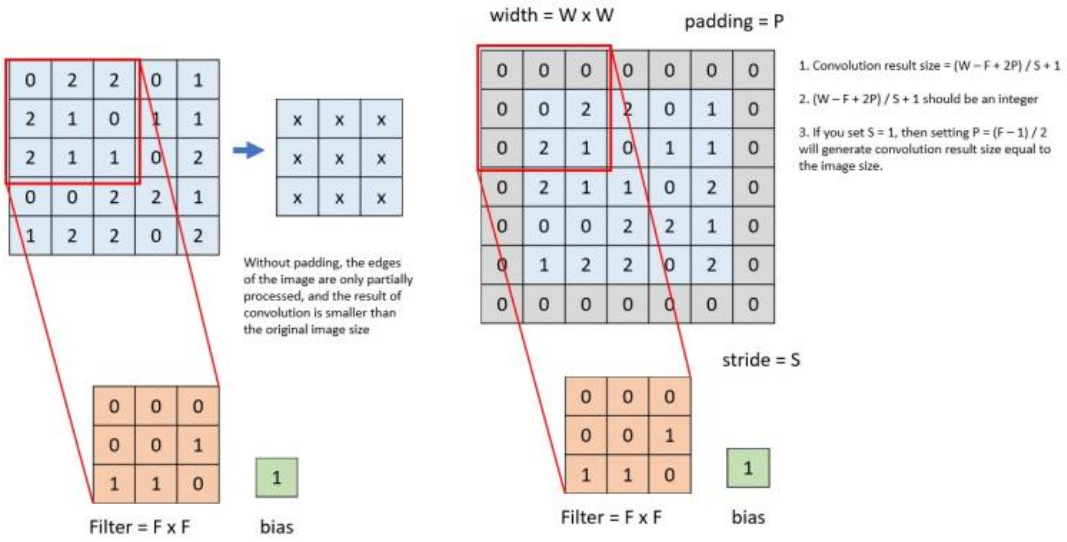


*Figure 13: Padding [46]*

3. Padding

   This is used to preserve dimensionality after convolution by adding zero values around the input image as shown in Figure 13.

2.3.2.3 Dropout Layer

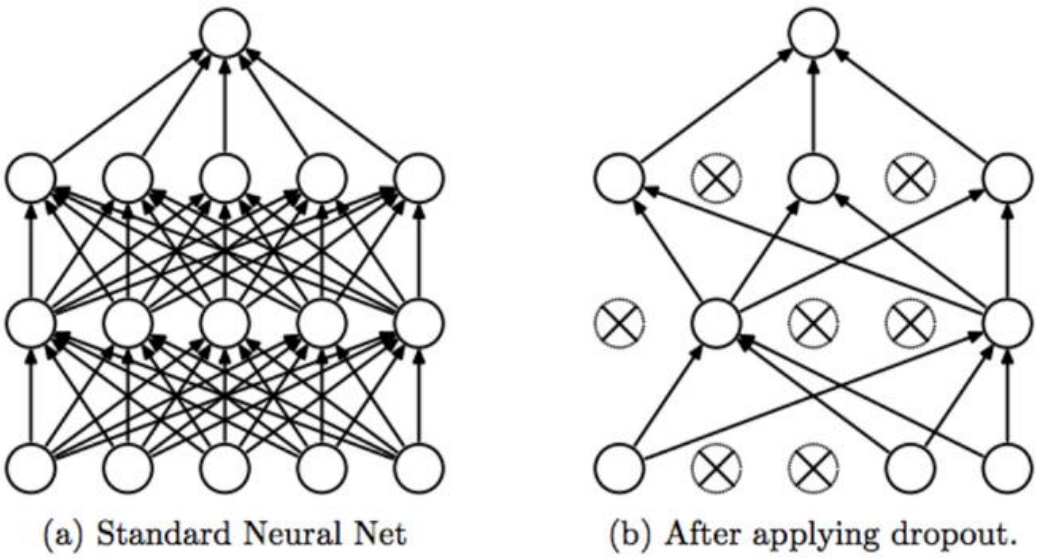


*Figure 14: Dropout [47]*

As shown in Figure 14, this layer randomly removes a fraction of input by nullifying input elements. This introduces noise that disrupts the over-reliance of few inputs, preventing overfitting and improving generalization. [47]

2.3.2.4 Pooling Layer

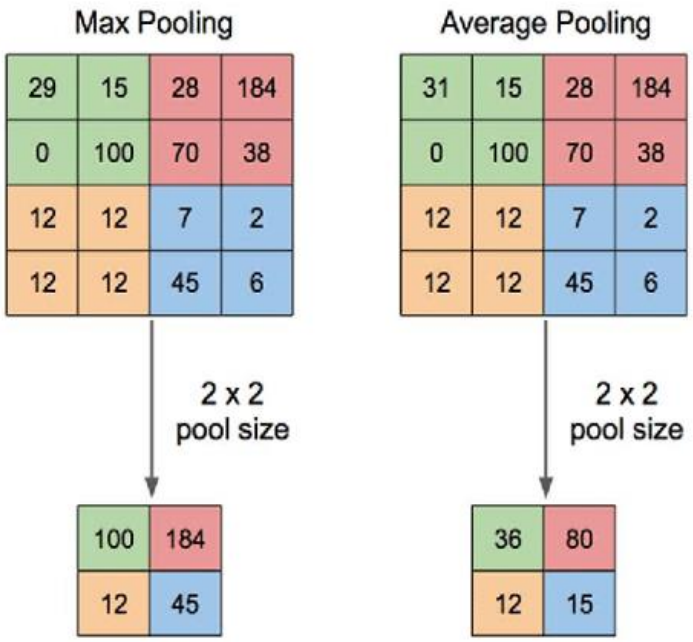


*Figure 15: Max Pooling and Average Pooling [48]*

High dimensional inputs greatly increase computation parameters causing overfitting and slow training speeds. The pooling layer serves as a method to reduce dimensionality by averaging or maximum values within a moving kernel as shown in Figure 15. This reduces overfitting as size reduction helps lessen features' location sensitivity from convolutional layers. [49]

2.3.2.5 Activation Layer

This layer is the source of machine learning's power and grants it the ability to recognize complicated patterns, it is applied alongside every other layer. By transforming values over complex non-linear functions, a model can be taught to understand convoluted connections between features and desired knowledge. There are many types of activation functions, the more popular ones being:

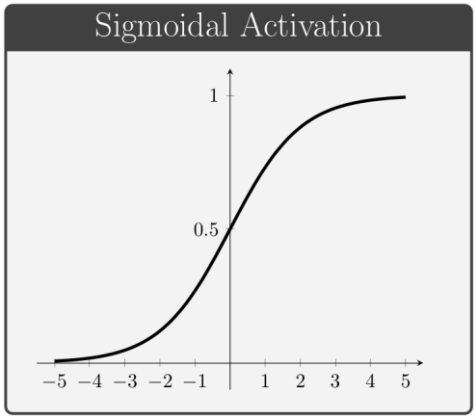


*Figure 16: Sigmoid Activation [50]*

1. Sigmoid

   This activation function can be modelled as $f(x) = \frac{1}{1+\exp(-x)}$ as shown in Figure 16. It has a smooth gradient and produces outputs within the 0 to 1 range. This makes it ideal for binary classification problems. However, due to function having small to no gradient at the lower and upper extremes, the vanishing gradient problem is a common occurrence, leading to slow learning speeds.

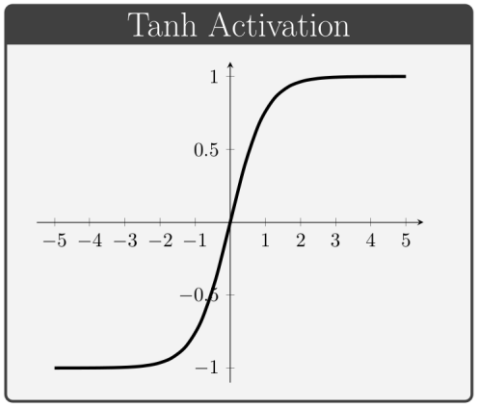


*Figure 17: Tanh Activation [50]*

2. Tanh/ Hyperbolic Tangent

   This activation function has an equation $f(x) = \tanh(x) = (\frac{2}{1+\exp(-2x)} - 1)$ and can be visualized as shown in Figure 17. As it is zero centred, the function gradient does not bias towards a particular direction. However, like sigmoid, it too suffers from vanishing gradient problem.

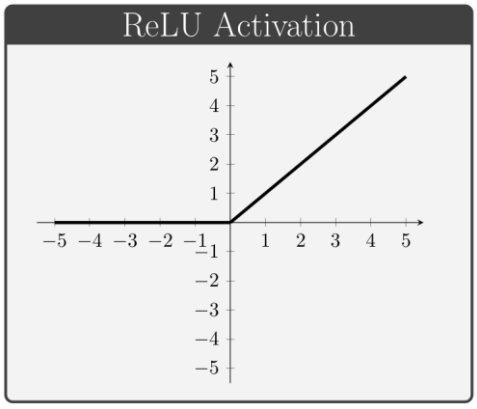


*Figure 18: ReLU Activation [50]*

3. Rectified Linear Unit (ReLU)

   This is the most popular activation function with the general equation being $f(x) = \max(0, x)$ as shown in Figure 18. It is computationally efficient and does not suffer from a vanishing gradient. Even though it has zero gradient for negative values, it still functions well for most application.

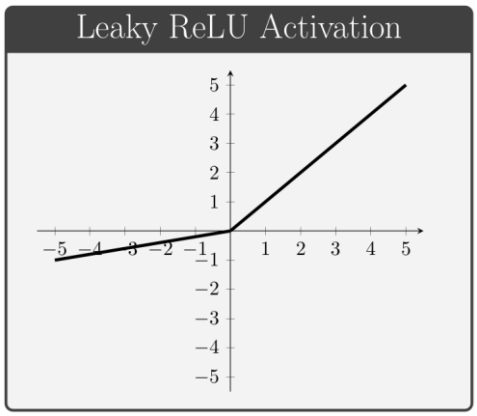


*Figure 19: Leaky ReLU Activation [50]*

   For special cases where input generally trends to zero or negative values, the Leaky ReLU function was proposed. [51] This function is shown in Figure 19 with the equation $f(x) = \max(0.01x, x)$ and helps solved the dying ReLU problem.

2.3.2.6 Up-Sampling Layer

This layer serves to undo pooling layers and regain spatial information from input. Such a layer is often encountered in semantic segmentation applications due to the need of mapping learned features to the original image. There are 3 types of methods to perform up-sampling:

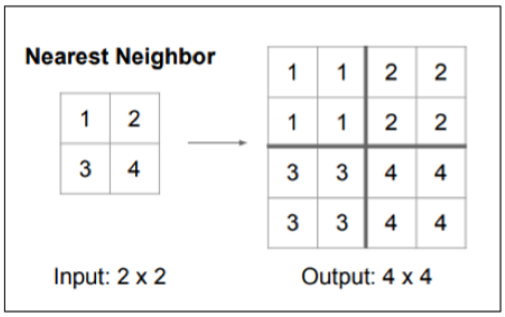


*Figure 20: Nearest Neighbour [52]*

1. Nearest Neighbour

   Every input value is expanded to its adjacent pixel as shown in Figure 20.

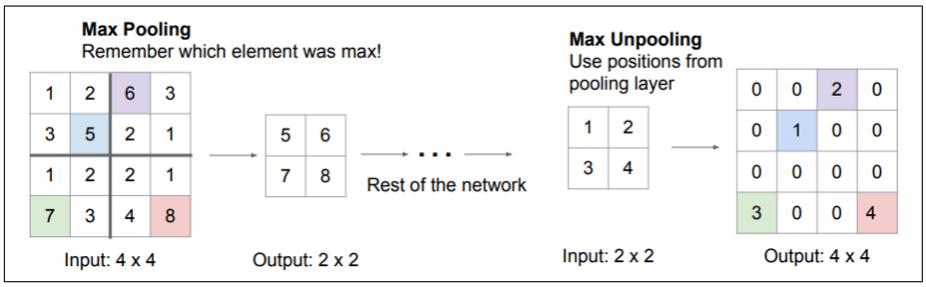


*Figure 21: "Bed of Nails" [52]*

2. "Bed of Nails"

   The input value is remapped to its pre-pooling position with the expanded regions zeroed as shown in Figure 21.

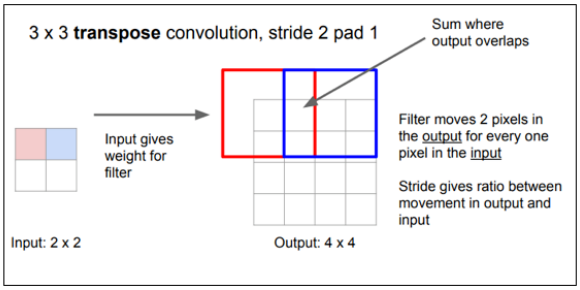


*Figure 22: Transpose Convolution [52]*

3. Transpose Convolution

   The most popular form of un-pooling as trainable parameters is assigned to kernel used to upsize input as shown in Figure 22. This allows the layer to learn relevant information required by the next layers to optimize model performance.

