

TRAFFIC SIGN RECOGNITION

BY

TAN JING JIE

JACYNTH THAM MING QUAN

TAN WEI MUN

NG JAN HUI

POR TEONG DEAN

A REPORT

SUBMITTED TO

Universiti Tunku Abdul Rahman

in partial fulfilment of the requirements

for the degree of

BACHELOR OF COMPUTER SCIENCE (HONS)

Faculty of Information and Communication Technology

(Kampar Campus)

MAY 2021

ABSTRACT

[Done By: Jacynth Tham Ming Quan]

In this technology-centred era, people have slowly become accustomed to having computer vision and artificial intelligence assist their every action. One of the prominent fields of artificial intelligence is computer vision, which is commonly found in image recognition and object detection. One significant application of object detection is intelligent cars, which are often integrated with traffic sign recognizers as protective measures. However, upon analysing existing traffic sign recognizers, it was discovered that most of the recognizers were unable to perform in real-time and were susceptible to weather and lighting conditions.

Motivated by the will to enhance the driving experience of drivers on the road, this project aims to overcome the problems associated with existing traffic sign recognizers as well as create an enhanced traffic sign recognizer altogether. Thus, this report proposes a novel traffic sign recognizer that uses image processing and computer vision technologies, aimed at becoming a module that can be integrated into smart cars in the future. In order to affirm the real-time capabilities of the proposed traffic sign recognizer, this project also includes a mobile application with the integration of the proposed traffic sign recognizer.

The proposed project is realized with the cohesive use of image processing and computer vision techniques in both desktop-based and mobile-based applications. First, still images or images from real-time video streams were pre-processed using bilinear interpolation. Then, the detected traffic signs were segmented out using HSV colour space segmentation. Then, the traffic sign classification process is handled using a combination of Hu Moments and various classifiers such as MobileNet (CNN), SVM and Random Forest. During this process, the traffic sign recognizer is able to perform real-time tracking of the detected traffic sign if video streams were used as the input. Finally, the output of the proposed traffic sign recognizer yields an image or video stream with the recognized traffic sign bounded and labelled.

TABLE OF CONTENTS

TITLE PAGE	i
ABSTRACT	ii
TABLE OF CONTENTS	iii
LIST OF TABLES	xi
LIST OF FIGURES	xiii
LIST OF ABBREVIATIONS	xviii
Chapter 1 Project Background	19
1.1 Problem Statement and motivation	19
1.2 Background	20
1.3 Project Objectives	22
1.4 Proposed Approach/ Study	22
1.5 Highlight of Achievement	23
1.6 Report Organisation	24
Chapter 2 Literature Review	25
2.1 Overview	25
2.2 Traffic Sign Recognition Using Support Vector Machine (SVM)	25
2.2.1 Real-Time Detection and Recognition of Road Traffic Signs	25
2.2.2 An Incremental Framework for Video-Based Traffic Sign Detection, Tracking, and Recognition	29
2.2.3 On Circular Traffic Sign Detection and Recognition	32
2.3 Convolutional Neural Network (CNN)	35
2.3.1 Traffic-Sign Detection and Classification in the Wild	35
2.3.2 Traffic Sign Detection and Recognition Using Fully Convolutional Network Guided Proposals	38
2.3.3 Traffic Sign Recognition with Hinge Loss Trained Convolutional Neural Networks	40

iii

Bachelor of Computer Science (Hons)

Faculty of Information and Communication Technology (Kampar Campus), UTAR

2.3.4	A Committee of Neural Networks for Traffic Sign Classification	43
2.3.5	Towards Real-Time Traffic Sign Detection and Classification	45
2.3.6	A Real-Time Chinese Traffic Sign Detection Algorithm Based on Modified YOLOv2	48
2.4	Random Forest Classifier	51
2.4.1	Real-Time Traffic Sign Recognition in Three Stages	51
2.4.2	Traffic sign detection and recognition based on random forests	54
2.5	Neural Network	57
2.5.1	An Efficient Method for Traffic Sign Recognition Based on Extreme Learning Machine	57
2.5.2	Traffic Sign Recognition using Perceptual Generative Adversarial Networks (GAN)	60
2.6	Fourier Descriptors	63
2.6.1	Using Fourier Descriptors and Spatial Models for Traffic Sign Recognition	63
2.7	Critical Remarks of Previous Works	66
2.7.1	Advantages	66
2.7.2	Shortages	67
2.7.3	Table of Comparison	69
2.7.4	Future Work	72
Chapter 3 System Design		73
3.2	System Design / Overview	78
3.2.1	System input	80
3.2.2	Bilateral blurring filter	81
3.2.3	Colour mask generate using Hue Saturation Value (HSV) colour space	81

3.2.4	Mask processing	82
3.2.5	Contour Finding	83
3.2.6	Elimination of small contour	84
3.2.7	Fill the contour	84
3.2.8	Segmentation	85
3.2.9	Further segmentation	86
3.2.10	Normalization of image	87
3.2.11	Extracting HoG Descriptors from segmented traffic sign	88
3.2.12	Computing Hu Moments from the internal symbol of traffic sign	89
3.2.13	Concatenating both features into a feature matrix	89
3.2.14	Data imputation using Sklearn	90
3.2.15	Loading dataset and data preparation	90
3.2.16	Dataset Splitting	91
3.2.17	Model Initialization	91
3.2.18	Model Training and Testing	92
3.2.19	Experimental Results Analysis	92
Chapter 4	Image Pre-processing	93
4.0	Overview	93
4.1	Methodology and Tools	93
4.2	Requirement	94
4.3	Design Block Diagram	94
4.4	Verification Plan	96
4.5	Implementation	97
4.5.1	The Bilateral Blurring filter	97
4.5.2	The Mask Generated by HSV Colour Space	99
4.5.3	The Mask Processing to Eliminate Noise	103
4.5.4	The Contour Finding	104

4.5.5	The Elimination of Insignificant Contour	104
4.5.6	The Filling of The Contour	105
4.5.7	The Segmentation by using Canvas Mask	107
4.5.8	The Further segmentation	110
4.5.9	The Normalisation Resize	110
4.6	Testing	112
4.7	Summary	117
Chapter 5 Feature Extraction		118
5.0	Overview	118
5.1	Methodology and Tools	118
5.2	Requirement	119
5.3	Design Block Diagram	119
5.4	Verification Plan	121
5.5	Implementation	122
5.5.1	Generating HoG and Hu moments features	122
5.5.2	Write feature matrix to CSV	123
5.6	Testing	125
5.7	Summary	127
Chapter 6 Classification		128
6.0	Overview	128
6.1	Methodology and Tools	128
6.2	Requirement	129
6.3	Design block diagram	129
6.4	Verification plan	131
6.5	Chinese Traffic Sign Feature Dataset Preparation	132
6.5.1	Dataset loading and splitting	132
6.5.2	Training and testing labels type conversion	133
6.5.3	Dataset Standardization	133
6.6	GTRSB Dataset Preparation	134

6.6.1	Dataset Exploration	134
6.6.2	Dataset Splitting	136
6.5.3	Dataset Preprocessing	137
6.7	SVM Model Classifier	138
6.7.1	SVM Model Initialization	138
6.7.2	SVM Model Training	139
6.7.3	SVM Model Evaluation	140
6.8	Random Forest Classifier	141
6.8.1	Random Forest Model Initialization	141
6.8.2	Random Forest Model Training	142
6.8.3	Random Forest Model Evaluation	143
6.9	Mobile Net Convolutional Neural Network (CNN)	145
6.9.1	Google Colab Project Setup	145
6.9.2	Model Configuration	145
6.9.3	Training the Model Using Transfer Learning	146
6.9.4	Testing the Model	149
6.9.5	Converting and Exporting the Model	150
6.10	Testing	152
6.11	Summary	156
Chapter 7 Desktop application		157
7.0	Overview	157
7.1	Methodology and Tools	157
7.2	Requirement	158
7.3	Block Diagram	159
7.4	Verification Plan	162
7.5	Implementation and Testing	164
7.5.1	User input static image	164
7.5.2	Video-Based Recognition	174
7.5.3	Camera Based Recognition	180

7.6 Summary	185
Chapter 8 Android application	186
8.0 Overview	186
8.1 Methodology and Tools	186
8.2 Requirement	187
8.3 Block Diagram	188
8.4 Verification Plan	190
8.5 Implementation	191
8.5.1 Project Setup (Android Studio)	192
8.5.2 Application Initialization and Permissions	192
8.5.3 Input	193
8.5.4 Image preprocessing – Segmentation of traffic sign	195
8.5.5 Classification of Traffic Signs	197
8.5.6 Output (Image-Based)	199
8.5.7 Output (Video-Based)	203
8.6 Testing	206
8.7 Summary	222
Chapter 9 Experimental Result	223
9.0 Overview	223
9.1 System Performance	224
9.1.1 Exporting predicted labels and ground truth labels	224
9.1.2 Computing classification metrics	225
9.1.3 Prediction performance visualization using Confusion Matrix	
227	
9.2 Model Comparison	230
9.2.1 Comparison between SVM, Random Forest and CNN	230
9.2.2 Benchmarking with Previous Works	231
9.3 Error Analysis	233

9.4	Future Work	235
9.5	Contribution	239
9.6	Summary	240
Chapter 10 Conclusion		241
10.0	Project Overview	241
10.1	Summary of Methodology and Implementation	241
10.2	Future Work and Concluding Note	242
Bibliography		243
Appendices		1
A	Poster	1
B	Work Code	1
B.1	Image preprocessing	1
B.2	Feature Extraction	9
B.3	Feature Dataset Imputation (Python)	11
B.4	End-to-end Model Training (SVM + RF)	13
B.5	Mobile Net Convolutional Neural Network Classification Training, Testing and Experimental result	24
B.6	Desktop application	38
B.7	Mobile application	68
B.8	Experimental Results (Python)	98
C	Image pre-processing flow	1
C.1	No horn sign	1
C.2	Car sign	1
C.3	U-turn	2
C.4	Crossing	3
C.5	No stop	3
C.6	No entry	4
C.7	Multiple traffic sign. (Combination of six traffic sign)	5

C.8	Small traffic sign (blue)	6
C.9	Small traffic sign (red).	6
C.10	Multiple traffic sign. (Real environment and overlapped traffic sign)	7
C.11	Multiple traffic sign. (Real environment and overlapped traffic sign)	7
C.12	Multiple traffic sign. (Real environment and overlapped traffic sign)	8
C.13	Multiple traffic sign. (Check the image preprocessing coverage)	9

LIST OF TABLES

Table Number	Title	Page Number
Table 2.7.3.1	The comparison table for previous works approach	69
Table 3.1.2.1	Hardware Specifications	75
Table 4.5.2.1	The colour range defined for each mask	100
Table 4.6.1	Test Case: Red Traffic Sign Input	112
Table 4.6.2	Test Case: Blue Traffic Sign Input	113
Table 4.6.3	Test Case: Yellow Traffic Sign Input	113
Table 4.6.4	Test Case: Multiple Traffic Signs Input of Varying Colours	114
Table 4.6.5	Test Case: Multiple Overlapped Traffic Signs Input of Varying Colours	115
Table 4.6.6	Test Case: Small Sized Traffic Signs Input	116
Table 5.4.1	Feature Extraction Test Cases (Desktop Application)	121
Table 5.6.1	Test case of Compute HoG features	125
Table 5.6.2	Test case of Compute Hu Moments	125
Table 5.6.3	Test Case of Concatenate Feature	126
Table 5.6.4	Test Case of CSV	127
Table 6.4.1	Classification Test Cases (SVM & Random Forest)	131
Table 6.4.2	Classification Test Cases (MobileNet)	131
Table 6.7.1.1	SVM classifier hyperparameters	138
Table 6.8.1.1	The hyperparameter list of Random Forest Model Initialization	141
Table 6.10.1	Test case of model completed the training and testing process	152
Table 6.10.2	Test case of export of trained model	153
Table 6.10.3	Test case of SVM and RF Training and Testing	154
Table 6.10.4	Test case of SVM and RF Model Save	155
Table 7.5.1.1	Classification results	166
Table 7.5.2.1	Video-based implementation testing	175
Table 7.5.3.1	Webcam-Based Traffic Sign Recognizer Results	181
Table 7.5.3.2	Webcam-Based Traffic Sign Recognizer Results	182
Table 7.5.3.3	Webcam-Based Traffic Sign Recognizer Results	183
Table 8.4.1	Image-Based Input Activity Test Cases	190
Table 8.4.2	Video-Based Input Activity Test Cases	191

Table 8.5.4.1	The HSV Range used	195
Table 8.6.1	Test Case: Red Traffic Sign Input (Image-Based)	206
Table 8.6.2	Test Case: Blue Traffic Sign Input (Image-Based)	207
Table 8.6.3	Test Case: Yellow Traffic Sign Input (Image-Based)	208
Table 8.6.4	Test Case: Multiple Traffic Sign with Various Lighting Input (Image-Based)	209
Table 8.6.5	Test Case: Multiple Input Images Containing Traffic Signs of Varying Color (Image-Based)	210
Table 8.6.6	Test Case: Speak out traffic sign (Image-Based)	211
Table 8.6.7	Test Case: No Valid Traffic Sign (Image-Based)	212
Table 8.6.8	Test Case: Export Each Segmented Traffic Sign (Image-Based)	213
Table 8.6.9	Test Case: Red Traffic Sign Input (Video-Based)	215
Table 8.6.10	Test Case: Blue Traffic Sign Input (Video-Based)	217
Table 8.6.11	Test Case: Yellow Traffic Sign Input (Video-Based)	218
Table 8.6.12	Test Case: Traffic Sign Tracking with Bounding Box (Video-Based)	219
Table 8.6.13	Test Case: No Valid Traffic Sign (Image-Based)	220
Table 8.6.14	Test Case: Button Redirection to File Input Activity (Video-Based)	221
Table 9.1.2.1	Classification Metrics for both classifiers	226

LIST OF FIGURES

Figure Number	Title	Page Number
Figure 2.1.2.1	The proposed schematic of the approach. (Greenhalgh & Mirmehdi, 2012)	26
Figure 2.1.2.2	The illustration of cascaded SVM classifiers. (Greenhalgh & Mirmehdi, 2012)	27
Figure 2.1.3.1	Component of the proposed TSR framework (2017)	29
Figure 2.1.3.2	Component of the proposed TSR framework (Yuan, et al., 2017)	30
Figure 2.2.3.1	Berkaya et al.'s Approach for Traffic Sign Recognition	32
Figure 2.2.3.2	Local Binary Pattern Sub-Region Divisions	33
Figure 2.3.1.1	Architecture of Zhu et al's Multi-Class CNN Model	36
Figure 2.3.1.2	Summary of Annotation Pipeline	37
Figure 2.3.2.1	The proposed method's pipework	38
Figure 2.3.2.2	Architecture of FCN	39
Figure 2.3.3.1	Proposed CNNs Architecture by Jin et al.	41
Figure 2.3.4.1	Architecture of a convolutional neural network	44
Figure 2.3.5.1	The pipeline of the proposed traffic sign recognition system. (a) Original image. (b) Probability maps (red + blue). (c) Proposals. (d) Detection result. (e) Classification.	46
Figure 2.3.5.2	The examples of sub-classes of GTSDB. (a) Prohibitory. (b) Danger (c) Mandatory.	46
Figure 2.3.5.3	The examples of sub-classes of CTSD. (a) Prohibitory. (b) Danger. (c) Mandatory.	47
Figure 2.3.6.1	Detection Algorithm for Chinese traffic signs (Zhang, et al., 2017)	48
Figure 2.3.6.2	Shows the comparison between German traffic signs (Left) and their corresponding Chinese traffic signs (Right)	49
Figure 2.3.6.3	Models created by (Zhang, et al., 2017)	49
Figure 2.4.1.1	Zaklouta and Stanciulescu's Three-Stage Approach for Traffic Sign Recognition	51
Figure 2.4.1.2	Summary of Zaklouta and Stanciulescu's Traffic Sign Detection Phase	52
Figure 2.4.2.1	Logical flow of the 3-stage algorithm (Ellahyani, et al., 2016)	54

Figure 2.4.2.2 Triangle, Circular and Rectangular Binary Patches of Traffic Signs (Ellahyani, et al., 2016)	55
Figure 2.4.2.3 HSI-HOG Descriptor Computation Flow (Ellahyani, et al., 2016)	55
Figure 2.4.2.4 Correct Classification Rates and Run Time for each descriptor (Ellahyani, et al., 2016)	56
Figure 2.5.1.1 Framework for the proposed TSR method using HOGv and ELM approach. (Huang, et al., 2016)	57
Figure 2.5.1.2 The extraction of HOGv descriptor (Huang, et al., 2016)	58
Figure 2.1.5.1 Novel Perceptual GAN Model of (Li, et al., 2017)	60
Figure 2.1.5.2 Super-resolved features spawned by the Generator Network (Li, et al., 2017)	61
Figure 2.1.5.3 Comparison of detection performance on traffic signs of different sizes (Li, et al., 2017)	62
Figure 2.6.2.1 Local features (green contours) was extracted and corresponding vectors (red arrow) pointing towards the center of the traffic sign.	64
Figure 2.6.2.2 Matching Sign Prototype Process	64
Figure 2.6.2.3 Examples of Result	65
Figure 3.1.1 System Overview Block Diagram	73
Figure 3.2.1 The Block Diagram for This Traffic Sign Recognition System	79
Figure 3.2.1.1 The sample in notepad file	80
Figure 3.2.2.1 The comparison of traffic sign before and after of bilateral blurring	81
Figure 3.2.3.1 The mask generated from a traffic sign.	82
Figure 3.2.4.1 The mask processing flow from original to eroded and dilated	83
Figure 3.2.5.1 Input and Output of the FindContours Function	83
Figure 3.2.6.1 The removal of insignificant contour.	84
Figure 3.2.7.1 The mask generated from a traffic sign.	85
Figure 3.2.2.8 The segmentation process	85
Figure 3.2.9.1 The further segmentation process image	86
Figure 3.2.10.1 The normalized resize process from further segmentation image	87
Figure 4.3.1.1 The overall flow of image segmentation.	95
Table 4.4.1 Image Pre-Processing Test Cases (Desktop Application)	96
Figure 4.5.1.1 The comparison of other blurring filter result and its generated mask.	

