Dash camera-based real-time video-assisted driving tools

Submitted by

Dibyanshu Patnaik (E024)
Harsh Praharaj (E027)
Kartikeya Prakash (E028)
Avi Seth (E038)

Under the Guidance of

Dr. Pravin Srinath
in partial fulfilment for the award of the degree
of

BACHELORS OF TECHNOLOGY
COMPUTER ENGINEERING
At

MUKESH PATEL SCHOOL OF TECHNOLOGY MANAGEMENT AND ENGINEERING

April 2020

DECLARATION

We, Avi Seth, Kartikeya Prakash, Harsh Praharaj, and Dibyanshu Patnaik, Roll No. E038, E028, E027, E024, B.Tech (Computer Engineering), VII semester understand that plagiarism is defined as any one or combination of the following:

1. Un-credited verbatim copying of individual sentences, paragraphs or illustration (such as graphs, diagrams, etc.) from any source, published or unpublished, including the internet.
2. Un-credited improper paraphrasing of pages paragraphs (changing a few words phrases, or rearranging the original sentence order)
3. Credited verbatim copying of a major portion of a paper (or thesis chapter) without clear delineation of who did wrote what. (Source: IEEE, The institute, Dec. 2004)
4. I have made sure that all the ideas, expressions, graphs, diagrams, etc., that are not a result of my work, are properly credited. Long phrases or sentences that had to be used verbatim from published literature have been identified using quotation marks.
5. I affirm that no portion of my work can be considered as plagiarism and I take full responsibility if such a complaint occurs. I understand fully well that the guide of the seminar/ project report may not be in a position to check for the possibility of such incidences of plagiarism in this body of work.

Signature of the Student:

Name: Dibyanshu Patnaik, Harsh Praharaj, Kartikeya Prakash, Avi Seth

Roll No. : E024, E027, E028, E038

Place:

Date:

CERTIFICATE

This is to certify that the project entitled "Dash camera-based real-time video-assisted driving tools" is the bonafide work carried out by Dibyanshu Patnaik, Harsh Praharaj, Kartikeya Prakash, and Avi Seth, of B.Tech (Computer Engineering), MPSTME (NMIMS), Mumbai, during the VIII semester of the academic year 2020, in partial fulfilment of the requirements for the award of the Degree of Bachelors of Engineering as per the norms prescribed by NMIMS. The project work has been assessed and found to be satisfactory.

Dr. Pravin Shrinath

Examiner 1

Examiner 2

Dean

Table of contents

List of Figures

Abbreviations

Abbreviation	Description
CNN	Convolutional Neural Network
BCNN	Branch Convolution Neural Network
SVM	Support Vector Machine
GTSRB	German Traffic Sign Recognition Benchmark
TSR	Traffic Sign Recognition
SOTA	State Of The Art
SVM	Support Vector Machine
ZJU	Zhejiang University
YOLO	You Only Look Once
UAV	Unmanned Aerial Vehicle
IoU	Intersection over Union

ABSTRACT

The purpose of this project is to build software tools which can be used for aiding a driver by using novel techniques in the domain of Image Processing and Deep Learning.

Broadly, the project focusses on four main issues: Traffic Sign Classification, Lane Detection, Vehicle Detection and Drowsiness Detection (of the driver).

We believe that with a robust system, many applications in the field of self-driving vehicles and military technology are possible.

CHAPTER 1

INTRODUCTION

1.1 Objective and Motivation:

India ranks 1 in the number of road accident deaths across the 199 countries reported in the World Road Statistics, 2018 followed by China and the US. Road traffic continues to be a major developmental issue, a public health concern and is a leading cause of death and injury across the World killing more than 1.35 million globally in 2016 as reported in the Global Status report on Road Safety 2018 with 90% of these casualties taking place in the developing countries. Road accidents in India kill almost 1.5 lakh people annually. Accordingly, India accounts for almost 11% of the accident-related deaths in the World.

The objective of this project is to develop a vision-based driver assistance system to enhance the driver's safety. The proposed system consists of several modules which perform various assistive functions namely nearby Vehicle Detection, Traffic Sign Classifier, Lane Detection, and measuring the Driver's Drowsiness Levels. For Traffic Sign Detection.

1.2 Project Overview

We build a deep convolution neural network model that can classify traffic signs present in the image into different categories. The dataset we have used has 50,000+ images of different traffic signs, each of which belongs to one of the 43 different classes.

For Vehicle Detection, we pick 300 random images from the KITTI dataset and annotate the images in a single "Vehicle" class. YOLOv3 is used to train the custom model, and a pre-trained model on the COCO dataset is used as a benchmark model, to compare the custom-built model.

Lane Detection works on a continuous video feed, working on each frame to perform basic image enhancement techniques to add visual attributes. The computer vision techniques are used to augment video output with a detected road lane, road radius curvature and road centre offset. It involves the use of Hough transform and perspective transform to be able to detect road lines, after applying OpenCV based techniques.

Drowsiness detection uses the distance between your eyelids to check over a short period whether you're falling asleep while driving and alerts you accordingly. It uses a linear SVM model (called EAR SVM – Eye Aspect Ratio SVM) trained from manually annotated sequences. Positive examples are collected as ground-truth blinks, while the negatives are those that are sampled from parts of the videos where no blink occurs. For accuracy evaluation, we use the 300 VW dataset which has 50 videos where each frame contains a precise annotation of facial landmarks. For landmark detectors, two SOTA detectors were tested – Intraface and Chehra

1.3 Software Specifications

Python==3.6+
Tensorflow==1.13.1
Keras==2.2.0
OpenCv==2+

1.5 Hardware Specifications

RAM: 8 GB
Processor: 10th Generation i5
OS: Windows 10 64bit

Camera: 1.2MP+

CHAPTER 2

LITERATURE SURVEY

2.1 Traffic Sign Classifier:

In this module, we investigated how the Convolutional Neural Networks (CNN) model has been used to detect and recognize traffic signs. CNN is a multi-layer neural network model, which is typically made up of two main parts. The first part contains alternating convolution and pooling. The second part is composed of connections and soft-max classification. Among the properties of CNN is its increased accuracy in vision tasks and improved learning ability. Another factor in evaluating the effectiveness of this technology is its performance under strict time constraints. Traffic Signal Recognition is a central feature of autonomous vehicles, and considering the risks of a late classification, the time it takes for this process to be completed is of equal importance to the accuracy of the process itself. However, many papers that present research about their model fail to provide the speed of classification. In this paper we observed a concise survey of recent research endeavours in traffic sign recognition, which implemented CNN architectures. The traffic sign recognition systems are usually comprised of two parts: traffic sign detection and classification. The regions of interest are first extracted in detection, and then identified correctly or rejected in classification.

In this paper, we observe how the CNN model has been employed in traffic sign recognition (TSR) systems. The recognition task consists of two parts: detection and classification, so we will divide the sections in this way:

1) CNN models used for Detection

In [1], the proposed algorithm is based on deep visual feature, which combines convolutional neural networks (CNN) and support vector machine (SVM). It can extract the characteristics of images in many cases and identify 4 types of arrows accurately. It is because the process of recognition is based on the deep visual features, the detection process is performed in two steps: 1) collect and process the images, then determine the keyframes of the images, and 2) the Convolutional Neural Network (CNN) is applied to extract the deep visual features of the images. There are two benefits for extraction by CNN: 1) the local perceptional vision, and 2) weights of Shared. The extraction of features is done by a series of convolutional layers. The features are then used to train the SVM classifier in a one-dimensional vector. The accuracy for this method is 71.42%.

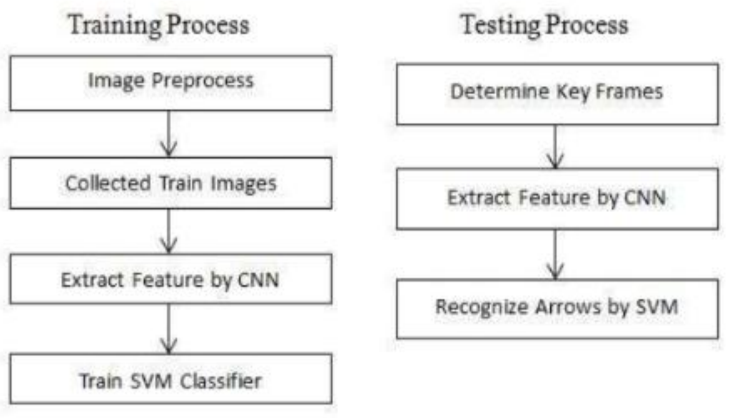


Fig 2.1.1 Framework of the proposed algorithm

Through this paper, an algorithm was proposed by combining convolutional neural networks and support vector machine for detecting and recognizing traffic signs. The algorithm was tested on a database containing various environments. The result shows that 71.42% of arrows have been detected correctly. This algorithm has a preferable generalization, and we believe it can be used to identify more generic traffic signs. At short-term, we will extends the set of training images to improve the parameter of CNN-SVM networks.

In [2], the CNN is used as the deep perceptual feature extractor its function is compared to the human visual cortex. It is claimed that a more selective CNN learned feature set is more economical and precise than previous TSR models. The accuracy for this model is 99.54%. The CNN architecture used is the one proposed in [3] adding an extra convolutional layer ahead of the FCL.

In [4], a traffic sign recognition system is proposed by applying a convolutional neural network (CNN). In comparison with previous methods which usually use CNN as feature extractor and multilayer perception (MLP) as the classifier, [4] proposed max-pooling positions (MPPs) as an effective discriminative feature to predict category labels. Through extensive experiments, MPPs demonstrates the ideal characteristics of small inter-class variance and large intra-class variance. Moreover, with the German Traffic Sign Recognition Benchmark (GTSRB), outstanding performance has been achieved by using MPPs. The accuracy of this model is 98.86%. The main motivation of this paper is to present a novel scheme for traffic sign recognition. The main characteristics of the proposed system include:

- A CNN model to learn a compact yet discriminative feature representation.

- A novel method to perform classification based on MPPs.

- A novel method to improve classification performance and speed using MPPs.

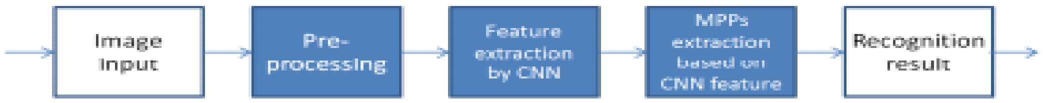


Image input
Pre-processing
Feature extraction by CNN
MPPs extraction based on CNN feature
Recognition result

Fig 2.1.2 System Overview

Network Architecture in Training Stage

The network consists of three convolution stages followed by fully connection layers and softmax layer. Each convolution stage includes convolutional layer, non-linear activation layer and max pooling layer. ReLU [5] is employed as the activation function for convolutional layers and full connection layers. Local response normalization (LRN) is used for normalizing feature maps. Dropout [5] is also adopted for preventing over-fitting. The final softmax layer has 43 outputs, corresponding to each category in GTSRB.

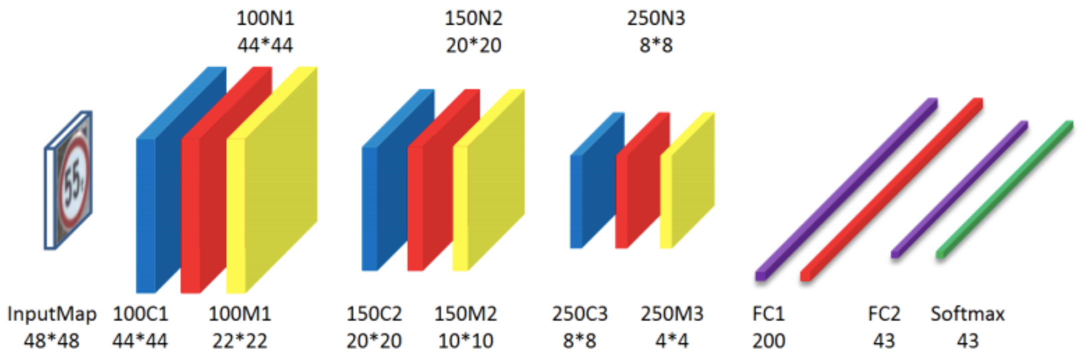
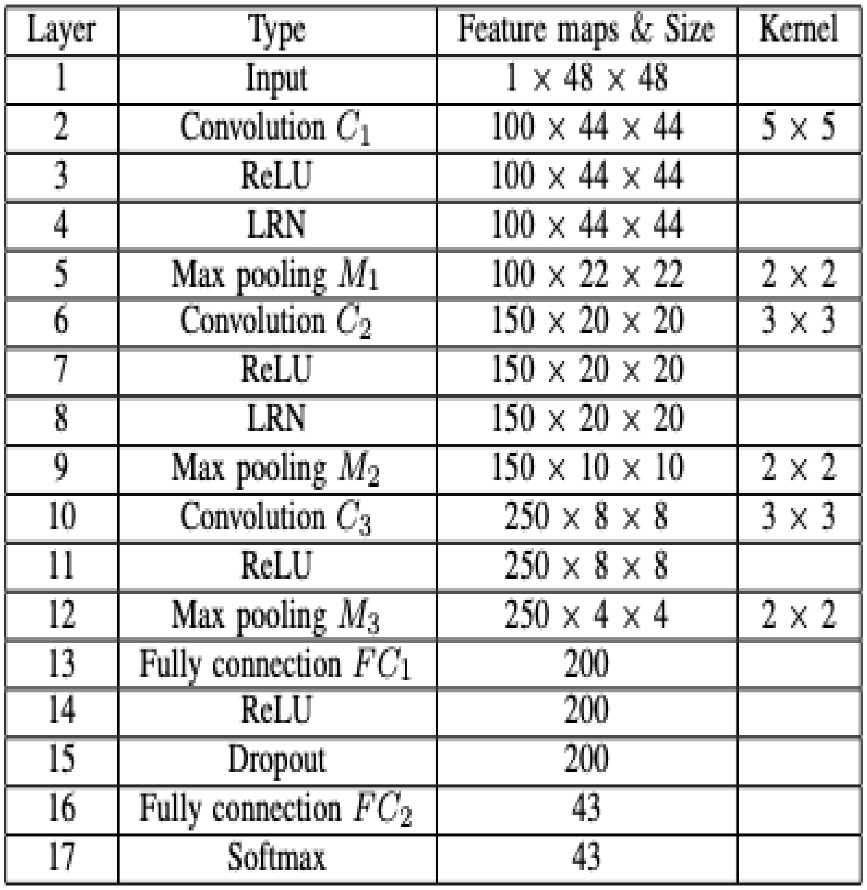


Fig 2.1.3 Network architecture in Training Phase



Layer	Type	Feature maps & Size	Kernel
1	Input	$1 \times 48 \times 48$	
2	Convolution C_1	$100 \times 44 \times 44$	5×5
3	ReLU	$100 \times 44 \times 44$	
4	LRN	$100 \times 44 \times 44$	
5	Max pooling M_1	$100 \times 22 \times 22$	2×2
6	Convolution C_2	$150 \times 20 \times 20$	3×3
7	ReLU	$150 \times 20 \times 20$	
8	LRN	$150 \times 20 \times 20$	
9	Max pooling M_2	$150 \times 10 \times 10$	2×2
10	Convolution C_3	$250 \times 8 \times 8$	3×3
11	ReLU	$250 \times 8 \times 8$	
12	Max pooling M_3	$250 \times 4 \times 4$	2×2
13	Fully connection FC_1	200	
14	ReLU	200	
15	Dropout	200	
16	Fully connection FC_2	43	
17	Softmax	43	

Fig 2.1.4 Selection of CNN Parameters

Network Architecture in Testing Stage

The network applied in testing stage is illustrated in Figure below. It is easily noticed that the original full connection layers and softmax layer are replaced with 903 new full connection layers. Each of the full connection layers can be regarded as a one-versus-all classifier. Instead of training all the weights by the standard back-propagation algorithm, all the parameters can be simply selected by using our MPPs method.

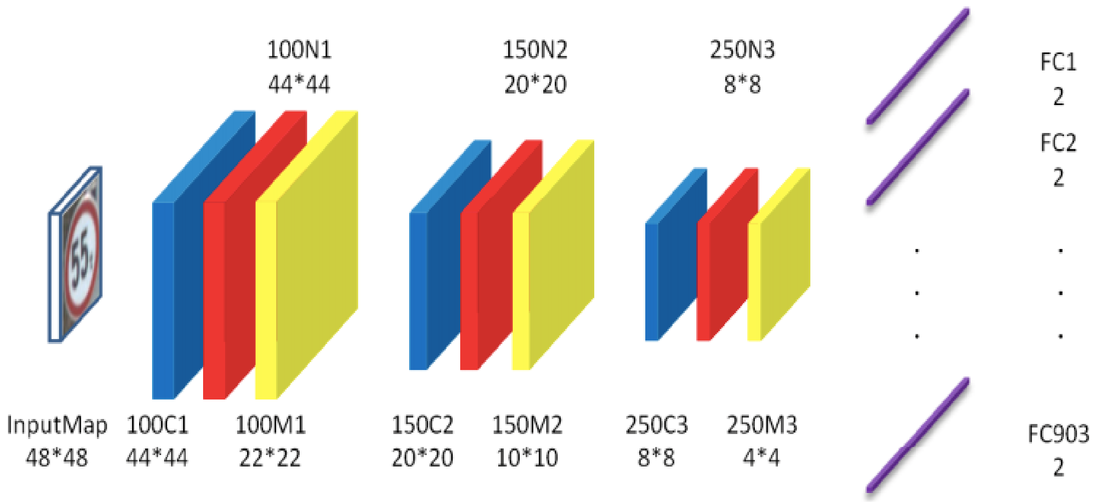


Fig 2.1.5 Network Architecture in Test Phase

Through this paper, a novel traffic sign recognition system is proposed, with main contributions including: (i) a CNN model to learn a compact yet discriminative feature representation; (ii) a novel method to perform recognition based on MPPs; (iii) a novel method to improve classification performance and speed using MPPs. By introducing MPPs for recognition, accuracy rate is significantly improved.

2) CNN models used for Classification

There have been variants of CNNs that have been used for the classification of images. In [5] and [6], a two-stage convolutional neural network architecture has been used for classification. For extraction, they used conventional feature extractors(Histogram of Oriented Gradients). The accuracy in [5] is 97% and the CNN architecture contains 2 sets of convolutional layers containing ReLu and pooling layers. In [7], a similar structure was used. However, it is different in that, after the traditional feature extraction, it used the CNN to learn stage-by-stage. The main modification in this architecture is that features from the 1st stage were fed into the classifier along with the 2nd stage features, and this produced more accurate results than any other mechanism. The accuracy achieved here was 99.17% by increasing the network's depth and by disregarding the color information.

3) CNN models used for Detection and Classification

Research [8] used a CNN model to do both: detection and classification. This allowed them to have a more selective feature representation. The detection is done using a color-based region proposal with a multi-tasking CNN, and classification is done simultaneously. The CNN architecture contains of 3 sets of convolution layers with pooling layers followed by 2 Fully connected layers. The accuracy obtained is 98.83%. The approach in [9] is different from others because it uses text-based traffic sign detection using two CNNs. The first CNN detects the signs and the regions of interest, and the second CNN detects the text. The second CNN is based on the model proposed in [10]. The first CNN model contains of 5-stages of convolutions and cascading them at the end to propose the ROI (regions of interest). The precision for this method is 90% and the time taken for classification is 0.15s.

4) Fast Branch CNN

In [11], the researchers add a branch-output mechanism to the deep CNN which speeds up the testing process of TSR. Inspired by the biological process, this mechanism increases speed and accuracy by introducing past knowledge of signs and their characteristics. Therefore, the detection of a small section of a sign or simply its shape can be enough to classify it, whereas other mechanisms must still process the entire image before classifying it. Before Fast Branch was developed, most mechanisms operated under the assumption that the system needed to see the entire image, and this turned out to be a drawback. The CNN architecture used in this model contains 3 sets of convolutional layers which comprise of convolutions followed by a pooling layer and a normalization layer. The branching out takes place after the convolution and before the pooling layer. In theory there can be n number of sets of these layers. The accuracy is 98.52%.

Overall architecture of a BCNN is illustrated in Fig. 3 and the core concept is an early stopping in classifying a sample. Suppose we have changed a pre-trained CNN model to a BCNN model, the difference between CNNs and BCNNs is that BCNNs predict some traffic signs in a previous layer instead of the original output layer. As illustrated in Fig. 3, samples may be predicted in one of the branches (Branch out 1, Branch out 2,…,Branch out n) instead of the original output layer (Softmax output layer). The dashed arrows in Fig. 3 mean that the corresponding branch may be nonexistent. Whether to branch or not depends on an optimization of time consumption and a criterion guaranteeing the accuracy. Clearly, the more samples the BCNN predict in previous layers, the more time the BCNN saves. That’s why BCNNs perform faster than general CNNs models1 on the same condition.