# Chapter 3: Methodology

## 3.1 Pipeline

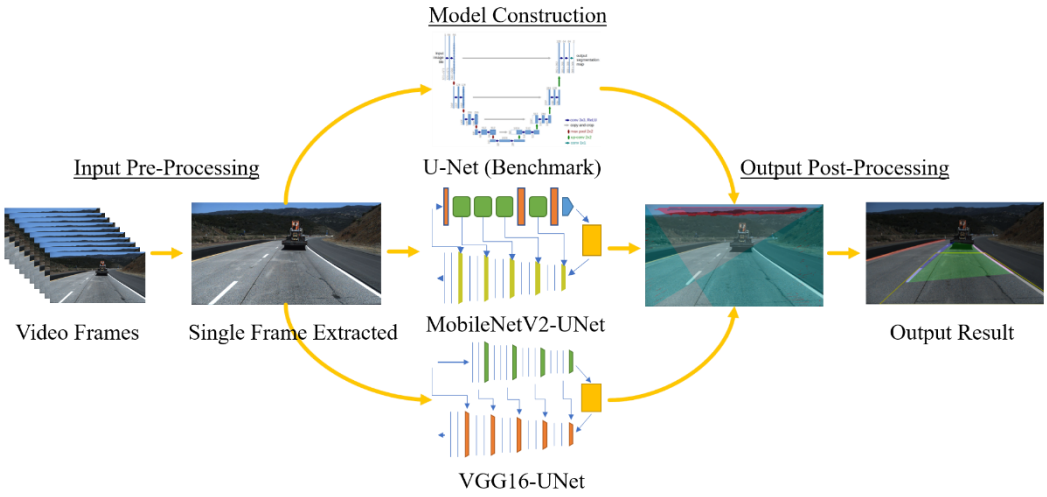


*Figure 23: Overview of Pipeline*

Figure 23 shows the overview of process pipeline, there are 3 stages involved. In the following section, input label generation, model build-up and output post-processing will be explored.

### 3.1.1 Input Label Generation

As videos from different distribution were of different image quality, file format and dimension. The VLC video player was first used to extract video frame in 720 x 1280 dimension and jpg format creating a uniform dataset throughout training samples. TuSimple's json label configuration was different from that generated by VIA, however, the process of creating trainable labels is similar. The steps are as follows:

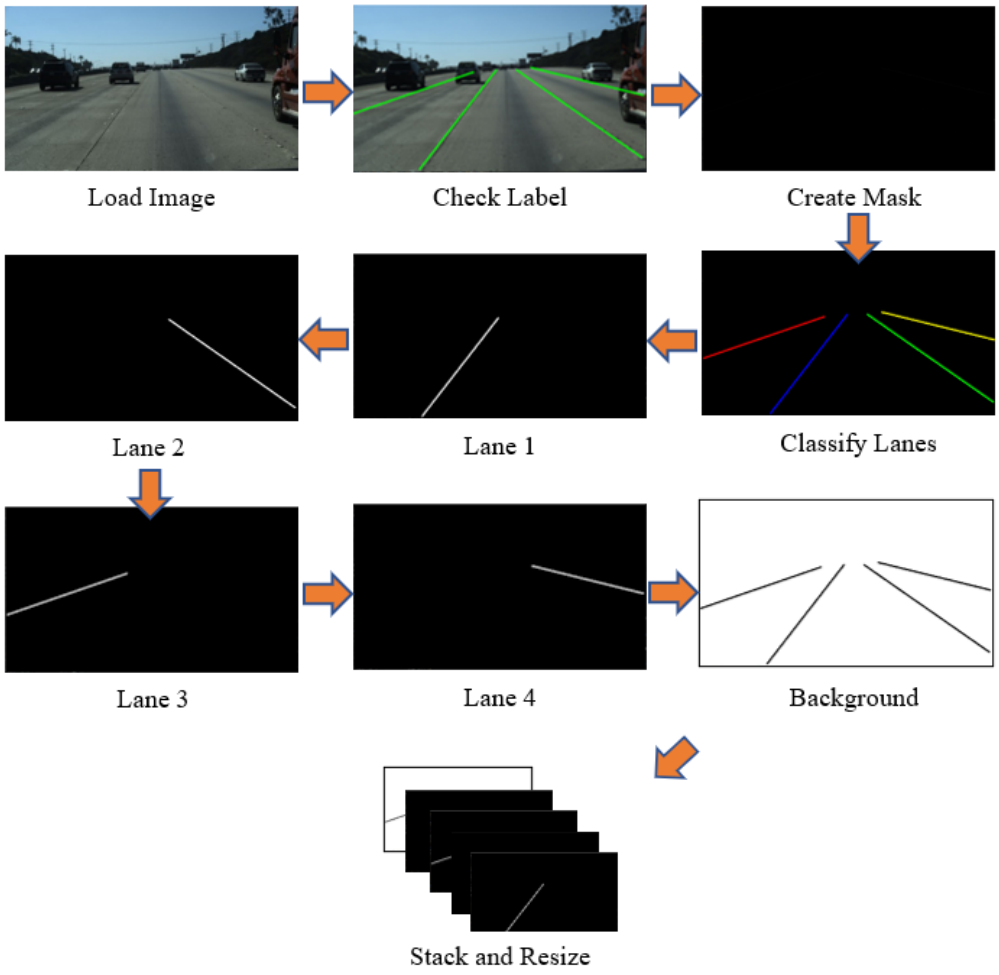


*Figure 24: Label Generation Process*

Code implementation:

Reading labels from TuSimple json file and generating semantic segmentation labels for training.

*Input: Json object containing labels from TuSimple (i.e., json_load(<file_path>))*
*Output: n×h×w×5 numpy array containing one-hot labels of 4 lanes and background,*
*where n = no. of labels, h = IMG_HEIGHT, w = IMG_WIDTH*

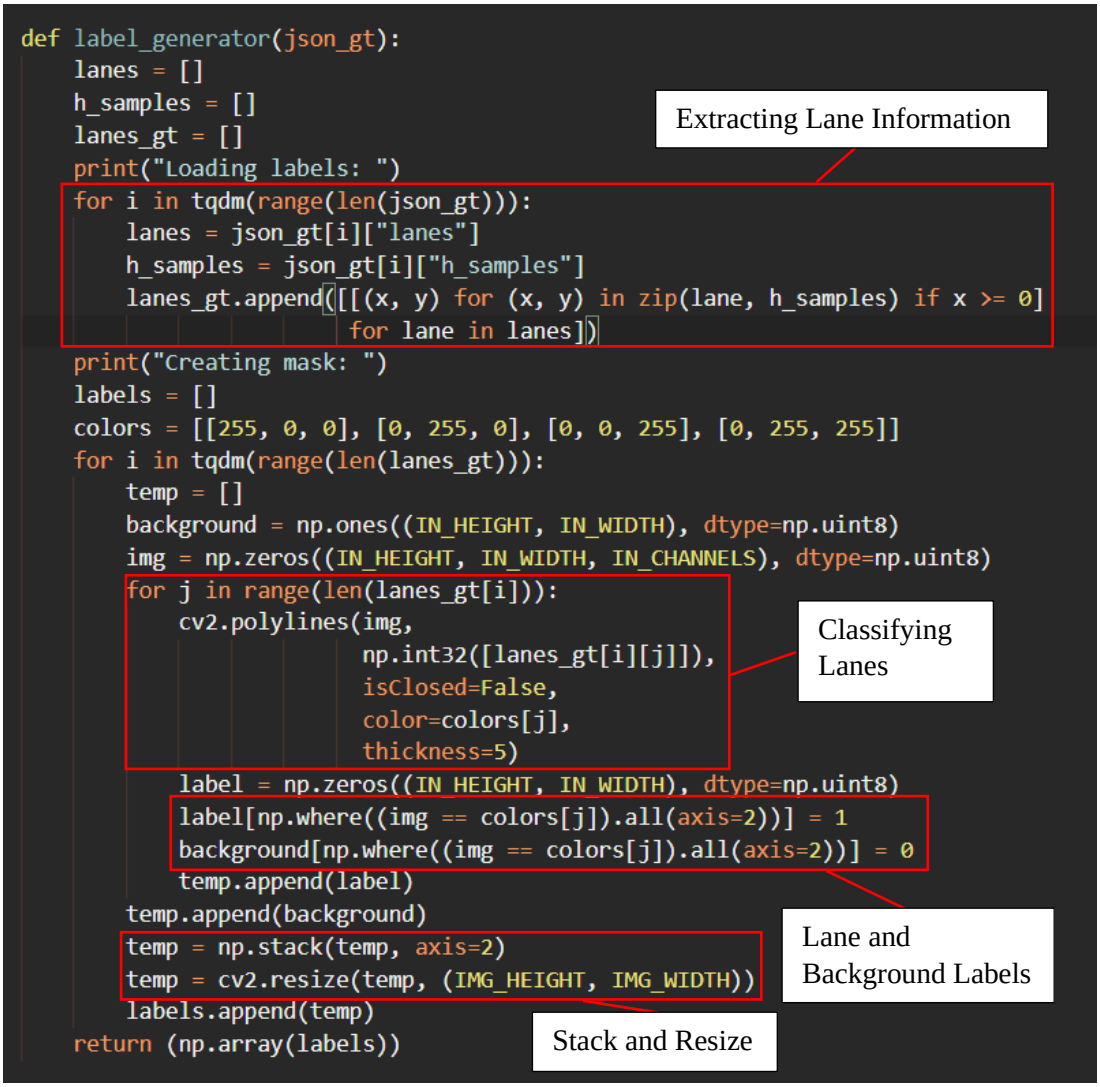


*Figure 25: Label Generation Code*

### 3.1.2 Model Build-Up

Both the MobileNetV2-UNet model and VGG16-UNet model were built with the help of tensorflow.keras.application module. Models came loaded with weights pre-trained on ImageNet, however, layers were set as trainable to enable the model to better fit semantic segmentation application.

Step 1 (MobileNetV2-UNet):

MobileNetV2 model is loaded from TensorFlow with the top removed. Skip connections are then extracted and concatenated with a decoder layer of similar size to restore spatial information and refine classified pixel mapping.

*Input: NULL*

*Output: MobileNetV2-UNet model*

```
def MobileNetV2_UNet():
    inputs = tf.keras.layers.Input(shape=(IMG_HEIGHT, IMG_WIDTH, IMG_CHANNELS),
                                   name="image_input")
    encoder = tf.keras.applications.MobileNetV2(input_tensor=inputs,
                                                weights="imagenet",
                                                include_top=False)
    skip_connection_names = [
        "image_input", "block_1_expand_relu", "block_3_expand_relu",
        "block_6_expand_relu", "block_13_expand_relu"
    ]
    encoder_output = encoder.get_layer("out_relu").output
    f = [32, 64, 128, 256, 512]
    x = encoder_output
    for i in range(1, len(skip_connection_names) + 1, 1):
        x_skip = encoder.get_layer(skip_connection_names[-i]).output
        x = tf.keras.layers.Conv2DTranspose(f[-i], (3, 3),
                                            strides=(2, 2),
                                            padding='same')(x)
        x = tf.keras.layers.Concatenate()([x, x_skip])
        x = tf.keras.layers.Dropout(0.2)(x)
        for j in range(2):
            x = tf.keras.layers.Conv2D(f[-i], (3, 3), padding="same")(x)
            x = tf.keras.layers.BatchNormalization()(x)
            x = tf.keras.layers.Activation("relu")(x)

    x = tf.keras.layers.Conv2D(5, (1, 1), activation='softmax')(x)
    model = tf.keras.models.Model(inputs, x)
    return (model)
```

*Figure 26: MobileNetV2-UNet Model Construction Code*

Step 1 (VGG16-UNet):

VGG16 model is loaded from TensorFlow with the top removed. Required layers are extracted as skip connection to decoder layer consisting of Transpose Convolution, Dropout, Convolution, Batch Normalization and Activation layers. The final layer is a pointwise SoftMax activation to classify output according to relative prediction probability.