Figure 4.5.1.2 The comparison between the before and after applying Bilateral blurring filter to a traffic sign.	99
Figure 4.5.2.2 Colour picker show the Hue and Saturation of red colour bound at left and right of the colour space	100
Figure 4.5.3.4 The comparison result of the before and after of the blue mask after applying morphology close function.	101
Figure 4.5.3.5 The combination of mask to segment multiple different colour traffic signs	102
Figure 4.5.3.1 The mask undergo the erode and dilate process.	103
Figure 4.5.4.1 The finding the contours from the masks.	104
Figure 4.5.4.1 The final canvas contour after eliminating noise	105
Figure 4.5.6.1 The detail process of fillHole() function	106
Figure 4.5.6.2 The conversion from final contour to final canvas mask.	106
Figure 4.5.6.3 The conversion from final canvas to final canvas mask.	107
Figure 4.5.6.4 The example approxPolyDP() function applied with epsilon value = 9.	
	108
Figure 4.5.7.1 The segmentation of each image and saved into local directory.	109
Figure 4.5.8.1 The 6 traffic sign images undergoing further segmentation.	110
Figure 4.5.9.1 The resize of segmented traffic sign.	111
Figure 4.5.9.2 The resize of further segmented traffic sign (logo).	111
Figure 4.5.9.3 Export the image preprocessing flow and the image preprocessing result.	
	111
Figure 5.3.1 Feature Extraction Flow	120
Figure 5.5.1 Feature matrices generated from traffic signs samples	123
Figure 5.5.2.1 Concatenated feature matrix of Hu Moments, HoG features and labels	
	124
Figure 6.3.1 The overall flow of classification	130
Figure 6.5.1.1 Example of first 10 training samples	132
Figure 6.5.1.2 Example of first 10 testing samples	132
Figure 6.6.1.1 Sample Batch of Images from GTSRB Dataset	134
Figure 6.6.1.2 Sample Batch of Images of Stop Sign Class	135
Figure 6.6.1.3 Contents of mobilenet_label_readable.txt	136
Figure 6.6.1 Number of Images In The Training Set and Testing Set	136
Figure 6.7.2.1 XML File that holds all configurations for the SVM classifier	140

Figure 6.7.3.1 SVM Classification Accuracy	140
Figure 6.8.2.1 XML File that holds all configurations for the RF classifier	143
Figure 6.8.3.1 Random Forest Classification Accuracy	143
Figure 6.9.3.1 Summary of Newly Defined MobileNet Model	147
Figure 6.9.3.2 Output of Model Training Process	147
Figure 6.9.3.3 Graph of Model Loss Against Training Steps	148
Figure 6.9.3.4 Graph of Model Accuracy Against Training Steps	149
Figure 6.9.4.1 Graph of Model Accuracy Against Training Steps	149
Figure 6.9.5.2 Contents of Output Folder After Exporting Model	150
Figure 6.9.5.3 Contents of Output Folder After Exporting Model	151
Figure 7.3.1 Block Diagram of Desktop Application	159
Figure 3.2.2 Proposed Block Diagram for Traffic Signs Recognizer System (Desktop Application)	161
Figure 7.5.1.1 User Input Image Preprocessing	165
Figure 7.5.1.2 Classification results based on user input	165
Figure 7.5.1.3 Sending 60 images for segmentation	170
Figure 7.5.1.4 Predictions (1/3)	172
Figure 7.5.1.5 Predictions (2/3)	173
Figure 7.5.1.6 Predictions (3/3)	173
Figure 8.3.1 The Overall Flow of Android Mobile Application	189
Figure 4.7.2.1 Mobile Application's Splash Screen	193
Figure 8.5.3.1 Input Option User Interface	193
Figure 8.5.3.2 File-Based Image Input Activity	194
Figure 8.5.3.3 Real-Time Video Stream Input Activity	194
Figure 4.6.3.1 Colour Mask Output in User Interface	195
Figure 8.5.5.1 Classification Process Flowchart	198
Figure 8.5.6.1 The output after classification by using single traffic sign	199
Figure 8.5.6.2 The output after classification by using single traffic sign	200
Figure 8.5.6.3 No Traffic Sign Detected Output in File-Based Image Input Activity	201
Figure 8.5.6.3 The output after classification by using single traffic sign	202
Figure 8.5.7.1 Output of Real-Time Video-Based Input Activity	203
Figure 8.5.7.2 Second Output of Real-Time Video-Based Input Activity	204
Figure 8.5.7.3 No Traffic Sign Detected Output in Video-Based Input Activity	204

Figure 9.0.1 Block Diagram illustrating experimental results analysis flow	223
Figure 9.1.3.1 SVM Confusion Matrix	227
Figure 9.1.3.2 Random Forest Confusion Matrix	228
Figure 9.1.3.3 Mobile Net CNN Confusion Matrix	229
Figure 9.2.1.1 Visualization of Classification Metrics	230
Figure 9.2.2.1 Benchmarking with Previous Works	232
Figure 9.4.1 The neural network segmentation algorithm segment the objects	235
Figure 9.4.1 Image augmentation technique applied to a speed limit of 50 traffic sign.	236
Figure 9.4.2 Flow of tracking approach	237
Figure 9.4.3 Fusion strategy approach	238

LIST OF ABBREVIATIONS

<i>ACF</i>	Aggregated Channel Feature
<i>ADAS</i>	Advanced driver-assistance systems
<i>CNN</i>	Convolutional Neural Network
<i>CPM</i>	Colour Probability Model
<i>CTSD</i>	Chinese Traffic Sign Dataset
<i>EDCircles</i>	Edge Drawing Circles
<i>ELM</i>	Extreme Learning Machine
<i>FCN</i>	Fully Convolutional Network
<i>GAN</i>	Generative Adversarial Network
<i>GTSRB</i>	German Traffic Sign Recognition Dataset
<i>HOG</i>	Histogram of Gradients
<i>HOG</i>	Histogram of Oriented Gradients
<i>HSI</i>	Hue-Saturation-Intensity
<i>HSV</i>	Hue-Saturation-Value
<i>IDE</i>	Integrate Development Environment
<i>LBP</i>	Local Binary Pattern
<i>MLP</i>	Multilayer Perceptron
<i>MSER</i>	Maximally Stable Extremal regions
<i>OpenCV</i>	Open-Source Computer Vision
<i>ReLU</i>	Rectified Linear Units
<i>RGB</i>	Red-Green-Blue
<i>ROI</i>	Region of Interest
<i>ROI</i>	Region of interest
<i>SGD</i>	Stochastic Gradient Descent
<i>SVM</i>	Support Vector Machine
<i>TSR</i>	Traffic sign recognition

Chapter 1

Project Background

1.1 Problem Statement and motivation

[Done By: Ng Jan Hui]

Traffic sign recognition (**TSR**) technology has grown increasingly popular and has ever since been in high demand due to the recent developments within the field of autonomous vehicles. The aspect of traffic sign recognition is considered an integral part of Advanced driver-assistance systems (**ADAS**), which are a group of technologies that aims to help the driver in driving and parking functions. The main issue is that the most sophisticated and mature TSR technologies can only be found in ADAS systems that are not commercially available for the public. We intend to address the issue mentioned by developing a lightweight traffic sign recogniser that can detect traffic sign images presented to it using computer vision techniques that are currently available and accessible by the public.

Driving is widely considered a visual intensive task. The driver will need to rely on the visual stimulus (Pedestrians, Road Signs, Landmarks, Roadblocks, etc) that are scattered across the surroundings of their current location to make sure that they are always driving safe and are also on the right course towards their destination. We are motivated to take on this project because traffic sign recognition can be used to enhance the driving experience of the drivers and also ensure their safety. Through the usage of computer vision to recognise and classify road signs, the computer can tell the driver about the road information conveyed by the road signs. The driver can divert more attention onto the road ahead of them and still be able to perceive the road signs that are around their location. This greatly enhances their driving experience while also maintaining their safety as they will be more focused on the road ahead. Hence, the facts mentioned above are the inspirations that drive us into doing this project.

1.2 Background

[Done By: Tan Jing Jie]

Nowadays, there are numerous applications that implement computer vision technology and image processing. Computer vision is a field of Artificial Intelligence that makes computers interpret the visual world through camera devices. There are a lot of underlying models to let machines be able to classify objects and output with a correct label. Meanwhile, image processing is a method to eliminate noise and obtain useful information from input images. This approach is essential for computer vision as it is a significant process to obtain the feature from a basic image which is meaningful for the computer to recognise.

There are many fields that require computer vision, such as medical diagnosis, Advanced driver-assistance systems (ADAS), and self-driving systems. In self-driving systems, computer vision plays a role in identifying the road situation, obstacles, and traffic signs. The traffic sign besides the road provides significant instruction to every road user especially for those who are driving. Although the self-driving system can get the road information through an online map. But, when there is a case that the road is under-maintained, a temporary traffic sign will be located to give instructions to the drivers. On the other hand, the online map might not be up to date about the latest road instructions. Hence, it will be dangerous to passengers or other road users. Therefore, a reliable TSR system is required to support the self-driving field.

This Traffic Sign Recognition (TSR) system implements image pre-processing – segmentation of traffic sign, feature extraction and classification techniques to enable computers to identify the real world's traffic sign. In general, Traffic Sign Recognition (TSR) is that when given an image that consists of one or more traffic signs, and with a set of models with matched labels and known by system, the system shall allocate the right labels to the respective sections in the image. This TSR system aims to work in real time with being able to recognise different types of traffic signs with high accuracy to users. The output of the recognised result can be provided in various forms including text-based output and spoken output. This model of system is able to assist in self-driving cars, police's law enforcement, vision-impaired notification, etcetera.

In this TSR system, cameras are used for capturing images. The captured image will be followed by image processing to remove noise or eliminate the interference of

Chapter 1 Project Background

lighting, shadow, etc. There are several libraries that can support the image processing of the system. For instance, OpenCV library consists of a lot of tools that can assist to segment out each traffic sign in an image. The colour segmentation tool provided by OpenCV library can be used to remove the background of the traffic sign. Besides, OpenCV library also aids in doing feature extraction for each segmented traffic sign. Moreover, by using by OpenCV library or TensorFlow library, a traffic sign classifier can be trained to recognize the traffic sign and output the result with an understood label.

In this project, 3 classifiers will be developed and compared which are Random Forest, Support Vector Machine and MobileNet Convolutional Neural Network (CNN) classifiers. This project implements both desktop based and mobile based traffic sign recognition systems. This system is able to receive two types of input, including, still image, and video stream. The approach for video stream and real-time video stream recognition is to extract each image frame by frame and do the recognition process for each image. This system targets to integrate in navigation applications or other visual assistive applications especially for visually impaired people. Hence, a mobile based TSR system is designed with voice output features.

1.3 Project Objectives

[Done By: Jacynth Tham Ming Quan]

The primary aim of this paper is to propose, design and implement an automatic traffic sign recogniser using image processing and machine learning techniques. This paper's goal is to realise the future works of existing traffic sign recognition systems in order to improve the overall accuracy and processing time. This traffic sign recognizer is enacted using various classifiers, including MobileNet (CNN), SVM and Random Forest. Moreover, the implementation of this project will be realised through the integration of the traffic sign recognition model into a desktop application and a mobile application with the ability to work on still images or real-time video streams. Listed below are the main objectives of this project:

- To develop a traffic sign recognizer that is able to recognize at least 43 traffic signs in different kinds of environments and weather conditions.
- To implement a traffic sign recognizing algorithm with more than 80% accuracy and a recognition time of fewer than 1 second per traffic sign.
- To develop a desktop-based traffic sign recognizer that is able to recognize traffic signs from still images.
- To develop a desktop-based traffic sign recognizer that is able to recognize traffic signs from file-based video input and webcam input.
- To develop a mobile-based traffic sign recognizing application that is able to recognize traffic signs from gallery-based image input and real-time video streams with voice outputs.

1.4 Proposed Approach/ Study

[Done By: Por Teong Dean]

Traffic sign recognisers are designed mainly to be implanted into vehicle safety systems in order to identify traffic signs on the road in real-time. These recognisers are critical in enhancing road safety amongst drivers by reminding them constantly about upcoming road signs and signalling warning messages when appropriate. A

combination of classification and image processing techniques are utilised in the proposed project to develop a novel traffic sign recogniser that is able to work on still images as well as real-time video streams. The proposed project is programmed using the C++ programming language with the use of the Open-Source Computer Vision Library, OpenCV, for the classification pipeline. The proposed project aims to recognize a minimum of 10 types of traffic signs varying in shapes and colours in both still images and video streams.

1.5 Highlight of Achievement

[Done By: Tan Wei Mun]

With the advance of the times and the development of science and technology, the popularity of vehicles has been enlarged to a certain extent that almost all the family in Malaysia has a car or motorbike as their transportation. Public transportation facilities like light rail transit (LRT) had yet to cover Malaysia, so moving from place to place has become difficult for the regions without LRT. Hence, owning a vehicle earliest after high school graduation has become a new norm today. However, it is harsh to ask the newbie drivers that have just passed the driving test to recognize all the traffic signs on the road. The project delivered a traffic sign recognition system to overcome the problems above that helps users recognize the traffic sign automatically.

This project adopts excellent image processing techniques to recognize traffic signs from images captured in different lighting conditions like overexposed, underexposed images and terrible weather like heavy rain. At the same time, the feature extractions are applied to improve further the accuracy of both laptops and mobile applications to above 80% accuracy. Other than computer applications, the traffic sign recognition model is realized on the mobile application, improving the usability of the recognition technology to real-life applications. The mobile application can utilize the phone camera to capture the input frame, identify and recognize traffic signs using a phone's camera, and notify users of any traffic signs observed in any weather situation. Furthermore, the implementation of the mobile application platform increased the mobility and accessibility of the recognition system, allowing users to utilize it more flexibly. The application can continuously detect the traffic sign on the road through the phone camera and notify users of the detection output of traffic signs. The

implemented method is critical, especially in bad weather, when drivers are less sensitive and can scarcely identify traffic signs in a low-visibility environment when they are focused on monitoring traffic conditions instead. By having this portable traffic sign recognition system, the drivers can have a lesser burden on driving and save their energy on other important tasks.