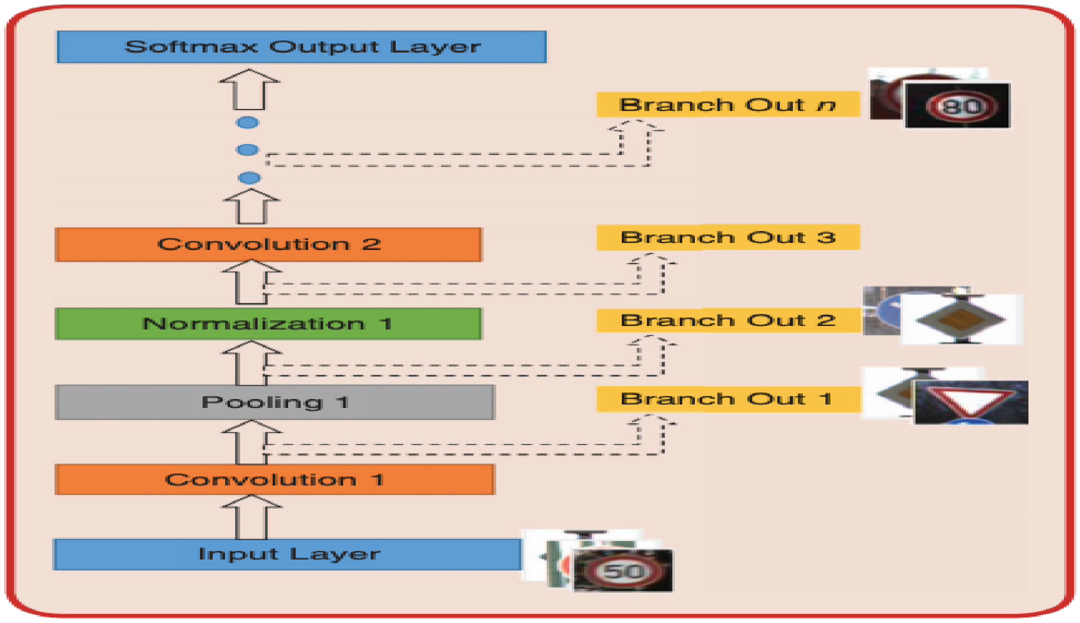


Fig 2.1.6 Structure Of BCNN

CNN is a multi-layer neural network model, which is made up of two main parts. The first part contains alternating convolution and pooling. The second part is fully-connection and soft-max classification. In convolution layer, each neuron connects a local of the input, and neurons in the same feature map share the weights, which functions like a convolution with a filter:

$$\mathbf{u}_{m,n} = \mathbf{f} * \mathbf{x} = \sum_{i=1}^{H} \sum_{j=1}^{W} f_{i,j} x_{i+m,j+n} + b$$

where **f ε R^(H*W)** is the weight matrix of filters, **x** is the input maps and **b** is bias. Pooling layers is a way to gain an invariant feature by aggregating low-level feature over a small neighborhood [35]. Average pooling and max-pooling are the most widely used pooling methods. Suppose the size of pooling kernel is **k** × **k** and stride is d, the result of pooling is:

$$\textit{average pooling}: \mathbf{v}_{m,n} = \frac{1}{k^2} \sum_{i=1}^{k} \sum_{j=1}^{k} x_{i+(m-1)\times d, j+(n-1)\times d}$$

$$\textit{max-pooling}: \mathbf{v}_{m,n} = \max_{i\in[1,k], j\in[1,k]} x_{i+(m-1)\times d, j+(n-1)\times d}$$

Deep CNNs are end-to-end models. Usually, the input is a raw image and the output is a one-hot vector. Besides, the output layer is commonly a softmax layer. Intuitively, deep CNN models transfer the raw images space to a linearly separable space, because softmax classifier is a multi-class linear classify. Conforming to the cognition, it is easy to identify the shape (triangle, rhombus and round), while it is relatively hard to identify the content of speedlimit signs (80 km/h, 70 km/h, 50 km/h and 30 km/h). So, we can find the following characteristics:

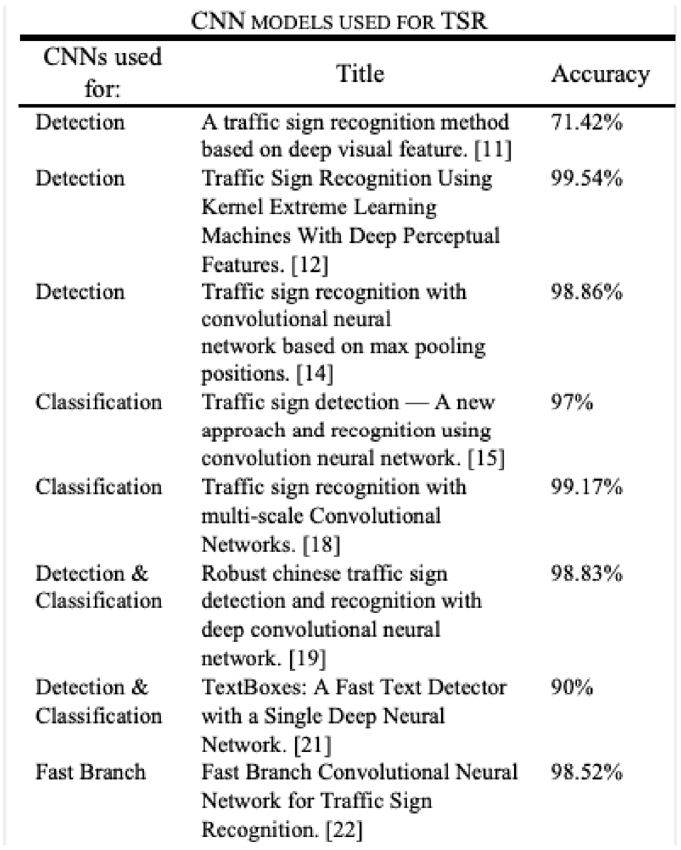
1) CNN models have an ability to gather samples with the same label together.

2) CNN models have a tendency to transfer the raw confused images space into a linear separable space.

3) The harder recognized a traffic sign is, the latter isolated it is.

This framework is faster than general deep CNN on the same condition. What's more, the change of data-flow can also help us to reduce the scale of a pre-trained CNN model. We test the framework on GTSRB. Experiments show that large numbers of traffic signs can be separated out in a shallow layer.

Table 2.1.1 Summary Of all the CNN models reviewed



CNN MODELS USED FOR TSR

CNNs used for:	Title	Accuracy
Detection	A traffic sign recognition method based on deep visual feature. [11]	71.42%
Detection	Traffic Sign Recognition Using Kernel Extreme Learning Machines With Deep Perceptual Features. [12]	99.54%
Detection	Traffic sign recognition with convolutional neural network based on max pooling positions. [14]	98.86%
Classification	Traffic sign detection — A new approach and recognition using convolution neural network. [15]	97%
Classification	Traffic sign recognition with multi-scale Convolutional Networks. [18]	99.17%
Detection & Classification	Robust chinese traffic sign detection and recognition with deep convolutional neural network. [19]	98.83%
Detection & Classification	TextBoxes: A Fast Text Detector with a Single Deep Neural Network. [21]	90%
Fast Branch	Fast Branch Convolutional Neural Network for Traffic Sign Recognition. [22]	98.52%

Convolutional Neural Networks (CNN) are better models for classification and detection of objects in visual tasks, specifically traffic sign recognition (TSR). Changing the order of connections in the convolutional and pooling layers can improve the accuracy, efficiency, and cost-effectiveness of the systems, considering their application in autonomous vehicles. In all, the accuracy of these different models is observed to be reliable and may have productive applications with autonomous vehicles and other real-world technologies. While conventional models continue to go step-by-step, Fast Branch CNNs present a more selective and time-efficient model, and they do so by eliminating past assumptions in the conventional wisdom. It is imperative that more research is done in this field for better solutions because, just as Fast Branch was able to make improvements, future systems will also have to eliminate hidden assumptions; only then will Traffic Sign Recognition through Convolutional Neural Networks achieve the most optimal results.

2.2 Lane Detection:

Identifying lanes on the road is a common task performed by all human drivers to ensure their vehicles are within lane constraints when driving, so as to make sure traffic is smooth and minimize chances of collisions with other cars due to lane misalignment. Similarly, it is a critical task for an autonomous vehicle to perform. It turns out that recognizing lane markings on roads is possible using well known computer vision techniques.

In this module, we propose a lane detection method based on the Hough Transform double edge extraction. The detection of lane lines is a hot topic in the field of computer vision. At present, there are two kinds of lane line detection methods proposed by scholars at home and abroad: model-based and feature-based. The model-based method is mainly used to extract the lane line by matching the feature point of the driveway line and the geometric model of the lane line. The feature-based method mainly detects lane lines through some low-level features such as gradients, directions, and gray values of the lane line edge points. The interested targets of the algorithm are always the nearest two lanes to the automobile, and it can detect the left and right lane separately.

1) Robust lane detection and tracking for lane departure warning

In the paper, we will present a real-time lane-detection and tracking system which is distinguished from the previous ones in the following ways:

1) It uses a more understanding algorithm to deal with the nearest two lanes, in case the typical road which has multiple lanes causes difficulty in generating appropriate departure decision.

2) It detects the left and right lane markings separately, whereas most of the previous work uses a fixed-width lane model. As a result, it can handle challenging scenarios such as merging or splitting lanes effectively even if one lane is gone, it can also provide warning based on the other lane.

3) It combines lane detection and tracking into a single algorithm, and there is more to be used than the information from a single image that can effectively deal with lane changes, such as

emerging, ending, merging, or splitting lanes.

A lot of papers use Hough transform to solve lane detection, while few of them utilized the prior knowledge which gets from the last image, and that's the biggest difference between the approach in this paper and the other ways. Since it is easy to obtain the initial and terminal points of the last scene, tracking lanes from frame to frame saves a lot of resources, and putting strong constraints on the likely location of the lane increases the accuracy of the algorithm.

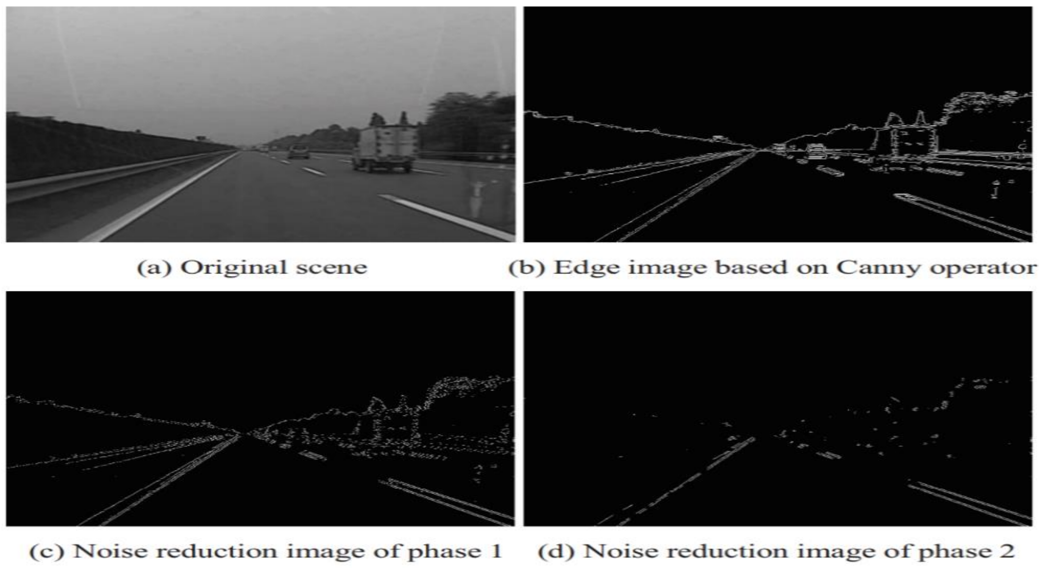

(a) Original scene
(b) Edge image based on Canny operator
(c) Noise reduction image of phase 1
(d) Noise reduction image of phase 2

Fig 2.2.1 Canny Edge Detection output

Edge Detection with Canny operator is used. Lane boundaries are defined by sharp contrast between the road surface and painted lines. The canny detector has a very desirable characteristic in that it does not produce noise like the other approaches. Canny operation provides the threshold automatically. However, it often produces far too much edge information including cars ahead and sceneries on the wayside as shown in (b), corresponding to the original image in (a). After observation and experiments, we summarize a scheme to decrease the redundant information. Firstly, delete the points that link with each other horizontally or vertically to reduce the horizontal and vertical line, which has been shown in (c). These lines turn out to be cars ahead and other road signs. Secondly, wipe off the stray points that scattering in the edge image alone as shown in (d).

To ascertain the initial points is the key to locate the lane markers precisely. They choose the search-area about a quarter of the image which belongs to the bottom of the scene. We model a scan-line as having an intensity profile with a uniform intensity I-marker distinct from the intensity I-road. The orientation of searching in one scan-line is important. It's convenient to look for one point from the midpoint of the line to one side, and then start to search the other point in the opposite direction, which may not worry about the influence from the obstacle in another lane.

The lane can be recognized based on two points. In fact, we can obtain the lines after detecting the initial ones. However, in order to prevent the distant scenery disturbing the surveillance of the road markers, the monitoring area is obtained which is about half of the road surface.

The rule above is suitable to estimate the location of lane in the next image. Since the displacement of the lanes between two successive scenes is small, the actual points of lane change a little. In the result of that, we can utilize the endpoints in the last image to obtain the lanes in the following one, and skip the phases of searching and confirming. The range of the endpoint is set as follows. The mid-value of the range should be the endpoint from the last one, and the span from the minimum to the maximum is about 10 pixels. In most instances, the endpoints in successive images changes little, so the optical searching choice is to start from the middle of the confines. If the model doesn't fit with the image, choose the one next to the mid-value on the left (or right) to see if this one fits or not, and next time is the one on the right (or left).

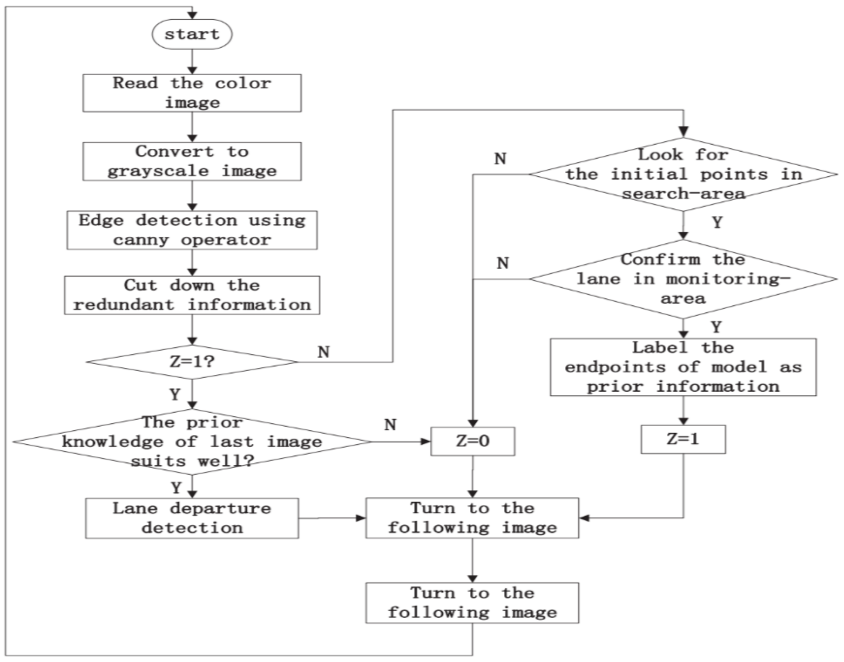

start
Read the color image
Convert to grayscale image
Edge detection using canny operator
Cut down the redundant information
Z=1?
N
Y
The prior knowledge of last image suits well?
N
Y
Lane departure detection
Look for the initial points in search-area
N
Y
Confirm the lane in monitoring-area
N
Y
Label the endpoints of model as prior information
Z=0
Z=1
Turn to the following image
Turn to the following image

Fig 2.2.2 System Flowchart of (A)

2) **Reduced resolution lane detection algorithm**

To detect obstacles and pedestrians, LIDAR sensors are widely used in the research, but there will still be some dead zone for it. For example, small objects that have lower height than the detection height of the LIDAR (2D LIDAR) will be ignored under the detection of the LIDAR. Therefore, a camera captured image with high quality would be a complement for the system to find the potential danger in front of the vehicle.

A typical lane detection method follows a process with the following steps:

- Capture a frame from the camera sensor.
- Convert the original image to grayscale image
- Threshold the image

- Find the lane mark by lane line or lane pixel
- Final filter.

With the above steps, most researchers usually focus on improving the accuracy of the detected lane mark and increase the detection speed. In order to improve the accuracy, one important step is to filter out the noise in the edge detection frame. Due to the natural light, environment, reflection and shadow, there could be many small noise spots in the edge detection result even it did not show too much noise for some other conditions. The paradox for this step is: a higher threshold value could filter out lane marks as well as other noise. The key point to locate the right position of the lane with the lane detection technique based on edge detection is to find and apply an appropriate threshold value. To obtain appropriate threshold value, iteration methods are usually used by researchers. However, with a high-resolution image, too many iterations will severely harm the efficiency of the image processing speed. Another issue brought by this approach is: after edge detection and Hough line transform method, an additional filter must be used in order to locate the lane marks from the lines found. Although most of the lines are on the lane mark, some noise could still exist, such as the outline of some vehicles, the buildings on the road side, barriers on the high way, etc. These noises will still affect the accuracy of the results of detection. In this paper, an effective and efficient Reduced Resolution lane detection algorithm (R2 algorithm) is employed.

The method includes two modes: The Iteration mode (I-mode) and the Color Threshold mode (C mode) as shown below:

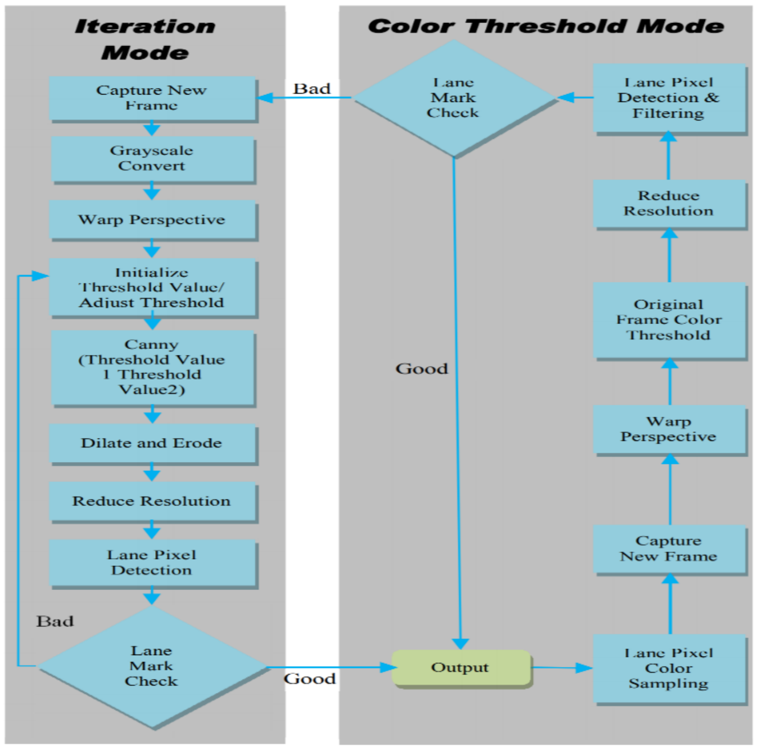


Fig 2.2.3 System Flowchart of (C)

I-mode is guaranteed to find the lanes, but it has a lower processing speed (usually 5-15 frames per second). The C-mode can filter out noise by color threshold, then apply the lane mark detection to the color threshold image. The C-mode could lose lane detection accuracy when the environment color is very close to the lane color, however, it offers a significantly increased level of processing speed (up to 100 frames per second). The program will mainly work under C-mode to maintain a high processing speed, and it will switch to I-mode when a sudden change occurs to the captured frame, like light condition change, big shadow from the environment or weird marks on the ground that look similar to the lane.

The BGR image needs to be converted to a grayscale image so that Canny edge detection could be applied. To make it easier to detect the lane and calculate the offset of the lines, a "bird view" will be very helpful. After the "bird view" transform, Canny edge detection, dilate and erode methods are applied. There might be some noise that affect the accurate detection of the lane marks after the first Canny edge detection, if so, a threshold value iteration will be applied after lane mark is located. Next step is the key step: reduce the resolution of the Canny threshold image. As shown in (e), the resolution of the threshold image is decreased by 10 times, but the shape of the lane remains the same. With the new threshold image, which has only 1/100 of the original pixel numbers, it will be very easy to find lane pixels on it with some of its characteristics.

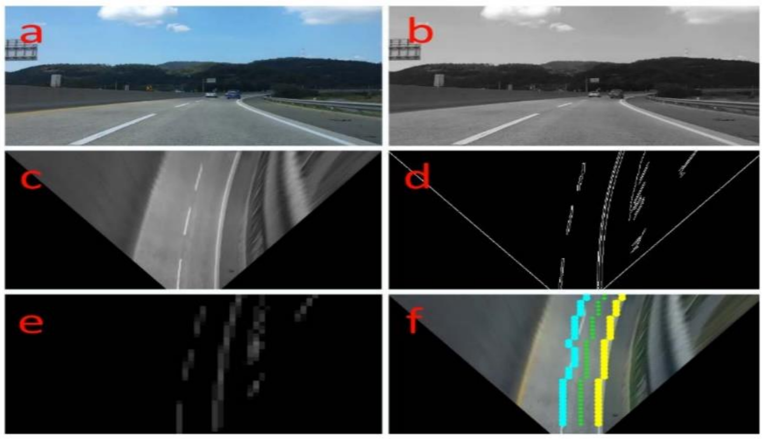


Fig 2.2.4 Perspective Transform output

The main difference between the C-mode and I-mode is: C-mode is using color threshold to create a threshold image for lane mark detection instead of employing Canny, dilating and eroding as the approach to create the threshold image. Every time the program jump into C-mode from I-mode, it will first pick BGR values from the lane mark as a sample lane color. Here it will first pick BGR values from the lane mark as a sample lane color. Then it will capture a new frame Figure (a), convert BGR to grayscale Figure (b) and warp perspective to a "bird view" image Figure (c). In the next step, the program will apply color threshold to the "bird view" image as shown Figure (d). Less elements to process other than the lane mark makes it much easier for the program to locate the lane marks. In addition to this, reduced resolution of the color threshold image significantly reduce the pixels needed to be processed.

The figure below shows the threshold image from the I-mode and C-mode. It is easy to find that the threshold image from the C-mode has less noise than that from the I-mode. However, because the color threshold could make some error accidentally, the I-mode will always provide a result that is more reliable.

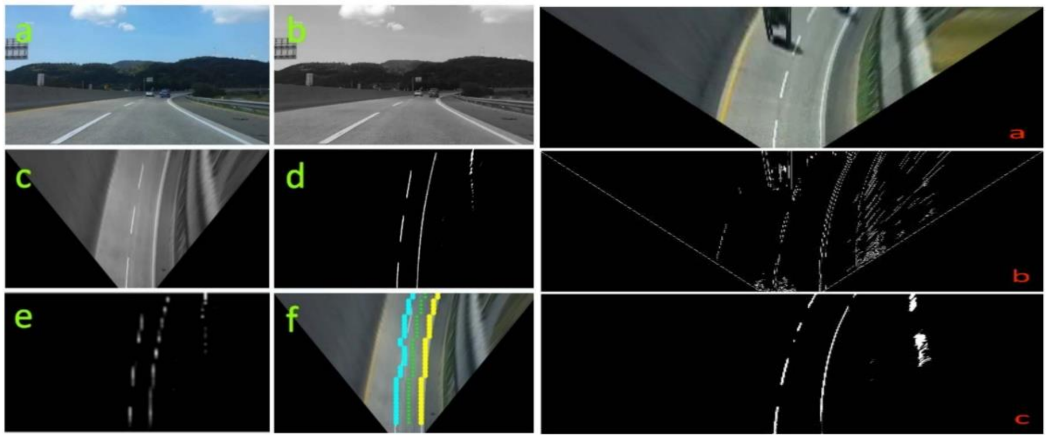


Fig 2.2.5 Output comparison between C and I modes

3) Double Lane Line Edge Detection Method Based on Constraint Conditions Hough Transform

Because the camera source is in the front side of the vehicle, this paper divides the lane line image into three parts: near field, far vision field and sky area. The near vision field area and far vision field area contain lane line information, and the sky area contains no important lane line information.

In the RGB image, the values of the R and G components of the white lane line and the yellow lane line are significantly higher than the R and G values of the road surface and the background. Therefore, according to the R, G components of the lane line image, it is possible to determine which pixel points in the image are the pixel points included in the lane line, and determine the color of the lane line through the B component.