*Input: NULL*

*Output: VGG16-UNet model*

```
def VGG16_UNet():
    inputs = tf.keras.layers.Input(shape=(IMG_HEIGHT, IMG_WIDTH, IMG_CHANNELS),
                                   name="image_input")
    encoder = tf.keras.applications.VGG16(input_tensor=inputs,
                                          weights="imagenet",
                                          include_top=False)
    skip_connection_names = [
        "image_input", "block2_conv2", "block3_conv3", "block4_conv3",
        "block5_conv3"
    ]
    encoder_output = encoder.get_layer("block5_pool").output
    f = [32, 64, 128, 256, 512]
    x = encoder_output
    for i in range(1, len(skip_connection_names) + 1, 1):
        x_skip = encoder.get_layer(skip_connection_names[-i]).output
        x = tf.keras.layers.Conv2DTranspose(f[-i], (3, 3),
                                            strides=(2, 2),
                                            padding='same')(x)
        x = tf.keras.layers.Concatenate()([x, x_skip])
        x = tf.keras.layers.Dropout(0.2)(x)
        for j in range(2):
            x = tf.keras.layers.Conv2D(f[-i], (3, 3), padding="same")(x)
            x = tf.keras.layers.BatchNormalization()(x)
            x = tf.keras.layers.Activation("relu")(x)

    x = tf.keras.layers.Conv2D(5, (1, 1), activation='softmax')(x)
    model = tf.keras.models.Model(inputs, x)
    return (model)
```

*Figure 27: VGG16-UNet Model Construction Code*

Step 2:

Both models are compiled using Adam optimizer with a learning rate of 0.0001 and categorical cross-entropy loss function. The metric of evaluation is set to pixel accuracy of prediction.

```
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=LR),
              loss=tf.keras.losses.categorical_crossentropy,
              metrics=['accuracy'])
```

*Figure 28: Optimizer and Loss Function Code*

Step 3:

Several call-back functions are called during training to improve training and for record keeping of training behaviour across epochs. ReduceLROnPlateau implemented here helps reduce the learning rate of training by a factor of 0.1 if validation loss shows no signs of improvement over 2 epochs whereas EarlyStopping functions to prevent the model from overfitting by stopping training and restoring best weights if validation loss shows no signs of improvement over 3 epochs. For tracking records, training parameters are saved as csv log and tensorboard to be later visualized for abnormality.

```
callbacks = [
    tf.keras.callbacks.TensorBoard(log_dir='vgg16_unet_board'),
    tf.keras.callbacks.ReduceLROnPlateau(monitor='val_loss',
                                         factor=0.1,
                                         patience=2),
    tf.keras.callbacks.EarlyStopping(monitor='val_loss',
                                     patience=3,
                                     restore_best_weights=True),
    tf.keras.callbacks.CSVLogger('vgg16_unet_logs.csv')
]
```

*Figure 29: Call-back Operations Code*

Step 4:

Both models are then trained in batch sizes of 16 over 30 epochs. A tenth of the training images and labels are also used to validate the model at every epoch to check for any signs of overfitting and underfitting. The best model is then saved for real-world lane detection.

```
results = model.fit(train_images,
                    train_labels,
                    validation_split=0.1,
                    batch_size=16,
                    epochs=30,
                    callbacks=callbacks)
model.save("vgg16_unet")
```

*Figure 30: Model Training Parameters Code*

### 3.1.3 Output Post Processing

As model output contained predictions at different levels of confidence post-processing must be carried out to filter low confidence readings. Guard zone and a threshold were also applied to output to allow for lane departure warning functionalities. Figures 31 shows the steps involved in transforming model output to useful knowledge:

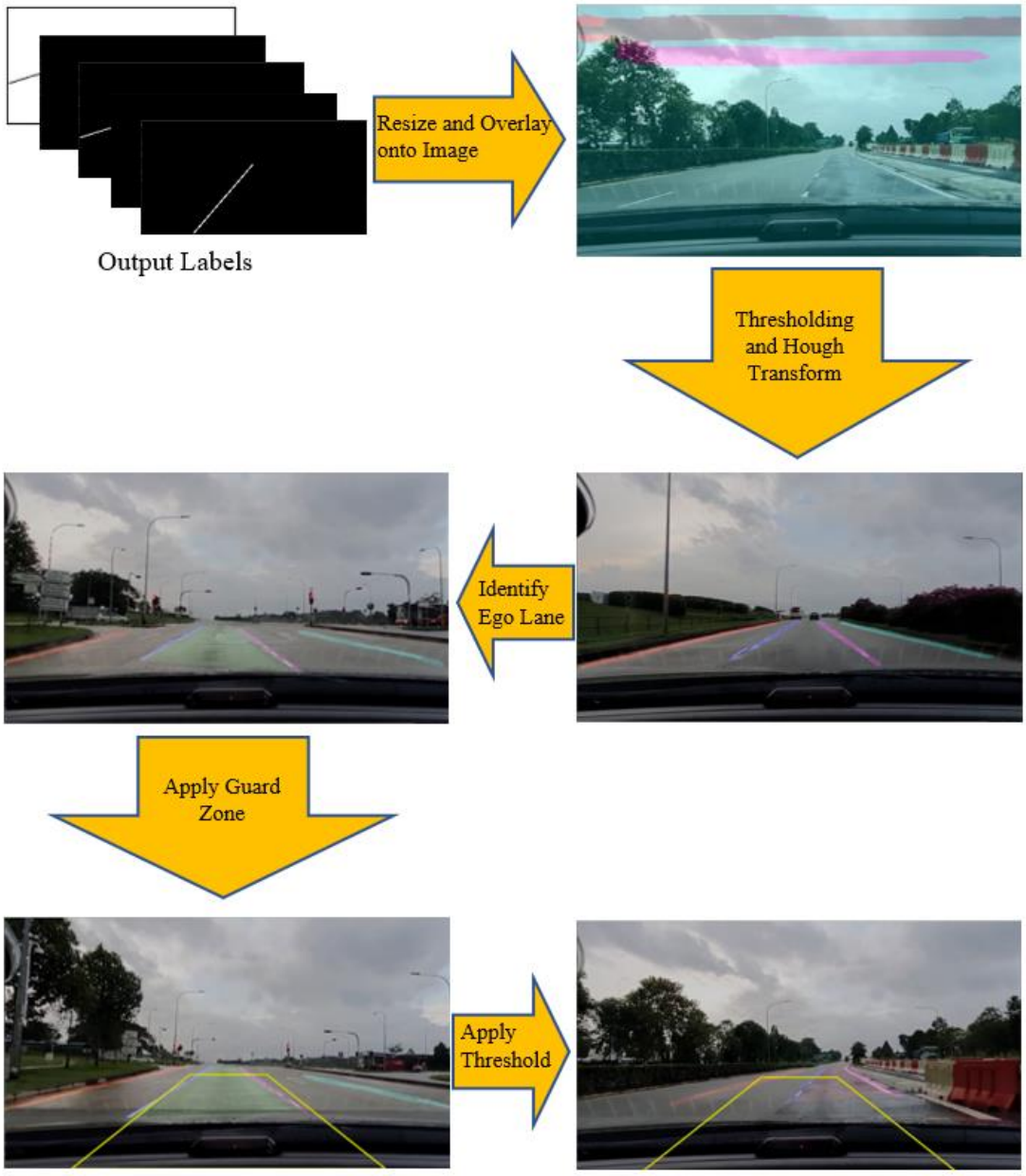


*Figure 31: Post Processing Procedure*

Code Implementation:

Video file is read and processed by model. The output of the model is then passed to the hough_output function to be further processed. Cv2.polylines here draws the guard zone to serve as “in lane” criteria.

```
cap = cv2.VideoCapture("20210321_181023.mp4")
while (cap.isOpened()):
    ret, frame = cap.read()
    img = cv2.resize(frame, (IMG_WIDTH, IMG_HEIGHT))
    resized = cv2.resize(frame, (IN_WIDTH, IN_HEIGHT))
    cv2.polylines(resized, [
        np.array([[530, 450], [190, 720], [1090, 720], [750, 450]], np.int32)
    ], True, (0, 255, 255), 3)
    img = np.expand_dims(img, axis=0) * 1.0 / 255
    prediction = new_model(img)
    prediction = np.squeeze(prediction, axis=0)
    prediction = cv2.resize(prediction, (IN_WIDTH, IN_HEIGHT))
    prediction = hough_output(prediction)
    overlay = cv2.addWeighted(np.array(resized, np.uint8), 0.8,
                              np.array(prediction, np.uint8), 0.2, 0)
    cv2.imshow("Output", overlay)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
cap.release()
cv2.destroyAllWindows()
```

Guard Zone

*Figure 32: Loading Video into Model Code*

hough_output function has several operations:

- Thresholding model output to remove low confidence predictions.
- Drawing polylines onto video frame.
- Drawing ego lane mask and threshold lane position from camera frame to identify lane departures.

*Input: Model output*

*Output: Labelled image containing lane lines and ego lane mask*

```
def hough_output(predict):
    image = np.zeros((predict.shape[0], predict.shape[1], 3), np.uint8)
    color = [[255, 0, 0], [255, 0, 255], [0, 0, 255], [255, 255, 0]]
    ego_lane = []
    for i in range(4):
        temp = np.zeros((predict.shape[0], predict.shape[1], 1), np.uint8)
        index = np.where(predict[:, :, i] * 255 > 80)
        for j in range(len(index[0])):
            temp[index[0][j], index[1][j], 0] = 255
        lines = cv2.HoughLinesP(temp, 1, np.pi / 180, 100, 100, 10)
        if i < 2:
            if lines is not None:
                top = np.ones(2, dtype=np.int32) * 3000
                bot = np.zeros(2, dtype=np.int32)
                for line in lines:
                    for x1, y1, x2, y2 in line:
                        if y1 > bot[1]:
                            bot[0] = x1
                            bot[1] = y1
                        if y1 < top[1]:
                            top[0] = x1
                            top[1] = y1
                        if y2 > bot[1]:
                            bot[0] = x2
                            bot[1] = y2
                        if y2 < top[1]:
                            top[0] = x2
                            top[1] = y2
                if i == 0:
                    ego_lane.append(top)
                    ego_lane.append(bot)
                else:
                    ego_lane.append(bot)
                    ego_lane.append(top)
        if lines is not None:
            for line in lines:
                for x1, y1, x2, y2 in line:
                    cv2.polylines(image,
                                  [np.array([[x1, y1], [x2, y2]], np.int32)],
                                  False, color[i], 5)
    mask = np.zeros((predict.shape[0], predict.shape[1], 3), np.uint8)
    if len(ego_lane) == 4 and ego_lane[0][0] > 530 and ego_lane[3][0] < 750:
        cv2.fillPoly(mask, [np.array(ego_lane, np.int32)], (0, 255, 0))
    elif len(ego_lane) == 4:
        cv2.fillPoly(mask, [np.array(ego_lane, np.int32)], (0, 0, 255))
    if len(ego_lane) == 4:
        image = cv2.addWeighted(image, 0.7, mask, 0.3, 0)
    return (image)
```

1. Thresholding

2. Lane Lines

3. Ego Lane Mask and Lane Departure

*Figure 33: Output Post Processing Code*

## 3.2 Model Construction

Lane detection from image frame involves classification of every image pixel as either lane or non-lane, in other words, it is a semantic segmentation problem. Such a binary classification provides good information of lane location but is not sufficient for application in lane departure warning systems. In this project, a 4-lane detector will be constructed to identify “lane to the left of the left ego lane”, “left ego lane”, “right ego lane” and “lane to the right of the right ego lane”. This will allow the driver assistance system to warn the driver of any unintended lane departures.

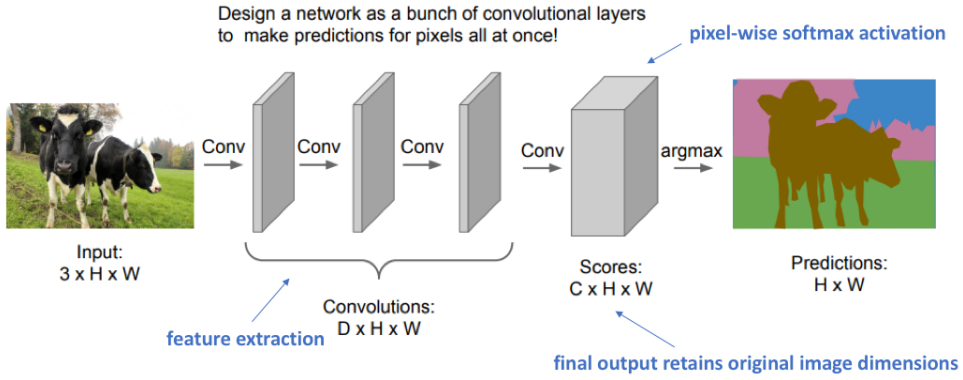


*Figure 34: Naïve Semantic Segmentation Approach [52]*

A naïve approach would be staking “same” convolutional layers in series to extract features and map classification to the original image as shown in Figure 34. This however is not computationally efficient as many parameters would have to be learnt. This is further exacerbated by the need to increase the width of each layer (channel count) to maintain model expressiveness. [53]

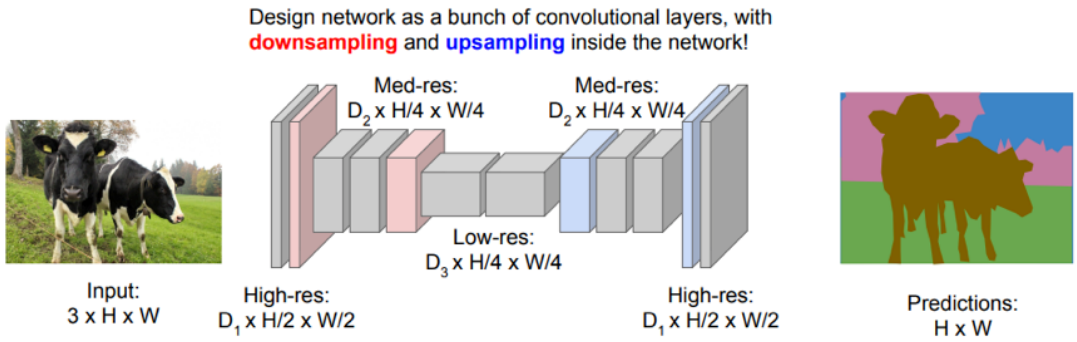


*Figure 35: Encoder-Decoder Approach [52]*

A better approach would involve down-sampling and up-sampling of input as shown in Figure 35. The down-sampling portion (encoder) serves as a semantic/contextual information extractor [54] whereas the up-sampling portion (decoder) helps recover spatial information. [55] This relieves much of the computational burden but suffers from poor segmentation mapping as coarse feature map produced by the encoder are not able to be accurately localized when up-sampled back to the original input dimension.