1.6 Report Organisation

[Done By: Jacynth Tham Ming Quan, Tan Jing Jie, Ng Jan Hui, Tan Wei Mun, Por Teong Dean]

This report consists of 10 chapters. The first chapter describes the background of the project, including the problem statement, motivation, and project objectives. Next, the literature review and comparison of existing traffic sign recognizers were reported. In chapter 3, the system design of the developed system was illustrated. Moving on, in chapter 4, the image pre-processing stage is described and tested. In chapter 5, the steps of the feature extraction stage are depicted. Next, in chapter 6, the training and testing processes of the classification models are explained. In chapter 7, the implementations of the image-based and video-based desktop applications are described. In chapter 8, the implementations of the image-based and video-based recognition in the mobile application are explained. In chapter 9, the experimental result of this project is verified and explained. Lastly, the project is summarized in chapter 10's conclusion.

Chapter 2

Literature Review

2.1 Overview

[Done by Tan Jing Jie]

In this chapter, 14 literature reviews had been done and a conclusion and future work had been made at the end of this chapter. The main classifier reviewed are Extreme Learning Machine (ELM), Support Vector Machine (SVM), Convolutional Neural Network (CNN), Extreme Learning Machine (ELM), Generative Adversarial Network (GAN), Random Forest Classifier, and Fourier Descriptors. Additional enhancement had been made on top of the main classifier and the advantages and limitation were summarized.

2.2 Traffic Sign Recognition Using Support Vector Machine (SVM)

[Done By: Jacynth Tham Ming Quan]

In this subsection, three existing traffic sign recognition systems are reviewed. All three of the systems used Support Vector Machine (SVM) as their main classifier.

2.2.1 Real-Time Detection and Recognition of Road Traffic Signs

[Done by: Tan Jing Jie]

Greenhalgh and Mirmehdi (2012) propose a system for automatic detection and recognition of traffic signs. This proposed system is able to detect the candidate region as maximally stable extremal regions (MSERs), which are able to work well in different lighting conditions. For the recognition algorithm, this paper proposed a cascade of support vector machine (SVM) classifiers that trained using Histogram of Oriented Gradients (HOG) features. This paper shed light on the training data that can be generated from synthetic template images that are freely available online. The overall approach can be illustrated as the figure below.

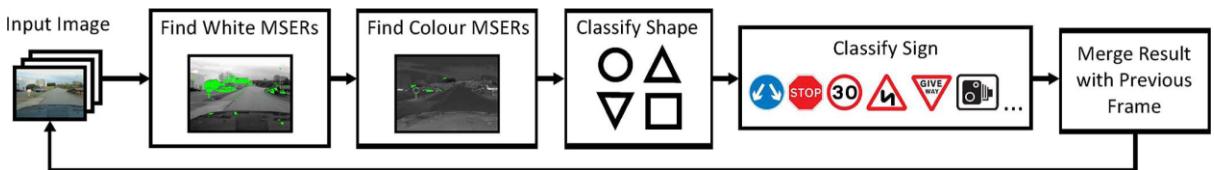


Figure 2.1.2.1 The proposed schematic of the approach. (Greenhalgh & Mirmehdi, 2012)

This system started with the detection of road signs as MSERs. Each frame was first binarized at a several number of different threshold levels. When the connected components are able to maintain their shape for some threshold levels are selected as MSERs. This aid in the event when there is a lighting issue. In addition, several features of the connected component including the width, height, and ratio were used to further reduce the number of candidates. Furthermore, other than transforming the red-green-blue (RGB) image into grey style image, the image was also transformed into a normalized red and blue image. However due to improving the processing speed, the number of thresholds selected was limited to 24 and the values were evenly spaced between 70 to 190 as the road signs are generally appear in this range.

Moving on to traffic sign recognition. First and foremost, the HOG features which represent the gradient orientation are extracted from the image. Although SVM is a binary classifier, a cascade of multiclass SVMs can achieve the multiclass classification by training many one-against-one binary SVMs. Indeed, obtaining enough real data is difficult and is time consuming as there are many possible variations (environment) for one of the traffic signs. The proposed solution was to synthetically generate variation and distortion of available dataset as training data for the classifier. There are 131 road signs used for training classification purpose and 1200 synthetically distorted images are generated for each road sign

As shown in the figure below, a classifier tree was constructed based on the set of type of traffic signs in the United Kingdom. This tree can adjust based on the individual country traffic sign. The images was first duplicated into colour MSER regions and white (grey style) MSER region. It is followed by shape classifier and more detailed classifier such as white circle classifier, colour triangle classifier and et cetera.

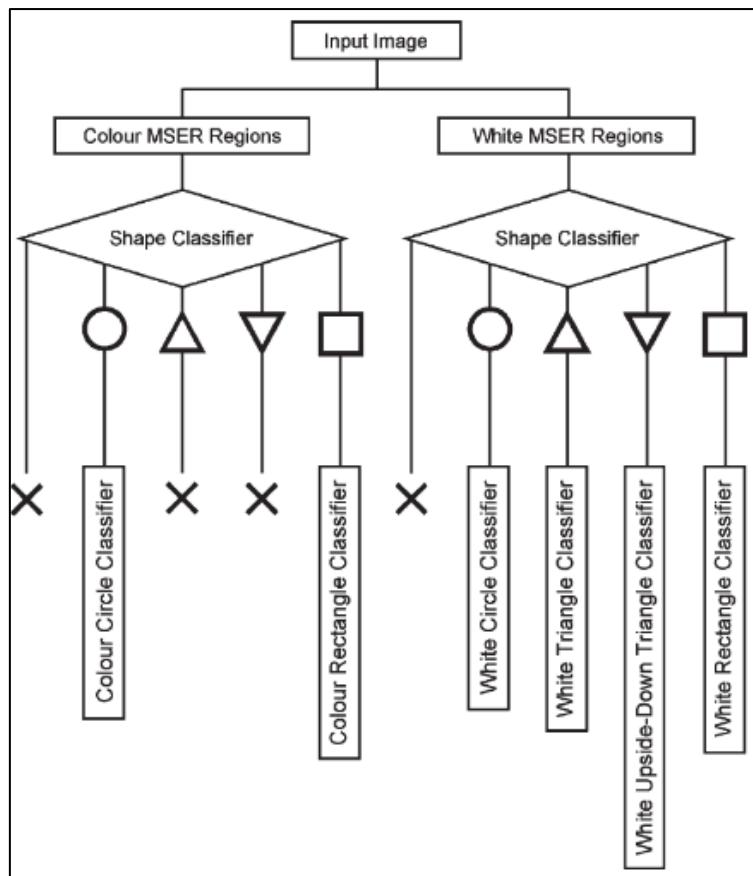


Figure 2.1.2.2 The illustration of cascaded SVM classifiers. (Greenhalgh & Mirmehdi, 2012)

This paper uses a test data set which included many challenging images affected by blurring, geometric distortion etc. The dataset was collected from frames of road video footage and road sign images from the internet. The proposed model achieves the accuracy of 89.2% and 92.1% for white and colour road signs respectively. On the other hand, the model is also evaluated using the data from the GTSRB test set, which was also used for synthetically generated train sets before. To ensure the model does not relate any background information, the background was modified, the accuracy achieved is 97.6%.

In summation, the novelties of this paper are classifying traffic sign images from images with various lighting conditions and maintain high accuracy in different vehicle speeds which are suitable for real-life implementation. Besides, in order to fully utilise the information from the dataset, 1200 synthetic distorted traffic sign images are generated for each road sign. All in all, the proposed system of this paper in real time still has space to do improvement in terms of classifier type and the tuning of models.

Chapter 2 Literature Review

The classifier tree can be improved by adding more branches (classes) for different characteristics such as colour and text of traffic signs.

2.2.2 An Incremental Framework for Video-Based Traffic Sign Detection, Tracking, and Recognition

[Done by: Tan Jing Jie]

Yuan, et al. (2017) have designed an incremental framework for Traffic Sign Recognition (TSR) system including detection, tracking and recognition in real world applications. The 3 major highlights of this paper include, utilizing the contextual information to enhance the detection performance, using incremental framework designed for detection, and tracking simultaneously and using scale-based intra-frame fusion method to get stable classification output. The overview of the proposed framework illustrated as the figure below.

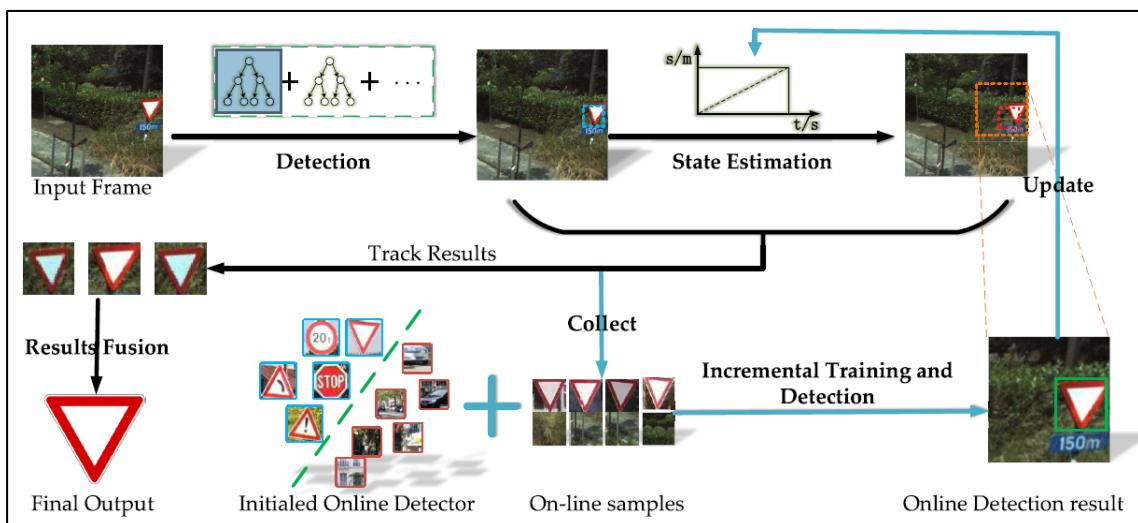


Figure 2.1.3.1 Component of the proposed TSR framework (2017)

First and foremost, offline detection was applied to detect the candidate traffic signs. By using the information of colour and shape, this paper develops a detector based on Aggregated Channel Feature (ACF) which uses a cascade of boosted weak tree classifiers. This detector is trained by gradient magnitude, HOG and LUV colour channels. After that, this system integrates with previous frame results by Kalman filter (offline motion model which is used to track the traffic sign). By this strategy, the online detection search takes advantage of not conducting detection for the whole image and this is very efficient for real-time application. In the end, it computes the Kalman gain and updates the state estimate. This can maximize the bounding box location more accurately by merging on-line strategy. Moreover, this system makes use of spatial distribution of traffic signs in images as this is a strong prior knowledge that the height

of a vehicle has only small variations. The overall illustration was shown as the figure below.

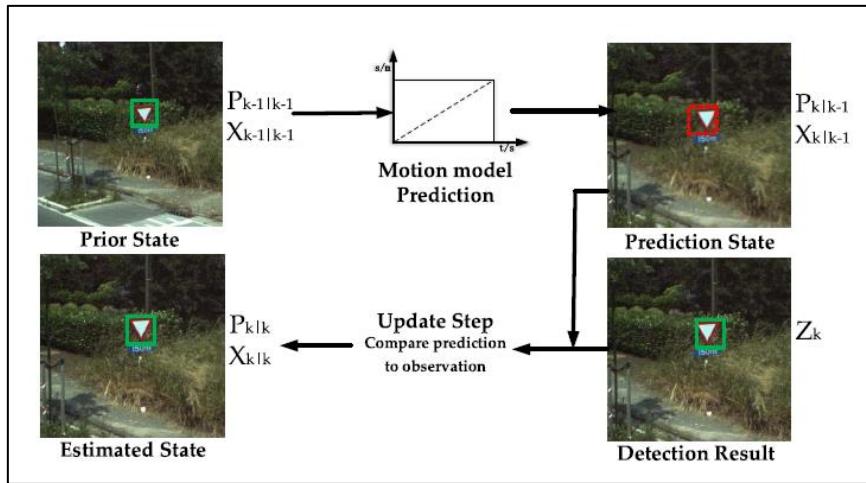


Figure 2.1.3.2 Component of the proposed TSR framework (Yuan, et al., 2017)

In this paper, the authors use Incremental Support Vector Machine (SVM) for training purposes. This paper is not focusing on classifier performance, but more study on the fusion strategy. It fused multiple frame results that belong to the same traffic sign to get better precision. As larger scale traffic signs usually contain richer information and lead to more high accuracy, it adopts a Gaussian-based weighting to fuse the classification together.

To examine the framework, three evaluation measures are applied for different stages of the system pipeline. Precision-recall measures are adopted for detection performance. The detection rate and false positives per frame are 98.29% and 0.00528 respectively, which is ultimately good and suitable to use in real-time. The use of Kalman filter and online strategy is able to reduce the localization error from 0.1972 to 0.1175. Moreover, using fusion techniques is significant to increase the accuracy result. For one of the instances, when there are 7 fusion frame numbers, the classification increases from 0.975 to 0.99.

The novelty of this framework is that it combines a lot of techniques in order to make this system work in real-time. This included using spatial distribution of traffic sign images and using Kalman filter (offline) to track the traffic sign and combine the result from the appearance model (online). This successfully increased the efficiency of detection due to the reducing of the scope to detect. They also use fusion techniques which combine several frame results to ensure the accuracy. These fusion techniques

not only use “mode” as strategy but using Gaussian-based weighting which gives more weightage to those frames which have higher chance to get correct classification.

However, the proposed TSR system had limitations. As this TSR system uses prior information for detection purposes, this limits the detection of some real traffic signs such as when a vehicle is turning around or drawing downhill. This can further improve by adding other machine learning models to improve or obtain related data from the gyro meter, speedometer, and et cetera.

The future work to resolve the limitation and improve the system is that using richer features for traffic sign detection is significant. The obvious information like the shape, text is always a good characteristic to get a more satisfying result in machine learning. For instance, the variety of traffic signs is limited, a red octagon always is a “STOP” sign. Besides, the saliency information of traffic signs such as edges and ratio can also be explored. All this prior knowledge is significant and valued to explore more to produce a more accurate model.

2.2.3 On Circular Traffic Sign Detection and Recognition

[Done By: Jacynth Tham Ming Quan]

In 2015, Berkaya et al. proposed novel approaches for a traffic sign detector and recognizer on coloured images (Berkaya et al, 2015). In their paper, Berkaya et al. utilized a circle detection algorithm and an RGB-based colour thresholding technique for the traffic sign detection phase. For the recognition phase, the paper proposed the use of an ensemble of features, including local binary patterns, histogram of oriented gradients (HOG) and Gabor features, coupled with a Support Vector Machine (SVM) classification framework. Berkaya et al. tested chose the German Traffic Sign Detection and Recognition Benchmark datasets as the benchmark of their approach. Figure 2.2.3.1 below illustrates the overview of Berkaya's traffic sign recogniser.