In this paper, Canny edge detection operator is used to filter out edges that meet the characteristics of lane lines. Therefore from the near field part, the neighboring 8 pixel points are evaluated for all non zero pixel values. When the total sum of all the nine pixel values are greater than 0, it is considered as the lane line edge pixel point.

The polar angle range used in this paper is -90° to 90°. The steps for detecting the straight line based on the constraint condition Hough transform are as follows:

Step 1: Quantize (p, theta) into equally spaced small grids by combining the polar Angle and the polar radius.

Step 2: Build a two-dimensional additive matrix A1. Each value of the matrix represents a small grid, and the value represents the number of grids passing through (p, theta). The pixels of the near-field part of the edge image are sequentially traversed, and the pixel points whose pixel value is not 0 are mapped as a sine curve of - space. As long as the curve passes through the small grid, the corresponding value of the accumulation matrix is incremented by one.

Step 3: Find the largest four groups (p, theta) in A1 by traversing the accumulation matrix A1. After obtaining the four largest groups (,), the starting and ending coordinates of the two edges of the left and right lane lines are determined by .

After obtaining the four largest groups (p, theta), the starting and ending coordinates of the two edges of the left and right lane lines are determined by theta.

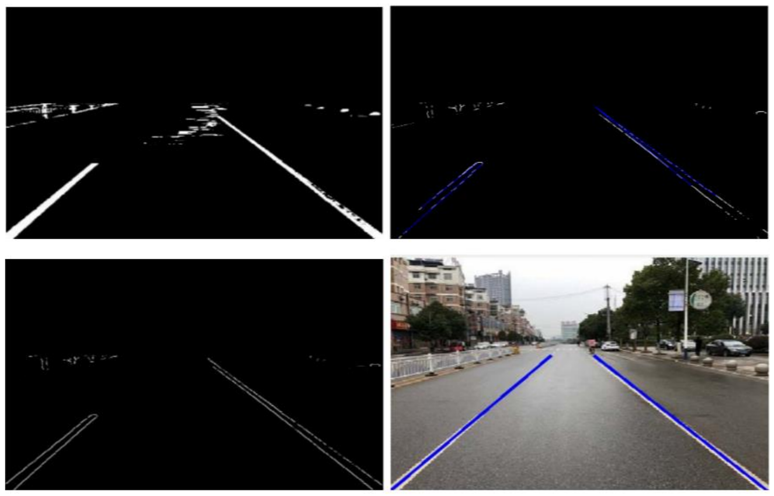


Fig 2.2.6 Final Output of (C)

2.3 Drowsiness Detection:

This implementation comes from the premise of detecting eye blinks in systems that monitor a human operator. Existing methods are active (reliable, but require special expensive hardware like IR cameras, special glasses) or passive (rely only on a remote camera). Several methods in the past have been proposed – some using motion estimation in the eye using a a Viola-Jones type detector. A major drawback of the previous approaches is that they usually implicitly impose too strong requirements on the setup, in the sense of a relative face-camera pose (head orientation), image resolution, illumination, motion dynamics, etc. Especially the heuristic methods that use raw image intensity are likely to be very sensitive despite their real-time performance.

Today, most of the SOTA landmark detectors formulate a regression problem where a mapping from one image into landmark positions or into another landmark parameterization is learned. These modern landmark detectors are trained on “in-the-wild datasets” and they are thus robust to varying illumination, various facial expressions, and moderate non-frontal head rotations. An average error of the landmark localization of a state-of-the-art detector is usually below five percent of the inter-ocular distance.

The eye blink is a fast closing and reopening of a human eye. Each individual has a little bit different pattern of blinks. The eye blink lasts approximately 100-400 ms. We propose to exploit state-of-the-art facial landmark detectors to localize the eyes and eyelid contours. From the landmarks detected in the image, the eye aspect ratio (EAR) that is used as an estimate of the eye-opening state, is derived. Since the per frame EAR may not necessarily recognize the eye blinks correctly, a classifier that takes a larger temporal window of a frame into account is trained.

For every frame, eye landmarks are detected. The EAR between height and width is computed:

$$\text{EAR} = \frac{||p_2 - p_6|| + ||p_3 - p_5||}{2||p_1 - p_4||},$$

Where *p1,…., p6* are the 2D landmark locations. This EAR is constant when an eye is open and tends to zero while closing an eye. Aspect ratio of the open eye has a small variance among individuals and it is fully invariant to a uniform scaling of the image and in-plane rotation of the face. Since eye blinking is performed by both eyes synchronously, the EAR of both eyes is averaged. A low value of the EAR may occur when a subject closes his/her eyes intentionally for a longer time or performs a facial expression. This is implemented by a linear SVM classifier (called EAR SVM) trained from manually annotated sequences. While testing, a classifier is executed in a scanning-window fashion. A 13-dimensional feature is computed and classified by

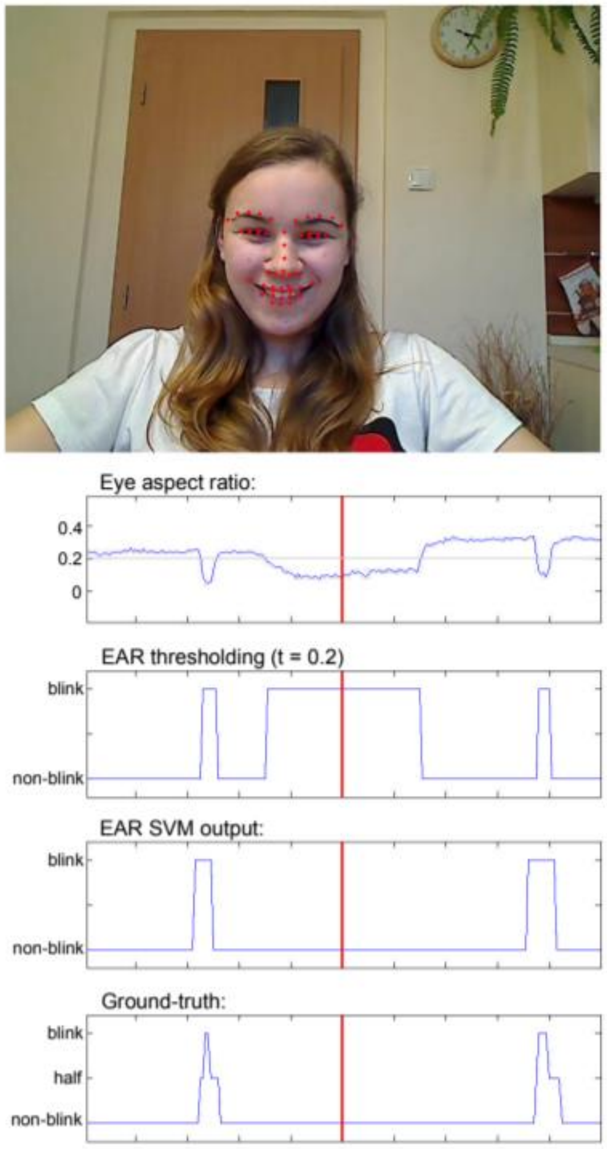


Figure 2.3.1 Ground truth vs SVM

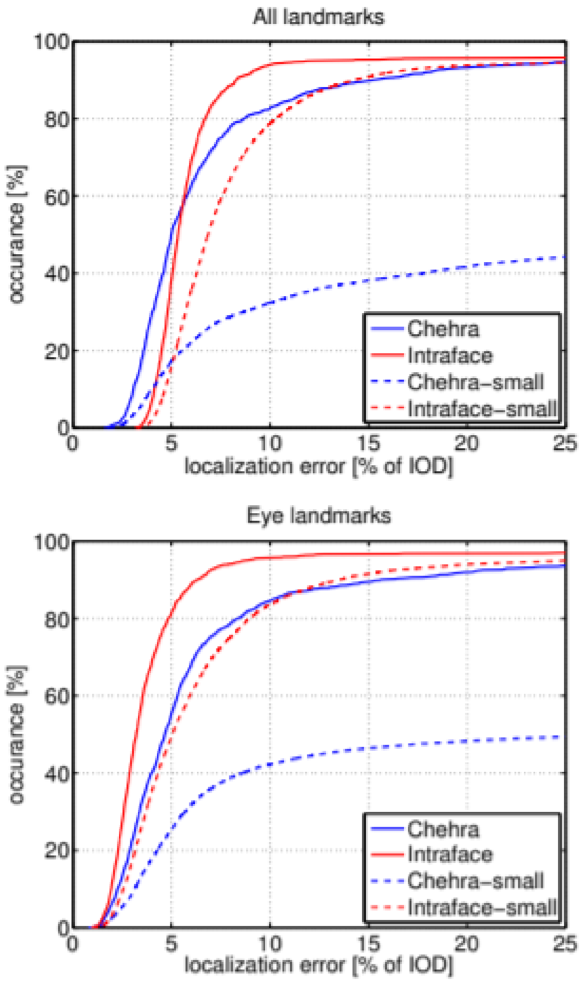


Figure 2.3.2 Occurance vs Error

EAR SVM for each frame except the beginning and ending of a video sequence. To evaluate accuracy of tested landmark detectors, we used the 300-VW dataset [19]. It is a dataset containing 50 videos where each frame has associated a precise annotation of facial landmarks. The videos are "in-the-wild", mostly recorded from a TV. Two state-of-the-art landmark detectors were tested: Chehra and Intraface. Faces are not always frontal to the camera, the expression is not always neutral. Sometimes people wear glasses, hair may occasionally partially occlude one of the eyes. Both detectors perform generally well, but the Intraface is more robust to very small face images.

The accuracy of the landmark detection for a face image is measured by the average relative landmark localization error:

$$\epsilon = \frac{100}{\kappa N} \sum_{i=1}^{N} ||x_i - \hat{x}_i||_2,$$

Xi = the ground-truth location of landmark i in the image, xˆi = estimated landmark location by a detector, N = number of landmarks and normalization factor κ = the inter-ocular distance (IOD), i.e. Euclidean distance between eye centers in the image.

The experiment with EAR SVM is done in a cross-dataset fashion. It means that the SVM classifier is trained on the Eyeblink8 and tested on the ZJU and vice versa. To evaluate detector accuracy, predicted blinks are compared with the ground-truth blinks. The number of true positives is determined as a number of the ground-truth blinks which have a non-empty intersection with detected blinks. The number of false negatives is counted as a number of the ground-truth blinks which do not intersect detected blinks. The number of false positives is equal to the number of detected blinks minus the number of true positives plus a penalty for detecting too long blinks. State-of-the-art on two standard datasets was achieved using the robust landmark detector followed by a simple eye blink detection based on the SVM. The algorithm runs in real-time, since the additional computational costs for the eye blink detection are negligible besides the real-time landmark detectors. The proposed SVM method that uses a temporal window of the eye aspect ratio (EAR), outperforms the EAR thresholding. On the other hand, the thresholding is usable as a single image classifier to detect the eye state, in case that a longer sequence is not available.

2.4 Vehicle Detection:

1) **Vision-based vehicle detection and counting system using deep learning in highway scenes**

   Vehicle detection and statistics in highway monitoring video scenes are of considerable significance to intelligent traffic management and control of the highway. With the popular installation of traffic surveillance cameras, a vast database of traffic video footage has been obtained for analysis. Generally, at a high viewing angle, a more-distant road surface can be considered. The object size of the vehicle changes greatly at this viewing angle, and the detection accuracy of a small object far away from the road is low. In the face of complex camera scenes, it is essential to effectively solve the above problems and further apply them.

At present, vision-based vehicle object detection is divided into traditional machine vision methods and complex deep learning methods. Traditional machine vision methods use the motion of a vehicle to separate it from a fixed background image. This method can be divided into three categories: the method of using background subtraction, the method of using continuous video frame difference, and the method of using optical flow. Using the video frame difference method, the variance is calculated according to the pixel values of two or three consecutive video frames. Moreover, the moving foreground region is separated by the threshold. By using this method and suppressing noise, the stopping of the vehicle can also be detected. When the background image in the video is fixed, the background information is used to establish the background model. Then, each frame image is compared with the background model, and the moving object can also be segmented. The method of using optical flow can detect the motion region in the video. The generated optical flow field represents each pixel's direction of motion and pixel speed.

The use of deep convolutional networks (CNNs) has achieved amazing success in the field of vehicle object detection. CNNs have a strong ability to learn image features and can perform multiple related tasks, such as classification and bounding box regression. The detection method can be generally divided into two categories. The two-stage method generates a candidate box of the object via various algorithms and then classifies the object by a convolutional neural network. The one-stage method does not generate a candidate box but directly converts the positioning problem of the object bounding box into a regression problem for processing. In the two-stage method, Region-CNN (R-CNN) uses selective region search in the image. The image input to the convolutional network must be fixed-size, and the deeper structure of the network requires a long training time and consumes a large amount of storage memory.

The YOLO network divides the image into a fixed number of grids. Each grid is responsible for predicting objects whose centre points are within the grid. YOLOv2 added the BN (Batch Normalization) layer, which makes the network normalize the input of each layer and accelerate the network convergence speed. YOLOv2 uses a multi-scale training method to randomly select a new image size for every ten batches. Our vehicle object detection uses the YOLOv3 network. Based on YOLOv2, YOLOv3 uses logistic regression for the object category.

The implementation of the highway vehicle detection framework used the YOLOv3 network. The YOLOv3 algorithm continues the basic idea of the first two generations of YOLO algorithms. The convolutional neural network is used to extract the features of the input image. According to the size of the feature map, such as 13*13, the input image is divided into 13*13 grids. The centre of the object label box is in a grid unit, and the grid unit is responsible for predicting the object. The network structure adopted by YOLOv3 is called Darknet-53. This structure adopts the full convolution method and replaces the previous version of the direct-connected convolutional neural network with the residual structure. The branch is used to directly connect the input to the deep layer of the network. Direct learning of residuals ensures the integrity of image feature information, simplifies the complexity of training, and improves the overall detection accuracy of the network. In YOLOv3, each grid unit will have three bounding boxes of different scales for one object. The candidate box that has the largest overlapping area with the annotated box will be the final prediction result. Additionally, the YOLOv3 network has three output scales, and the three scale branches are eventually merged. Shallow features are used to detect small objects, and deep features are used to detect large objects; the network can thus detect objects with scale changes. The detection speed is fast, and the detection accuracy is high. When using YOLO detection, images are resized to the same size, such as 416*416, when they are sent to the network. Since the image is segmented, the size of the remote road surface becomes deformed and larger. Therefore, more feature points of a small vehicle object can be acquired to avoid the loss of some object features due to the vehicle object being too small.

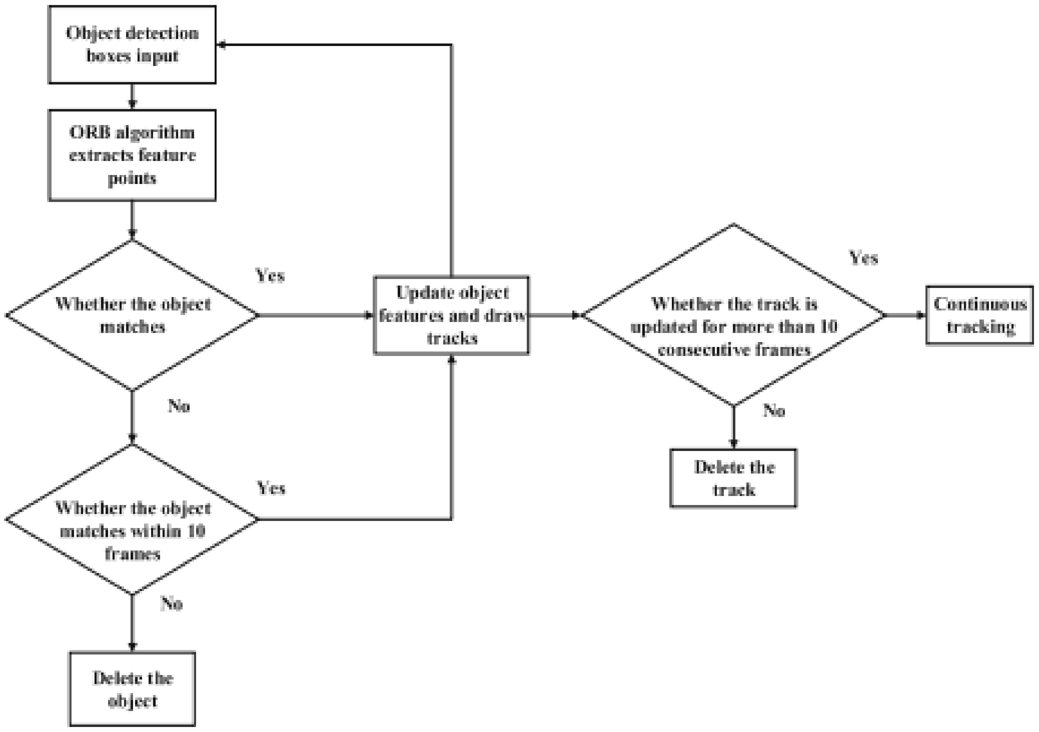

Object detection boxes input
ORB algorithm extracts feature points
Whether the object matches
Yes
No
Update object features and draw tracks
Whether the track is updated for more than 10 consecutive frames
Yes
Continuous tracking
No
Delete the track
Whether the object matches within 10 frames
Yes
No
Delete the object

Fig 2.4.1 Process of multi object tracking

In this study, the ORB algorithm was used to extract the features of the detected vehicles, and good results were obtained. The ORB algorithm shows superior performance in terms of computational performance and matching costs. This algorithm is an excellent alternative to the SIFT and SURF image description algorithms. The ORB algorithm uses the Features From Accelerated Segment Test (FAST) to detect feature points and then uses the Harris operator to perform corner detection. After obtaining the feature points, the descriptor is calculated using the BRIEF algorithm. The coordinate system is established by taking the feature point as the center of the circle and using the centroid of the point region as the x-axis of the coordinate system.

2) Vehicle Detection and Car Type Identification System using Deep Learning and Unmanned Aerial Vehicle

Vehicle detection and type-identification systems through UAVs with advantages in time, space and cost have recently gained attention for their active research areas in establishing traffic flow monitoring systems. In recent years, the performance improvements of UAVs have been enabled the recording of ultra-high-resolution aerial images at low cost, which can cover a wide area. airships and UAVs can provide UHD-4K aerial images with relatively low costs. The proposed method consists of vehicle detection and type-identification method throughout deep learning-based networks trained in UHD-4K aerial images. In general, the detection speed and accuracy of object detection systems should be considered when designing object detection systems. The two performance indicators have closely correlated. Due to the complexity of detection algorithms, object detection accuracy is reduced if fast object detection is desired, an object detection speed is slowed if high object detection accuracy is desired. Therefore, the performance of the system depends on which object detection algorithm is selected, and the general object detection methods are divided into single frame analysis and multi-frame analysis techniques. Sophisticated and robust object detection algorithms provide high accuracy, but their detection speed may not guarantee to make them suitable for the real-time system. Simple object detection algorithms allow fast, real-time object detection but accuracy is down. The overall performance of the vehicle detection system that relies on quick, vehicle detection and classification is more influenced by the vehicle detector algorithms. In recent years, vehicle detection and classification methods that use deep learning algorithms have emerged to provide fast vehicle detection without loss of precision. Various methods have been conducted to develop accurate and fast vehicle detection and identification systems based on deep learning algorithms. Having a fair comparison among different vehicle detection algorithms performance is robust. There is no straight answer on which model is the best. For real applications, we make choices the detection algorithm to balance accuracy and speed.

Deep learning algorithms have been attracted a lot of attention in object detection and classification research groups recent. The basic concepts of deep learning algorithms are based on neural networks, developed in the 1980s. Early neural network algorithms have a disadvantage of some shortcomings such as overfitting, local minima, initial weights determination, and slow learning rates, and they could not recognize patterns of high complexity. But, in the mid-2000s, some researchers developed the learning methods that made better learning of many layers in neural networks with deep layers networks. Also, the development of GPGPU with substantial processing power has even much more decreased the calculation time for complex matrix operations recent, and deep learning algorithms have developed in a variety of object detection and classification algorithms. There are numerous deep learning algorithms, most of which are derived from early neural network algorithms. General deep learning networks consist of multiple layers. In general, deep learning networks are trained using the back-propagation that method used in neural networks to compute a gradient. Deep neural algorithms then train biases and weights using the gradient descent method. General deep learning algorithms have introduced concepts, such as dropout, mini batches, and pre-training for initial weights to overcome the problems in traditional neural network algorithms.

YOLO (you only look once) is one of the faster object detection algorithm recently. It is a good algorithm when you need rapid real-time detection without loss of much accuracy. YOLOv2 used a custom deep architecture darknet-19, 19-layer network supplemented with 11 more layers for object detection. With a 30-layer architecture, YOLOv2 often struggled with small object detections. It was attributed to a loss of fine-grained features as the layers downsampled the input. To overcome this, YOLOv2 used an identity mapping, concatenating feature maps from a previous layer to capture low-level features. However, YOLOv2's architecture was still lacking some of the most critical elements that are now stapled in most of the state-of-the-art algorithms (no residual blocks, no skip connections and no upsampling). But YOLOv3 incorporated all of these. Darknet-53 used in YOLOV3 is an advanced model from Darknet-19 and consists of 53 convolutional layers. It still relies on successive 3×3 and 1×1 filters but added residual blocks.

For the implementation of the proposed system, the deep learning network is trained various types of car images which are short cars (sedan, wagon, light van, etc.), medium cars (short towing, short trailer, caravan, bus, etc.) and long vehicles using Darknet-53 algorithm with real vehicle images. The vehicle images were taken in an almost orthogonal direction driving cars about the intersection at an altitude of 100-120m using the DJI Phantom 3 Professional. The test video is composed of UHD 3840×2160 images with a refresh rate of 30 FPS. The performance of a vehicle detector is dependent mainly on the performance of its deep neural network.