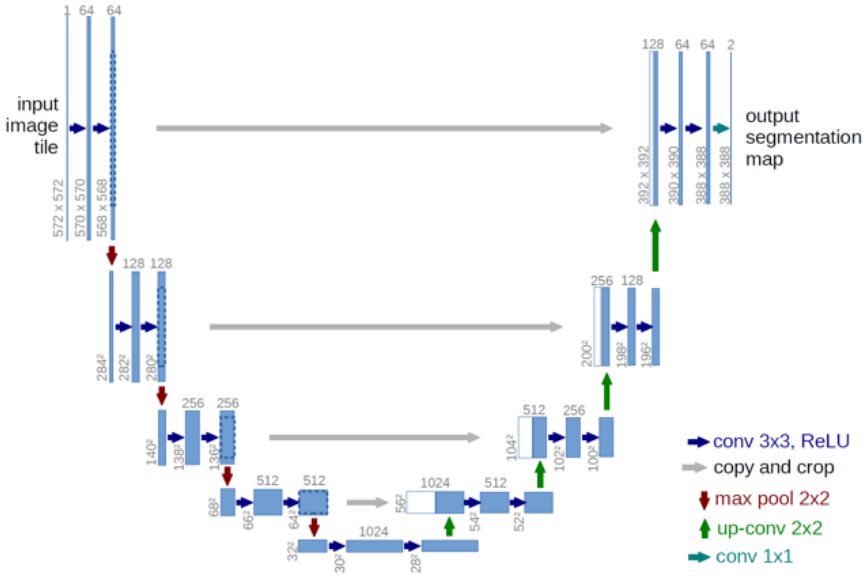


*Figure 36: U-Net [56]*

To overcome this shortcoming, Ronneberger et al. [56] proposed the U-Net architecture that improves upon the encoder-decoder blueprint by adding skip connections as shown in Figure 36. It showed significant improvements over previous architectures and was able to produce a more refined semantic segmentation.

This project seeks to improve upon the U-Net architecture by using VGG 16 and MobileNetV2 as the encoder. VGG 16 has been considered one of the best model architectures and is widely used in image classification problems to extract complex features for final layer reasoning. MobileNetV2 is known for being faster than other model architectures without compromising on its accuracy.

### 3.2.1 VGG16

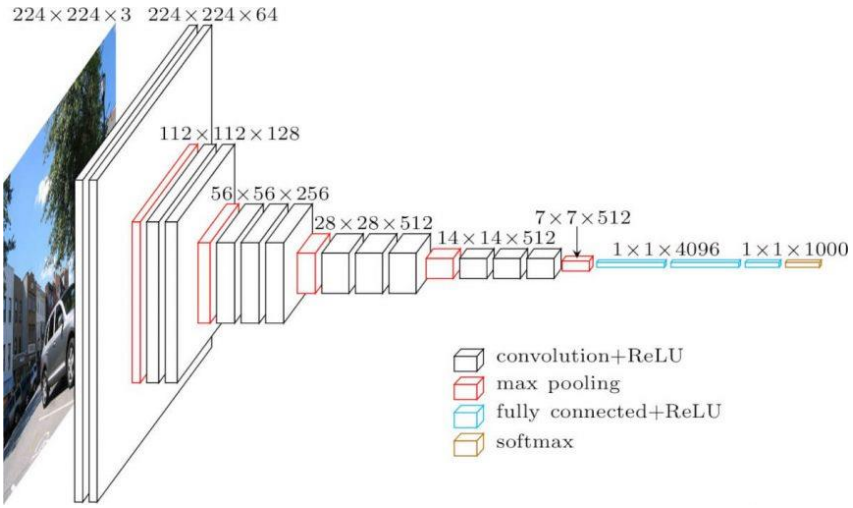


*Figure 37: VGG16 Model Architecture [57]*

Visual Geometry Group with 16 trainable layers (VGG16) is a convolutional neural network proposed by Simonyan and Zisserman [57]. It consists of layers as follows:

→ 2 x convolution layer of 64 channel of 3x3 kernel and same padding*

→ 1 x maxpool layer of 2x2 pool size and stride 2x2

→ 2 x convolution layer of 128 channel of 3x3 kernel and same padding

→ 1 x maxpool layer of 2x2 pool size and stride 2x2

→ 3 x convolution layer of 256 channel of 3x3 kernel and same padding

→ 1 x maxpool layer of 2x2 pool size and stride 2x2

→ 3 x convolution layer of 512 channel of 3x3 kernel and same padding

→ 1 x maxpool layer of 2x2 pool size and stride 2x2

→ 3 x convolution layer of 512 channel of 3x3 kernel and same padding

→ 1 x maxpool layer of 2x2 pool size and stride 2x2

→ 3 x dense layer (fully connected layer)

*Figure 38: VGG16 Layer Breakdown [57]*

### 3.2.2 MobileNetV2

MobileNetV2 is a neural network designed for mobile devices to all for vision-based segmentation, classification, and detection. It prides itself on being a lightweight model capable of achieving high-performance results. The model architecture is as follows:

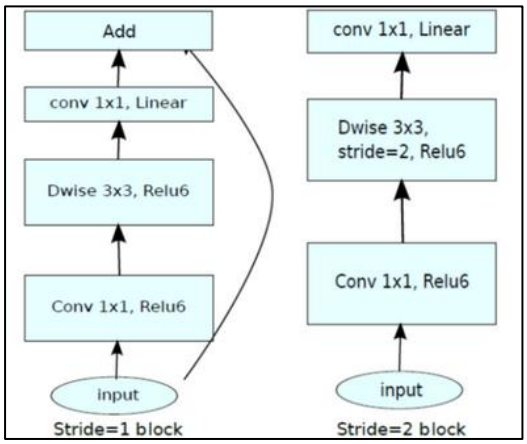


*Figure 39: MobileNetV2 Building Block [58]*

| Input | Operator | $t$ | $c$ | $n$ | $s$ |
|---|---|---|---|---|---|
| $224^2 \times 3$ | conv2d | - | 32 | 1 | 2 |
| $112^2 \times 32$ | bottleneck | 1 | 16 | 1 | 1 |
| $112^2 \times 16$ | bottleneck | 6 | 24 | 2 | 2 |
| $56^2 \times 24$ | bottleneck | 6 | 32 | 3 | 2 |
| $28^2 \times 32$ | bottleneck | 6 | 64 | 4 | 2 |
| $14^2 \times 64$ | bottleneck | 6 | 96 | 3 | 1 |
| $14^2 \times 96$ | bottleneck | 6 | 160 | 3 | 2 |
| $7^2 \times 160$ | bottleneck | 6 | 320 | 1 | 1 |
| $7^2 \times 320$ | conv2d 1x1 | - | 1280 | 1 | 1 |
| $7^2 \times 1280$ | avgpool 7x7 | - | - | 1 | - |
| $1 \times 1 \times 1280$ | conv2d 1x1 | - | k | - | |

*Figure 40: MobileNetV2 Layer Breakdown [58]*

where t = expansion factor, c = output channels, n = repetition, s = stride (block type)

Figure 39 shows 2 convolutional blocks used by MobileNetV2. Stride 1 block (bottleneck residual block) performs an efficient form of convolution while stride 2 block (bottleneck block) mainly functions as a downsizer. ReLU6 was also proposed due to its robustness for low-precision computation [59]. It works exactly like ReLU but with an upper limit of 6 as shown in Figure 41.

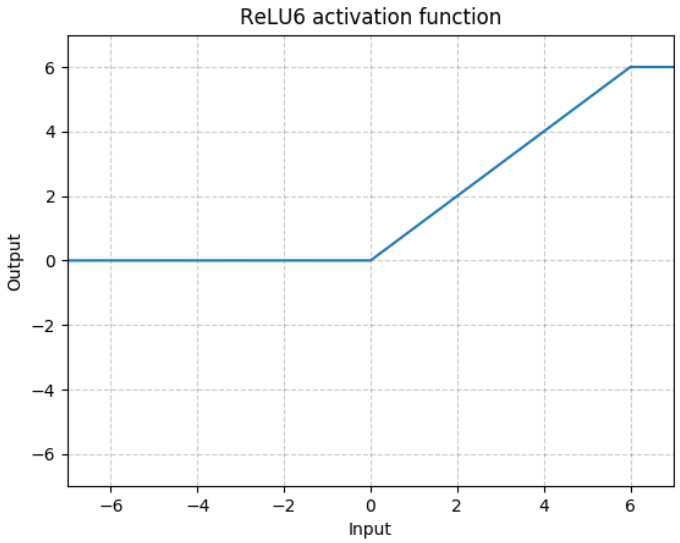


*Figure 41: ReLU6 [58]*

The secret to MobileNetV2's processing speed and lightweight-ness is Depthwise Separable Convolution. This is carried out in several steps as shown in Figure 42 - 44:

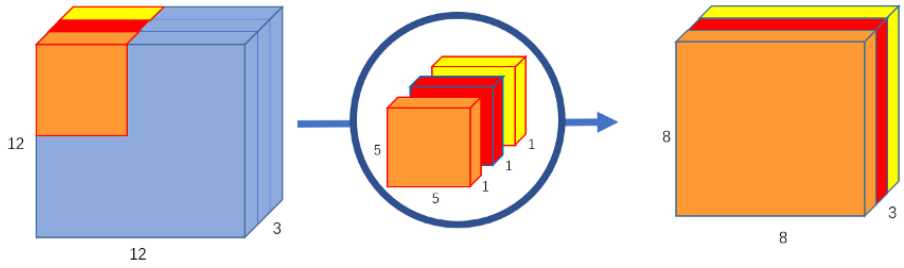


*Figure 42: Depthwise Convolution [60]*

1. Depthwise convolution is carried out on input (i.e., convolution over every individual channel is performed and then stack together)

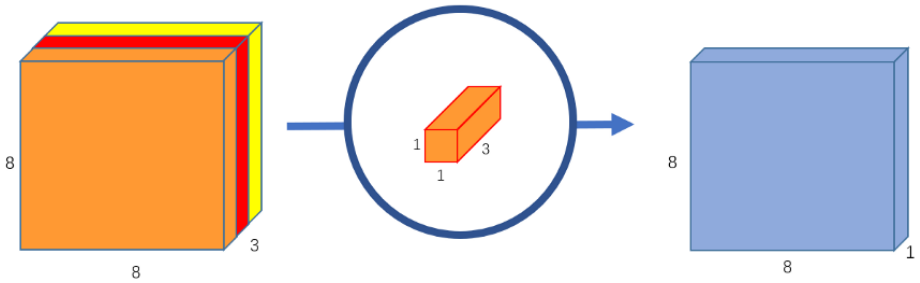


*Figure 43: Pointwise Convolution (1 Kernel) [60]*

2. Pointwise convolution is then carried out (i.e., 1x1xd kernel is passed over input, where d = depth of input)

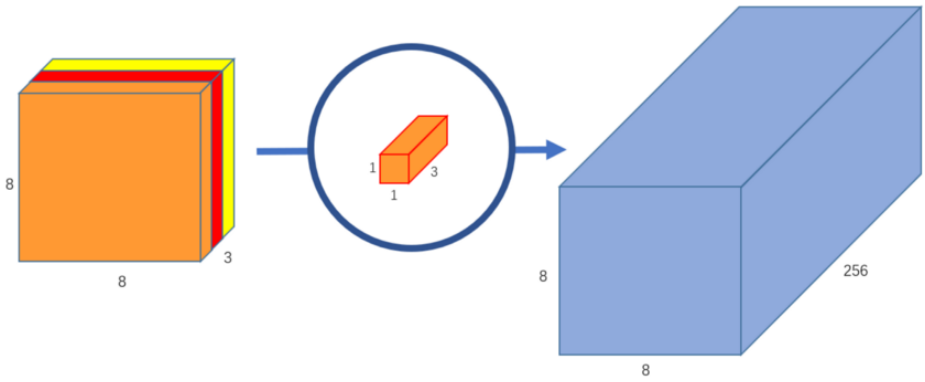


*Figure 44: Pointwise Convolution (256 Kernels) [60]*

3. Output channel number can be expanded by introducing a different number of pointwise kernels (i.e., 254 different kernels of size 1x1xd is used to produce output with 254 channels)

### 3.2.3 VGG16-UNet

This model is built on the U-Net backbone with VGG16 serving as the encoder.

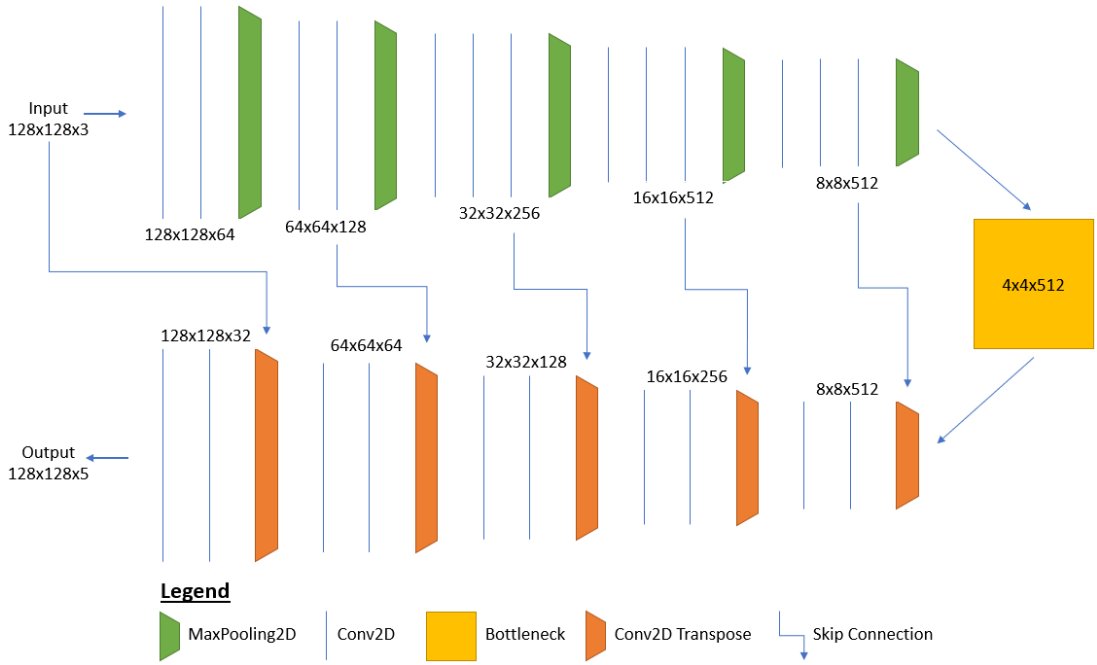


*Figure 45: VGG16-UNet Model Architecture*

### 3.2.4 MobileNetV2-UNet

This model is built on the U-Net backbone with MobileNetV2 serving as the encoder.