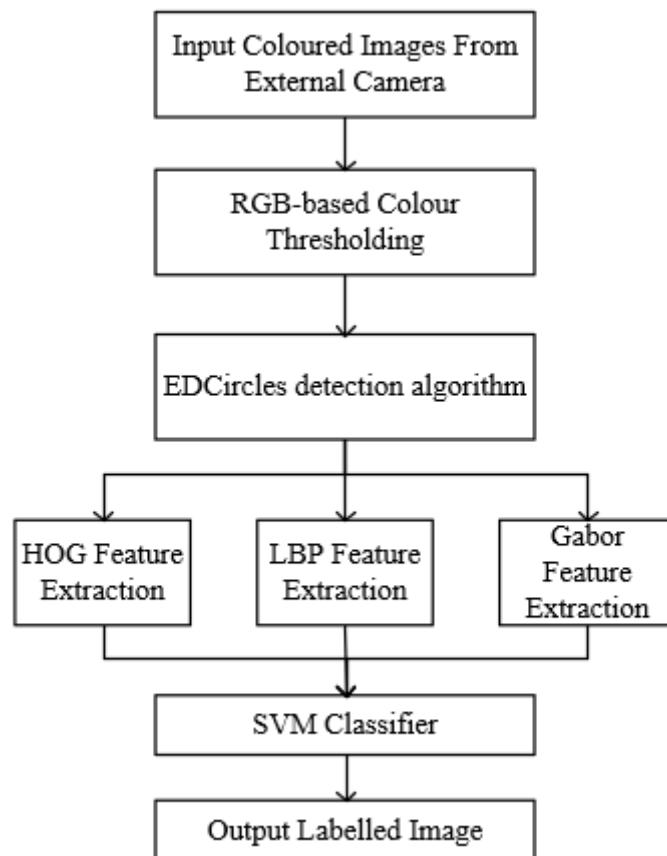


Figure 2.2.3.1 Berkaya et al.'s Approach for Traffic Sign Recognition

In the detection phase, the parameter-free EDCircles algorithm was implemented. EDCircles used greyscale images as inputs and focused on edge detection. As the inputs of the proposed paper were coloured images, colour

thresholding techniques had to be performed first before the images could be fed into the EDCircles algorithm. Berkaya et al. selected the Normalized RGB colour space to decrease the effects of lightings and to accurately find the thresholds needed. After colour thresholding was performed on the input image, a binary image output was fed into the EDCircles detector for edge detection. The edge detection was performed in 5 steps, from the detection of edge segments, to extracting and combining the circular arcs to form candidate circles and finally, outputting a set of perceptually valid circles. Once the possible traffic signs were detected, the recognition phase was initiated.

For the recognition phase, Berkaya et al. proposed an ensemble of HOG, LBP and Gabor features within a SVM classification framework. First, the HOG method was used to express the image with the detected traffic signs as a group of local histograms which consisted of gradient orientations and magnitudes. Next, the LBP method was used to compute the feature vector for the very same input image. The LBP method divided the input image into several sub-regions, starting at 2X2 to 4X4, and computed the LBP histogram for each region. Figure 2.2.3.2 below shows the sub-region division used by Berkaya et al.

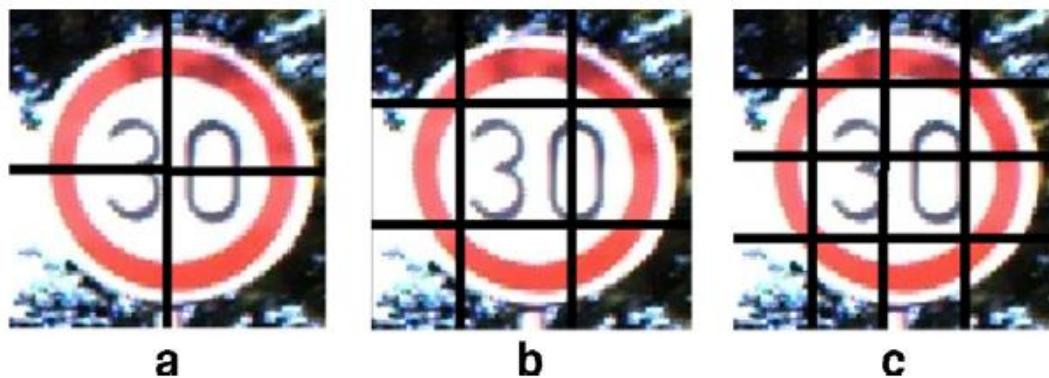


Figure 2.2.3.2 Local Binary Pattern Sub-Region Divisions

Finally, the Gabor filter feature extraction technique was used to obtain a feature vector. The Gabor filter utilized a Gabor filter bank with different angles and frequencies to achieve the results. In Berkaya et al's paper, the Gabor filter bank was used with the following parameters – 18 filters, 6 angles, 30-degree increments and 3 octave intervals.

Once all the three different feature extraction techniques had been performed on the detected traffic sign, the outputs were combined into the SVM classification

framework. Each of the feature vectors were normalized to values between 0 and 1 and appended to each other. When tested on the German Traffic Sign Detection and Recognition Benchmark, Berkaya et al.'s proposed traffic sign recognizer achieved an accuracy of 97.04% for all signs and a feature extraction time of 3.60 milliseconds per 40X40 image.

In short, the traffic sign recogniser developed by Berkaya et al. was able to detect traffic signs using a combination of the EDCircles circular detector algorithm and the RGB-based colour thresholding technique. The traffic sign recognizer was then able to further classify the signs using a combination of three feature extraction techniques in a SVM classification framework. The novelty of Berkaya et al's traffic sign recognizer was the use of a parameter-free detection algorithm. This gave the proposed recognizer an advantage as no training images were required to fine-tune the algorithm's parameters. Furthermore, the use of three feature extraction techniques simultaneously effectively decreased the chances of false positives in the classification phase. The future works of this paper proposed a deeper analysis of the combination of different circle detection algorithms, colour models, feature sets and classifiers.

2.3 Convolutional Neural Network (CNN)

[Done By: Jacynth Tham Ming Quan]

In this subsection, six existing traffic sign recognition systems are reviewed. All six of the systems used Convolutional Neural Network (CNN) as their main classifier.

2.3.1 Traffic-Sign Detection and Classification in the Wild

[Done By: Jacynth Tham Ming Quan]

In 2016, Zhu et al. developed a computer-based traffic sign recogniser using a Convolutional Neural Network (CNN) model (Zhu, et al., 2016). The main logic behind the classification of Zhu et al.'s system was that the traffic signs were able to be divided into different categories according to function – warning, prohibitory and mandatory – and each category were to be further divided into subclasses according to the signs' generic shapes, colour, and appearance. Pertaining to this logic, Zhu et al. came up with a two-phase traffic sign recogniser (detection phase and the classification phase).

In their paper, Zhu et al. created their own benchmark named Tsinghua-Tencent 100K to train and test their traffic sign recogniser. Their benchmark consisted of 100000 Tencent Street View panoramas, 30000 of which contain traffic-sign instances. According to Zhu et al., the absence of traffic signs in part of the images in the training dataset would ensure that the system was able to distinguish real traffic signs from other similar real-world objects.

In Zhu et al.'s project, two CNNs were trained – one for multi-class detection and another for simultaneous detection and classification. The two CNNs were similar and only differed in terms of the branches in the last layer. The architecture of the neural network used by Zhu et al. was based on the Caffe learning framework and OverFeat framework with slight modifications and an added bounding box regression step. The CNN was fully convolutional, with only the last layer branched into three streams – a bounding box layer, a label layer and a pixel layer. The outputs of the pixel layer represented the probabilities of a 4 X 4-pixel region containing a traffic sign. The output of the label layer was a classification vector with n elements, where each element represented the probability that the input belongs to a specific class of traffic signs. As for the bounding box layer, the outputs represented the distance between the 4 X 4-

pixel region mentioned earlier and the four sides of the predicted bounding box of the target object. Figure 2.3.1.1 below illustrates the architecture of Zhu et al's multi-class CNN model.

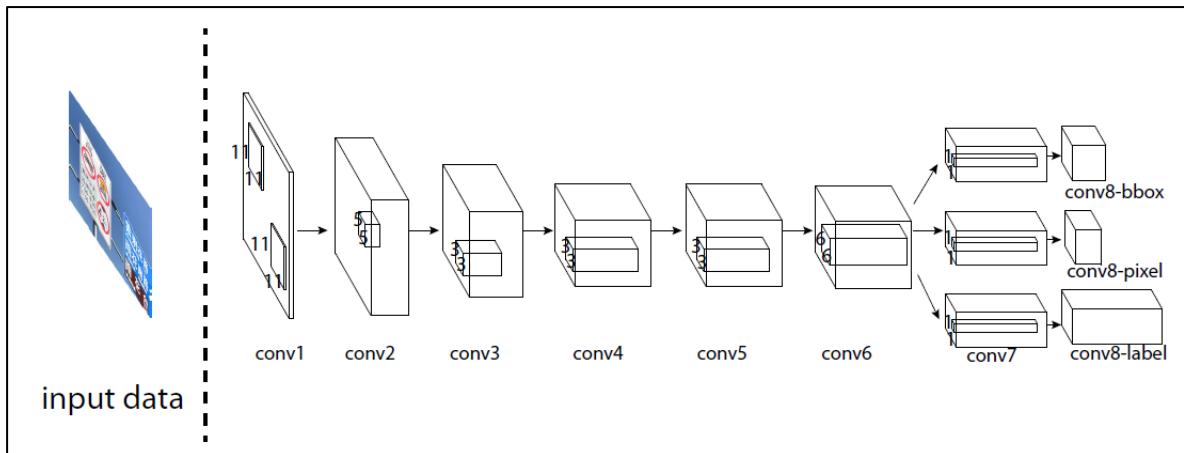


Figure 2.3.1.1 Architecture of Zhu et al's Multi-Class CNN Model

According to Zhu et al., deeper networks yield better performance and increased capabilities. However, branching too early in the network would lead to an increased training time and GPU consumptions. Therefore, to achieve a balance between speed and performance, Zhu et al. made their system's neural network starting branch at the sixth layer, instead of the seventh layer as in the original OverFeat framework.

There were two versions of the trained CNN model in the figure above. The complete network was able to detect and recognise signs simultaneously. When only a traffic sign detector was needed later on in the implementation, the label layer in the last layer was removed to omit the model's classification capabilities. Figure 2.3.1.2 below summarizes the three primary steps of the annotation pipeline – locating the bounding box, marking the contours and finally, attaching the class label.

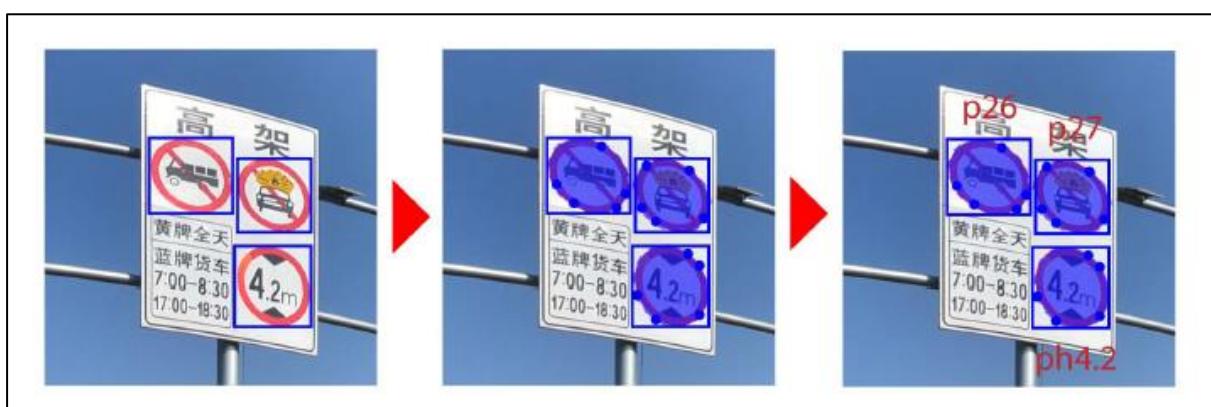


Figure 2.3.1.2 Summary of Annotation Pipeline

In the model training process, Zhu et al. proposed the use of data augmentation techniques. To augment the data, the standard templates of the traffic signs were rotated and scaled randomly to create a reasonable amount of perspective distortion. Their detection network achieved an 84% accuracy and 94% recall at a Jaccard similar coefficient of 0.5 when tested on images with smaller traffic sign regions. The network was also tested on 90000 images that had no traffic signs, and the detector was able to detect the absence of traffic signs with a 100% accuracy. Finally, the detection and classification process were merged so that the system could perform both simultaneously. When tested on a test set of 10000 images, the network resulted in an accuracy of 88% and a recall of 91%.

All in all, the traffic sign recogniser developed by Zhu et al. was able to detect and recognise traffic signs in still images efficiently. The project's novelty was that the trained CNN models were able to classify traffic signs from images that covered large variations in illuminance and weather conditions and pictures in which the traffic sign only took up a small fraction of the whole image. The future works of this paper proposed the inclusion of more different types of signs into the benchmark as well as the acceleration of the entire traffic sign recognition process in order to implement it into real-time smart driving systems.

2.3.2 Traffic Sign Detection and Recognition Using Fully Convolutional Network Guided Proposals

[Done By: Tan Wei Mun]

In 2016, Zhu et al. developed a novel framework based on object proposals for traffic sign detection and identification that operates in a coarse-to-fine way (Zhu, et al., 2016). They primarily decreased the sign-seeking area by using FCN to create guided heat maps of traffic signs. Traffic sign suggestions are retrieved under the guidance of a FCN, and traffic signs always appear on both sides of the road in their work. This technique effectively decreased the number of recommendations and improved efficiency by limiting the search range of traffic signs. As indicated in Figure 2.3.2.1, the suggested technique is divided into two stages: a proposal extraction stage guided by FCN and EdgeBox; a traffic sign classification stage utilizing a CNN model. Unlike the popular R-CNN object detection proposal, the suggested technique uses a quicker EdgeBox and FCN guided object proposal that is fast and accurate. FCN first generates coarse regions of traffic signs from the input picture, after which EdgeBox extracts the proposals. Finally, a CNN classifier is trained using a bootstrapping strategy to classify traffic signs, and false positives are removed. At the same time, the optimal bounding boxes are retained by non-maximal suppression.

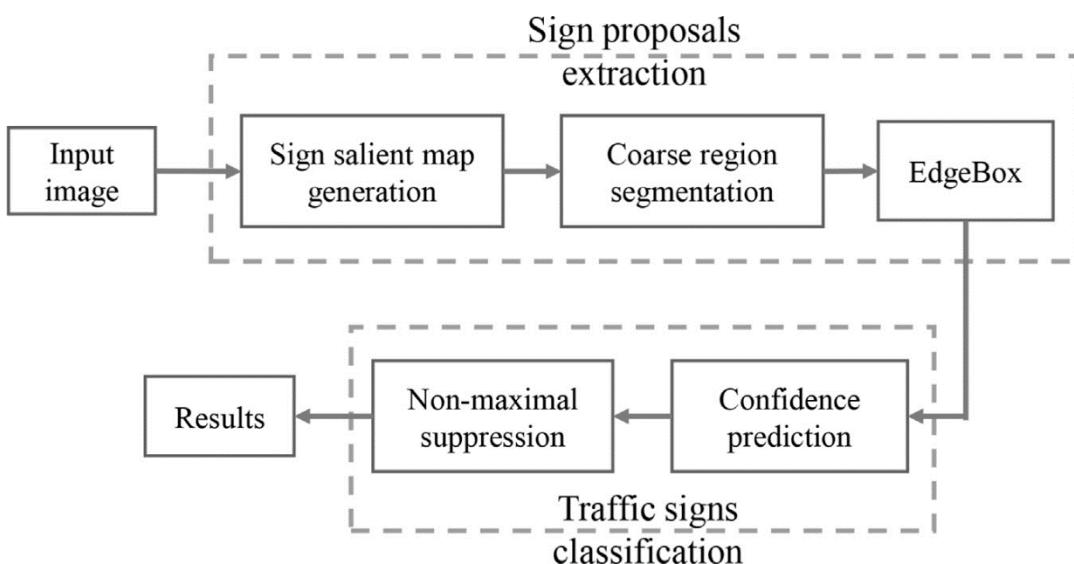


Figure 2.3.2.1 The proposed method's pipework

In the proposal extraction, object proposal techniques give fewer candidate bounding boxes with high recall than traditional object detection algorithms. A FCN is utilized to discover coarse sign regions instead of extracting object proposals from the

whole image, drastically reducing the search range for traffic signs. FCN is trained using bounding box level supervision to identify regions of interest. Although this approach is inadequate and noisy, it is suited for coarse object detection since the bounding box encompasses non-object regions. Besides, the holistically-nested edge detection network was adopted to generate coarse sign regions. As shown in Figure 2.3.2.2, a deconvolutional layer fuses various feature maps from each convolutional stage to produce rich and hierarchical features, resulting in feature maps of the same size. The fused features are then input into a 1 X 1 convolutional layer. When training, pixels inside the ground truth bounding boxes are considered positive, while those outside pixels are considered negative.