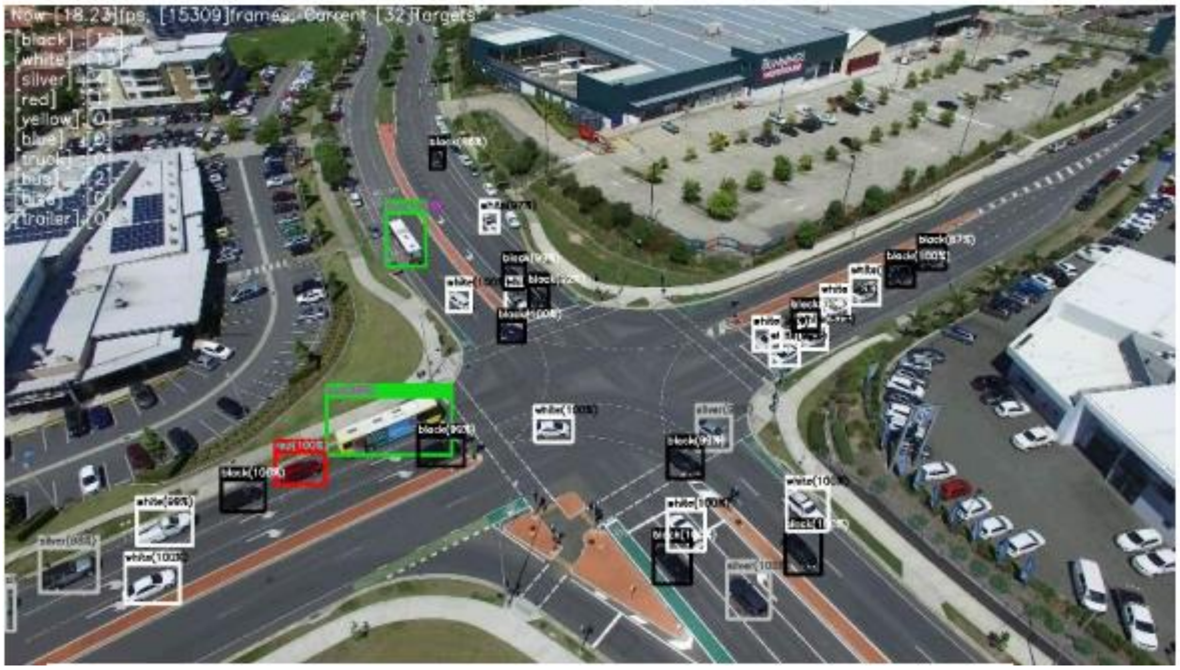


Fig 2.2.2 Result of vehicle classification by YOLOv3

The method proposed in this paper successfully detected and classified vehicle which moves at 23 FPS maximum and 17-20 FPS on average with UHD-4k video. The method with variable search ranges made minor errors than the method with fixed search ranges. Even though the experiment dataset used in the test was not enough big and diverse to make a clear statement, the method performed better than other reviewed methods.

3) YOLOv3: An Incremental Improvement

YOLOv3 predicts an objectness score for each bounding box using logistic regression. This should be 1 if the bounding box prior overlaps a ground truth object by more than any other bounding box prior. If the bounding box prior is not the best but does overlap a ground truth object by more than some threshold, we ignore the prediction. We use the threshold of 0.5. Unlike Faster-RCNN, this system only assigns one bounding box prior for each ground truth object. If a bounding box prior is not assigned to a ground truth object it incurs no loss for coordinate or class predictions, only objectness. Each box predicts the classes the bounding box may contain using multilabel classification. It do not use a softmax as the authors have found it is unnecessary for good performance, instead they simply use independent logistic classifiers. During training they use binary cross-entropy loss for the class predictions. This formulation helps when they move to more complex domains like the Open Images Dataset. In this dataset there are many overlapping labels (i.e. Woman and Person). Using a softmax imposes the assumption that each box has exactly one class which is often not the case. A multilabel approach better models the data.

YOLOv3 predicts boxes at 3 different scales. The system extracts features from those scales using a similar concept to feature pyramid networks. From the base feature extractor, add several convolutional layers. The last of these predicts a 3-d tensor encoding bounding box, objectness, and class predictions. Next, they take the feature map from 2 layers previous and upsample it by 2×. The next step is to take a feature map from earlier in the network and merge it with the upsampled features using concatenation. This method allows the authors to get more meaningful semantic information from the upsampled features and finer-grained information from the earlier feature map. They then add a few more convolutional layers to process this combined feature map, and eventually predict a similar tensor, although now twice the size. They perform the same design one more time to predict boxes for the final scale. Thus, the predictions for the 3rd scale benefit from all the prior computation as well as fine grained features from early on in the network. The system uses k-means clustering to determine our bounding box priors. It just sorts of chosen 9 clusters and 3 scales arbitrarily and then divides up the clusters evenly across scales.

Each network is trained with identical settings and tested at 256×256, single crop accuracy. Run times are measured on a Titan X at 256 × 256. Thus Darknet-53 performs on par with state-of-the-art classifiers but with fewer floating-point operations and more speed.

Darknet-53 is better than ResNet-101 and 1.5× faster. Darknet-53 has similar performance to ResNet-152 and is 2× faster. Darknet-53 also achieves the highest measured floating-point operations per second. This means the network structure better utilizes the GPU, making it more efficient to evaluate and thus faster. That's mostly because ResNets have just way too many layers and aren't very efficient.

4) Vehicle target detection in complex scenes based on YOLOv3 algorithm

With the popularity of family cars, traffic congestion is increasing, and the demand for intelligent traffic monitoring system is increasing, among which the real-time detection technology of vehicle targets is particularly critical. With the development of unmanned driving technology, real-time detection of vehicle targets is also the primary problem to be solved. In the actual scene, the vehicle target in the picture or video frame will encounter complex problems such as occlusion, background noise, light mutation, shooting angle, etc., which brings great difficulties to the traditional target detection methods, which will bring bad effect on detection. Therefore, different from traditional detection methods and combined with the current rapidly developing in deep learning detection algorithm, this paper proposes a vehicle target detection method based on YOLOv3 algorithm in complex scenes. Traditional vehicle target detection methods include: frame difference method, optical flow method, background difference method. These methods are mainly based on the change of information between video frames, rather than detection and recognition through the characteristics of vehicles. Such methods are fast, but have low detection accuracy and poor robustness for complex scenes. Therefore, machine learning method extracts target features, such as HOG features, SIFT and other methods, then inputs the extracted features into classifiers, such as SVM, for classification detection. With the rapidly development of deep learning theory and practice, target detection methods based on deep learning algorithm emerge rapidly. Different from the traditional feature extraction and matching algorithm, the deep convolution network has a certain degree of tolerance for geometric transformation, deformation and illumination. Driven by the training data, it can construct the feature description adaptively and has a good flexibility and expansibility.

The vehicle target detection process designed in this paper is shown below. Firstly, screening out samples which contain car targets from the data sets of VOC2007 and VOC2012 to constitute the VOC car data sets. Then putting the training samples into the YOLOv3 training network, and the training is carrying out until the network convergence. Finally, the weight.h5 file was loaded into the YOLOv3 model to test samples.

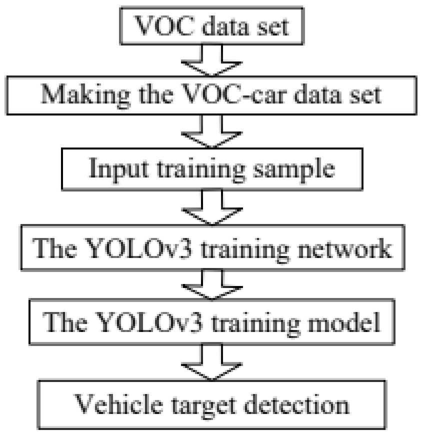
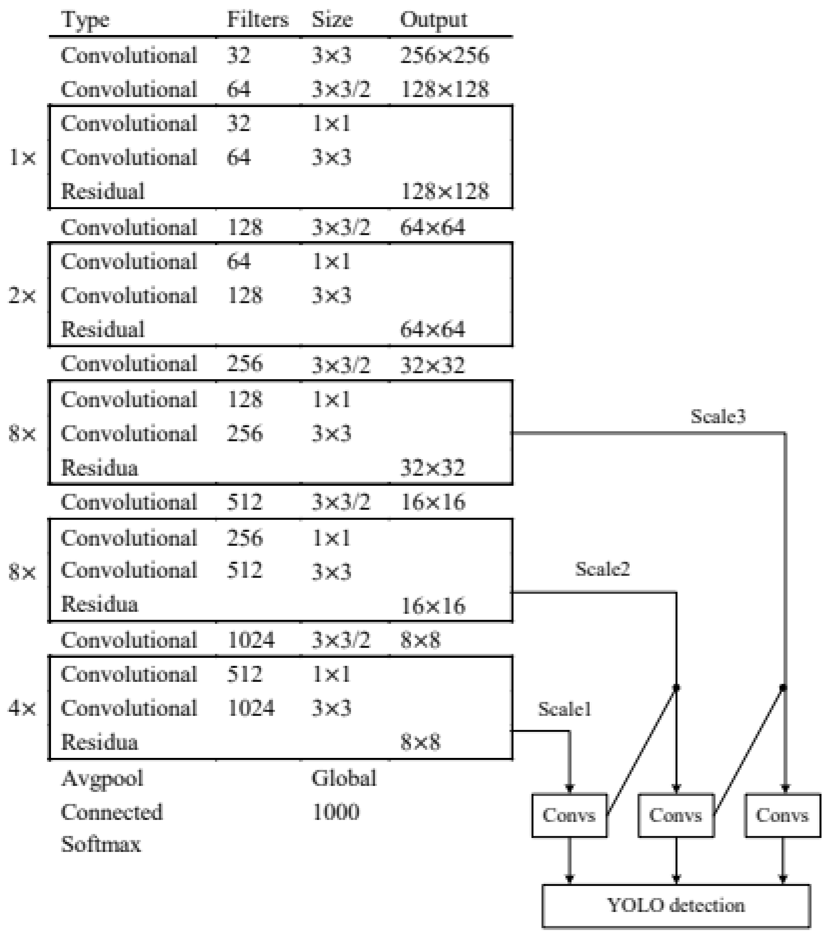

VOC data set
Making the VOC-car data set
Input training sample
The YOLOv3 training network
The YOLOv3 training model
Vehicle target detection

Fig 2.2.3 Flow chart of car target detection

The YOLO algorithm is a convolutional neural network that can predict multiple Box locations and categories at one time. In a real sense, it realizes the end-to-end target detection and plays the advantage of fast speed, but its accuracy decreases. However, the YOLO9000 algorithm is an improvement on the original YOLO algorithm, which maintains the speed and improves the accuracy at the same time. In this way, YOLO9000 can simultaneously train in the data sets of COCO and ImageNet, and the trained model can detect 9000 objects. YOLOv3 predicts four coordinate values for each of the bounding box, which is based on the top left corner of the image offset, as well as the bounding box by wide and high. YOLOv3 predicts the score of an object for each bounding box by logistic regression. If the prediction of the bounding box coincides better with real border than others, then its value is one. Compared with the previous YOLO algorithm, YOLOv3 has made many improvements. Firstly, it carries out multi-scale prediction, and predicts three boxes for each scale. The design method of anchor still uses clustering method, and obtains nine cluster centres, which are divided into three scales according to size. Secondly, YOLOv3 choose better basic classification network and classifier darknet-53. Thirdly, Softmax is replaced by logistic regression for classifier in category prediction.



	Type	Filters	Size	Output
	Convolutional	32	3×3	256×256
	Convolutional	64	3×3/2	128×128
	Convolutional	32	1×1	
1×	Convolutional	64	3×3	
	Residual			128×128
	Convolutional	128	3×3/2	64×64
	Convolutional	64	1×1	
2×	Convolutional	128	3×3	
	Residual			64×64
	Convolutional	256	3×3/2	32×32
	Convolutional	128	1×1	
8×	Convolutional	256	3×3	
	Residua			32×32
	Convolutional	512	3×3/2	16×16
	Convolutional	256	1×1	
8×	Convolutional	512	3×3	
	Residua			16×16
	Convolutional	1024	3×3/2	8×8
	Convolutional	512	1×1	
4×	Convolutional	1024	3×3	
	Residua			8×8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Fig 2.4.4 YOLOv3 detection structure diagram

Data training is the key in deep learning networks. In this paper, for the vehicle target detection, the pictures containing the vehicle targets are selected from the PASCAL VOC2007 and VOC2012 data sets to make a separate VOC-car data set. The YOLOv3 algorithm uses the typical Darknet-53 model to train and fine-tune the parameters of the network. In the experiment, the pre-training model on the ImageNet dataset is used to initialize the Darknet-53 model. With the iterative training, the network reaches convergence.

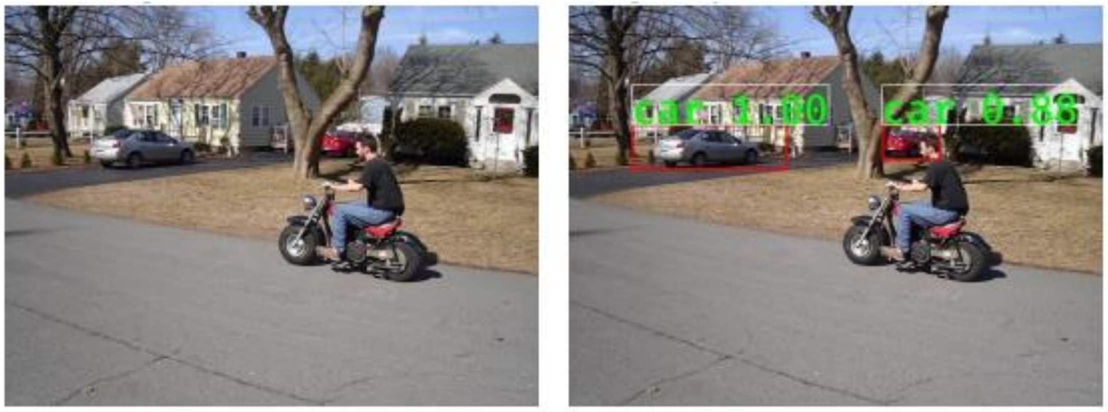


Fig 2.4.5 Vehicle target detection based on YOLOv3 algorithm

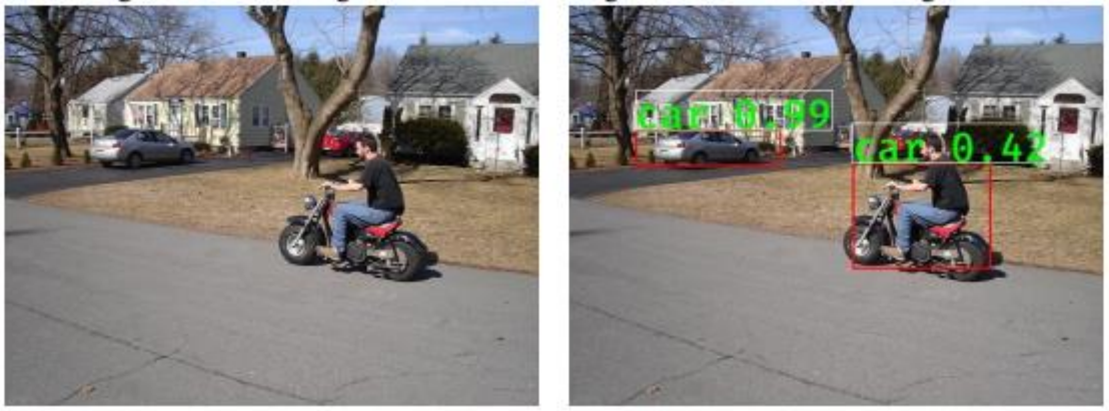


Fig 2.4.6 Vehicle target detection based on YOLOv3-tiny algorithm

It can be seen from the figure that for the complex environment vehicle target detection, the detection algorithm based on YOLOv3 is more accurate than the detection algorithm based on YOLOv3-tiny. The former can better detect the target position, while the latter is prone to false detection in complex environments such as occlusion and poor light.

5) Object Detection, Classification, and Tracking for Autonomous Vehicle

For a car to be a truly autonomous, it must make sense of the environment through which it is driving. The autonomous car must be able to both localize itself in an environment and identify and keep track of objects (moving and stationary). The process of path planning and autonomous vehicle guidance depends on three things: localization, mapping, and tracking objects. Localization is the process of identifying the position of the autonomous vehicle in the environment. Mapping includes being able to make the sense of the environment. Tracking of moving objects involves being able to identify the moving objects and track them during navigation. The processes of localization and mapping have been explored through the use of simultaneous localization and mapping (SLAM). SLAM enables autonomous vehicles to simultaneously build the map of the unknown environment and localize itself in the unknown environment. The development of SLAM has been significant in the development of autonomous robots. Most of the research in SLAM assumes the environment to be static and considers the movement of objects as noise. The detection and tracking of moving objects (DATMO) are one of the most challenging issues and is important for safety and essential for avoiding collisions. SLAM combined with DATMO helps solve this problem.

The algorithm developed on this paper and the results of the algorithm are all based on the datasets provided by Oxford University Robotcar. The Robotcar, an autonomous capable Nissan LEAF, traversed a route through central Oxford, UK twice a week on average over the period of May 2014 to December 2015 to collect data of different driving environments.

A single image will have multiple objects that need to be identified. Detecting multiple objects in an image is a challenging task; however, using current methods, it is achievable with accuracy in real time. Object detection is one of the most researched topics in the area of Computer vision. The object detection usually starts with extracting features from the input image using different algorithm such as Haar or HOG.These features are fed to the classifier to identify the object. With advancement in deep learning, there are many convolutional neural network (CNN) architectures that have outperformed the object detection method using a feature extractor. These CNNs attempt to imitate the working of human neurons. The CNN learns the feature in object during training process and can generalize the feature for the object and detect the object.

The R-CNN , Faster R-CNN , YOLO algorithms are all based on CNN and perform very well at multiple object detection. This work uses YOLO algorithm for the detection and classification of objects. The advantage of using YOLO is that it is orders of magnitude faster (45 frames per seconds) than other algorithms while maintaining a high accuracy. Other algorithms detect objects using region-proposal techniques or sliding-windows method and iterate over the image for many times. YOLO sees the entire image and only once hence the name. This makes YOLO faster compared to iterative methods. The entire image is fed through the CNN and detects multiple objects simultaneously. YOLO works by taking an image and splitting it into an SxS grid. For each grid cell, it predicts bounding boxes and confidence for those boxes and class probability. Objects are located within the image where the bounding boxes have a class probability above a threshold value. For every object detected YOLO classifies the object, gives the confidence and the bounding box for the object.

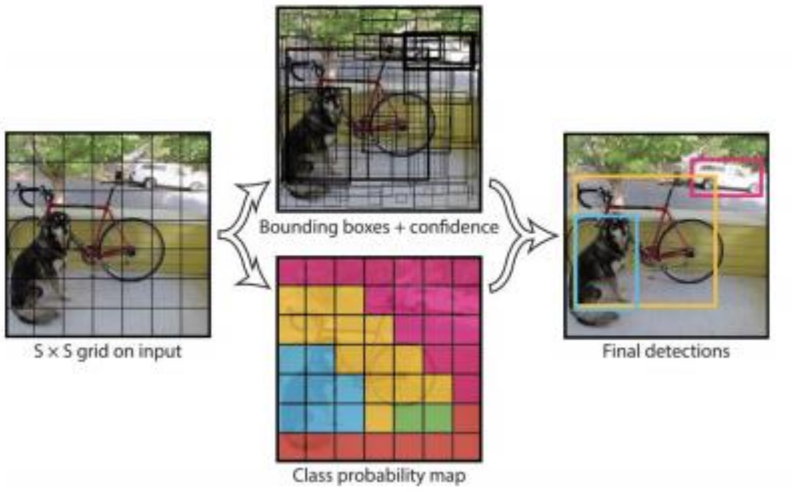


Fig 2.4.7 Illustration of YOLO algorithm

You only look once is an algorithm based on CNN used to detect, classify, and localize objects in an image. When objects are to be detected in an image, first it is resized into the size of the input layer of the YOLO architecture, then the image is run through the CNN and the output is the bounding box for each detected object with classification and probability score.

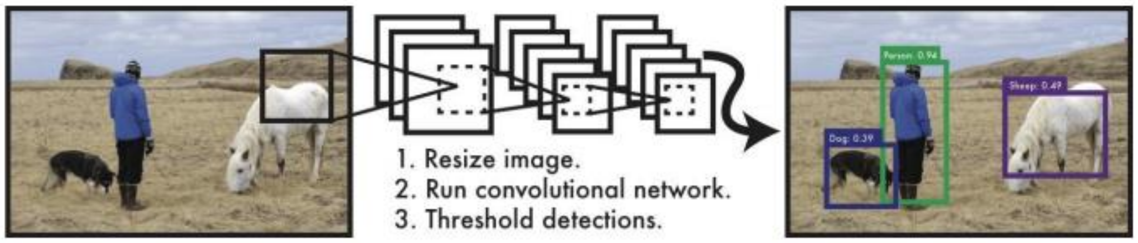
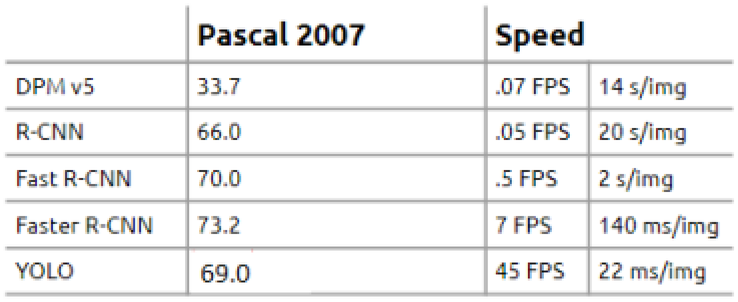


Fig 2.4.8 YOLO detecting, classifying and localizing objects in an image

YOLO can see an entire image at once and simultaneously predicts the multiple bounding boxes and class probabilities for each box, making it faster compared to other region-based method or sliding window methods such as DPM, R-CNN, Fast R-CNN, and Faster R-CNN. The figure below shows the performance of each of these method speed and accuracy on Nvidia Titan X GPU and the same test dataset Pascal 2007. The Faster R-CNN is accurate by 4 percent compared to YOLO, but YOLO's Figure 19: YOLO detecting, classifying and localizing objects in an image 35 speed is far more superior. This makes YOLO suitable for using in real-time systems such as autonomous vehicles.



	Pascal 2007	Speed	
DPM v5	33.7	.07 FPS	14 s/img
R-CNN	66.0	.05 FPS	20 s/img
Fast R-CNN	70.0	.5 FPS	2 s/img
Faster R-CNN	73.2	7 FPS	140 ms/img
YOLO	69.0	45 FPS	22 ms/img

Fig 2.4.9 Performance of different object detection algorithms

Each object detected has a bounding box, which localizes the object in the image. This information is helpful in finding the distance of the objects from the RobotCar. Once the objects are detected and classified and located in the image, the projected laser scans to the image can be isolated so that only laser bouncing back from the objects detected remain.