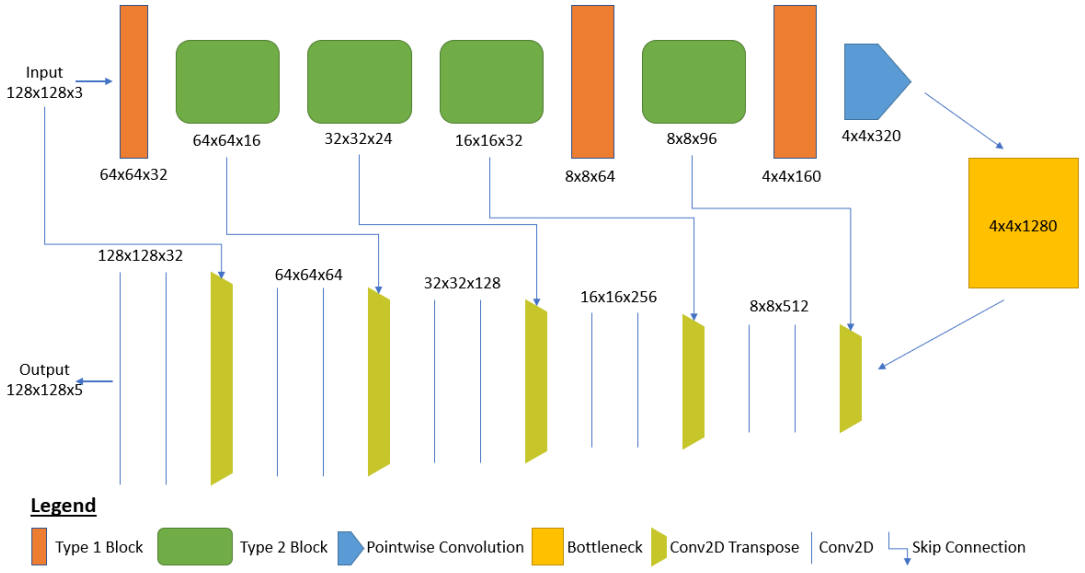


*Figure 46: MobileNetV2-UNet Model Architecture*

# Chapter 4: Experiment & Discussion

## 4.1 Implementation Environment

In this section, all main libraries including their version and hardware used are listed.

### 4.1.1 Software

This project was fully programmed in Python 3.7.9 with Anaconda package manager to allow compatible implementation of TensorFlow GPU. The table below lists all major libraries used.

Table 1: Libraries and their Descriptions

| Library | Description |
|---|---|
| TensorFlow GPU 2.1.0 | Machine learning framework that facilitates the construction of complex models. It includes functional APIs that can perform tedious operations such as pooling, convolution, concatenation and so on. |
| NumPy 1.19.2 | Fundamental package used in scientific computation. It allows the efficient creation of multidimensional array objects and provides a collection of logical operations to perform on them. |
| JSON | JavaScript Object Notation (JSON) is a lightweight data storage format. This library is required to perform read, write, and parse operations for such file formats. |
| OpenCV 3.4.2 | Computer vision and machine learning software library designed to solve computer vision problems. It contains highly optimized numerical operations that could efficiently resize, transform, and extract features from images. |
| OS | The main library used to perform operating system related task for example switching search directory, creating folders, removing folders and many more. |
| tqdm | A library that displays the number of iterations ran in a specified section of code as a progress bar. |

### 4.1.2 Hardware

The main hardware involved in this project was a powerful computer to run the many computationally expensive layers in a machine learning model. The training was mainly done on the computer at Rehabilitation Research Institute of Singapore (RRIS) lab equipped with Nvidia GeForce RTX 2080 GPU to enable GPU enhancement for fast training of the model. Code implementation and testing were largely done on my personal computer with Nvidia GeForce GTX 1650Ti GPU.

## 4.2 Dataset

Dataset used in this project were images of road lanes and their respective lane labels to supervise the training of the model. Dataset was mainly collected from two sources:

1. Online - TuSimple dataset
2. Self-collected road video footage

### 4.2.1 TuSimple Dataset (Competitions for CVPR2017)

This dataset was used for a lane detection challenge during the 2017 Conference on Computer Vision and Pattern Recognition. Images were provided in jpg format with dimensions 720 x1280 whereas labels were in a json file format. Dataset features were as follows:

Table 2: TuSimple Dataset Features

| **Complexity** | **Dataset Size** |
|---|---|
| • Good and medium weather conditions<br>• Different daytime<br>• 2/3/4 or more lane highway roads<br>• Different traffic conditions | • 3626 labelled frames<br>• 2782 unlabelled frames |

As supervised learning was conducted, only labelled image frames were used.

*Figure 47: TuSimple Image Sample*

*Figure 48: TuSimple Image Sample*

### 4.2.2 Self-Collected Road Video Footage

A total of 6 hours of dashboard camera video footage from 4 different cars driven on Singapore roads were obtained. The video frame was then captured once every 10 seconds in 720 x1280 dimension using the VLC video player's scene filter function. Blurry images were manually removed to finally acquire 2067 images. Dataset was more representative of local conditions and came with a whole range of challenges, below are some of them:

Table 3: Self-Collected Dataset Features

| Environment Complexity | Hardware Complexity |
|---|---|
| • Good and bad weather conditions<br>• 2/3/4 lane roads<br>• Urban traffic conditions<br>• Complex lane markings (i.e., zigzag) | • Varied videos dimensions<br>• Varied video format<br>• Varied camera resolution<br>• Varied mount position |

*Figure 49: Singapore Road Sample*


*Figure 50: Singapore Road Sample*


*Figure 51: Singapore Road Sample*

*Figure 52: Singapore Road Sample*

Lanes were manually labelled using an online VGG Image Annotator (VIA) with polyline region shape. Annotations were then exported as json file to be later processed into image labels for training. Labelling was done as such:

- Left ego-lane = 1
- Right ego-lane = 2
- Lane left of left ego-lane = 3
- Lane right of right ego-lane = 4


*Figure 53: Singapore Road Labelled*

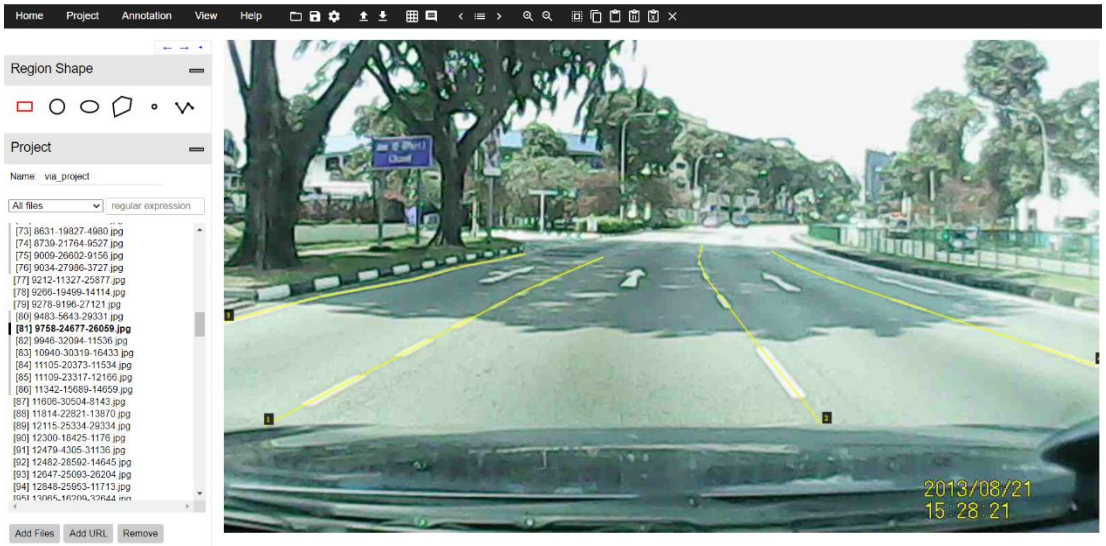


*Figure 54: Singapore Road Labelled*

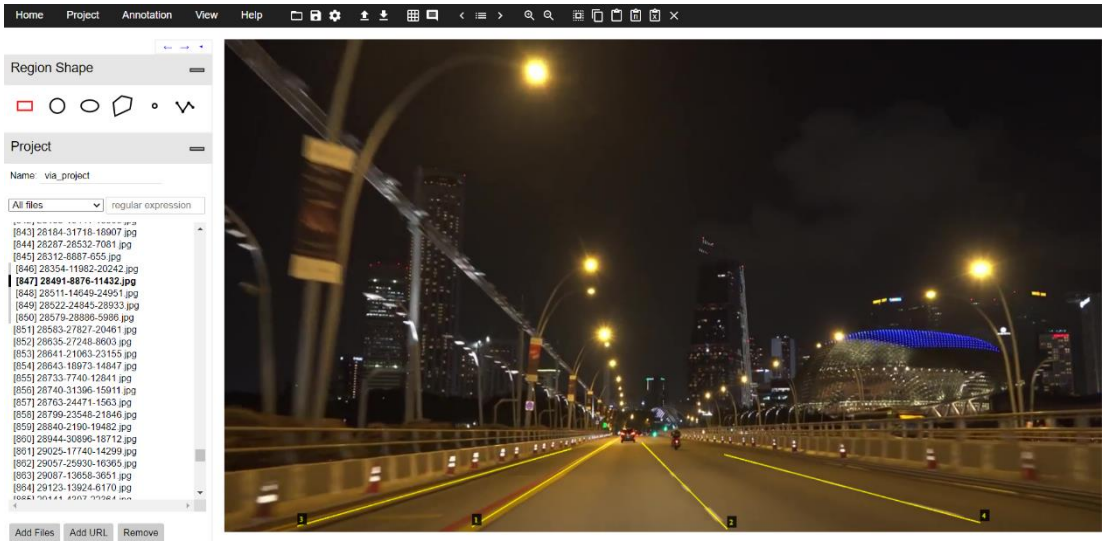


*Figure 55: Singapore Road Labelled*

## 4.3 Results

The project was carried out in 2 different environments. First, using TuSimple dataset which simulates optimum lane detection conditions. And next, using self-collected images of real-life driving conditions.

### 4.3.1 First Phase (TuSimple Simulated Environment)

The first phase was implemented to allow models to learn lane marking characteristics quickly as images showed a greater distinction between lanes and non-lanes. The high-definition images used also helped models identify nuances in lane features better leading to more effective learning. Models were then evaluated on the side lane, centre lane and curved lane to determine its ability to generalize well. The results of the detection were as shown below:

**Side Lane**

*Figure 56: U-Net Side Lane*

*Figure 57: VGG16-UNet Side Lane*

*Figure 58: MobileNetV2-UNet Side Lane*

## Centre Lane

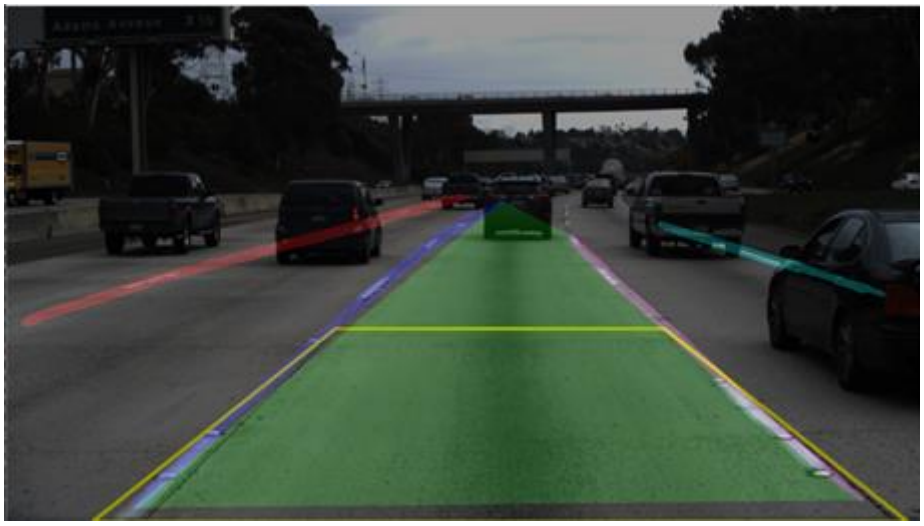

*Figure 59: U-Net Centre Lane*

*Figure 60: VGG16-UNet Centre Lane*

*Figure 61: MobileNetV2-UNet Centre Lane*

## Curved Lane

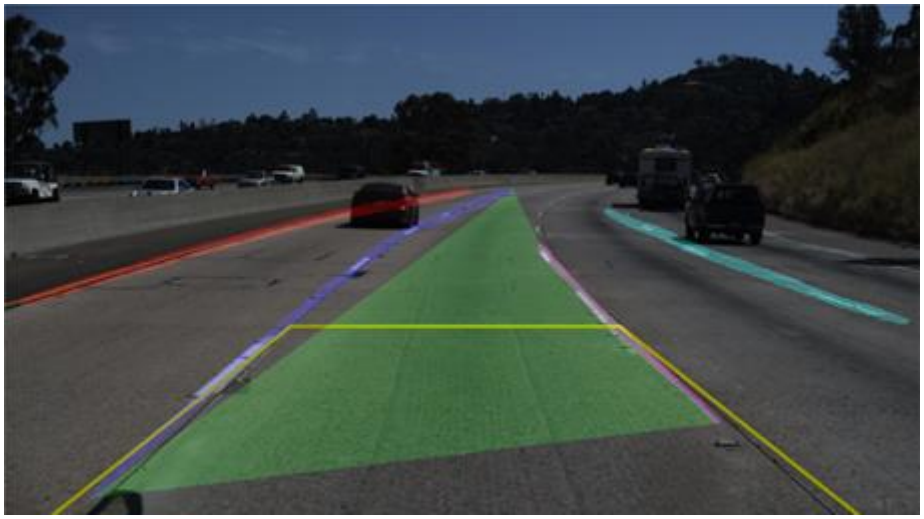

*Figure 62: U-Net Curved Lane*

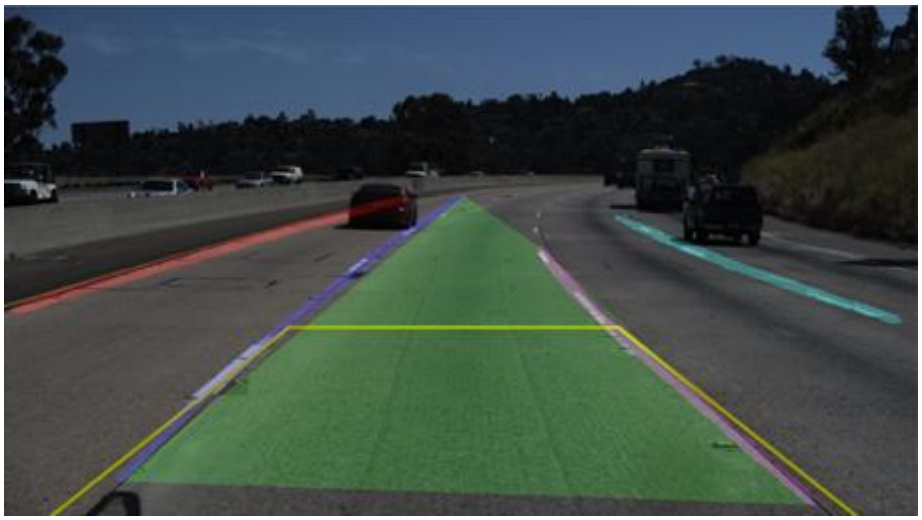

*Figure 63: VGG16-UNet Curved Lane*

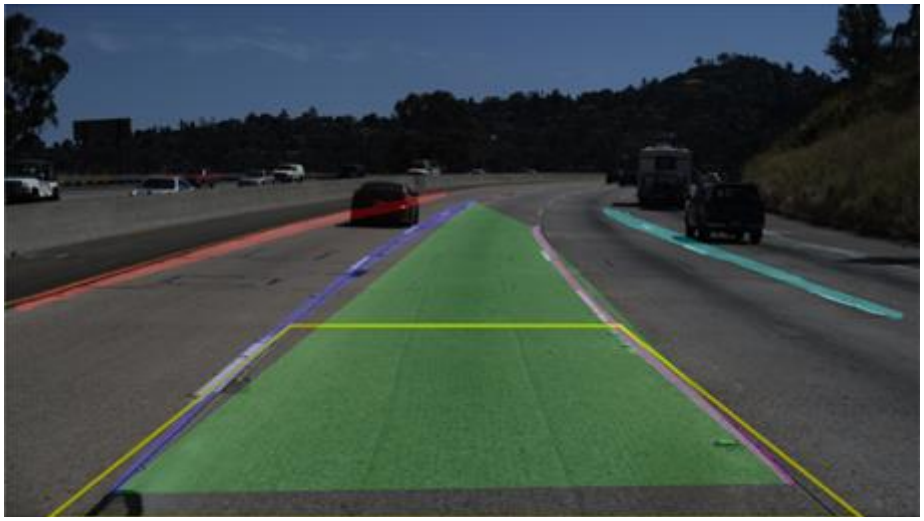

*Figure 64: MobileNetV2-UNet Curved Lane*

### 4.3.2 Second Phase (Singapore Real-Life Environment)

The second phase was conducted using images captured from dashboard cameras of 4 different cars driven in Singapore. This was primarily done to teach lane detection models Singapore's unique road design and enable it to better cater to the Singapore audience. Commercially available dashboard cameras were also chosen as the image capturing tool to realize the potential of making these devices capable of lane detection and lane departure warning.

Below shows lane predictions of the models before (i.e., trained with TuSimple dataset) and after (i.e., trained with TuSimple + Singapore road images) customized training in broad daylight conditions:

**U-Net**

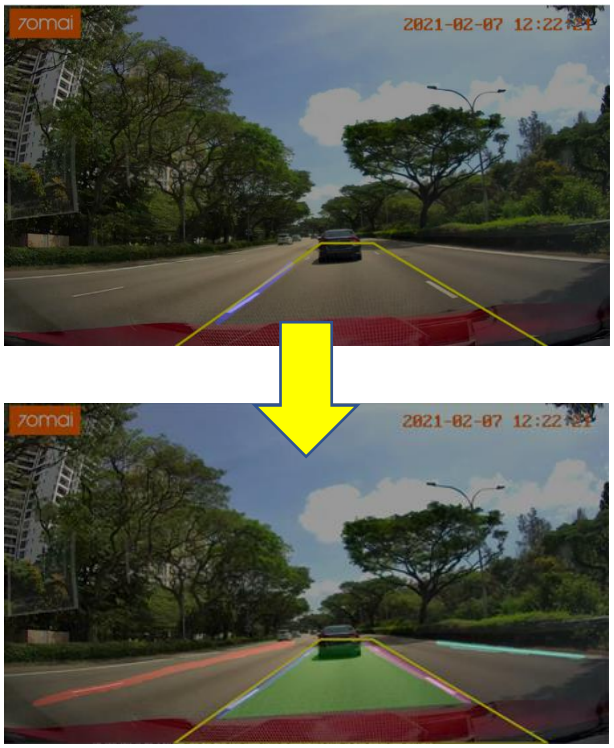


*Figure 65: U-Net Training Outcome*

### VGG16-UNet

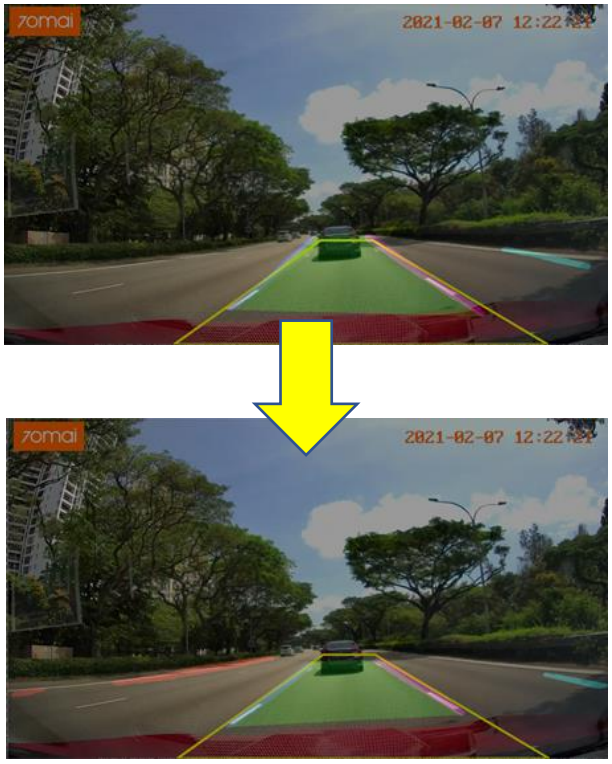


*Figure 66: VGG16-UNet Training Outcome*

### MobileNetV2-UNet

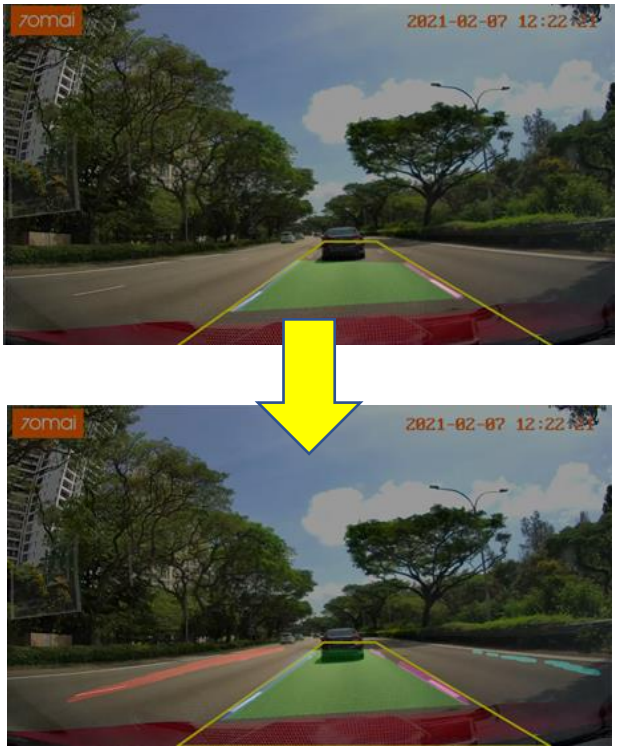


*Figure 67: MobileNetV2-UNet Training Outcome*

## 4.4 Discussion

In this section, models will be compared based on their speed, accuracy, and robustness to different driving conditions.

### 4.4.1 Computation and Training Speed

The 3 models were first evaluated based on their total parameters, computation time and training speed. These metrics are crucial as it determines the model's feasibility for real-time application as well as its ability to learn quickly in ever-changing environments such as that on roads. Below is the summary of this evaluation:

Table 4: Model Parameters, Computation Time, and Training Time

| **Model** | **Total Parameters** | **Computation Time (seconds per frame)** | **Training Time (seconds per epoch of 2858 images)** |
|---|---|---|---|
| U-Net | 34,513,605 | ~3.5 | ~ 32.38 |
| MobileNetV2-UNet | 19,337,957 | ~2.2 | ~ 23.45 |
| VGG16-UNet | 28,845,541 | ~2.6 | ~ 30.14 |

From table 4, it can be inferred that computation time and training time decreases with the number of parameters. However, due to the computation load of the model, real-time lane prediction could not be achieved even for the relatively lightweight models proposed. A possible fix to this problem would include using a better graphics card. Computation times displayed were based on a model running on Nvidia GeForce GTX 1650Ti GPU which is far inferior to newer processers currently available. Apart from that, video image frames could be extracted at a lower rate to not overwhelmed the model. For example, sampling and feeding of images into a model could be done at a rate of 3 frames per second instead of 30 frames per second.

### 4.4.2 Accuracy

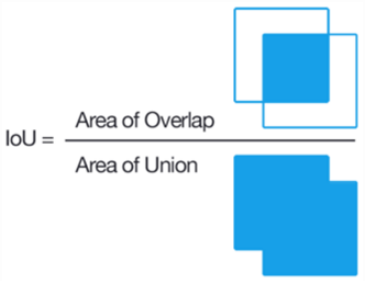


*Figure 68: Intersection over Union (IoU) [61]*

The model prediction accuracy was then examined using 2 metrics, namely pixel accuracy and mean Intersection over Union (mIoU). Pixel accuracy is determined by the number of pixels accurately classified in each class over total image pixels. This serves as a good metric to assess machine learning models but fails to provide a reliable estimate when it comes to small object predictions. Mean IoU on the other hand, calculates the IoU (as shown in Figure 68) for each class and then averages it over the total number of class. This solves the problem of class imbalance and provides a better overview of model performance. Below outlines the model accuracies based on these metrics:

Table 5: Model Pixel Accuracy and Mean IoU

| **Model** | **Pixel Accuracy (%)** | **Mean IoU (%)** |
|---|---|---|
| U-Net | ~ 99.00 | ~ 70.07 |
| MobileNetV2-UNet | ~ 98.73 | ~ 86.52 |
| VGG16-UNet | ~ 98.86 | ~ 75.78 |

From table 5, it is observed that mean IoU and pixel accuracy differs greatly, this suggests that training data suffers from class imbalance, specifically non-lane class dominates image pixels. To prevent this, under-sampling of the majority class can be performed, for example by removing the top portion of the image before feeding it into the model as road markings would not appear on top of the image. Also, more training data could be fed into the model to increase model recognition of diverse road conditions and thus improving accuracy.

### 4.4.3 Robustness to Different Driving Conditions

In real life, road conditions are unpredictable. To account for this, lane detection models were assessed in several challenging situations.