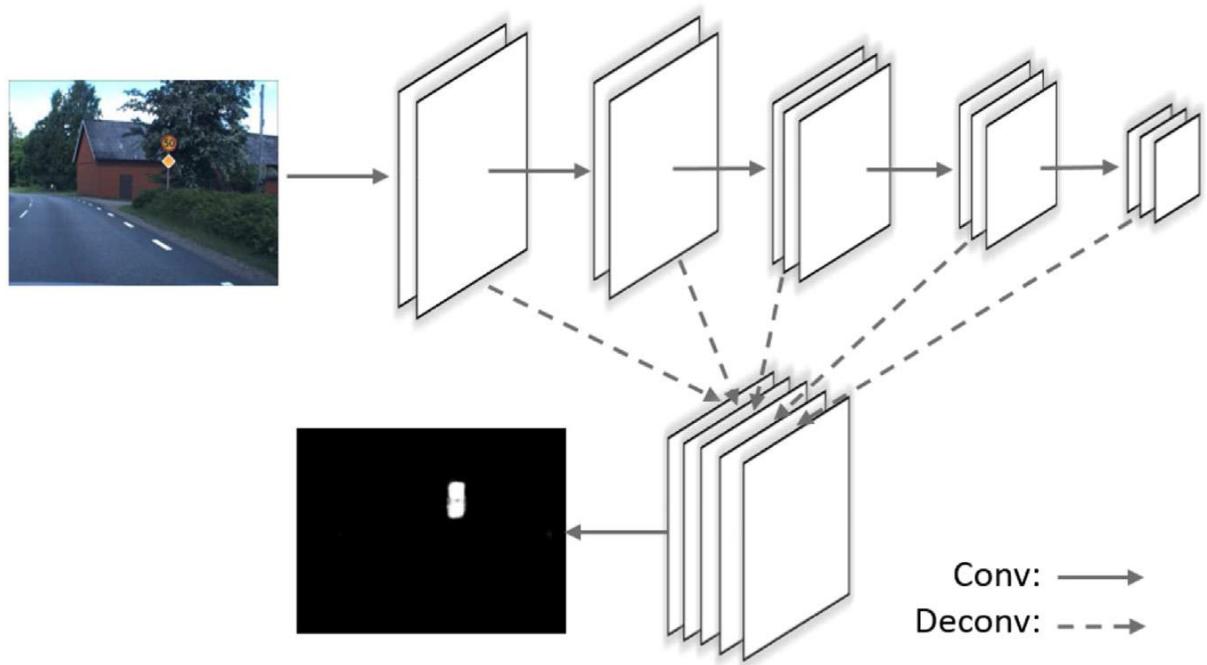


Figure 2.3.2.2 Architecture of FCN

Next, Zhu et al. used EdgeBox in traffic sign proposals to produce candidate bounding boxes with good recall in a short amount of time. As the proposals being extracted using coarse sign regions, the recall of traffic sign proposals is readily achieved with minimal proposals compared to the original EdgeBox. The proposed FCN guided object proposal increases object detection speed and accuracy by eliminating most background areas and decreasing false positives. The use of FCN in this study, which extracts coarse sign areas, substantially decreased the number of EdgeBox proposals while maintaining the detection rate. In brief, the FCN and Edgebox combination reflects the detection rate by varying the maximum number of EdgeBox boxes from each coarse sign region retrieved by FCN. A significant reduction in the

number of proposals lowered the number of negative samples for training, providing a balance of positive and negative data and increasing CNN classification accuracy.

Deep CNN is utilized by Zhu et al. to perform the task of traffic sign classification. A batch normalization layer, a ReLU layer, and a max-pooling layer with two strides follow each convolutional layer in the network architecture. It significantly improves convergence time. The suggested CNN model is small, and the classification task is more accessible due to enhanced object localization accuracy and noise background removal by the FCN guided object proposal. Therefore, the small network has a quick detection time instead. Furthermore, the bootstrapping method was adopted to improve CNN discriminative ability and mining hard negative for training CNN. After the convergence of network training, the wrongly classified samples were added to the training set to optimize the network. Zhu et al. also used non-maximum suppression to remain the local optimal bounding boxes of traffic signs as the final output of the proposed traffic sign detection system.

Zhu et al. had evaluated their design on Swedish Traffic Signs Dataset (STSD) and have obtained an average precision and recall of 98.67% and 93.27%, which outperformed the popular R-CNN method by 5%. Lastly, future work improvements might include building a real-time traffic sign recognition system based on the proposed algorithms and implementing an end-to-end network, creating the suggested FCN guided recommendations.

2.3.3 Traffic Sign Recognition with Hinge Loss Trained Convolutional Neural Networks

[Done By: Tan Wei Mun]

Jin et al. proposed a hinge loss stochastic gradient descent (SGD) technique to train CNNs to extract features and categorize traffic signs simultaneously in 2014 (Jin, et al., 2014). In traffic sign classification, the technique achieves excellent accuracy rates and outperforms humans.

As inspired by previous works, Jin et al. have designed a CNN with three stages after going through series of cross-validation about the layer size, full-link layer hidden neurons, convolution layer kernel size, max-pooling layer kernel size, normalization layer kernel size, and normalization layer parameters. As illustrated in Figure 2.3.3.1, there are three phases in which each with four layers: convolutional layer, max-pooling

layer, non-linear layer, and normalization layer. Then comes one stage of a full-link layer and a non-linear layer, followed by one stage of a full-link layer and a soft max layer. Convolution layers and full-link layers, which learn feature detectors and classifiers, respectively, have a total of 1162284 parameters to learn.

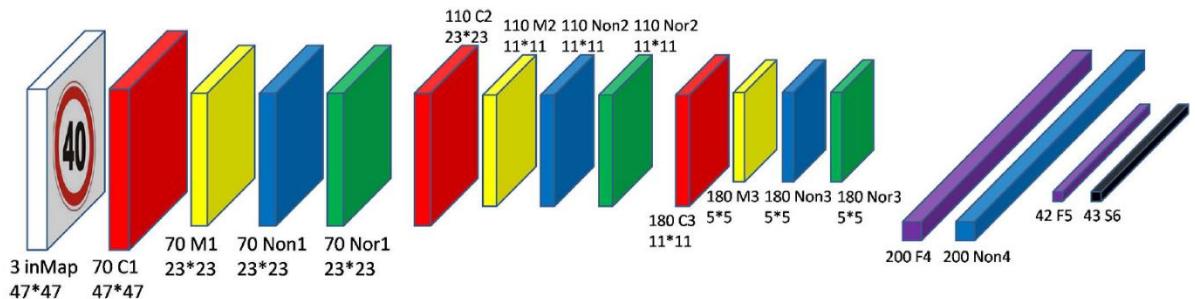


Figure 2.3.3.1 Proposed CNNs Architecture by Jin et al.

Through experiments, Jin et al. found that most output probabilities for a CNN model trained with cross-entropy will be incredibly near to 1.0, but many training samples will be misclassified, wasting model capacity. Therefore, they proposed a support vector machines hinge loss alike cost function. The proposed cost function optimized the model capacity, which focused on the misclassified training data to recognize them correctly instead of recognizing accurate classification more accurately like CNN trained by cross-entropy cost, improving the model generalization. Besides, the proposed method also shortened the training time required. Usually, SGD iterations spend a long time on forward and backward propagation no matter the training sample is helpful or not. However, hinge loss SGD iterations could be stopped when the training sample is helpless after the forward propagation, significantly reducing the time cost.

Jin et al. tested their model with the cross-entropy technique using the German Traffic Sign Recognition Benchmark (GTSRB) dataset, which included a bounding box annotation for traffic signs detected on the pictures. They used the bounding box to scale the pictures to square images as the input to the CNNs model and found that the hinge loss approach produced a validation accuracy of 98.89 percent, which is greater than cross-entropy. The cross-entropy drives all the examples mentioned above, and many parameters update operations are close to each other, which wrongly predicted samples are seldom learned. On the contrary, the hinge loss is learned from wrongly predicted samples and continuously improving its accuracy until final convergence. Besides, they have also tested their algorithm with different image pre-processing

methods on the inputs and obtained better accuracy of 99.65% and a lower error rate of 35.19% compared with the winner of the IJCNN 2011 recognition competition that obtained the best results on the GTSRB dataset. Apart from supplying the benchmark dataset, the Institut für Neuroinformatik Real-Time Computer Vision research group also organized many people skilled at identifying traffic signs to deliver a performance on the dataset to compare computer and human abilities. The human performance is 98.84%, which is lower than Jin et al.'s proposed algorithm.

Nevertheless, Jin et al. pointed out that pictures with low resolution, intense sunshine, and improper bounding box annotations are the primary causes of incorrectly predicted traffic signs. That is where Jin et al.'s approach falls short, and they also highlighted that the error rate might be further improved by focusing the algorithm on precision, which could be a future enhancement on this technique. Furthermore, since they utilized scaled pictures with a single traffic sign to train the CNN model, Jin et al.'s system may have trouble identifying numerous traffic signals in a single input. Another project in the future may be to replace the training data with pictures of several traffic signs and see how this affects the algorithm's accuracy.

2.3.4 A Committee of Neural Networks for Traffic Sign Classification

[Done by Por Teong Dean]

A fast and fully parameterisable GPU implementation of a CNN with a high recognition rate of the traffic sign was developed by (Cires, et al., 2011). It learnt in a supervised way instead of requiring careful design of pre-wired extractors and in the preliminary phase of the German traffic sign recognition benchmark, it obtain a better-than-human recognition rate of 98.98%.

In (Cires, et al., 2011)'s paper, for having the highest successful recognition rate, the localised features will be extracted by the hierarchical visual object recognition systems from the input images. Then, by repeating sub-sampled and re-filtered the responded convolving image patches during filtration, it will have architecture of deep-forward network with their eventually classified output feature.

According to (Cires, et al., 2011), novelty of their development is they developed a fast GPUbased CNN with high flexibility and fully online such as weight updates after each images instead of using hard coding for satisfying the GPU hardware constraints like the others previous GPUbased CNN implementation. Cires, an al.'s GPUbased CNN will allowed for training a lot of CNNs within days instead of months, so it can check the influence of difference structural parameters and performing error analysis on repeated experiments.

Based on (Cires, et al., 2011)'s description, alternation of a convolutional networks with subsampling layers reminiscent of complex and simple cells in the primary visual cortex CNNs hierarchical networks. The factors that cause the CNNs to be different from each other is how the CNNs were trained and realisation of convolutional and subsampling layers. Figure 2.3.4.1 shown the main building blocks description by Cires,an et al.

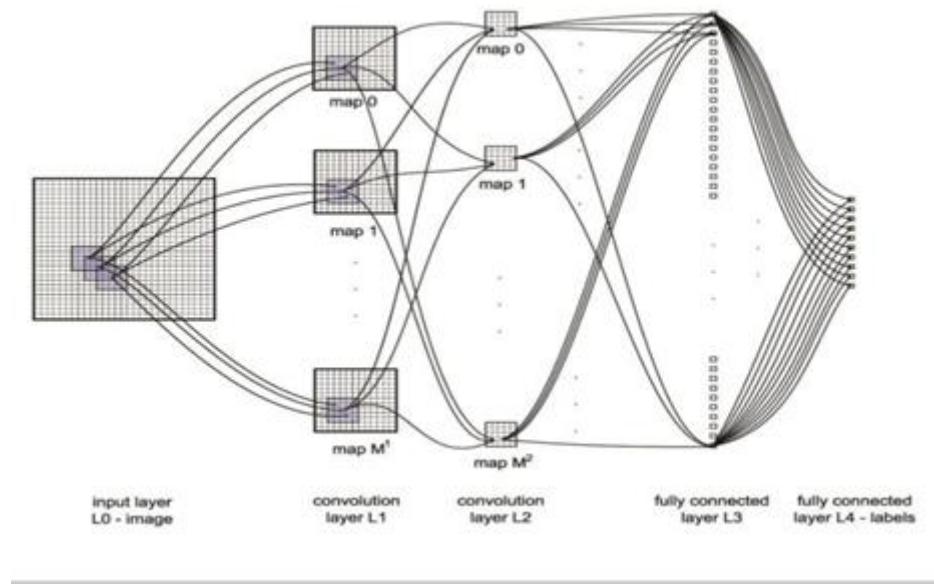


Figure 2.3.4.1 Architecture of a convolutional neural network

Here the convolutional layers are fully connected. Both convolutional layers are using a 5×5 kernel and skipping factors of 1. Through the experiment, the best of their CNNs achieves a 98.73% recognition rate in the German traffic sign recognition benchmark. Further improvements can be done by using a committee of CNNs trained on raw pixel intensities and the multilayer perceptron (MLP) trained on descriptors offering complementary information, the recognition rates could be able to increase to 99.15%.

However, in the experiment, a lot of traffic signs with “no vehicles” were falsely classified by both CNN and the committee. It also shows that nearly all those traffic signs were confused with the “no overtaking sign” by closer inspection of the output activation of the CNNs. This is because of the CLAHE introduces artifacts which will makes these two classes difficult to classify.

2.3.5 Towards Real-Time Traffic Sign Detection and Classification

[Done by Por Teong Dean]

(Yang, et al., 2015) developed a fast-processing time and extremely fast detection's real-time traffic sign recognition module. The main idea in Yang et al.'s is to build their detection module with it's based on traffic sign proposal extraction and classification built upon a colour probability model and a colour HOG in detection phase. Then after detecting the traffic sign by using CNN, the traffic sign will be further classified into subclasses in between each superclass according to their appearance during the classification phase.

In Yang et al.'s, 2015 paper, their main contribution work can be summarising into the according step. At first, for the dealing with colour information of traffic signs, they propose a colour probability model. This can minimise the detection time and the following algorithm's search space as well as enhancing the traffic signs' s specific colors and suppress its background colour. Second, instead of sliding windows detection, they propose to extract traffic signs proposals and combine some machine learning algorithms such as CNN and SVM for detection and classification of the traffic signs. Third, a Chinese Traffic Sign Dataset (CTSD) also construct by them to evaluate their method on Chinese roads.

In the traffic sign recognition module of Yang et al, it contains two main important modules which is the detection module and classification module. According to Yang et al., they first change the input colour image into traffic sign probability maps by using colour probability model (CPM). Then in the probability maps, they find the Maximally Stable Extremal Regions (MSERs) to extract traffic sign proposals. Thirdly, they filtered out the false positives and the remaining traffic sign proposals will be classified into their respective super class by using novel colour HOG based Support Vector Machine (SVM). By employing CNN in their classification module, they further classify the detected traffic signs into their specific sub-classes.

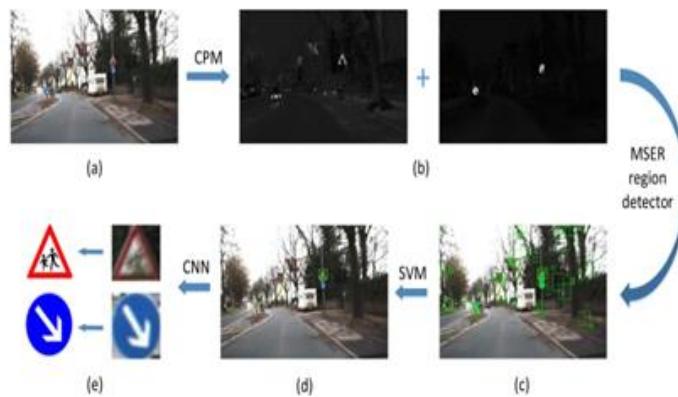


Figure 2.3.5.1 The pipeline of the proposed traffic sign recognition system. (a) Original image. (b) Probability maps (red + blue). (c) Proposals. (d) Detection result. (e) Classification.

According to (Yang, et al., 2015)'s project, they use two traffic sign dataset, German Traffic Sign Detection Benchmark (GTSDB) and Chinese Traffic Sign Dataset (CSTD). Then, the detected traffic signs will be needed to classify into three sub-classes (Prohibitory, Danger, Mandatory) respectively as shown in Figure 5.2 and 5.3 or background class in their detection module due to there were be some false alarms was detected. Three CNNs were trained for the three sub-classes respectively.