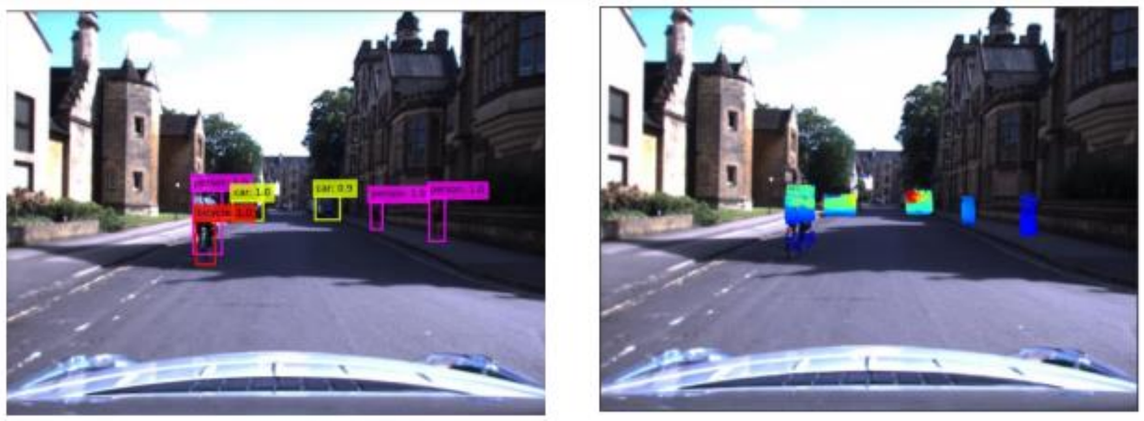


Fig 2.4.10 Object Detection by YOLO (left) and Laser Projection on detected objects (right)

6) Self-Driving Cars: Evaluation of Deep Learning Techniques for Object Detection in Different Driving Conditions

Computer Vision has become increasingly important and effective in recent years due to its wide-ranging applications in areas as diverse as smart surveillance and monitoring, health and medicine, sports and recreation, robotics, drones, and self-driving cars. Visual recognition tasks, such as image classification, localization, and detection, are the core building blocks of many of these applications and recent developments in Convolutional Neural Networks (CNNs) have led to outstanding performance in these state-of-the-art visual recognition tasks and systems. CNNs are primarily used for image classification tasks, such as, looking at an image and classifying the object in it. In contrast, the term object localization refers to the task of identifying the location of an object in the image.

An object localization algorithm will output the coordinates of the location of an object with respect to the image. In computer vision, the most popular way to localize an object in an image is to represent its location with the help of bounding boxes. The task of object detection combines both classification and localization to identify multiple objects in an image and locate each object within the image. A well-known application of multiple object detection is in self-driving cars, where the algorithm not only needs to detect the cars but also pedestrians, motorcycles, trees and other objects in the frame and then draw bounding boxes around each of the detected objects.

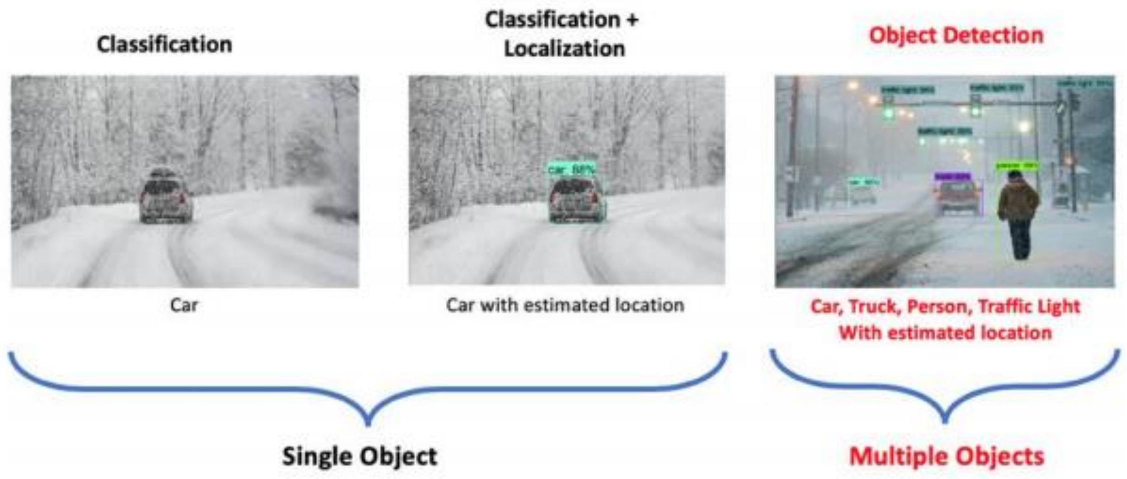
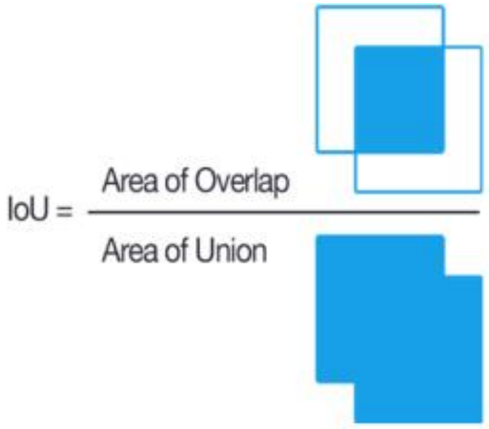

Classification
Classification + Localization
Object Detection
Car
Car with estimated location
Car, Truck, Person, Traffic Light With estimated location
Single Object
Multiple Objects

Fig 2.4.11 Different roles of computer vision in self driving cars

A typical object detection pipeline can be mainly divided into three stages: informative region selection, feature extraction, and classification. This choice gives rise to many combinations of feature extractors and region selection techniques. With these differences, it can be difficult for practitioners to decide what architecture is best suited to their applications. Two key metrics: mean average precision (mAP) and inference time are used to evaluate these algorithms. The mAP provides a single number summary of how well the object detector performed in detecting and classifying the objects. Inference time measures the speed of detection. A key challenge with object detection is that labelled data is scarce and it takes substantial effort to gather and annotate each image to be used in the training phase. Moreover, training an object detector from scratch takes a significant amount of time and compute resources.

Most object detection models use Convolutional Neural Network (CNN) methods to train images and detect objects by identifying a region of interest within that image and determine the class of the object present within that region. Testing the pretrained object detection models requires test images that contain reasonable clarity and visibility of the classes being identified – which the probability of detection depends. To measure the performance of the object detection algorithms, a reference to the location of objects in the images is needed. COCO annotation format was followed to label target object classes using bounding boxes. These reference bounding boxes indicating the true location of objects in the images are referred to as Ground Truth. Comparing this Ground Truth bounding boxes to the predicted bounding boxes that are generated by the algorithms is what gives us the accuracy of the Object detection algorithm summarized by the mAP score.

To evaluate the model on the task of object localization, we must first determine how well the model predicted the location of the object. Intersection over Union is an evaluation metric used to measure the accuracy of a localization task. IoU is a ratio of the area of overlap between the predicted bounding box and the ground-truth bounding box divided by the area of the union, or more simply, the area encompassed by both the predicted bounding box and the ground-truth bounding box.



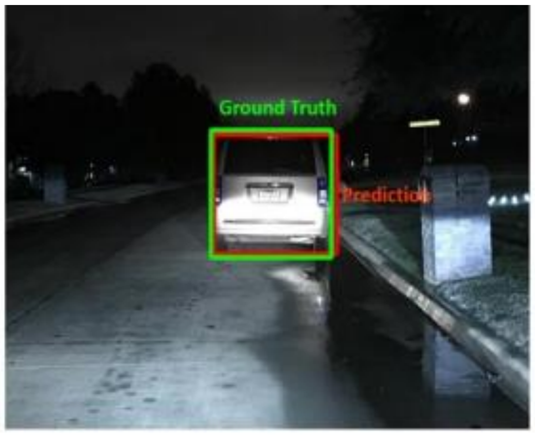


Fig 2.4.12 Computing the intersection over union and sample image showing predicted vs ground-truth bounding boxes overlaid

CHAPTER 3
Analysis and Design

3.1 Traffic Sign Classifier

3.1.1 Dataset Used:

In this module, we developed a deep learning model that will train on German traffic sign images and then classify the unlabeled traffic signs. The German Traffic Sign Benchmark is a multi-class, single-image classification challenge held at the International Joint Conference on Neural Networks (IJCNN) 2011. The benchmark has the following properties:

- Single-image, multi-class classification problem
- More than 40 classes (43)
- More than 50,000 images in total
- Large, lifelike database

3.1.2 Data Pre-Processing:

The acquired data are usually messy and come from different sources. To feed them to the neural network, they need to be standardized and cleaned up. More often than not, preprocessing is used to conduct steps that reduce the complexity and increase the accuracy of the applied algorithm. We can't write a unique algorithm for each of the condition in which an image is taken, thus, when we acquire an image, we tend to convert it into a form that allows a general algorithm to solve it.

1)Gray Scaling:

Gray scaling of images is done to reduce the information provided to the pixels and also reduces complexity.

2)Bilateral Filtering:

Bilateral filtering is a noise reducing , edge preserving smoothening of images.

3) Image Translation:

This is a technique by which we shift the location of the image. In layman terms, if the image's location is (x1,y1) position, after translation it is moved to (x2,y2) position.

4)Rotation:

We used 10 degrees rotation of images. It would not make much sense to rotate images more than that as that might lead to wrong representations of the traffic signs.

3.1.3 CNN Architecture:

The next step in the model building process would be to use a much sophisticated architecture to boost our model performance. Research in the field of computer vision has established that Convolutional neural networks performs exceedingly well at image recognition challenges and hence was our first choice.

```
#Building the model
model = Sequential()
model.add(Conv2D(filters=32, kernel_size=(5,5), activation='selu', input_shape=X_train.shape[1:]))
model.add(Conv2D(filters=32, kernel_size=(5,5), activation='selu'))
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(rate=0.25))
model.add(Conv2D(filters=64, kernel_size=(3, 3), activation='selu'))
model.add(Conv2D(filters=64, kernel_size=(3, 3), activation='selu'))
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(rate=0.25))
model.add(Flatten())
model.add(Dense(256, activation='selu'))
model.add(Dropout(rate=0.5))
model.add(Dense(43, activation='softmax'))
```

The proposed CNN model includes an input layer, four convolutional layers, three dropout layers, two Maxpool Layers and two fully connected layers before the output layer.The input layer is composed of a series of input images, which are resized to the size of 76*76 pixels. Each convolutional layer is composed of convolutional kernels bank and a non-linear transform function. The choice of non-linear transform is the sigmoid/relu/selu/tanh function(s). The shape of the training data is (39209 X 30 X 30 X 3) , which implies that there are 39209 images in the training set each of size 30 X 30 having 3 channels (RGB).

Layers of CNN:

1) Convolutional layer:

Convolution is the first layer to extract features from an input image. Convolution preserves the relationship between pixels by learning image features using small squares of input data. It is a mathematical operation that takes two inputs such as image matrix and a filter or kernel.

Consider a 5 x 5 whose image pixel values are 0, 1 and filter matrix 3 x 3 :

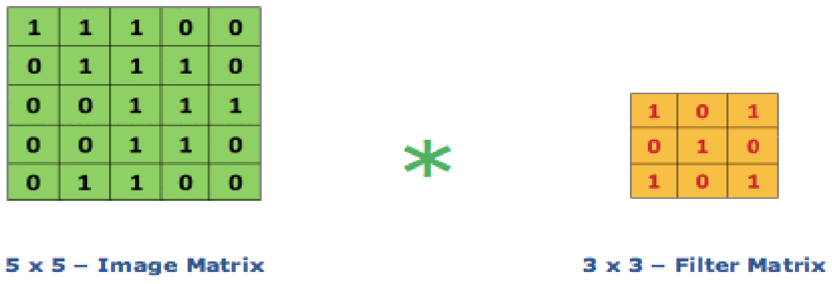


Fig 3.1 5*5 Image Matrix and 3*3 Filter Matrix

Then the convolution of 5 x 5 image matrix multiplies with 3 x 3 filter matrix which is called "Feature Map" as output will be

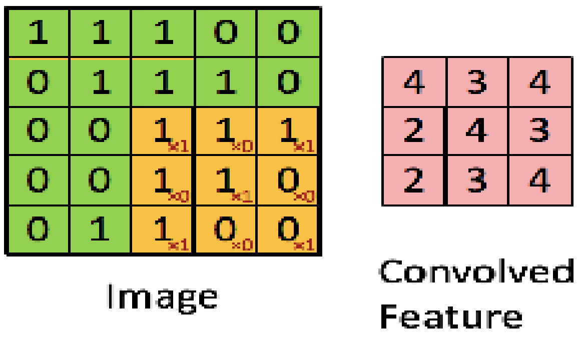


Fig 3.1.3.2 Feature Matrix Obtained

2) Pooling Layer:
Pooling layers section would reduce the number of parameters when the images are too large. Spatial pooling also called subsampling or downsampling which reduces the dimensionality of each map but retains important information. Spatial pooling can be of different types:

- Max Pooling
- Average Pooling
- Sum Pooling

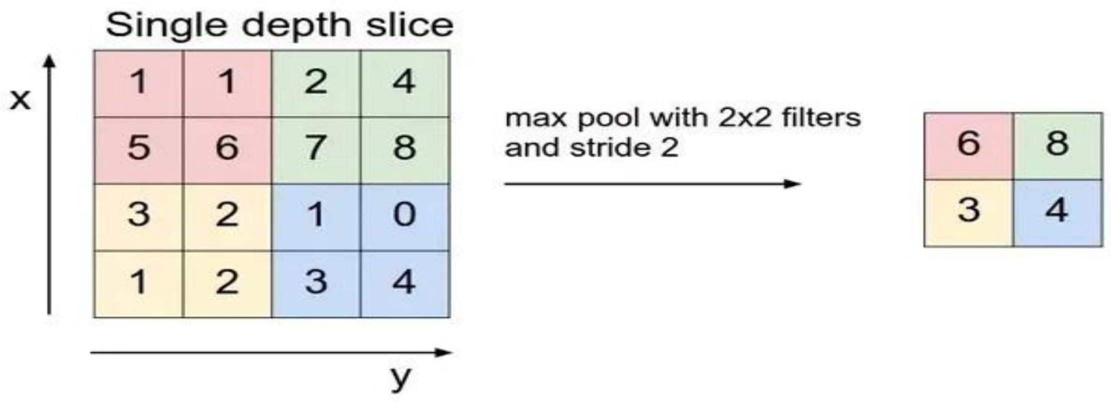


Fig 3.1.3.3 Max Pooling using a 2*2 Filter

3) Fully Connected Layer:
The layer we call as FC layer, we flattened our matrix into vector and feed it into a fully connected layer like a neural network.

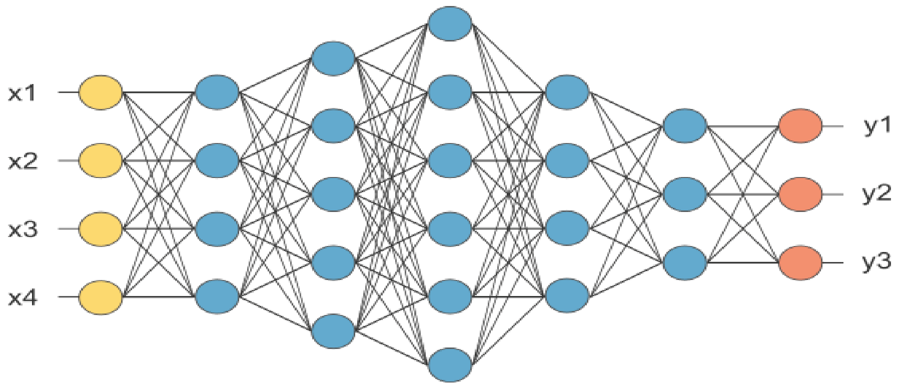


Fig 3.1.3.4 Fully Connected Layers

In the above diagram, the feature map matrix will be converted as vector (x1, x2, x3, …). With the fully connected layers, we combined these features together to create a model. Finally, we have an activation function such as softmax or sigmoid to classify the outputs as cat, dog, car, truck etc. (In our case, traffic signs)

The complete overview of the whole architecture looks something like this:

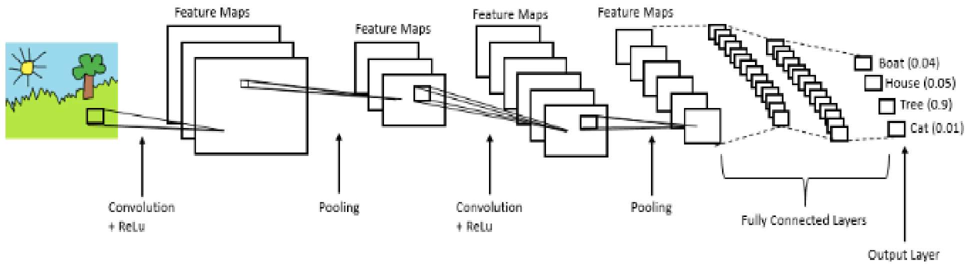


Fig 3.1.3.5 General Architecture Of a CNN

3.2) Lane Detection

We've researched multiple papers and have finally come up with our implemented model.

1) Converting to HSL

The HSL color space has intensity decoupled from color information. Hue, Saturation, and Lightness. Makes it easier to compare objects with similar hue but varying lighting conditions. Note that the hue is a continuous angular scale of Hue where 0 and 360 degrees meet at red, so that red (360 degrees) has maximum hue while orange has minimum.

In general all image processing methods used for grey level images can be used for color images. They can be carried out on each of the color channels or for example on the intensity only.

2) Isolate yellow and white from HSL image

For the yellow mask,

Low threshold = 15, 38, 115

High threshold = 35, 204, 255

For the white mask,

Low threshold = 0, 200, 0

High threshold = 180, 255, 255

3) Combine isolated HSL with original image

Combine those two masks using an OR operation and then combine with the original image using an AND operation to only retain the intersecting elements.

4) Convert image to grayscale for easier manipulation

5) Apply Gaussian Blur to smoothen edges

The Gaussian distribution in 1-D has the form:

$$G(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{2\sigma^2}}$$

where σ is the standard deviation of the distribution.

6) Apply Canny Edge Detection on smoothed gray image

   a. Intensity Gradient

      Image smoothening with Sobel kernel

      Find edge gradient and direction for each pixel

   b. Non Maximum Suppression

      Pixels are checked if they're the local maxima. A is on edge, hence gradient direction is normal to the edge. Point B and C are in gradient directions. So point A is checked with point B and C to see if it forms a local maximum. If so, it is considered for next stage, otherwise, it is suppressed (put to zero).

7) Trace Region Of Interest and discard all other lines identified by our previous step that are outside this region

8) Perform a Hough Transform to find lanes within our region of interest and trace them in red

9) Separate left and right lanes

10) Interpolate line gradients to create two smooth lines

3.3 Vehicle Detection:

In this module, we have picked 300 random images from the KITTI dataset. The number of classes decided for the model was one: Vehicles.

The KITTI dataset has been recorded from a moving platform while driving in and around Karlsruhe, Germany. It includes camera images, laser scans, high-precision GPS measurements and IMU accelerations from a combined GPS/IMU system. The main purpose of this dataset is to push forward the development of computer vision and robotic algorithms targeted to autonomous driving.

Data Pre-processing:

The acquired data has been self-annotated according to the class mentioned above. The annotation tool used for this purpose is LabelImg.

LabelImg is a graphical image annotation tool. It is written in Python and uses Qt for its graphical interface. Annotations are saved as XML files in PASCAL VOC format, the format used by ImageNet. Besides, it also supports YOLO format.

Annotation format for YOLO:

```
<object-class-id> <x-centre> <y-centre> <width> <height>
height, width - Actual height and width of the image
 x, y - centre coordinates of the bounding box
 h, w - height and width of the bounding box

<x-centre> : x / width
 <y-centre> : y / height
 <width> : w / width
 <height> : h / height
```

Flow of module:

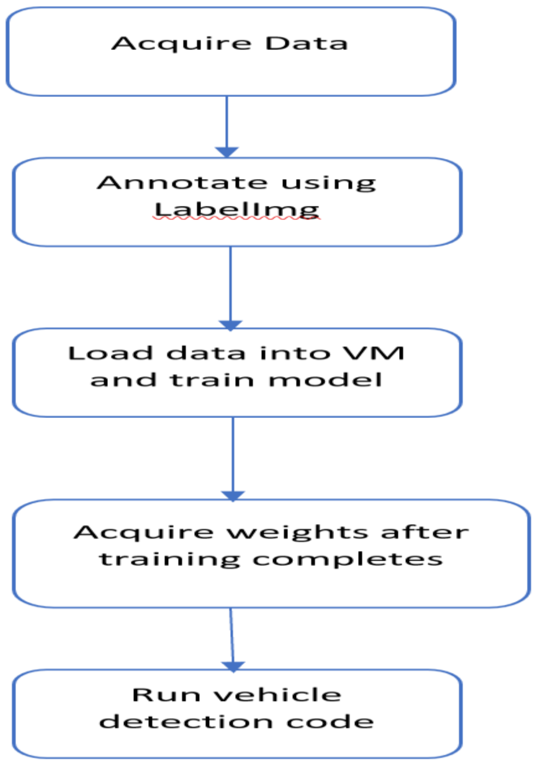

Acquire Data
Annotate using LabelImg
Load data into VM and train model
Acquire weights after training completes
Run vehicle detection code

Fig 3.3.1 Flow of the vehicle detection module

3.4 Drowsiness Detector:

What is Dlib?

It‘s a landmark’s facial detector with pre-trained models, the dlib is used to estimate the location of 68 coordinates (x, y) that map the facial points on a person’s face. These points are identified from the pre-trained model where the iBUG300-W dataset (shown below) was used.