**Glaring Light**

*Figure 69: U-Net (Glaring Light)*

*Figure 70: VGG16-UNet (Glaring Light)*

*Figure 71: MobileNetV2-UNet (Glaring Light)*

From Figure 69 to 71, VGG16-UNet distinguished itself as a power feature extractor outperforming the other 2 models despite the extreme lighting conditions.

**Congested Road**

*Figure 72: U-Net (Congested Road)*

*Figure 73: VGG16-UNet (Congested Road)*

*Figure 74: MobileNetV2-UNet (Congested Road)*

Figure 72 to 74 shows that all 3 models performed well in congested road conditions will some minor misclassification and label omission by U-Net and VGG16-UNet.

**Night-Time**

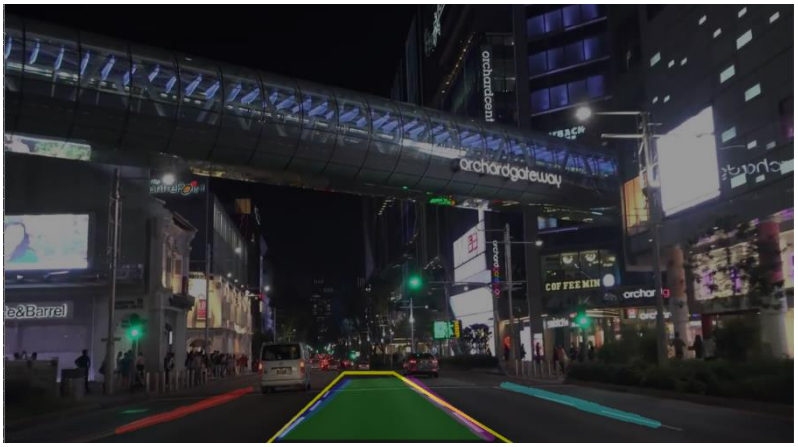


*Figure 75: U-Net (Night-Time)*

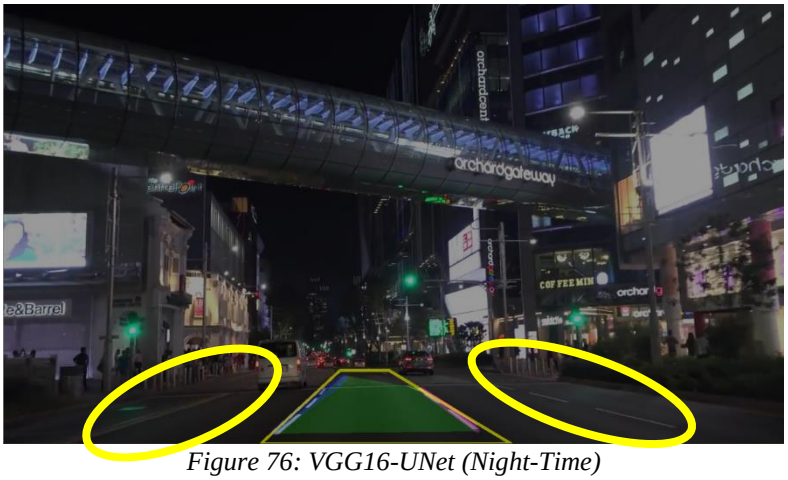


*Figure 76: VGG16-UNet (Night-Time)*


*Figure 77: MobileNetV2-UNet (Night-Time)*

As observed in Figure 75 to 77, VGG16-UNet was the only model that failed to detect non-ego lanes during night-time conditions.

**Rainy**

*Figure 78: U-Net (Rainy)*

*Figure 79: VGG16-UNet (Rainy)*

*Figure 80: MobileNetV2-UNet (Rainy)*

As seen in Figure 78 to 80, all models performed well in rainy weather.

**Tunnel**

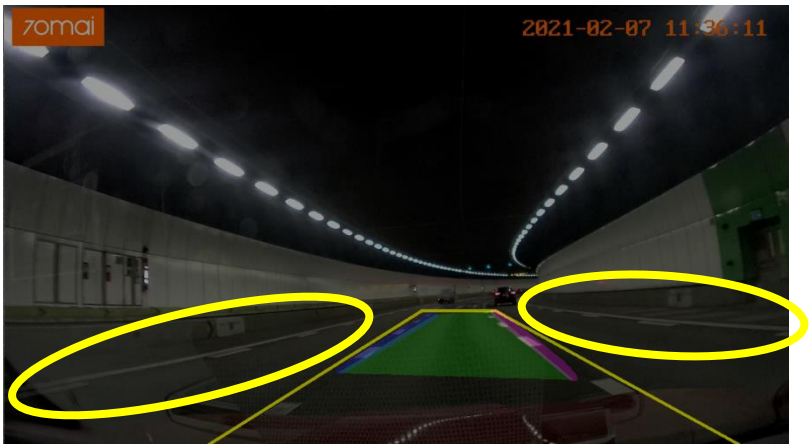


*Figure 81: U-Net (Tunnel)*


*Figure 82: VGG16-UNet (Tunnel)*


*Figure 83: MobileNetV2-UNet (Tunnel)*

Figure 81 to 83 shows that all 3 models suffered in accuracy when passing through a tunnel. VGG16-UNet also made some serious errors classifying the left ceiling as right lane and left lane as left ego lane.

**Wet Road**

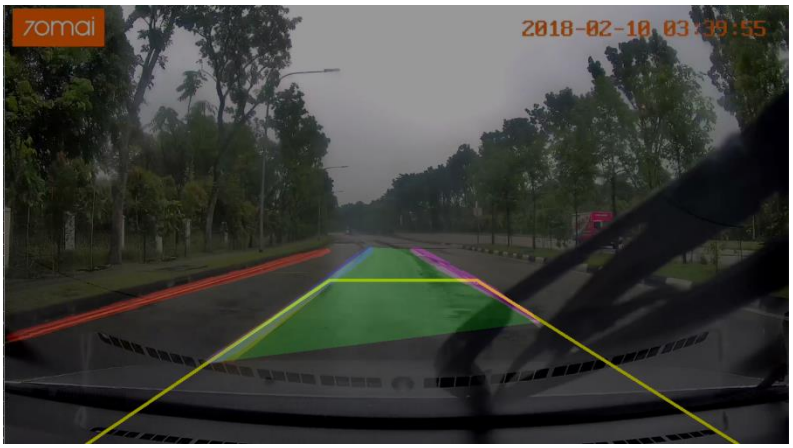


*Figure 84: U-Net (Wet Road)*

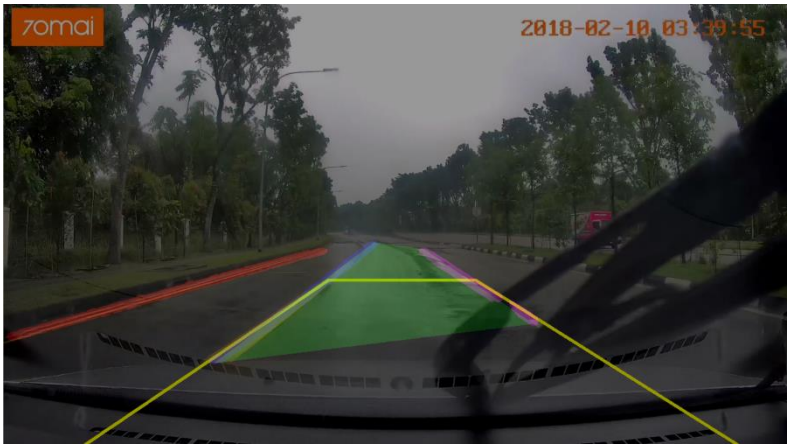


*Figure 85: VGG16-UNet (Wet Road)*


*Figure 86: MobileNetV2-UNet (Wet Road)*

As perceived from Figure 84 to 86, all 3 models perfectly detected lane markings despite the reflective road surface and obstruction of the windscreen wiper.

Table 6 summarizes model performance in the challenging conditions above:

Table 6: Model Performance in Challenging Driving Conditions

| Driving Condition | Models | | |
|---|---|---|---|
| | U-Net | VGG16-UNet | MobileNetV2-UNet |
| Glaring Light | | ✓ | |
| Congested Road | | ✓ | ✓ |
| Night-Time | ✓ | | ✓ |
| Rainy | ✓ | ✓ | ✓ |
| Tunnel | | | ✓ |
| Wet Road | ✓ | ✓ | ✓ |

Based on the assessment above, MobileNetV2-UNet was found to be the most robust model, detecting lanes in most of the challenging driving conditions tested. It was also the fastest both in training and computation. This is due to its novel convolutional layer and ability to learn useful features with a fraction of the parameters required by other models. This unique feature can be seen in Figure 87 with steep accuracy improvements early during the training phase.

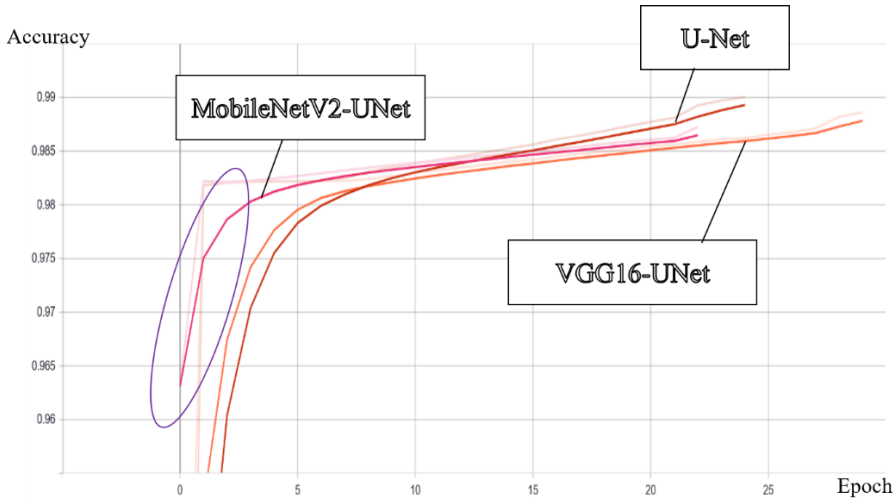


*Figure 87: Training Accuracy Curve of Models*

However, results from Chapter 4.3.2 show that model predictions were only as good as the amount and quality of labelled data it was trained on. This reinforces the importance of quality and quantity of data for model training.

# Chapter 5: Future Work

This project explored the improvements that could be derived from old model architecture as new improved techniques are implemented on it. The implementation of the MobileNetV2 and VGG16 model here in this project proved to significantly enhance semantic segmentation of image frames both in speed and accuracy such that it is viable as a lane departure warning system. Future work would include:

1. Construction of Different Machine Learning Models

   As lane marking location do not vary much in successive frames, a model constructed with spatial-temporal (i.e., Recurrent Neural Networks) capabilities could be constructed to achieve better lane predictions. Besides that, the model could be constructed to receive several inputs in a different colour scheme (e.g., HSV image) or transformation (e.g., canny edge transformed image) to aid the model in extracting relevant features and connecting inputs to the desired output.

2. Increment of Training Data Variety and Number

   As seen from results in Chapter 4.3.2, the models were not able to detect Singapore road lanes accurately and consistently when it was only trained on high-definition images of roads in America. Self-collected data varied widely in camera quality, weather conditions and camera location in the car which adds noise and complexity to detection operation. Models trained on a diverse dataset including strong shadows, different lighting conditions, different weather conditions, different road colour and so on will be prepared to perform well in such conditions, improving driver's confidence in visual-based lane detection.

3. Improvement on Camera Quality and Implementation

   Machine learning models are limited by the accuracy of the input data. If the image quality is bad, the model is unlikely capable of extracting relevant information from it, therefore a high-resolution camera capable of keeping focus regardless of driving speed and lighting conditions should be used as an input retriever. Also, adding 2 or more cameras from different positions could

be explored to acquire texture, depth and other information that could contribute to detection accuracy.

4. Include Additional Input Pre-Processing Processes

   As mentioned in the discussion in Chapter 4.4.2, machine learning-based lane detection tends to suffer from class imbalance. This is due to lanes only covering a small portion of the input image. As such regions of interests (ROI) could be introduced to model pre-processing by removing irrelevant portions of the image (i.e., topside of the captured image), using a reliable classifier to first crop lane sections out, or first dividing the image into left and right portions to reduce class dominance.

5. Integrate Object Classification

   Cars, road signs, traffic light, trees and many other objects provide a visual cue of possible lane regions. By adding object classification functionalities into lane detection models, much other useful information can be derived from camera input which could help to further improve lane detection accuracy.

# Chapter 6: Conclusion

Lane detection is a critical component of any driver assistance or autopilot system. As car development continue to trend towards autonomy, more reliable and efficient ways of lane detection must be introduced. This is paramount as the cost of error is high, ranging from damage to the vehicle to loss of life. Currently, machine learning techniques are resource-intensive, requiring highly specialized hardware and lots of training data to perform well. To overcome this challenge, this project examined several new machine learning models in combination with image processing techniques to perform lane detection and lane departure warning from dashboard camera footage.

The two new semantic segmentation models proposed showed significant improvements in computation plus training time and manage to achieve lane pixel accuracy of approximately 98% with a mean IoU of around 80%. MobileNetV2-UNet model distinguished itself as the fastest model both in terms of training and prediction and displayed robustness in many common yet challenging driving conditions. The VGG16-UNet model, on the other hand, performed poorly in low luminosity environments however show substantial improvements in computation time and detection in glaring light conditions. Using thresholding over guard zone and hough transform model fitting, these models capable of providing partial lane departure warning with high reliability.