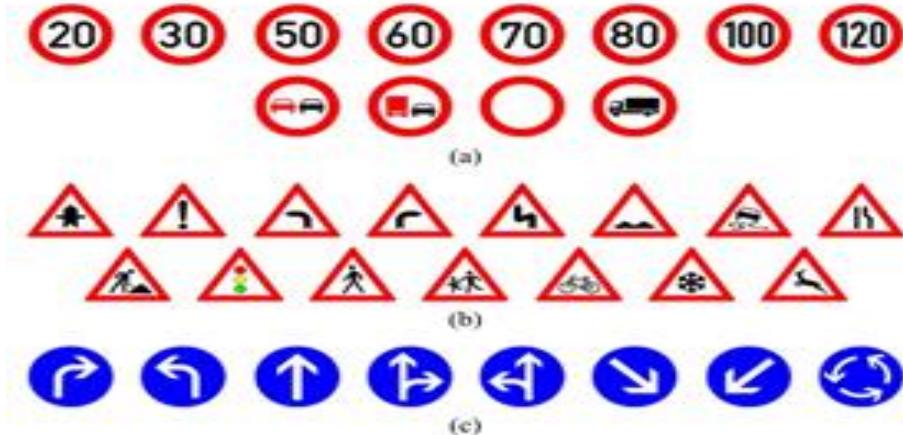


Figure 2.3.5.2 The examples of sub-classes of GTSDB. (a) Prohibitory. (b) Danger (c) Mandatory.



Figure 2.3.5.3 The examples of sub-classes of CTSD. (a) Prohibitory. (b) Danger. (c) Mandatory.

For minimising the processing time, they only use gray images in their experiment due to colour supplies little distinctive information for classification. Also, all the images were resized to 32x32 due to the input of CNN should not have different size. Additionally, for adjusting the images contrast due to some images might caught under different lighting conditions, contrast histogram equalisation (CLAHE) was also use in the experiment.

For the GTSDB dataset, 340 traffic signs were detected in their detection module which including 62 Danger signs, 48 Mandatory signs, 160 Prohibitory signs and 70 false alarms (Background signs). Also, in the CTSD dataset, 568 traffic signs were detected in the detection module which it's included 135 Mandatory signs, 128 Danger signs, 258 Prohibitory signs and 47 false alarms (Background signs). The overall classification accuracies of their proposed method on GTSDB dataset are 98.24% and 98.77% for CTSD dataset respectively. While, for the falsely classified sign, most of them were blurry or unfinish capture image.

2.3.6 A Real-Time Chinese Traffic Sign Detection Algorithm Based on Modified YOLOv2

[Done by Ng Jan Hui]

Based on the works of (Zhang, et al., 2017), they have constructed a single convolutional neural network that is built on top of YOLOv2 to perform real-time detection on Chinese and German traffic signs (CTSB and GTSDB datasets). (Zhang, et al., 2017) proposed the following pipeline for the detection of Chinese traffic signs based on the concept of YOLOv2 bounding boxes and also fine grid, the mentioned algorithms were aimed towards detecting small sized traffic signs. Figure 2.3.6.1 illustrates their algorithm.

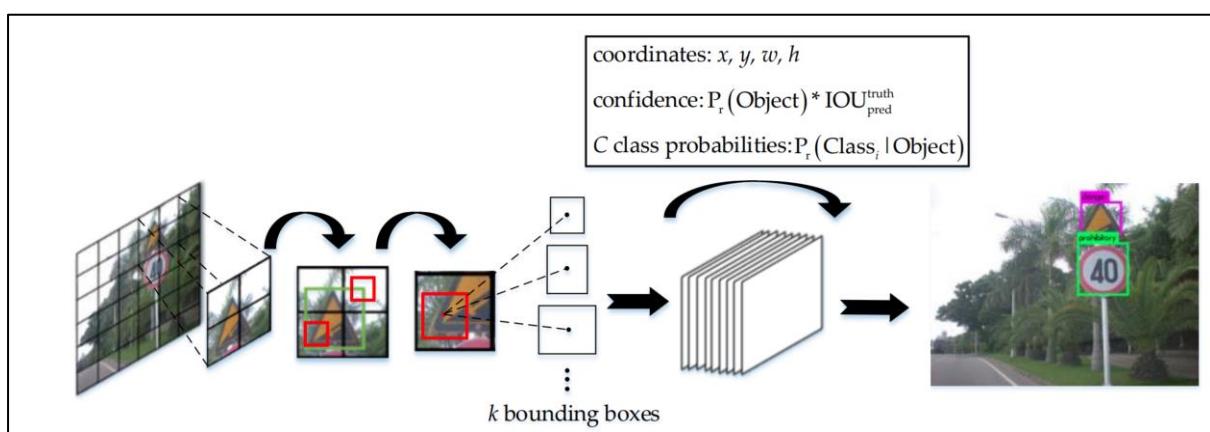


Figure 2.3.6.1 Detection Algorithm for Chinese traffic signs (Zhang, et al., 2017)

The major contribution by (Zhang, et al., 2017) in their works was the expansion of the CTSD dataset to the CCTSDB (Changsha University of Science and Technology Chinese traffic sign detection benchmark). As mentioned by (Zhang, et al., 2017), their reasoning behind the expansion of the CTSD dataset was because there was already a lot of effort placed by other researchers into studying the more common GTSDB dataset. They stated that since certain traffic signs within the CTSD dataset contained Chinese characters and mostly have distinct representation forms compared to the German traffic signs of the GTSDB dataset. Hence, they wanted to implement a YOLOv2 based detection framework for Chinese traffic signs as a novel idea. Figure 2.3.6.2 shows the comparison between Chinese traffic signs and German traffic signs.



Figure 2.3.6.2 Shows the comparison between German traffic signs (Left) and their corresponding Chinese traffic signs (Right)

Figure 2.3.6.3 illustrates the three models created by (Zhang, et al., 2017) based on YOLO architecture. (Zhang, et al., 2017) used multiple 1 x 1 convolutional layers in the intermediate layers and fewer convolutional layers in the top layer to construct their network. (Zhang, et al., 2017) explained during the creation of Model-B, the intuition behind stacking 1 x 1 convolutional layers was to learn more features via cross channels. Multiple 1 x 1 convolutional layers can increase the nonlinear learning ability of the model and will obviously reduce the running time. As quoted by (Zhang, et al., 2017), they said that the number of parameters regarding convolution operations in Model-B decreased as least 21 times compared to Model-A and 27 times compared to the vanilla YOLOv2.

	YOLO	YOLOv2	Model-A	Model-B	Model-C
Bottom	Conv7/2-64	Conv3-32	Conv3-32	Conv3-32	Conv3-32
Maxpool/2	Maxpool/2	Maxpool/2	Maxpool/2	Maxpool/2	Maxpool/2
Conv3-192	Conv3-64	Conv3-64	Conv3-64	Conv3-64	Conv3-64
Maxpool/2	Maxpool/2	Maxpool/2	Conv1-32	Conv1-32	Conv1-32
Conv1-128	Conv3-128	Conv3-128	Maxpool/2	Maxpool/2	
Conv3-256	Conv1-64	Conv1-64	Conv3-64	Conv3-64	
Conv1-256	Conv3-128	Conv3-128	Conv1-32	Conv1-32	
Conv3-512	Maxpool/2	Maxpool/2	Conv1-32	Conv1-32	
Maxpool/2	Conv3-256	Conv3-256	Conv3-64	Conv3-64	
Conv1-256	Conv1-128	Conv1-128	Maxpool/2	Maxpool/2	
Conv3-512	Conv3-256	Conv3-256	Conv3-128	Conv3-128	
Conv1-256	Maxpool/2	Maxpool/2	Conv1-64	Conv1-64	
Conv3-512	Conv3-512	Conv3-512	Conv1-64	Conv1-64	
Conv1-256	Conv1-256	Conv1-256	Conv3-128	Conv3-128	
Conv3-512	Conv3-512	Conv3-512	Maxpool/2	Maxpool/2	
Conv1-256	Conv1-256	Conv1-256	Conv3-256	Conv3-256	
Conv3-512	Conv3-512	Conv3-512	Conv1-128	Conv1-128	
Conv1-512	Maxpool/2	Maxpool/2	Conv1-128	Conv1-128	
Conv3-1024	Conv3-1024	Conv3-1024	Conv3-256	Conv3-256	
Maxpool/2	Conv1-512	Conv1-512	Maxpool/2	Maxpool/2	
Conv1-512	Conv3-1024	Conv3-1024	Conv3-512	Conv3-512	
Conv3-1024	Conv1-512	Conv1-512	Conv1-256	Conv1-512	
Conv1-512	Conv3-1024	Conv3-1024	Route	Route	
Conv3-1024	Conv3-1024	Route	Conv3(256)	Conv3(128)	
Conv3-1024	Conv3-1024	Conv3(512)	Conv1-64	Conv1-64	
Conv3/2-1024	Route	Conv1-64	Reorg/Route	Route	
Conv3-1024	Conv1(64)	Reorg/Route	Conv3-512	Conv3(256)	
Conv3-1024	Reorg/Route	Conv3-1024	Conv1-40	Conv1-64	
Local	Conv3-1024	Conv1-40	Detection	Reorg	
Dropout	Conv1	Detection		Conv1(512)	
Conn	Detection			Reorg/Route	
Detection				Conv3-1024	
				Conv1-40	
Top				Detection	

Figure 2.3.6.3 Models created by (Zhang, et al., 2017)

Chapter 2 Literature Review

According to (Zhang, et al., 2017), the performance of their Model-B was the best. The recall and precision were optimal, scoring 90.37% and 95.31% respectively. The most notable aspect of model B was the processing time, it was able to perform detection within a mere 0.017 seconds. Furthermore, Model-B was able to detect subclass of traffic signs that were not marked, this meant that Model-B was able to generalize well to unknown inputs.

As work for future studies, (Zhang, et al., 2017) had plans to improve the CCTSDB data set for further research purposes. Additionally, plans to include high resolution traffic sign images and street videos containing traffic signs were considered. Finally, (Zhang, et al., 2017) planned to alter their network structure accordingly in order to accommodate for the detection of small traffic signs in the newly integrated high-resolution inputs.

2.4 Random Forest Classifier

[Done By: Jacynth Tham Ming Quan]

In this subsection, two existing traffic sign recognition systems are reviewed. Both of the systems used the Random Forest classifier as their main classifier.

2.4.1 Real-Time Traffic Sign Recognition in Three Stages

[Done By: Jacynth Tham Ming Quan]

In 2012, Zaslavskiy and Stanciu developed a three-stage, real-time traffic sign recognition system as a component of an Advanced Driver Assistance System (ADAS). The three stages included in the system were the segmentation phase, detection phase and classification phase. In their proposed system, Zaslavskiy and Stanciu used an efficient linear Support Vector Machine (SVM) with Histogram of Oriented Gradients (HOG) features for the detection process and two tree classifiers – K-d tree and Random Forest. Figure 2.4.1.1 below illustrates the overview of Zaslavskiy and Stanciu's proposed approach.

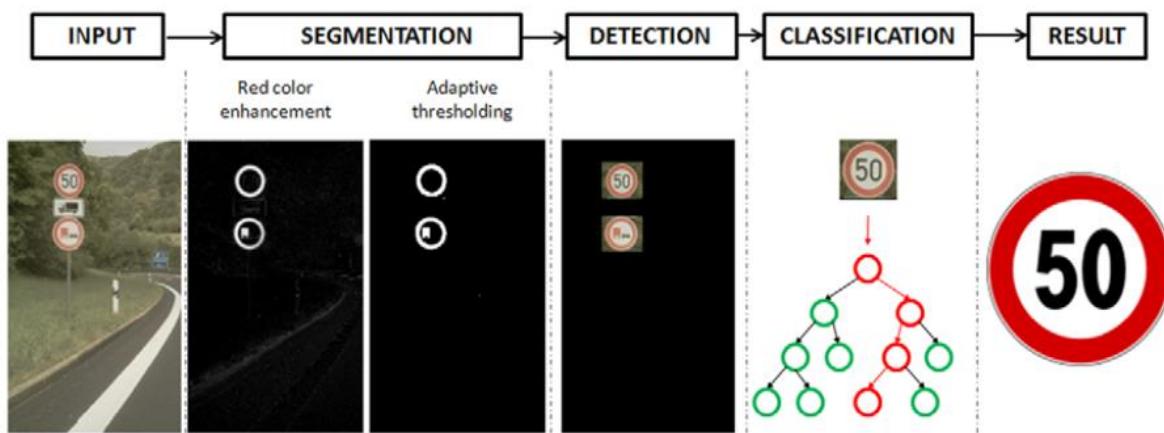


Figure 2.4.1.1 Zaslavskiy and Stanciu's Three-Stage Approach for Traffic Sign Recognition

The first step of the traffic sign recognition was image segmentation. In this phase, the potential return on investments (ROI) were extracted from the input image and red colour enhancement was performed. The result was a binary mask that allowed the system to reduce the search space and the number of false alarms of the detector in the second phase. In this phase, three types of enhancements were done to the input image – colour enhancement,

chromatic filter, and morphological filter. The colour enhancement process extracted the pixels that had dominant red components. Next, the chromatic filter used chromatic segmentation to obtain only the red parts of the signs. Last but not least, the morphological filter was used to obtain the binary mask that designated the ROIs within the image.

Next, in the detection process, two detectors were trained, one to detect circular signs and one to detect triangular signs. According to Zaklouta and Stanciulescu , the SVM and HOG detectors were selected because of their efficiency when used with the tree classifiers. The training and testing datasets contained 14763 and 1584 signs respectively. First, the binary mask created in the segmentation phase previously was used in a sliding window in the detection phase. Each image was resized into a subwindow and scanned at different scales. In each subwindow, the sum of the pixel values was computed with respect to the width and height of the subwindow. If the ratio of the white pixels to the area of the subwindow exceeds a certain threshold value, then the subwindow was considered to contain a potential traffic sign. Then, the potential subwindows were passed to the HOG and SVM detectors, which verified the presence of the respective shapes of the signs. According to Zaklouta and Stanciulescu, the combination of the linear SVM and HOG detector with the red colour enhancement segmentation technique proved to be the best compromise between performance and resource efficiency. Figure 2.4.1.2 below summarizes the traffic sign detection process.

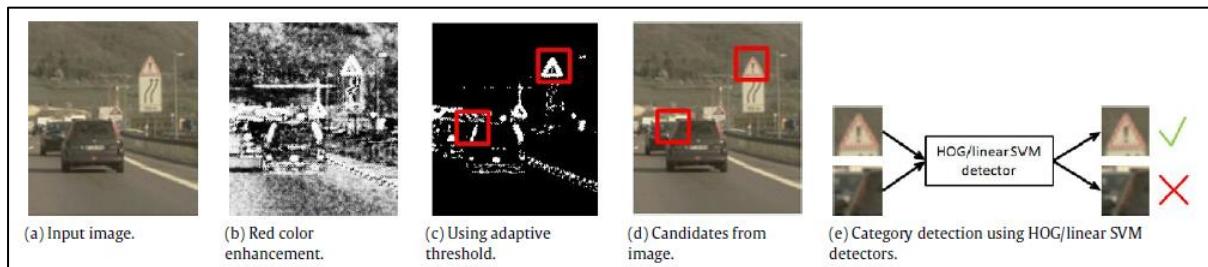


Figure 2.4.1.2 Summary of Zaklouta and Stanciulescu's Traffic Sign Detection Phase

In the classification phase, Zaklouta and Stanciulescu implemented a Random Forest classifier to recognise the traffic signs. The random trees were grown using a subset $I \subset I_N$ of the training samples that were randomly chosen with replacement. The random trees grown were not pruned and in each of the node, a subset of features was randomly selected. The previously selected subset was then split into the left nodes and right nodes using the feature chosen and the pre-set threshold. To classify a traffic sign, each of the random trees in the Random Forest were traversed and the probabilities were stored in the leaves of each tree

Chapter 2 Literature Review

traversed. The combination of having 500 trees and 100 features yielded the highest accuracy of 97.2% when used with the HOG detector. In the testing phase, the model was tested against the German Traffic Sign Recognition Benchmark.

In order to minimise memory consumption and improve accuracy rates even further, Zaklouta and Stanciulescu also implemented Fisher's Criterion for feature space reduction. Fisher's Criterion ranked the features according to their ratio of inter-class to intra-class variance. Coupled with the use of the spatial weighting, the Fisher Scores of the HOG feature vector evidenced that the central image blocks (pixels) were given a higher variable importance than the marginal image blocks. The highest accuracy in the testing phase was achieved using a combination of Random Forest classifier and Fisher Criterion feature selection with 1000 features.