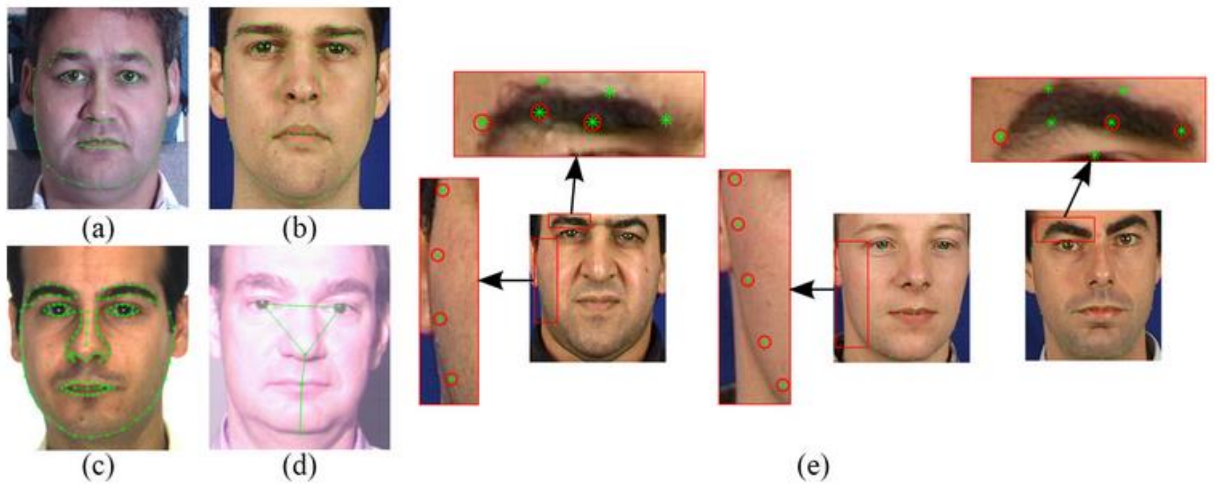


Figure 3.4.1 iBUG300-W dataset examples

Flow of program:

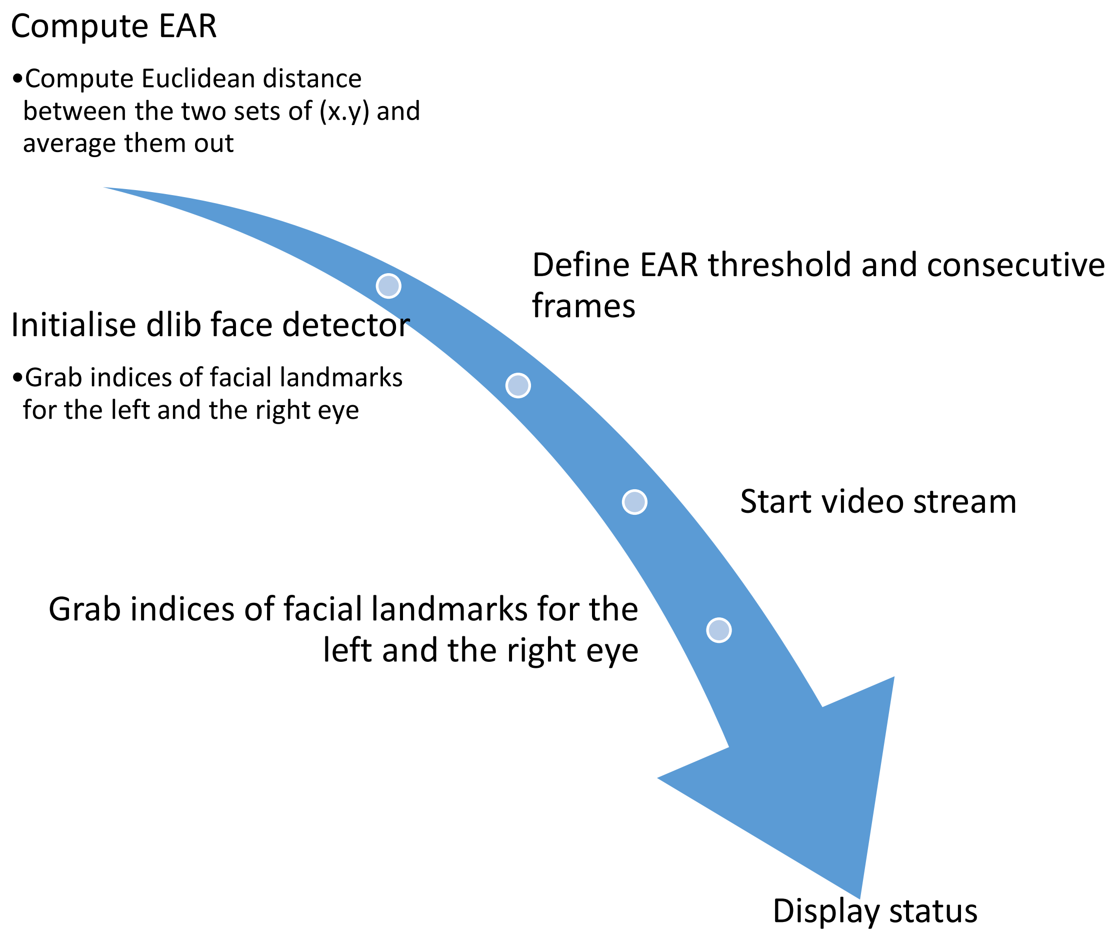

Compute EAR
•Compute Euclidean distance between the two sets of (x.y) and average them out
Define EAR threshold and consecutive frames
Initialise dlib face detector
•Grab indices of facial landmarks for the left and the right eye
Start video stream
Grab indices of facial landmarks for the left and the right eye
Display status

Figure 3.4.2 Flowchart of Drowsiness Detector

CHAPTER 4

Implementation

4.1 Traffic Sign Classifier

After building the model architecture, we then train the model using model.fit(). We experimented with different batch sizes (32 and 64.) Our model performed better with 64 batch size. Roughly after 15 epochs the accuracy was stable.

```
#Compilation of the model
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

epochs = 15
```

The optimizer controls the learning rate. We will be using 'adam' as our optmizer. Adam is generally a good optimizer to use for many cases. The adam optimizer adjusts the learning rate throughout training.

The learning rate determines how fast the optimal weights for the model are calculated. A smaller learning rate may lead to more accurate weights (up to a certain point), but the time it takes to compute the weights will be longer.

For our loss function, we will use 'categorical_crossentropy'. Cross entropy is another way to measure how well your Softmax output is. That is how similar is your Softmax output vector is compared to the true vector [1,0,0], [0,1,0],[0,0,1] for example if there are only three classes. Categorical crossentropy will compare the distribution of the predictions (the activations in the output layer, one for each class) with the true distribution, where the probability of the true class is set to 1 and 0 for the other classes. To put it in a different way, the true class is represented as a one-hot encoded vector, and the closer the model's outputs are to that vector, the lower the loss.

Now we will train our model. To train, we will use the 'fit()' function on our model with the following five parameters: training data (train_X), target data (train_y), validation split, the number of epochs and callbacks.

```
history = model.fit(X_train, y_train, batch_size=64, epochs=epochs, validation_data=(X_test, y_test))
```

The validation split will randomly split the data into use for training and testing. During training, we will be able to see the validation loss, which give the mean squared error of our model on the validation set. We will set the validation split at 0.2, which means that 20% of the training data we provide in the model will be set aside for testing model performance.

The number of epochs is the number of times the model will cycle through the data. The more epochs we run, the more the model will improve, up to a certain point. After that point, the model will stop improving during each epoch. In addition, the more epochs, the longer the model will take to run. To monitor this, we will use ‘early stopping’.

Early stopping will stop the model from training before the number of epochs is reached if the model stops improving. We will set our early stopping monitor to 8. This means that after 8 epochs in a row in which the model doesn’t improve, training will stop. Sometimes, the validation loss can stop improving then improve in the next epoch, but after 8 epochs in which the validation loss doesn’t improve, it usually won’t improve again.

We then test our model and plot two graphs namely, Accuracy Vs Epoch and Loss Vs Epoch which will be discussed in the next chapter.

```
#plotting                     graphs                   for                 accuracy
plt.figure(0)
plt.plot(history.history['accuracy'],           label='training                accuracy')
plt.plot(history.history['val_accuracy'],          label='val                  accuracy')
plt.title('Accuracy')
plt.xlabel('epochs')
plt.ylabel('accuracy')
plt.legend()
plt.show()

plt.figure(1)
plt.plot(history.history['loss'],               label='training                    loss')
plt.plot(history.history['val_loss'],              label='val                      loss')
plt.title('Loss')
plt.xlabel('epochs')
plt.ylabel('loss')
plt.legend()
plt.show()
```

We then create a graphical user interface for our traffic signs classifier with Tkinter. Tkinter is a GUI toolkit in the standard python library. With this GUI, any user of the application can introduce an image of a traffic sign that has never been seen by the model, and the model will try to predict the image.

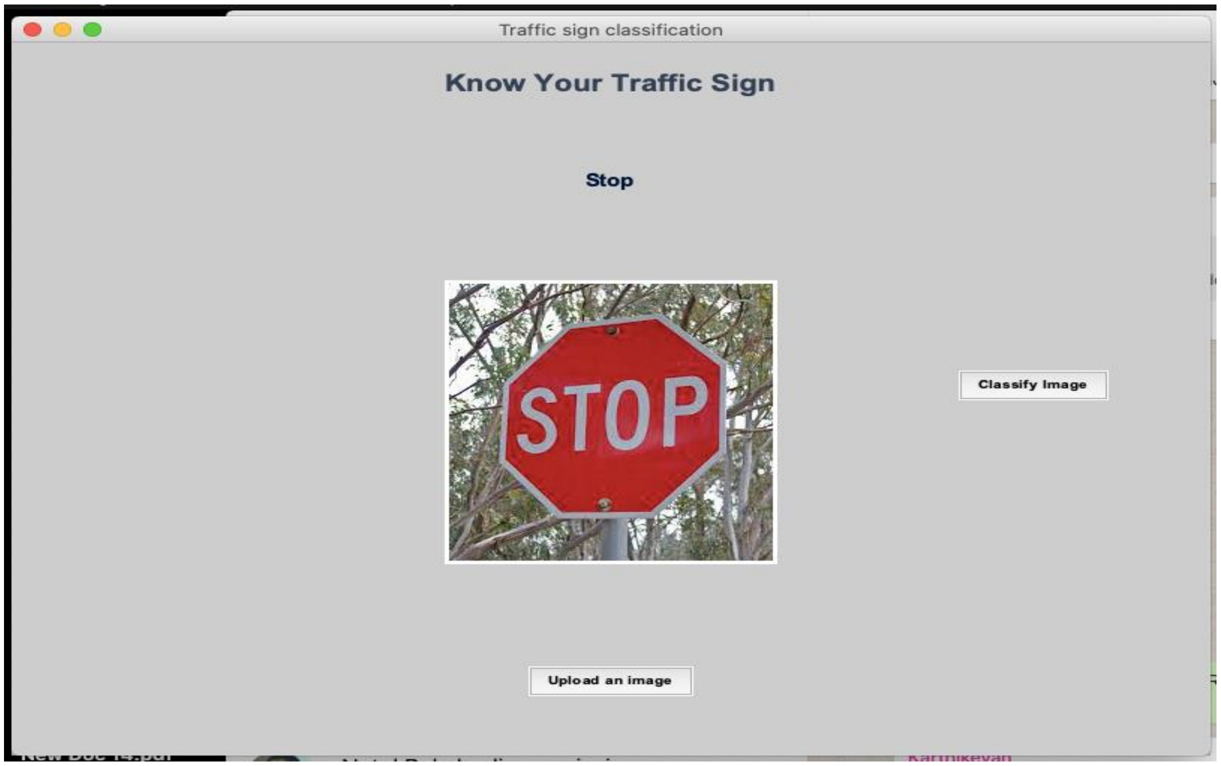


Fig 4.1.1 Output Of the Classifier

4.2) Lane Detection

The lane detection module was implemented on Jupyter Notebook as it offers a convenient way of looking at the results at each step. The python libraries used are OpenCv2, Matplotlib and numpy. Inbuilt functions offered by OpenCv have been used throughout, with tweaks in the parameters as per the literature survey.

Original to color:

```
cv2.cvtColor(img, cv2.COLOR_RGB2HLS)
```

Isolate Yellow and White:

```
low_threshold = np.array([15, 38, 115])
high_threshold = np.array([35, 204, 255])
yellow_mask = cv2.inRange(img, low_threshold, high_threshold)
low_threshold = np.array([0, 200, 0])
high_threshold = np.array([180, 255, 255])
white_mask = cv2.inRange(img, low_threshold, high_threshold)
```

Combining the frames:

```
hsl_mask = cv2.bitwise_or(hsl_yellow, hsl_white)
output_mask = cv2.bitwise_and(img, img, mask=hsl_mask)
```

Gray Scale and Gaussian Blur:

```
cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
cv2.GaussianBlur(grayscale_img, (kernel_size, kernel_size), 0
```

Canny Edge Detector:

```
def canny_edge_detector(blurred_img, low_threshold, high_threshold):
   return cv2.Canny(blurred_img, low_threshold, high_threshold)
canny_images1 = list(map(lambda img: canny_edge_detector(img, 50, 150),
blurred_images2))
```

Hough Transform:

```
def hough_transform(canny_img, rho, theta, threshold, min_line_len, max_line_gap):
   return cv2.HoughLinesP(canny_img, rho, theta, threshold, np.array([]),
minLineLength=min_line_len, maxLineGap=max_line_gap)
```

4.3 Vehicle Detection:

Hardware used:

The entire training process has been carries out via Google Colaboratory notebook, giving a 12GB-DDR5 RAM support and 35GB storage in virtual machine. Along with this, CUDA 10.0 was used to meet the GPU requirements. NVIDIA cudNN 7.5 was installed which is a GPU accelerated library of primitives built for deep neural networks. The processors used were Intel Xenon CPU: 2.20 GHz each (2 cores).

Training process:

Darknet-53 CNN was used since YOLOv3 is built upon the layers of Darknet-53. Execution permissions for the Darknet-53 were set in the VM.

```
!chmod +x ./darknet
```

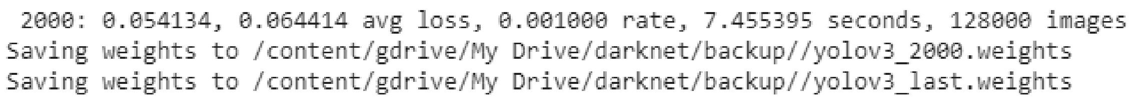
The cfg file of YOLOv3 was configured according to the model being built. The number of classes in the cfg file from 80 to 1. Along with this, the number of filters were changed as well according to the following formula:

```
filters = (classes + 5) * 3
```

Finally, the training process was started

```
!./darknet detector train "/content/gdrive/My Drive/darknet/obj.data" "/content/gdrive/My Drive/darknet/cfg/yolov3.cfg" "/content/darknet/weights/darknet53.conv.74" -dont_show
```

The model was trained for 2000 iterations since the average loss dropped below 0.07 beyond that.



```
 2000: 0.054134, 0.064414 avg loss, 0.001000 rate, 7.455395 seconds, 128000 images
Saving weights to /content/gdrive/My Drive/darknet/backup//yolov3_2000.weights
Saving weights to /content/gdrive/My Drive/darknet/backup//yolov3_last.weights
```

Pre-trained model:

A Pre-trained model was also used built on the COCO dataset as the YOLO9000 uses data with a lot of data pre-processing and the results observed are faster than any other algorithm (45 FPS). Hence, the best results could be provided by this model and thus served the purpose of a benchmark model against our custom-built model.

4.4. Drowsiness Detector:

To build our blink detector, we'll be calculating an eye aspect ratio (EAR), introduced in *Real-Time Eye Blink Detection Using Facial Landmarks*.[1] Each eye is represented by 6 *(x, y)*-coordinates, starting at the left-corner of the eye, and then working clockwise around the rest of the region: It checks 20 consecutive frames and if the Eye Aspect ratio is less than 0.25, Alert is generated.

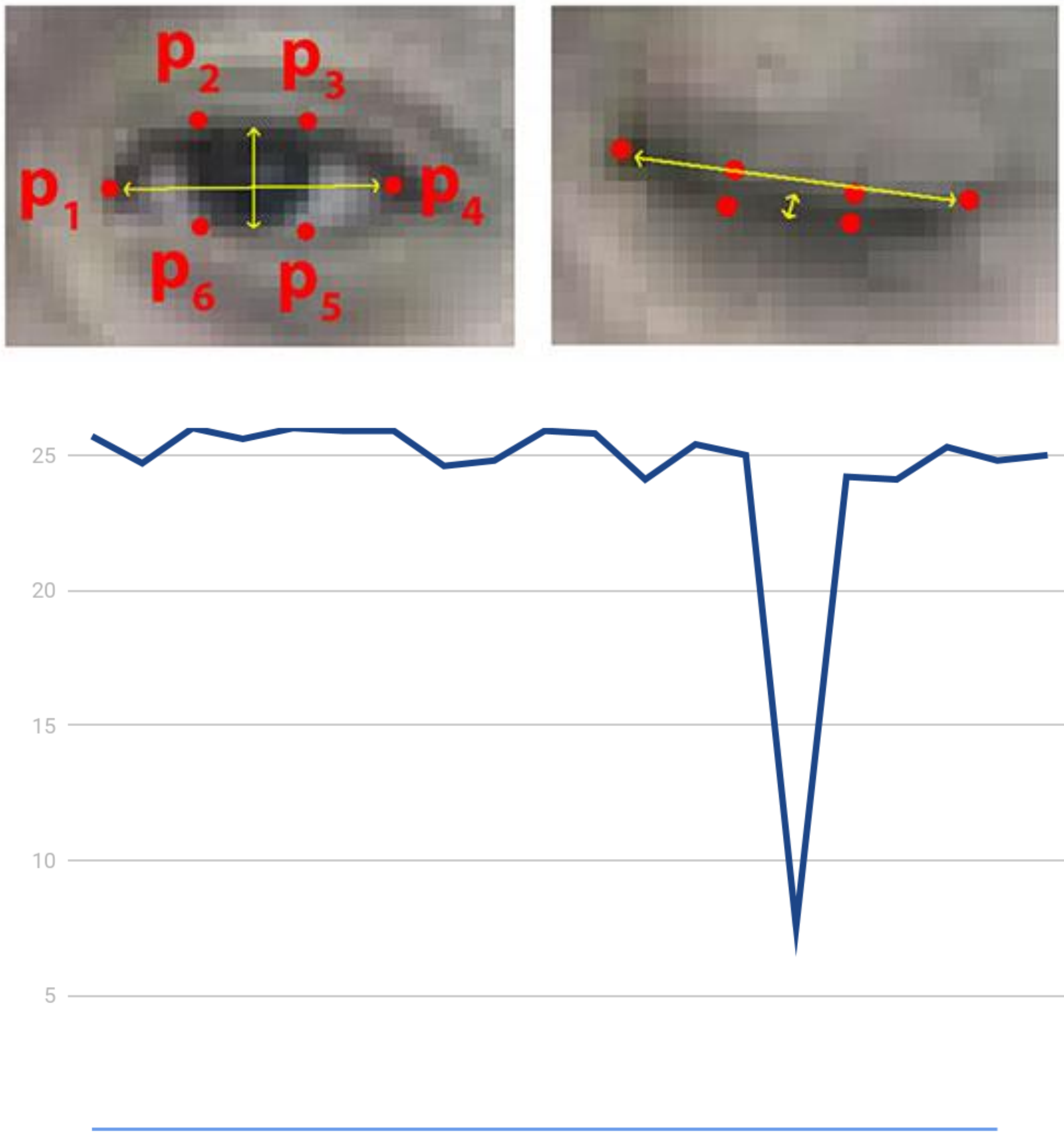


Figure 4.4.1 Formula depiction and graph

The eye aspect ratio is nearly constant while the eye is open, but will swiftly fall to zero when a blink is taking place. The *bottom* figure plots a graph of the eye aspect ratio over time for a video clip.

We use the SciPy package to compute the Euclidean distance between facial landmark points in the EAR computation. The following dataset (and for other detection and localization of facial landmarks) we use the dlib library. Quite simply, this is how we compute the EAR:

```
def eye_aspect_ratio(eye):
     A = dist.euclidean(eye[1], eye[5])
     B = dist.euclidean(eye[2], eye[4])
     C = dist.euclidean(eye[0], eye[3])
     ear = (A + B) / (2.0 * C)
     return ear
```

The return value of this function will remain constant when the eye is open and drop to almost zero during a blink.

We then define an EAR Threshold, the value of which decides when we start counting the number of frames for which the person has his/her eyes closed. This value of frames is also set by us. By experimentation, we found that the value for the threshold works well for just about everyone at 0.25 and the number of consecutive frames at 46. The higher this value, the less sensitive the detector is.

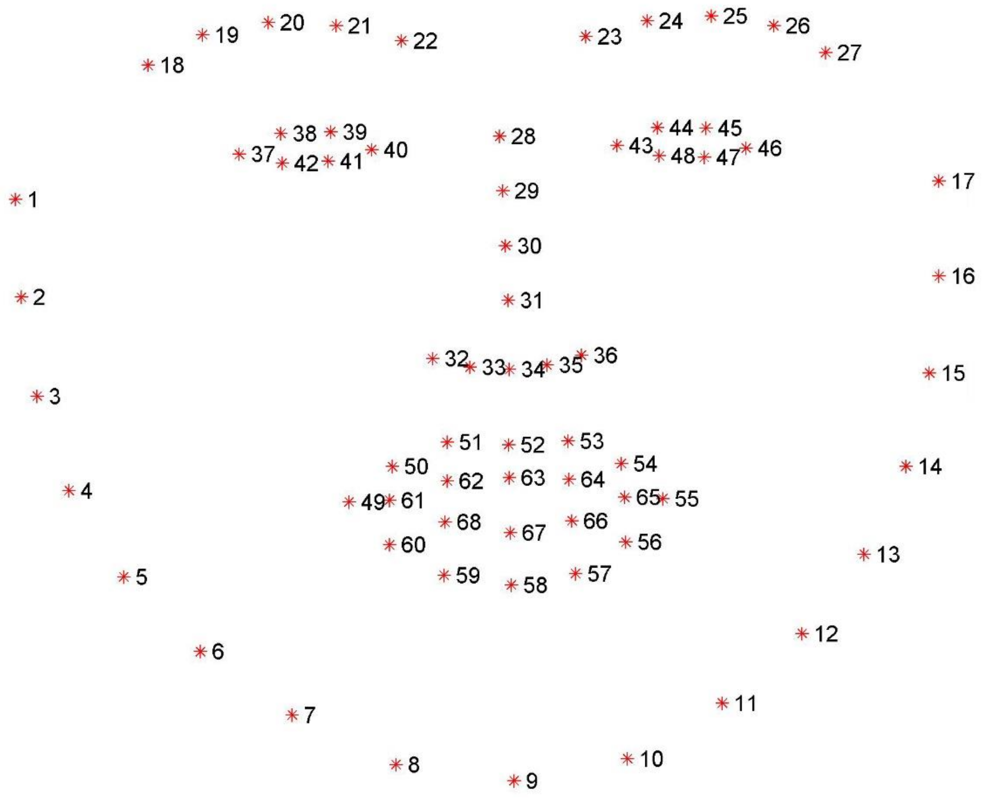


Figure 4.4.2 68 landmarks dataset

We can determine the starting and ending array slice index values for extracting *(x, y)*-coordinates for both the left and right eye below:

```
(lStart, lEnd) = face_utils.FACIAL_LANDMARKS_IDXS["left_eye"]
(rStart, rEnd) = face_utils.FACIAL_LANDMARKS_IDXS["right_eye"]
```

This is how the testing is visualized:

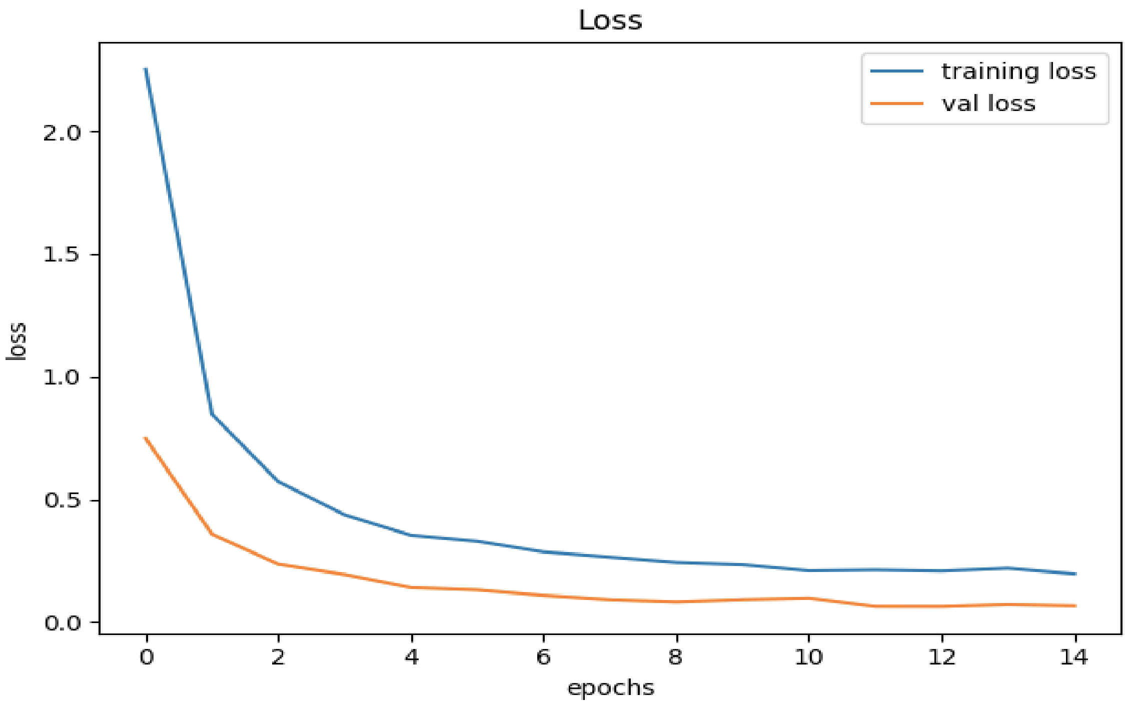
```
leftEyeHull = cv2.convexHull(leftEye)
rightEyeHull = cv2.convexHull(rightEye)
cv2.drawContours(frame, [leftEyeHull], -1, (0, 255, 0), 1)
cv2.drawContours(frame, [rightEyeHull], -1, (0, 255, 0), 1)
```

Chapter 5

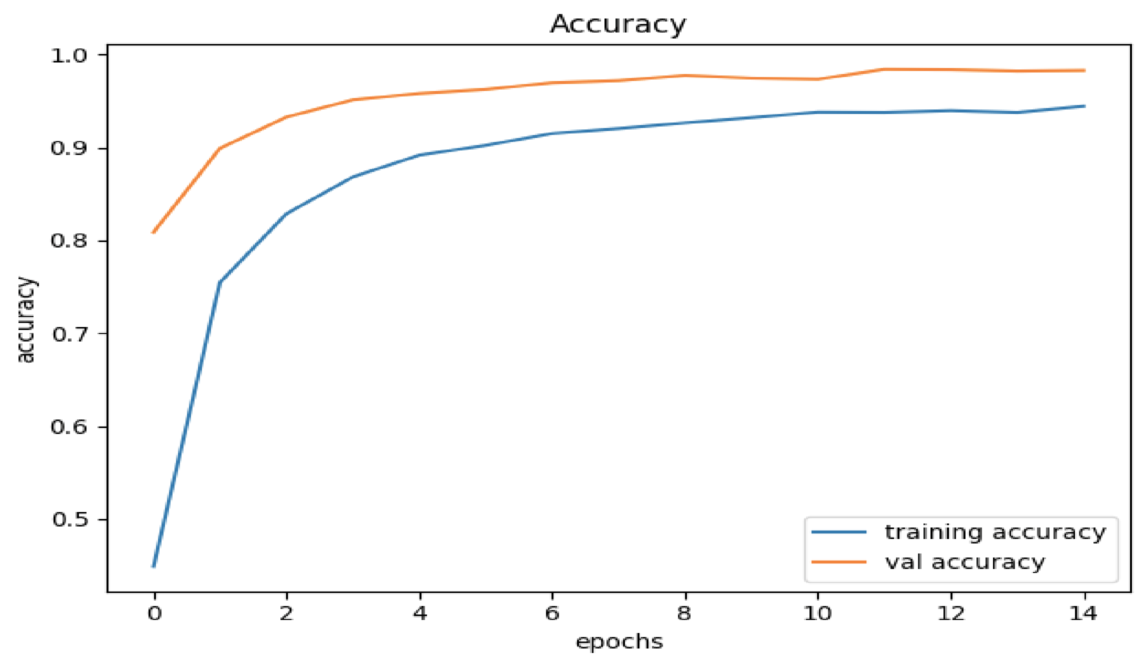
RESULTS

5.1 Traffic Sign Classifier:

5.1.1 RELU:



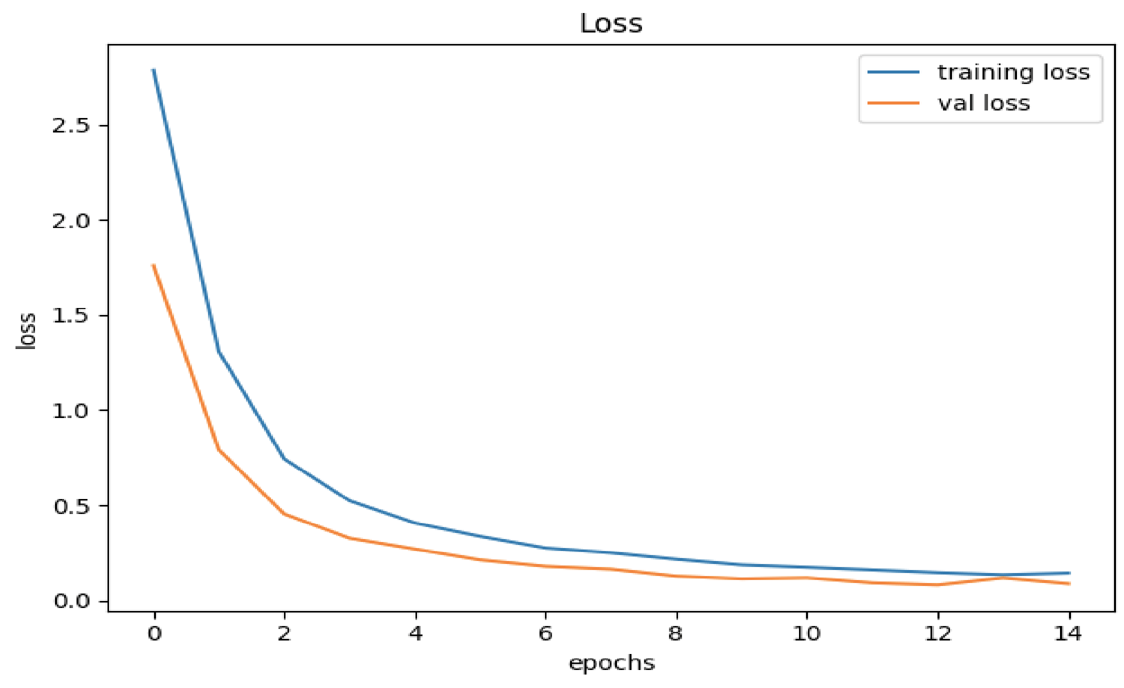
(a)



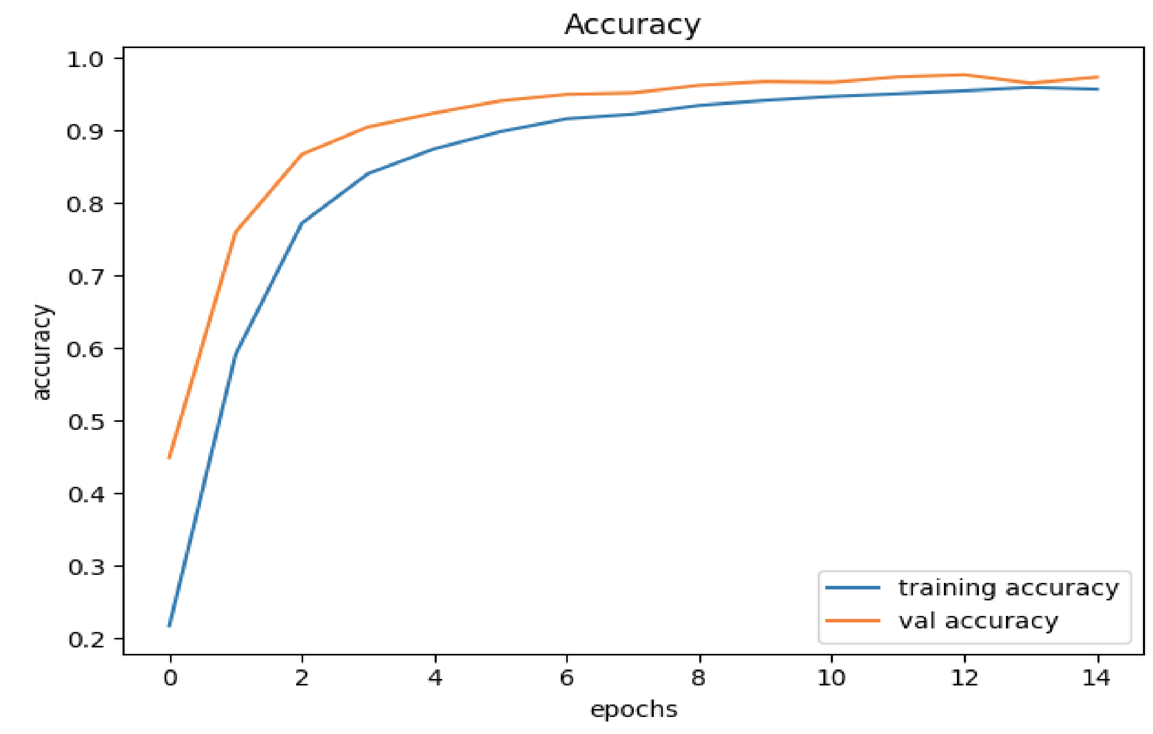
(b)

Fig 5.1.1 (a) Loss vs Epoch and (b) Accuracy vs Epoch

5.1.2 Sigmoid



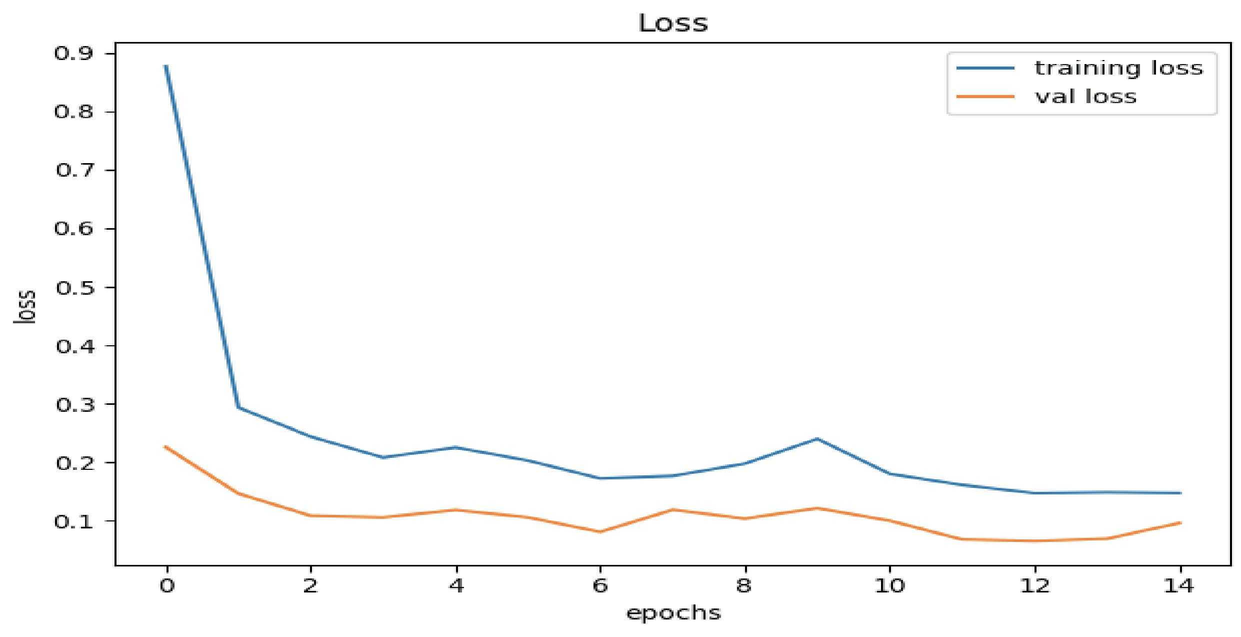
(a)



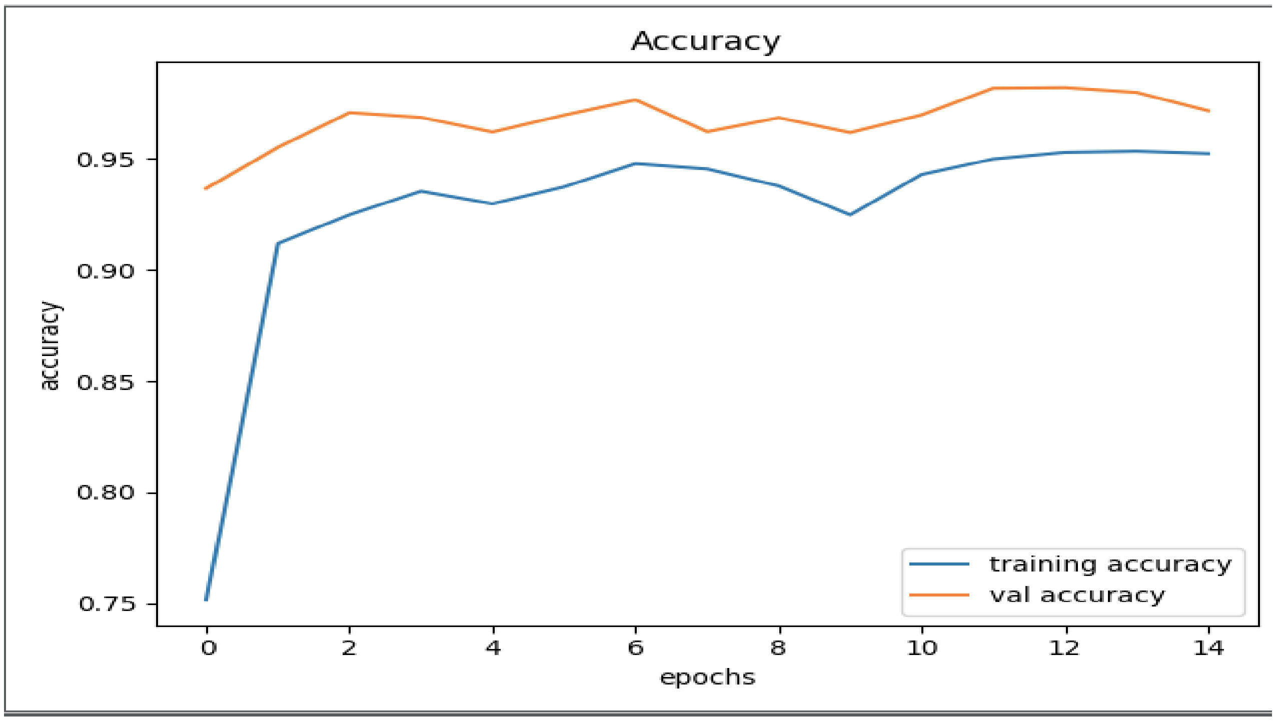
(b)

Fig 5.1.2 (a) Loss vs Epoch and (b) Accuracy vs Epoch

5.1.3 Tanh:



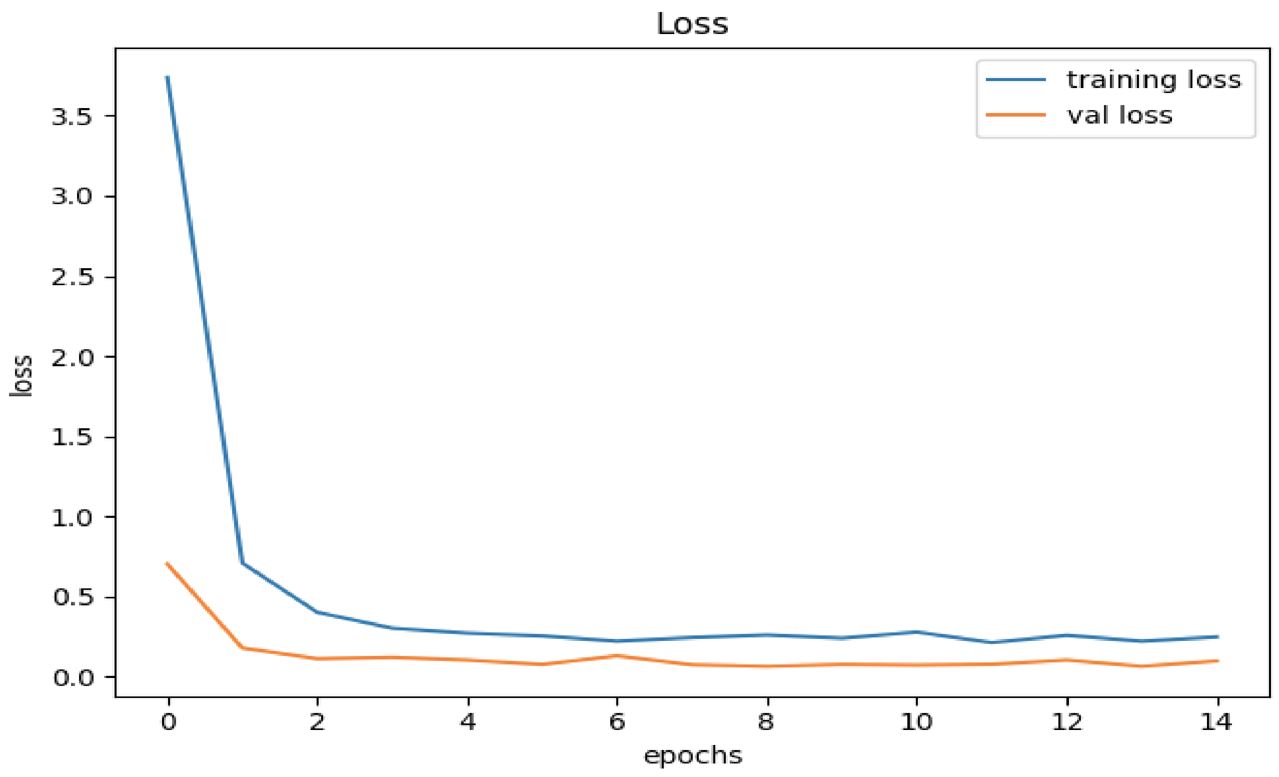
(a)



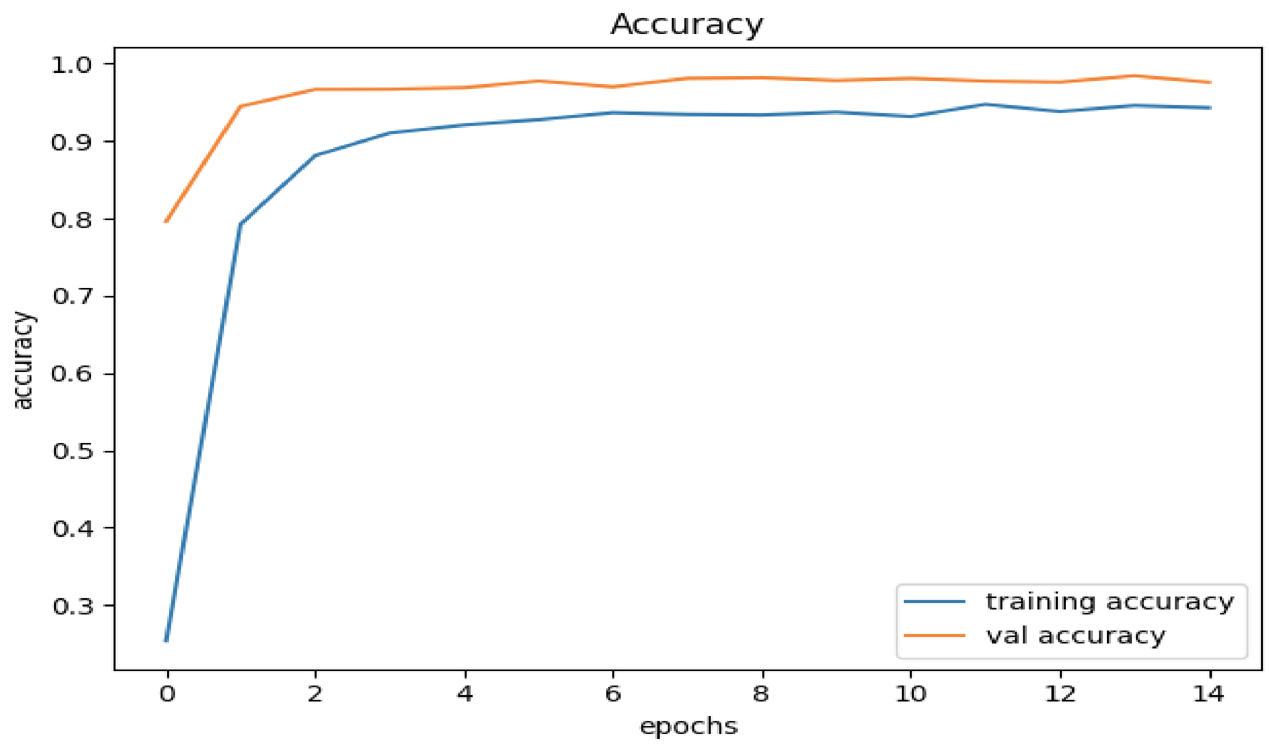
(b)

Fig 5.1.3 (a) Loss vs Epoch and (b) Accuracy vs Epoch

5.1.4 Selu:



(a)



(b)

Fig 5.1.4 (a) Loss vs Epoch and (b) Accuracy vs Epoch

Convolutional Neural Networks (CNN) are better models for classification and detection of objects in visual tasks, specifically traffic sign recognition (TSR). Changing the order of connections in the convolutional and pooling layers can improve the accuracy, efficiency, and cost-effectiveness of the systems, considering their application in autonomous vehicles. In all, the accuracy of these different models is observed to be reliable and may have productive applications with autonomous vehicles and other real world technologies.