Both these models showed improvements over the current state-of-the-art semantic segmentation model, U-Net, however, were still incapable of real-time application due to its model size. Large models are required to perform the complex task as deeper networks can relate input information better, however, this increases computation cost. This accuracy-speed trade-off remains one of the major hurdles in machine learning and unless better algorithms or better equipment are introduced, it would continue to remain so.

# Reference


[1] World Health Organization. (2020, February 7). Road traffic injuries. https://www.who.int/news-room/fact-sheets/detail/road-traffic-injuries

[2] E. Bellis and J. Page, National motor vehicle crash causation survey (NMVCCS), SAS analytical user's manual, U.S. Department of Transportation, National Highway Traffic Safety Administration, Washington, DC, USA, Tech. Rep. No. HS-811 053, Dec. 2008.

[3] J. E. Gayko, "Lane departure and lane keeping," in Handbook of Intelligent Vehicles, A. Eskandarian, Ed. London, UK: Springer, 2012, pp. 689−708.

[4] Chen, W., Wang, W., Wang, K., Li, Z., Li, H., & Liu, S. (2020). Lane departure warning systems and lane line detection methods based on image processing and semantic segmentation: A review. Journal of Traffic and Transportation Engineering (English Edition), 7(6), 748–774. https://doi.org/10.1016/j.jtte.2020.10.002

[5] Baili, Jamel & Marzougui, Mehrez & Sboui, Ameur & Lahouar, Samer & Hergli, M. & Jaganathan, Subash Chandra Bose & Besbes, Kamel. (2017). Lane Departure detection using image processing techniques. 238-241. 10.1109/Anti-Cybercrime.2017.7905298.

[6] B. He, R. Ai, Y. Yan, and X. P. Lang, "Lane marking detection based on Convolution Neural Network from point clouds," in Proc. 19th Int. Conf. Intelligent Transportation Systems (ITSC), Rio de Janeiro, Brazil, 2016, pp. 2475−2480.

[7] J. Li, X. Mei, D. Prokhorov, and D. Tao, "Deep Neural Network for Structural Prediction and Lane Detection in Traffic Scene," IEEE Transactions on Neural Networks and Learning Systems, vol. 28, no. 3, pp. 690-703, 2017.

[8] McCall, J., Trivedi, M.: Video-based lane estimation and tracking for driver assistance: survey, system, and evaluation. IEEE Trans. Intell. Transp. Syst. 7, 20–37 (2006)

[9] Evans, A. (2021). Four tricks for fast blurring in software and hardware [Digital image]. https://www.gamasutra.com/view/feature/3102/four_tricks_for_fast_blurring_in_.php?print=1

[10] Holder, Ross & Tapamo, Jules-Raymond. (2017). Improved gradient local ternary patterns for facial expression recognition. EURASIP Journal on Image and Video Processing. 2017. 10.1186/s13640-017-0190-5.

[11] Q. Lin, Y. Han, H. Hahn, Real time lane detection based on extended edge-linking algorithm, in: IEEE International Conf. on Computer Research and Development, 2010, pp. 725–730.

[12] Freeman, W., & Adelson, E. (1991). The Design and Use of Steerable Filters. IEEE Trans. Pattern Anal. Mach. Intell., 13, 891-906.

[13] J. C. McCall and M. M. Trivedi, "An integrated, robust approach to lane marking detection and lane tracking," IEEE Intelligent Vehicles Symposium, 2004, Parma, Italy, 2004, pp. 533-537, doi: 10.1109/IVS.2004.1336440.

[14] M. Beyeler, F. Mirus, and A. Verl, “Vision-based robust road lane detection in urban environments,” in Proc. IEEE Int. Conf. Robotics and Automation (ICRA), Hong Kong, China, 2014, pp. 4920−4925.
[15] K.Y. Chiu, S.F. Lin, Lane detection using color based segmentation, IEEE Proc. Intell. Veh. Symp. (2005) 706–711.
[16] P.C. Wu, C. Chang, C.H. Lin, Lane mark extraction for automobiles under complex conditions, Pattern Recognition 47 (2756–2767) (2014) 2756-2767.
[17] A. Bar Hillel, R. Lerner, D. Levi, and G. Raz, “Recent progress in road and lane detection: A survey,” Mach. Vis. Appl., vol. 25, no. 3, pp. 727−745, Apr. 2014.
[18] Shi, X., Kong, B., Zheng, F.: A new lane detection method based on feature pattern. In: International Congress on Image and Signal Processing, pp. 1–5 (2009)
[19] D.C. Tseng, C.W. Lin, Versatile lane departure warning using 3D visual geometry, Int. J. Innovative Comput.Inf.Control 9 (5) (2013) 1899–1917.
[20] Rasmussen, C., Korah, T.: On-vehicle and aerial texture analysis for vision-based desert road following. In: CVPR Workshop on Machine Vision for Intelligent Vehicles, vol. III, p. 66 (2005)
[21] Borkar, A., Hayes, M., Smith, M.: Robust lane detection and tracking with RANSAC and Kalman filter. In: International Conference on Image Processing, pp. 3261–3264 (2009)
[22] Jiang, Y., Gao, F., Xu, G.: Computer vision-based multiple-lane detection on straight road and in a curve. In: Image Analysis and Signal Processing, pp. 114–117 (2010)
[23] Kim, Z.: Robust lane detection and tracking in challenging scenarios. IEEE Trans. Intell. Transp. Syst. 9, 16–26 (2008)
[24] Huang, A.S., Moore, D., Antone, M., Olson, E., Teller, S.: Finding multiple lanes in urban road networks with vision and LiDAR. Auton. Robots 26, 103–122 (2009)
[25] Dr. George Bebis, University of Nevada, http://www.cse.unr.edu/~bebis/CS791E/Notes/DeformableContours.pdf
[26] Sawano, H., Okada, M.: A road extraction method by an active contour model with inertia and differential features. IEICE Trans. Inf. Syst. E 89D, 2257–2267 (2006)
[27] K. H. Lim, K. P. Seng, and L. M. Ang, “River flow lane detection and Kalman filtering-based B-spline lane tracking,” Int. J. Veh. Technol., vol. 2012, pp. 465819, Nov. 2012.
[28] Civera, J., & Davison, A. (2010). 1-Point RANSAC for EKF Filtering. Application to Real-Time Structure from Motion and Visual Odometry.
[29] Derpanis, K., 2010. Overview of the RANSAC Algorithm. York University - EECS, [online] pp.1-2. Available at: http://www.cse.yorku.ca/~kosta/CompVis_Notes/ransac.pdf
[30] Y. U. Yim and S. Y. Oh, “Three-feature based automatic lane detection algorithm (TFALDA) for autonomous driving,” IEEE Trans. Intell. Transp. Syst., vol. 4, no. 4, pp. 219−225, Dec. 2003.

[31] Johnsson, M. (2013, March 13). ”how to draw the line” with ggplot2 [Digital Image]. https://onunicornsandgenes.blog/2013/05/31/how-to-draw-the-line-with-ggplot2/
[32] F. Bounini, D. Gingras, V. Lapointe, and H. Pollart, “Autonomous vehicle and real time road lanes detection and tracking,” in Proc. IEEE Vehicle Power and Propulsion Conf. (VPPC), Montreal, QC, Canada, 2015, pp. 1−6.
[33] Nieto, M., Salgado, L., Jaureguizar, F., Arrospide, J.: Robust multiple lane road modeling based on perspective analysis. In: International Conference on Image Processing, pp. 2396–2399 (2008)
[34] Kacmajor, T. (2017, June 6). Hough lines transform explained - tomasz kacmajor. Medium. https://medium.com/@tomasz.kacmajor/hough-lines-transform-explained-645feda072ab
[35] Wikipedia contributors. (2021, February 3). Hough transform. In Wikipedia, The Free Encyclopedia. Retrieved from https://en.wikipedia.org/w/index.php?title=Hough_transform&oldid=1004703028
[36] X. An, E. Shang, J. Song, J. Li, H. He, Real-time lane departure warning system based on a single fpga, EURASIP J. Image Video Process. 38 (2013) 1–18.
[37] M. Aly, “Real time detection of lane markers in urban streets,” in Proc. IEEE Intelligent Vehicles Symp., Eindhoven, Netherlands, 2008, pp. 7−12.
[38] Schubert, E., Sander, J., Ester, M., Kriegel, H. P., & Xu, X. (2017). DBSCAN revisited, revisited. ACM Transactions on Database Systems, 42(3), 1–21. https://doi.org/10.1145/3068335
[39] Agarwal, V. (2019, June 27). Let’s cluster data points using DBSCAN - Vibhor Agarwal. Medium. https://medium.com/@agarwalvibhor84/lets-cluster-data-points-using-dbscan-278c5459bee5
[40] J. Niu, J. Lu, M. Xu, P. Lv, X. Zhao, Robust lane detection using two-stage feature extraction with curve fitting, Pattern Recognit. 59 (C) (2016) 225–233.
[41] Sabita-AI, S. (2021, March 25). Everything you need to know about machine learning. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2021/03/everything-you-need-to-know-about-machine-learning/
[42] A. Gurghian, T. Koduri, S. V. Bailur, K. J. Carey, and V. N. Murali, “DeepLanes: End-to-end lane position estimation using deep neural networks,” in Proc. IEEE Conf. Computer Vision and Pattern Recognition Workshops (CVPRW), Las Vegas, NV, USA, 2016, pp. 38−45.
[43] Jordan, J. (2018, January 26). Normalizing your data (specifically, input and batch normalization). Jeremy Jordan. https://www.jeremyjordan.me/batch-normalization/
[44] Jin, J., Li, M., & Jin, L. (2015). Data normalization to accelerate training for linear neural net to predict tropical cyclone tracks. Mathematical Problems in Engineering, 2015, 1–8. https://doi.org/10.1155/2015/931629

[45] McCaffrey, J. D. (2018, February 1). Understanding image convolution and mushy brains. James D. McCaffrey. https://jamesmccaffrey.wordpress.com/2018/02/01/understanding-image-convolution-and-mushy-brains/#jp-carousel-7494
[46] McCaffrey, J. D. (2018, May 30). Convolution image size, filter size, padding and stride. James D. McCaffrey. https://jamesmccaffrey.wordpress.com/2018/05/30/convolution-image-size-filter-size-padding-and-stride/
[47] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. Journal of Machine Learning Research, 15, 1929–1958. Retrieved from http://jmlr.org/papers/v15/srivastava14a.html
[48] Yani, Muhamad & Irawan, S, & S.T., M.T.. (2019). Application of Transfer Learning Using Convolutional Neural Network Method for Early Detection of Terry's Nail. Journal of Physics: Conference Series. 1201. 012052. 10.1088/1742-6596/1201/1/012052.
[49] Yu, D., Wang, H., Chen, P., & Wei, Z. (2014). Mixed Pooling for Convolutional Neural Networks. Rough Sets and Knowledge Technology, 364–375. https://doi.org/10.1007/978-3-319-11740-9_34
[50] Winovich, N. (2016). Deep learning. Purdue University. https://www.math.purdue.edu/%7Enwinovic/deep_learning.html
[51] A. Maas, A. Hannun, and A. Ng, "Rectifier Nonlinearities Improve Neural Network Acoustic Models," in International Conference on Machine Learning (icml), 2013.
[52] Fei-Fei, L., Johnson, J., & Yeung, S. (2017, May 10). Semantic segmentation idea: Fully convolutional [Digital Image]. http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture11.pdf
[53] Kawaguchi, K., Huang, J., & Kaelbling, L. P. (2019, July 1). Effect of depth and width on local minima in deep learning. MIT Press Direct. https://direct.mit.edu/neco/article/31/7/1462/8499/Effect-of-Depth-and-Width-on-Local-Minima-in-Deep
[54] Long, J., Shelhamer, E., & Darrell, T. (2014, November 14). Fully convolutional networks for semantic segmentation. ArXiv.Org. https://arxiv.org/abs/1411.4038
[55] Noh, H., Hong, S., & Han, B. (2015, May 17). Learning deconvolution network for semantic segmentation. ArXiv.Org. https://arxiv.org/abs/1505.04366
[56] Ronneberger, O., Fischer, P., & Brox, T. (2015, May 18). U-Net: Convolutional networks for biomedical image segmentation. ArXiv.Org. https://arxiv.org/abs/1505.04597
[57] Simonyan, K., & Zisserman, A. (2014, September 4). Very deep convolutional networks for Large-Scale image recognition. ArXiv.Org. https://arxiv.org/abs/1409.1556

[58] Tsang, S. (2019, May 19). Review: MobileNetV2 — Lightweight model (image classification). Medium. https://towardsdatascience.com/review-mobilenetv2-light-weight-model-image-classification-8febb490e61c
[59] Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., & Adam, H. (2017, April 17). MobileNets: Efficient convolutional neural networks for mobile vision applications. ArXiv.Org. https://arxiv.org/abs/1704.04861
[60] Wang, C. (2018, August 14). A basic introduction to separable convolutions - towards data science. Medium. https://towardsdatascience.com/a-basic-introduction-to-separable-convolutions-b99ec3102728
[61] Wikipedia contributors. (2021, April 16). Jaccard index. In Wikipedia, The Free Encyclopedia. Retrieved from https://en.wikipedia.org/w/index.php?title=Jaccard_index&oldid=1018153019