In a nutshell, the traffic sign recogniser developed by Zaklouta and Stanciulescu was able to detect traffic signs using the SVM/HOG detector and classify traffic signs according to the German Traffic Sign Recognition Benchmark using the Random Forest classifier with Fisher's Criterion feature selection technique. The novelty of Zaklouta and Stanciulescu's project revolved around the use of a two-fold (Random Forest and Fisher's Criterion) classification technique to reduce feature space dimension and achieve higher accuracies. The limitation of this project was that it was only able to recognize red traffic signs from the prohibitory category. The future works of this paper proposed the integration of temporal data into the system in order to track detected traffic signs and the combination of the adaptive threshold with other colour enhancement techniques in order to detect traffic signs of colours other than red.

2.4.2 Traffic sign detection and recognition based on random forests

[Done by Ng Jan Hui]

Pertaining to the works of (Ellahyani, et al., 2016), a novel traffic sign detection and recognition algorithm was introduced. (Ellahyani, et al., 2016)'s recognition algorithm was broken down into 3 main steps. As a brief explanation, during the first step, the ROI of the traffic sign image was segmented based on the thresholding of HSI colour space components. During the second step, the traffic signs' shapes are detected based on the processed blobs generated by the first step. During the final step, the recognition on the detected traffic signs was carried out using random forest classifiers. Figure 2.4.2.1 shows the flow of (Ellahyani, et al., 2016)'s algorithm.

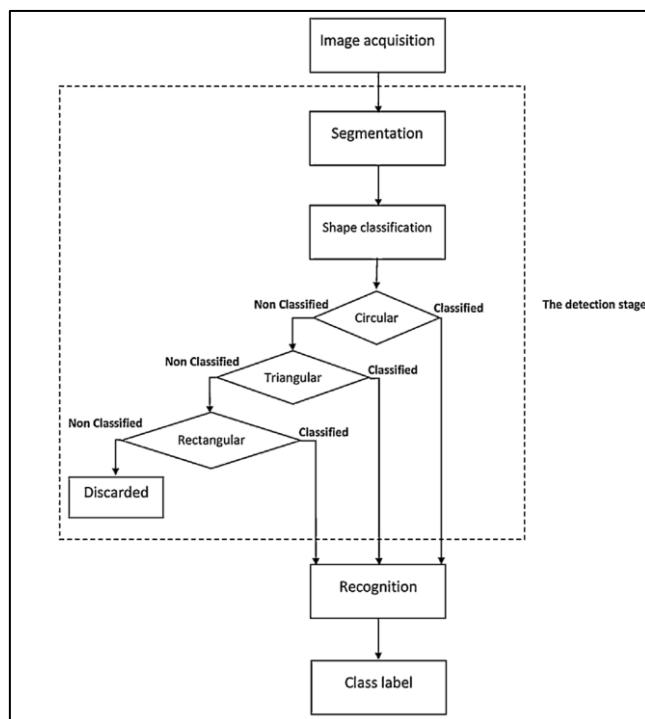


Figure 2.4.2.1 Logical flow of the 3-stage algorithm (Ellahyani, et al., 2016)

Two novelties within (Ellahyani, et al., 2016)'s works can be identified. During the second step of their algorithm, (Ellahyani, et al., 2016) proposed the usage of invariant geometric moments with a simple metric to classify the traffic sign's shapes instead of using conventional machine learning methods. Since traffic signs are mostly limited to triangular, circular, rectangular, and octagonal forms. The usage of invariant geometric moments results in a lower processing time compared to using algorithms such as SVMs and Neural Networks. Next, the histogram of oriented gradients (HOG) features was extended to the HSI colour space,

(Ellahyani, et al., 2016) then combined the HSI-HOG features with the local self-similarity (LSS) features to form a novel descriptor called HSI-HOG+LSS.

During the segmentation phase of their algorithm, the appropriate threshold value was obtained based on the analysis of the histograms of hue, saturation and intensity components corresponding to the red and blue manually segmented signs from the GTSDB data set. Achromatic decomposition was then utilized to extract the white ROIs (Shape of Sign) from the base traffic sign images. The binary patches of the sign shapes extracted is shown in figure 2.4.4.2

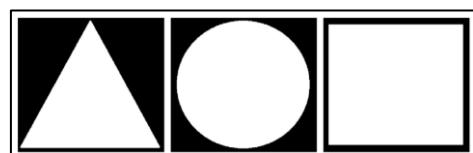


Figure 2.4.2.2 Triangle, Circular and Rectangular Binary Patches of Traffic Signs (Ellahyani, et al., 2016)

During the shape classification phase, (Ellahyani, et al., 2016) pointed out that since the binary patches extracted are of simple shapes (Triangle, Circular and Rectangular), invariant moments is more than sufficient to recognize the shapes. Moving on to the recognition phase, a variety of features such as HOG, LBP, LSS and a combination of those 3 was computed and used for recognition. Among the features, the most notable was the HSI-HOG descriptor, (Ellahyani, et al., 2016) had proposed the following flow as shown in figure 2.4.4.3 to compute the HSI-HOG descriptor.

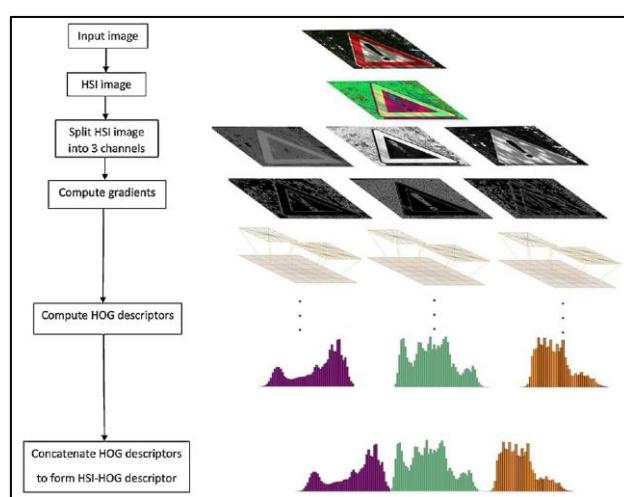


Figure 2.4.2.3 HSI-HOG Descriptor Computation Flow (Ellahyani, et al., 2016)

Chapter 2 Literature Review

(Ellahyani, et al., 2016) further concatenated the LSS descriptors with the HSI-HOG descriptors to form their novel HSI-HOG+LSS descriptor. The usage of such a combination of features resulted in a low execution time and good performance as both colour and texture-based features was utilized in tandem. The results are shown in figure 2.4.4.4

Feature	CCRs(%) of all subsets		Run time (ms/frame)	
	Random forest	SVM	Random forest	SVM
HOG	96.11	95.93	21.24	46.18
HSI-HOG	96.73	96.29	28.51	51.96
HSI-HOG+LBP	97.06	96.72	29.75	53.87
HSI-HOG+LSS	97.43	96.91	28.93	53.12

Figure 2.4.2.4 Correct Classification Rates and Run Time for each descriptor (Ellahyani, et al., 2016)

To wrap up, (Ellahyani, et al., 2016)'s works achieved 94.21% AUC on the German GTSDB data set with a processing rate of 8-10 frames per second. Although the work of (Ellahyani, et al., 2016) was able to produce good results, they did mention a few improvements to consider as part of future work. For example, adaptive thresholding could be used to overcome the colour segmentation issues that was caused by the variable weather conditions, time of day, shadows, and the orientation of the object with respect to the sun during step one of their algorithm. Besides, temporal information could also be incorporated to track the detected traffic signs and be used in aiding the decision-making process of the classifier. The integration of temporal information can also serve as a means of restricting the search space of the traffic sign image considering previous detections information, which ultimately helps in quickening the detection speed. (Ellahyani, et al., 2016) also mentioned that feature selection can also be considered in the recognition phase as it is able to reduce the size of the descriptor vector. Lastly, they mentioned that other classifiers such as Neural Networks can be combined with their current algorithm.

2.5 Neural Network

2.5.1 An Efficient Method for Traffic Sign Recognition Based on Extreme Learning Machine

[Done by: Tan Jing Jie]

Huang, et. al. (2016) had introduced a computationally efficient method for Traffic Sign Recognition (TSR). This proposed method includes the two modules – Extraction of Histogram of Oriented Variant (HOGv) feature and Extreme Learning Machine (ELM) algorithm. The proposed HOGv features success to maintain a good balance between the redundancy and local details. Moreover, compared to other deep learning methods such as Convolutional Neural Network (CNN), this approach achieves high accuracy prediction with low computation complexity with. The overall approach can be illustrated as the figure below.

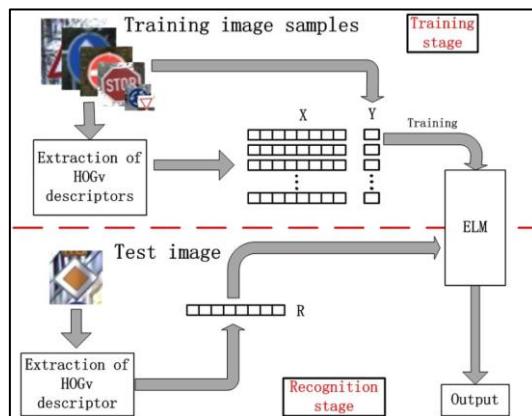


Figure 2.5.1.1 Framework for the proposed TSR method using HOGv and ELM approach.

(Huang, et al., 2016)

In order to resolve contaminated condition including bad weather, viewpoint variations, etc., HOG was used to represent more local detail information – more dimensions. However, this representation is redundant and will affect the classification performance. Hence, they proposed HOGv descriptor with 5 steps – image pre-processing, gradient accumulation, normalization, dimensionality reduction and concatenation. The extraction of HOGv descriptor as shown in the figure below.

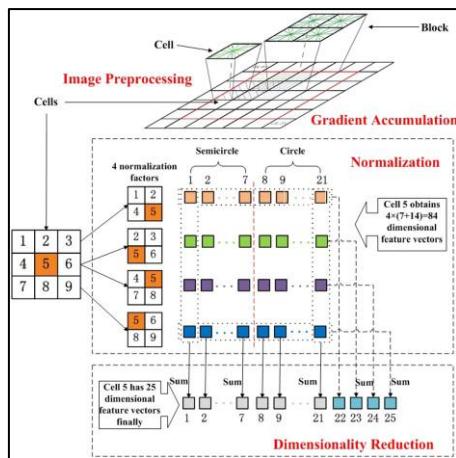


Figure 2.5.1.2 The extraction of HOGv descriptor (Huang, et al., 2016)

This paper makes improvements to the ordinary HOG descriptor in order to include more detailed local information of traffic signs into the accumulated histogram. Assuming a valid traffic sign image used as input, firstly, bilinear interpolation was used to scale the traffic sign into fixed size. The approach includes both contrast sensitive and contrast insensitive orientation of gradients. Second, each cell's-oriented histogram was normalized by their respective neighbour block. Followed by these normalized histograms using the principal component analysis strategy to remove redundant information in meanwhile maintaining the information. Hence, this method can effectively reduce the need of computational power on the following classification task while ensuring the result's accuracy.

Moving on, this paper proposed ELM which is a single-hidden-layer feedforward neural network (SFNNs) as the classification algorithm. The ultimate advantage of this algorithm is that only the output weight between hidden and output layers are trained, so layer-by-layer back-propagation used in deep learning algorithms is no longer required. As it has only one hidden layer involved in training, this increases the recognition speed and also decreases the computational cost of training. Second, the advantage is that the cost function includes the norm of output weight, hence improved generalization. The two advantages bring ELM able to obtain optimal and generalized solutions for multiclass recognition.

There are several comparisons made by authors to their proposed method – HOGv descriptor and ELM descriptor. The HOG descriptors used are HOGv (based on circle gradient orientation) and HOGs (semicircle gradient orientation). In the meanwhile, extra testing for the HOGv descriptor is further done for the rotated version indicated by HOGv+r. Moreover, the experiment is conducted for different types of classification methods, including the ELM

Chapter 2 Literature Review

approach. There were two types of ELM approach, Kernel ELM with Gaussian function, and normal ELM. The following diagram tabulated the overview of the comparison result using the GTSRB dataset. The overall result is identical when they use BTSC and MASTIF dataset.

Table 2.1.1.1 The recognition accuracy obtained by different HOG descriptors and different classifiers on the GTSRB dataset. (Huang, et al., 2016)

	HOGs_nb	HOGc_nb	HOGv_nb	HOGv+r
Kernel ELM based	97.52±0.30%	98.44±0.16%	98.99±0.10%	99.49±0.05%
Kernel SVM based	96.96±0.30%	98.01±0.25%	98.41±0.25%	98.95±0.15%
ELM based	96.75±0.16%	97.77±0.11%	98.54±0.12%	98.89±0.10%
SVM based	96.12±0.20%	97.43±0.17%	97.65±0.25%	97.80±0.21%
LDA based	94.01±0.15%	96.24±0.25%	97.14±0.21%	97.56±0.20%

From the table, it is easy to notice that HOGv detectors have the best result and HOGv+r has the higher accuracy, 99.49%. Besides, it is easy to notice that Kernel ELM based classifiers have the best result to recognise the traffic sign. On the other hand, compared with CNN based classifiers, CNNs are very sensitive to the surrounding conditions such as lighting, and shadow occlusion. ELM based classification only misclassifies when the providing images are blurred and which are difficult to recognize by humans. This paper achieved low recognition time, 1.0ms/frame which is much shorter than CNN, 11.0ms/frame.

In brief, the traffic sign recogniser developed by Huang, et. al. (2016) was able to recognise traffic signs in images efficiently. This project's novelty was to reduce the computational complexity while maintaining the accuracy. Hence, this increases the real-time-use performance and reduces the computation time needed. Beside increasing the computational efficiency, this trained ELM also gave highly satisfied results on classifying traffic signs from images that in variant illuminance and weather conditions.

The future works of this paper proposed learning the number of hidden nodes in ELM and extending this ELM-classifier in doing the traffic sign detection. By learning the number needed in the hidden nodes, can do further improvement on balancing the computational cost and the accuracy. On the other hand, the balance of detection speed and accuracy is not satisfied nowadays. Since the ELM classifier reaches a similar accuracy compared to CNN which both have 99.5% accuracy and above. Hence, ELM classifier has high potential to outperform in traffic sign detection. The resolving of the limitation on detecting traffic signs is pivotal in improving the real-time system performance especially when there is occlusion or bad weather.

2.5.2 Traffic Sign Recognition using Perceptual Generative Adversarial Networks (GAN)

[Done by Ng Jan Hui]

With regards to the works of (Li, et al., 2017), the authors proposed the usage of a novel perceptual generative adversarial network (Perceptual GAN) model that works well on detecting small objects such as traffic signs and pedestrians. As stated by the authors, their novel perceptual GAN can improve small object detection by narrowing the representation difference of small objects from its larger counterparts and by fully exploiting the structural correlations between objects of different scales during network training.

The authors disagreed with the notion of applying data augmentation or simply increasing the feature dimension of the small objects as the methods of achieving better detection results. Instead, the authors argued that the effective way of lifting the representation of small objects is to unveil the intrinsic structural correlations between the small-scale versions and large-scale versions of the same object for each feature identified. Each category of the features will then be transformed to upgrade the capability of the perceptual GAN in a more intuitive way.

The novel perceptual GAN model proposed by (Li, et al., 2017) is composed of two subnetworks: namely the generator network, and the perceptual discriminator network. Figure 2.1.5.1 depicts the perceptual GAN and its two subnetworks.