2) Lane Detection

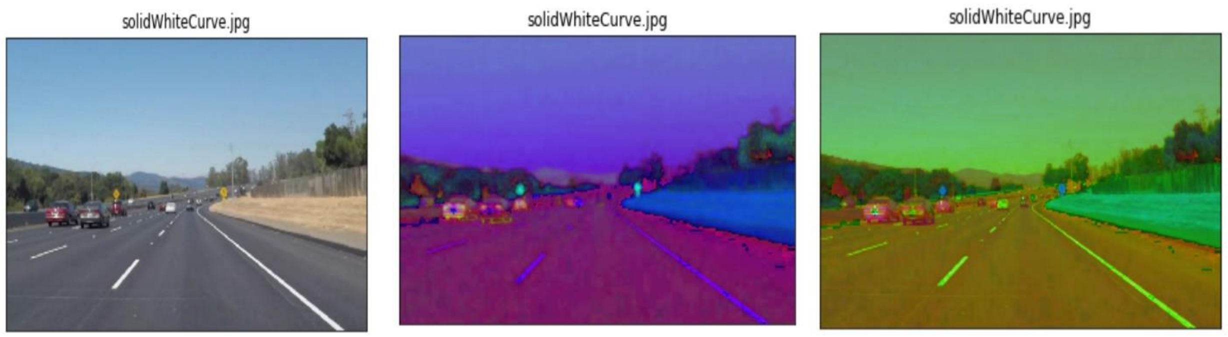


Fig 5.2.1 Original vs HSV vs HSL

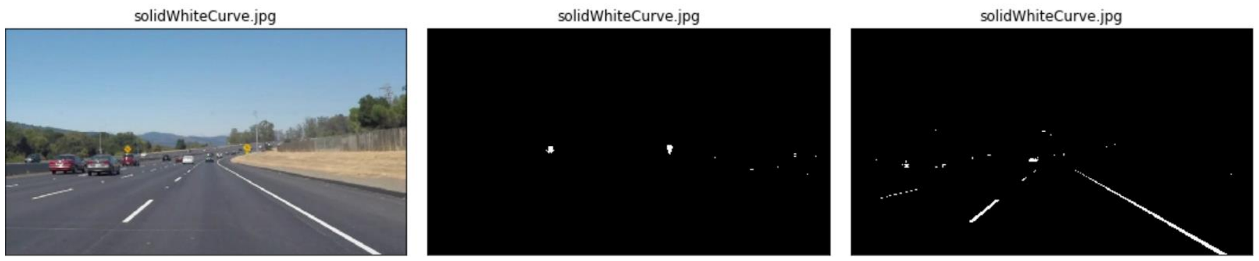


Fig 5.2.2 Isolating Yellow and White

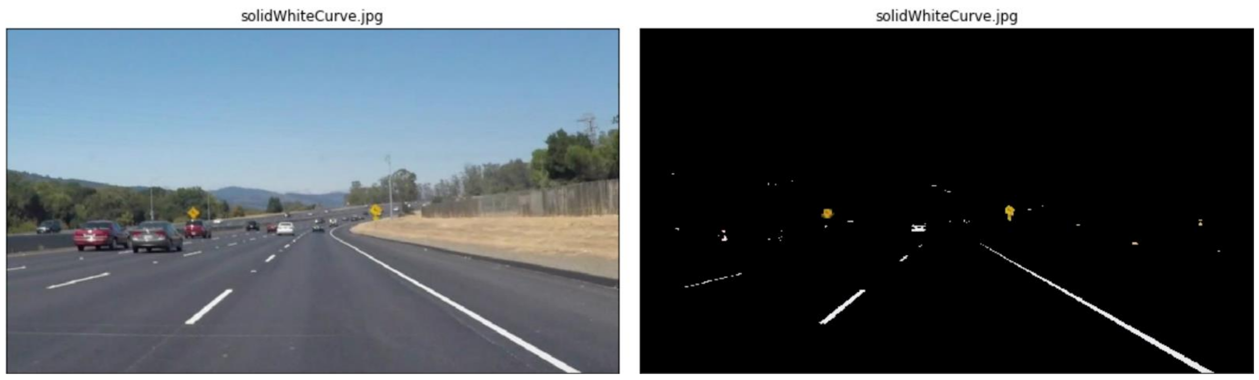


Fig 5.2.3 Combining the frames

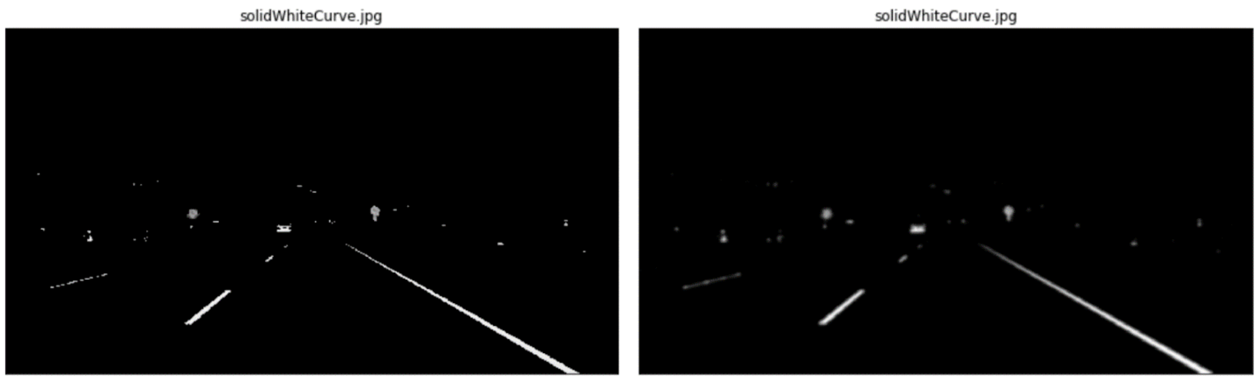


Fig 5.2.3 Gray Scale and Gaussian Blur

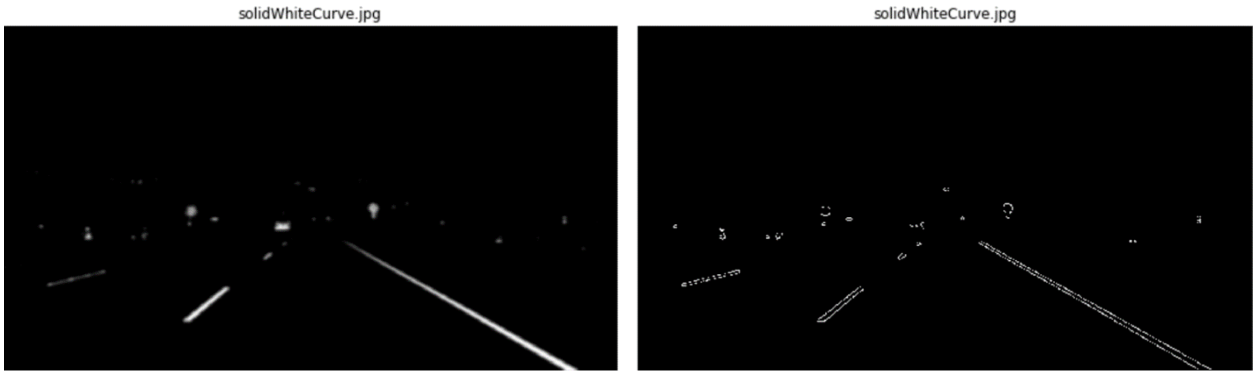


Fig 5.2.4 Original vs HSV vs HSL

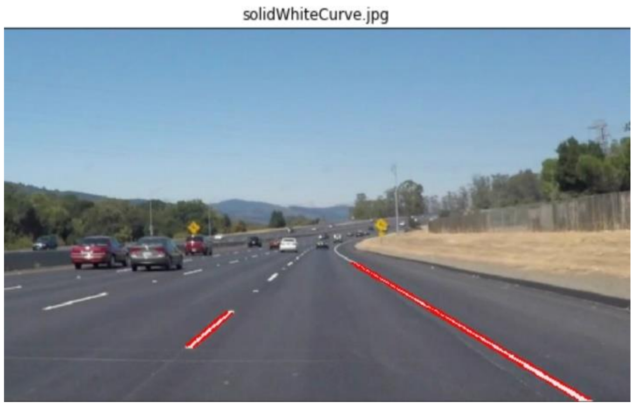


Fig 5.2.5 Original vs HSV vs HSL Hough Transform

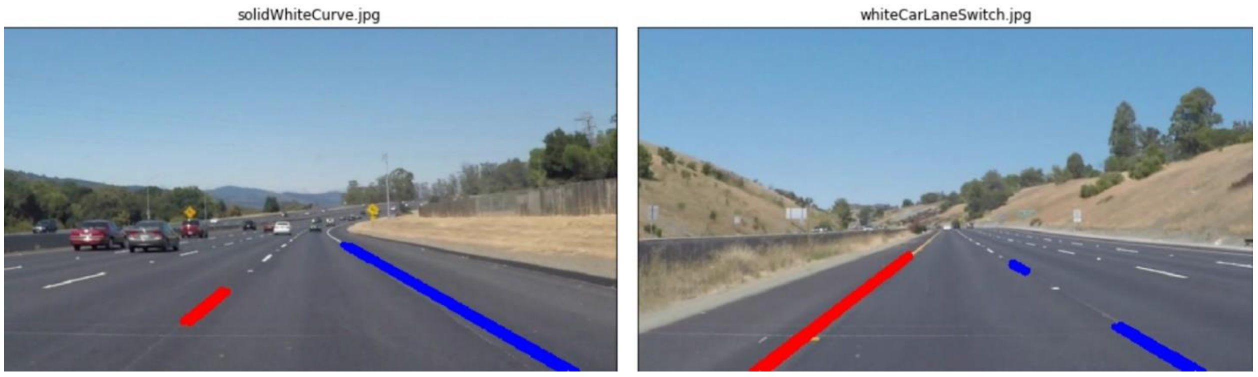


Fig 5.2.6 Left and Right Edge Lane Separator

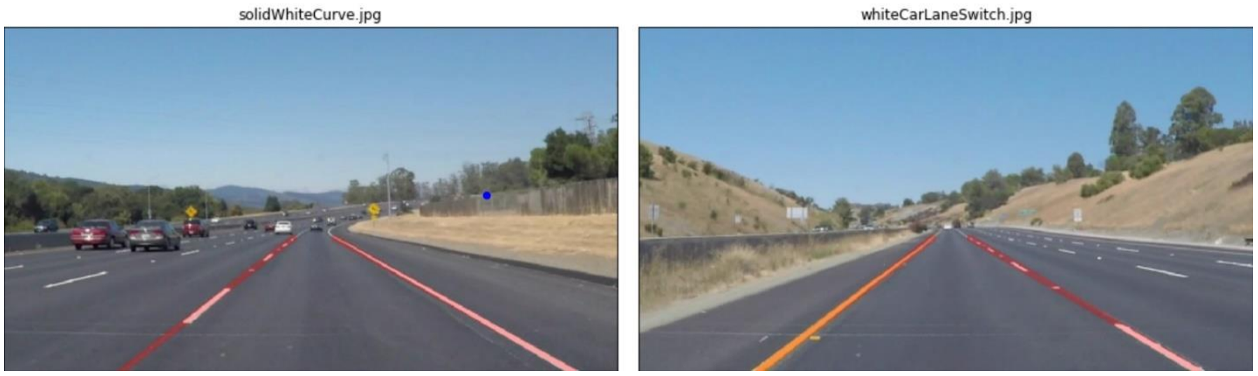


Fig 5.2.7 Regression based extrapolation

5.3. **Drowsiness Detection:**

This detector works in a variety of conditions and only slightly depends on the camera hardware used. Shown below are two situations, one where the EAR > 0.25, and the other where the person's eyes were closed and the EAR dropped below 0.25 for atleast 46 frames, and DROWSINESS ALERT was displayed:

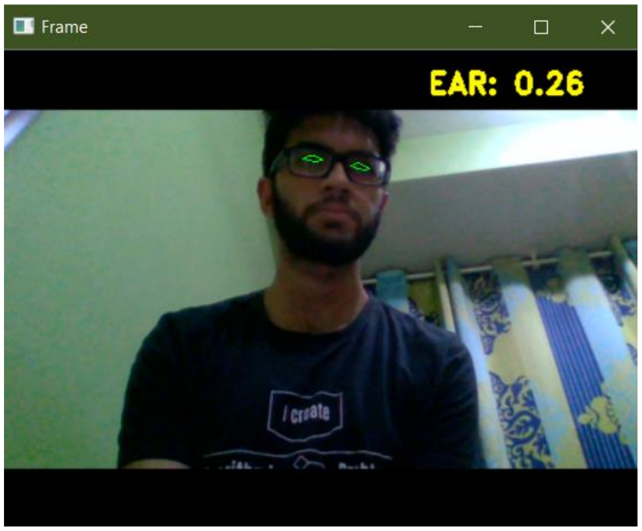


Figure 5.3.1 Not drowsy

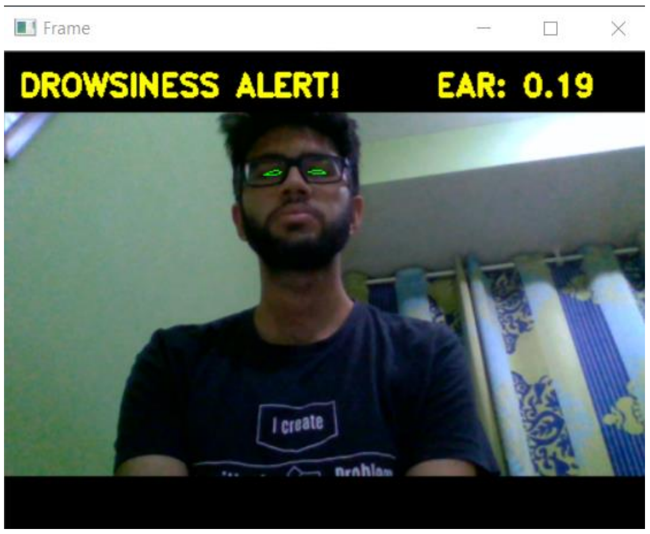


Figure 5.3.2 Drowsy

5.4 Vehicle Detection:

:

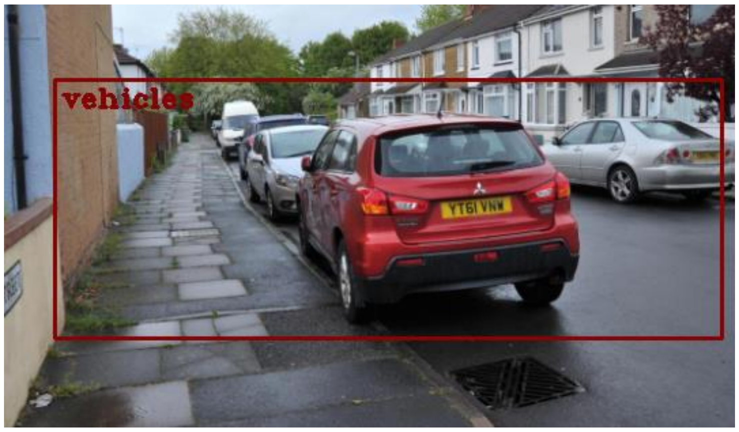


Fig 5.4.1 Results observed from the custom model

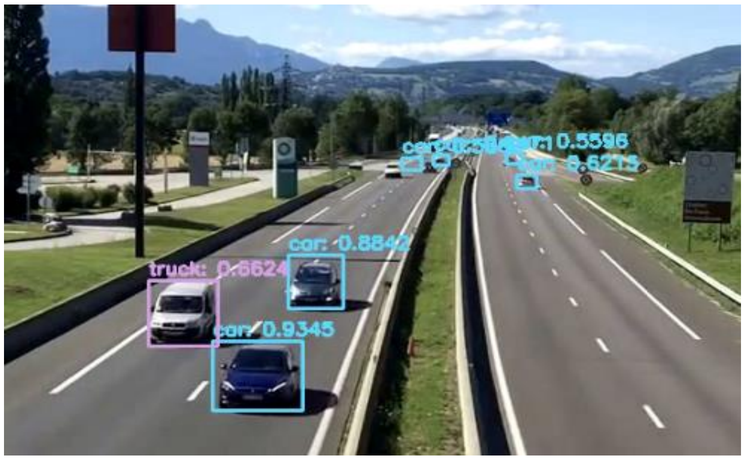


Fig 5.4.2 Results observed from the pretrained model

The main shortcoming in our custom model was with the amount of data pre-processing. Since the number of training images taken were 240, the validation of the model cannot be perfect. Vehicular data is one of the most complex datasets, since vehicles include many different sizes and forms, and every vehicle has four different orientations. For any real-world application to be built which uses vehicle detection in real time, we therefore infer that YOLO9000 built on the COCO dataset would be perfect as it run on high precision and speed. The custom model was built from scratch with a research purpose to understand the working and structure of YOLOv3 and to get hands on experience with training a deep neural network model, both of which objectives were completed successfully.

Chapter 6

CONCLUSION AND FUTURE WORK

The main objective of this project was to create software which can be used in assisted driving systems, hence consisting of the four modules: Traffic Sign Classification, Vehicle Detection, Lane Detection and Drowsiness Detection. The output of the entire project was successful in terms of getting the results which can be used in a real time system with the support of a dash cam for image sources. The trained modules can further be improved by expanding the size of the dataset and by performing novel data pre-processing steps therefore making the data more suitable for the training process, which would result in extended iterations before the average loss goes below a threshold, and hence the output achieved would improve.

In terms of future work, the project right now encompasses of modules which are only focusing on the software aspect of the solution. The project can be extended further to make sure that the outputs of all the software modules can be translated into machine understandable output. This could then be paired with hardware devices which could further execute these instructions in an automotive. For example, if the Traffic Classification modules reads a “Stop” Sign, brakes can be applied in real time in the vehicle. Along with this, pedestrian detection and navigation systems can also be embedded in the entire system to make it more complete with all the driving tools integrated.

REFERENCES

1. Lin, F., Lai, Y., Lin, L., & Yuan, Y. (2016, August). A traffic sign recognition method based on deep visual feature. In *2016 Progress in Electromagnetic Research Symposium (PIERS)* (pp. 2247-2250). IEEE.
2. Zeng, Y., Xu, X., Shen, D., Fang, Y., & Xiao, Z. (2016). Traffic sign recognition using kernel extreme learning machines with deep perceptual features. *IEEE Transactions on Intelligent Transportation Systems*, *18*(6), 1647-1653.
3. Cireşan, D., Meier, U., Masci, J., & Schmidhuber, J. (2011, July). A committee of neural networks for traffic sign classification. In *The 2011 international joint conference on neural networks* (pp. 1918-1921). IEEE.
4. Qian, R., Yue, Y., Coenen, F., & Zhang, B. (2016, August). Traffic sign recognition with convolutional neural network based on max pooling positions. In *2016 12th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD)* (pp. 578-582). IEEE.
5. Dhar, P., Abedin, M. Z., Biswas, T., & Datta, A. (2017, December). Traffic sign detection—A new approach and recognition using convolution neural network. In *2017 IEEE Region 10 Humanitarian Technology Conference (R10-HTC)* (pp. 416-419). IEEE.
6. Jung, S., Lee, U., Jung, J., & Shim, D. H. (2016, August). Real-time Traffic Sign Recognition system with deep convolutional neural network. In *2016 13th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI)* (pp. 31-34). IEEE.
7. Sermanet, P., & LeCun, Y. (2011, July). Traffic sign recognition with multi-scale convolutional networks. In *The 2011 International Joint Conference on Neural Networks* (pp. 2809-2813). IEEE.
8. Qian, R., Zhang, B., Yue, Y., Wang, Z., & Coenen, F. (2015, August). Robust Chinese traffic sign detection and recognition with deep convolutional neural network. In *2015 11th International Conference on Natural Computation (ICNC)* (pp. 791-796). IEEE.
9. Zhu, Y., Liao, M., Yang, M., & Liu, W. (2017). Cascaded segmentation-detection networks for text-based traffic sign detection. *IEEE transactions on intelligent transportation systems*, *19*(1), 209-219.
10. Liao, M., Shi, B., Bai, X., Wang, X., & Liu, W. (2017, February). Textboxes: A fast text detector with a single deep neural network. In *Thirty-First AAAI Conference on Artificial Intelligence*.
11. Hu, W., Zhuo, Q., Zhang, C., & Li, J. (2017). Fast branch convolutional neural network for traffic sign recognition. *IEEE Intelligent Transportation Systems Magazine*, *9*(3), 114-126.
12. Redmon, Joseph & Farhadi, Ali. (2018). YOLOv3: An Incremental Improvement

13. Seo, Chang. (2019) “Vehicle Detection and Car Type Identification System using Deep Learning and Unmanned Aerial Vehicle”, International Journal of Innovative Technology and Exploring Engineering (IJITEE) ISSN: 2278-3075, Volume-8 Issue-8S2
14. Simhambhatla, Ramesh; Okiah, Kevin; Kuchkula, Shravan; and Slater, Robert (2019) "Self-Driving Cars: Evaluation of Deep Learning Techniques for Object Detection in Different Driving Conditions," SMU Data Science Review: Vol. 2 : No. 1 , Article 23
15. Aryal, Milan, "Object Detection, Classification, and Tracking for Autonomous Vehicle" (2018). Masters Theses. 912
16. Lecheng Ouyang and Huali Wang 2019 “Vehicle target detection in complex scenes based on YOLOv3 algorithm” IOP Conf. Ser.: Mater. Sci. Eng. 569 052018
17. Song, H., Liang, H., Li, H. *et al.* Vision-based vehicle detection and counting system using deep learning in highway scenes. *Eur. Transp. Res. Rev.* **11,** 51 (2019)
18. Geiger, Andreas & Lenz, P & Stiller, Christoph & Urtasun, Raquel. (2013). Vision meets robotics: the KITTI dataset. The International Journal of Robotics Research. 32. 1231-1237. 10.1177/0278364913491297.
19. Soukupová, T., & Cech, J. (2016). Real-Time Eye Blink Detection using Facial Landmarks.
20. M. FENICHE and T. MAZRI, "Lane Detection and Tracking For Intelligent Vehicles: A Survey," 2019 International Conference of Computer Science and Renewable Energies (ICCSRE), Agadir, Morocco, 2019, pp. 1-4.
21. L. Dang, G. Tewolde, X. Zhang and J. Kwon, "Reduced resolution lane detection algorithm," 2017 IEEE AFRICON, Cape Town, 2017, pp. 1459-1464.
22. G. Deng and Y. Wu, "Double Lane Line Edge Detection Method Based on Constraint Conditions Hough Transform," 2018 17th International Symposium on Distributed Computing and Applications for Business Engineering and Science (DCABES), Wuxi, 2018, pp. 107-110.
23. J. Canny, "A Computational Approach to Edge Detection," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. PAMI-8, no. 6, pp. 679-698, Nov. 1986.
24. https://github.com/anilkay/ComputerVisionExamples/blob/master/beatian.ipynb
25. https://ibug.doc.ic.ac.uk/resources/facial-point-annotations/
26. http://dlib.net/face_landmark_detection.py.html

ACKNOWLEDGEMENT

We would like to express our sincere gratitude to our mentor, Dr. Pravin Shrinath, for helping us and guiding us throughout the project, along with proving us periodic feedback which could drive the project in the right direction.

We would also like to thank the Computer Department of MPSTME, for providing us with knowledge material thereby enabling us to execute this project to the best of our abilities.