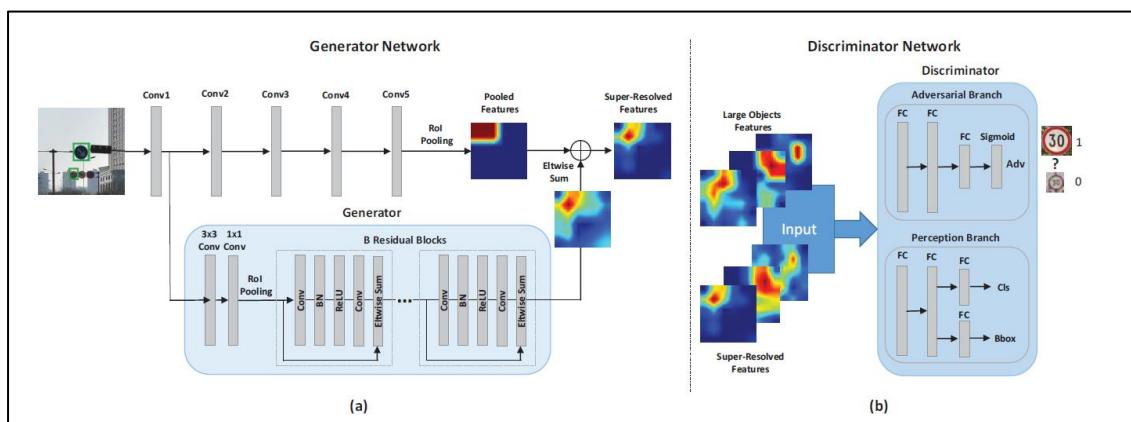


Figure 2.1.5.1 Novel Perceptual GAN Model of (Li, et al., 2017)

Basically, the generator subnetwork aims to transform the original poor features of the small objects to highly discriminative versions by introducing fine-grained details from lower-

level layers. On the other hand, the discriminator subnetwork acts as a critic and evaluates the fine-grained details produced by the generator subnetwork. The novelty of (Li, et al., 2017)'s perceptual GAN is the inclusion of a new perceptual loss which is specifically used for detection purposes. This means that the discriminator subnetwork can also be used to justify the detection accuracy. To put it simply, the generators of the perceptual GAN can learn to transfer perceived poor representations of the small objects to super-resolved versions. The super-resolved version of the small objects is similar enough to the actual large objects which can fool the discriminator subnetwork and at the same time improving the detection accuracy. The demonstration of the super resolution of small object features is shown in figure 2.1.5.2.

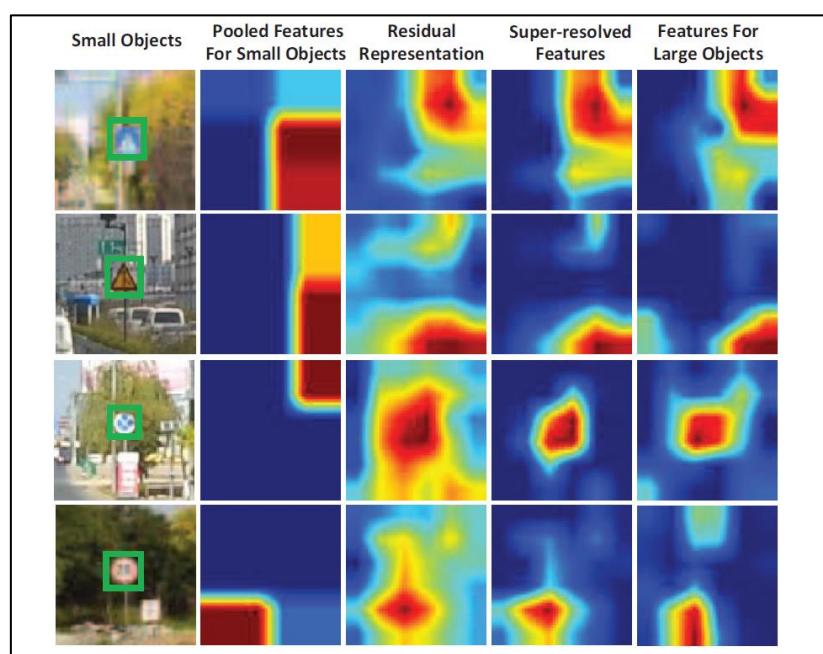


Figure 2.1.5.2 Super-resolved features spawned by the Generator Network (Li, et al., 2017)

Furthermore, (Li, et al., 2017) optimized the parameters of the generator subnetwork and discriminator subnetwork alternatively to curb the min-max problem of their model. The competition evoked through the alternative optimization of both subnetworks is the key to produce super-resolved large-object like representations from the same small-object.

During the experimentation phase, (Li, et al., 2017) used the Tsinghua-Tencent 100K traffic sign benchmark as the dataset for their Perceptual GAN. The dataset contains traffic signs of variable sizes. 3270 traffic signs which have areas of less than 32 by 32 pixels are considered as suitable small instances. The ample amount of small traffic signs provides a good testbed for evaluating the performance of their model. Figure 2.1.5.2 shows the comparison of

Chapter 2 Literature Review

recall and accuracy between (Li, et al., 2017)'s model and models from other works. The performance of (Li, et al., 2017) model was superior to the other state-of-the-art algorithms during that time. It even surpassed (Zhu, et al., 2016)'s CNN model.

Object size	Small	Medium	Large
Fast R-CNN [11] (R)	46	71	77
Fast R-CNN [11] (A)	74	82	80
Faster R-CNN [32] (R)	50	84	91
Faster R-CNN [32] (A)	24	66	81
Zhu <i>et al.</i> [45] (R)	87	94	88
Zhu <i>et al.</i> [45] (A)	82	91	91
Ours (R)	89	96	89
Ours (A)	84	91	91

Figure 2.1.5.3 Comparison of detection performance on traffic signs of different sizes (Li, et al., 2017)

Lastly, (Li, et al., 2017) extended their work beyond into detecting other diverse small object categories which include boats, bottles, chairs, and plants. As expected, their model achieves an average precision of 69.4%, 60.2%, 57.9% and 41.8% respectively for each object. Their model performs very well against other competitive baseline models such as the Fast R-CNN. (Li, et al., 2017) mentioned during the implementation of their work, they only utilized the features that are produced by the “Conv1” layer of the generator network. Perhaps in future studies, experimenting on the features given by other layers from “Conv2” to “Conv5”, and then evaluating its performance during detection can be done.

2.6 Fourier Descriptors

[Done By: Jacynth Tham Ming Quan]

In this subsection, one existing traffic sign recognition system using Fourier Descriptors as its main classifier is reviewed.

2.6.1 Using Fourier Descriptors and Spatial Models for Traffic Sign Recognition

[Done by Por Teong Dean]

(Larsson & Feslberg, 2011) claimed that a robust vision system was needed and essential for traffic sign recognition even though there is an existing GPS navigator system. But in some cases, the GPS navigator system will always fail in traffic sign recognition. Therefore, they developed a traffic sign recognition system using locally segmented contours, which the contours described by Fourier descriptors. Then, by using exhaustive search, a query image can be matched to the sign prototype database. It was done with high efficiency by enforcing the spatial requirements with a quick cascaded matching scheme and Fourier descriptors using the correlation-based matching scheme.

Their main contribution in the experiments was extending previous Fourier descriptors' work with an implicit star-shaped object model for performance improvement of the Fourier descriptors. Secondly, for changing the previous Fourier descriptors into a fully automatic system, the need for an ROI detector was removed. Thirdly, a database exceeding 20 thousand frames with 20 percent hand-labelled with the contents of a total of 3488 traffic signs was released by them. The method consists of three steps: FDs extractions, FDs matching, and lastly, matching prototypes with spatial models.

In figure 5.5, a simple implicit star-shaped object model was shown; each local feature was connected to the object's center.



Figure 2.6.2.1 Local features (green contours) was extracted and corresponding vectors (red arrow) pointing towards the center of the traffic sign.

The rough idea of Larson and Felsberg for the matching sign prototype was shown in Figure 6.2.2:



Figure 2.6.2.2 Matching Sign Prototype Process

The upper left was the Query image. Then, extracted contours were shown in the upper right picture. Thirdly, the non-yellow colour in the picture lower left shown the contours that match any of the contours in the pedestrian crossing prototype. Finally, the picture lower right shows the final result after matching against all sign prototypes.

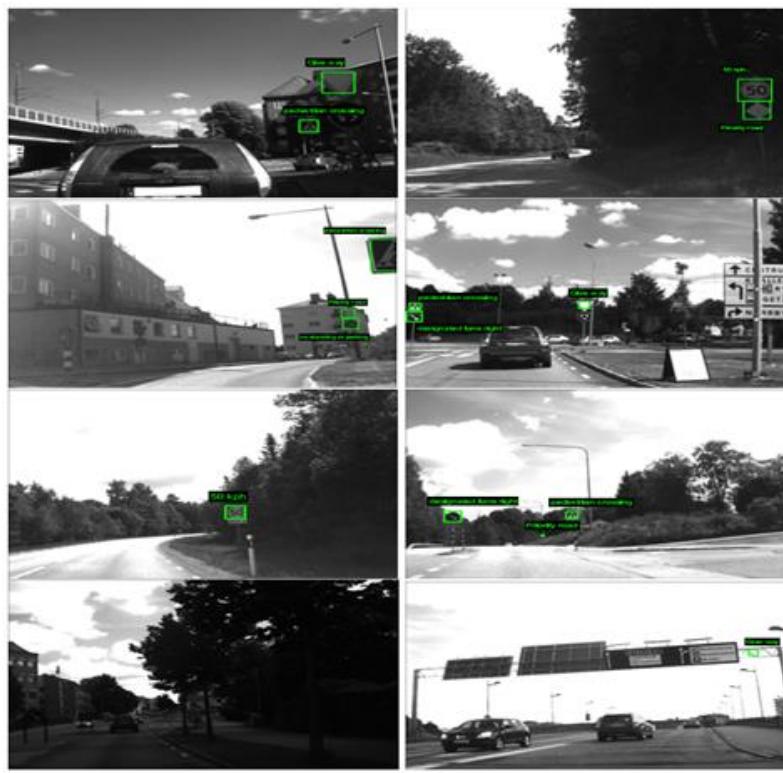


Figure 2.6.2.3 Examples of Result

The first 3 rows were the correct detected sign. While the lowest left shown the result of miss detection, and the lowest right shown the result of false-positive give way sign in their experiment.

False contour extraction will lead to False Positive, FP, and it will cause missed detection like the error example in the lowest left of Figure 5.7. At the same time, insufficient contours extraction in the images will also lead to missed designated signs, as shown in the error example in the lowest right of Figure 5.7. The proposed traffic sign recognition shows a false classification as a give-way sign. This is because the Giveaway class of the traffic signs has only a single contour, so it cannot benefit from the spatial model method proposed by Larson and Felsberg. It is possible to increase the recall rate for the Giveaway class, but this will also lead to the number of false positives, FPs growing dramatically.

According to (Larsson & Feslberg, 2011), their presented methods will get an excellent result for traffic signs with multiple distinct contours. However, it still needs improvement to detect the traffic signs with a few or have only a single contour. Also, future work can be done by extending the database to include different ambient conditions, such as rain conditions and night conditions, and have other countries' traffic signs.

2.7 Critical Remarks of Previous Works

[Done by Tan Wei Mun]

In the literature review, 14 papers related to traffic signs classification were reviewed and categorized five sub-sections according to the classifier used in the papers. The five general classifiers are SVM, CNN, random forest, Fourier descriptors and neural network classifier. Each paper has proposed different methods to train and improve the classifier's accuracy, and some even proposed the traffic sign detection algorithm using object detection. Hence, different classifiers trained using various methods will be analyzed by comparing their pros and cons in this section.

2.7.1 Advantages

[Done by Por Teong Dean]

The traffic sign classification and detection system that uses Support Vector Machines (SVM) techniques were employed by Greenhalgh & Mirmehdi (2012), Yuan et al. (2017) and Berkaya, et al. (2015). The main advantage of the SVM classifier was that it was able to classify traffic signs accurately in different lighting conditions. This was due to the ability of the classification pipeline to detect candidate regions as maximally stable extremal regions (MSERs). In addition, the use of feature extraction techniques with the SVM also yielded a lower chance of false positives.

Moving on, the random forest classifier used by Zaslavskiy & Stanciulescu (2012) and Ellahyani et al. (2016) had several advantages compared to the other classifiers. The use of random forest classifier, paired with Fisher's Criterion for feature space reduction, from Zaslavskiy & Stanciulescu's paper resulted in higher accuracies and lower memory consumptions. Similarly, using a combined feature descriptor in Ellahyani et al.'s paper also produced better performance in terms of classification speed and accuracy.

Next, Convolutional neural networks (CNN) were used in 6 of the reviewed papers. In the proposed system by (Zhu, et al., 2016), the main advantage of their proposed system is their trained CNN models can classify traffic signs from images that covered large variations in illuminance and weather conditions, as well as pictures in which the traffic sign only took a small fraction of the whole image. In Jin et al.'s paper, a hinge loss stochastic gradient descent (SGD) technique was used with the CNN model, which led to an optimized model capacity as

well as higher accuracies compared to the vanilla CNN. The other papers shared similar approaches, which was the use of feature extraction methods to yield higher accuracies, lower searching durations and the ability to classify traffic signs in real-time.

Furthermore, the neural network-based approaches were employed by (Huang, et al., 2016) and (Li, et al., 2017), with the integration of HOG features and ELM algorithm, respectively. The former successfully maintained a balance between local details and redundancy in the context of traffic signs, all while decreasing the need for high computational powers during the classification process. Similarly, the latter achieved the same results, with an additional ability to obtain an optimal and generalized solution for multiclass recognition with the use of the ELM algorithm.

Last but not least, (Larsson & Feslberg, 2011) proposed a solution by using Fourier descriptors and spatial models for traffic sign recognition. The main advantage is that their proposed solution had a high-performance accuracy for the multiple distinct contours traffic sign.

2.7.2 Shortages

[Done by Ng Jan Hui]

Despite the advantages mentioned in the reviewed works, a few underlying flaws that hinders the performance of the systems could be pointed out.

The usage of Support Vector Machines to classify traffic signs was employed by (Greenhalgh & Mirmehdi, 2012), (Yuan, et al., 2017) and (Berkaya, et al., 2015). The disadvantage of using the fusion strategy by (Yuan, et al., 2017) is that detection could not be performed when the vehicle is making drastic changes in motion, for instance when the vehicle is turning around or going downhill.

Moving on, Convolutional neural networks (CNN) were also used in 6 of our reviewed works. Out of the 6 CNN methodologies, (Jin, et al., 2014)'s work had some slight disadvantages. The disadvantage is when their prediction algorithm was given low resolution traffic sign images, traffic sign images that are under intense sunshine, or when improper labelling was done, it would result in a falsely predicted traffic sign. Besides, (Jin, et al., 2014)'s prediction algorithm could only classify one traffic sign per image and is unable to correctly predict multiple traffic signs in one image. This is because (Jin, et al., 2014) utilized scaled pictures of only a single traffic sign during the training of their CNN.

Chapter 2 Literature Review

Next, random forest classifiers were also used by (Zaklouta & Stanciulescu, 2012) and (Ellahyani, et al., 2016). The weakness of (Zaklouta & Stanciulescu, 2012)'s work is that their classification model could not recognize other colours apart from red coloured signs of the prohibitory traffic sign category. This is contributed by the fact that simple thresholding was used instead of adaptive thresholding. (Ellahyani, et al., 2016) also faced a similar problem encountered by (Zaklouta & Stanciulescu, 2012). Their model could not work well on noisy images caused by variable environment conditions. Similarly, adaptive thresholding could also be used during the image pre-processing phase of (Ellahyani, et al., 2016)'s classification pipeline to mitigate the effects colour segmentation issues.

Then, neural network-based approaches were employed by (Huang, et al., 2016) and (Li, et al., 2017). Between these 2 works, (Huang, et al., 2016)'s algorithm had some weaknesses. Although their algorithm can work in real time, however when presented with traffic sign images that are exposed under intense sunlight, blocked by shadow or when there is under bad weather conditions, the performance of their algorithm will drop. They proposed to learn the number of hidden nodes needed in order to curb the weakness.

Lastly, a Fourier descriptor based approached was utilized by (Larsson & Feslberg, 2011), the weakness of their works was obvious as their algorithm faced trouble in detecting traffic signs with 1 or few contours. This is due to the nature of Fourier descriptors to implicitly work on star-shaped objects (stars naturally have distinct and multiple contours). The low number of contours will not be able to benefit from the spatial model proposed by the authors. Thus, hindering the classification performance.

2.7.3 Table of Comparison

[Done by: Tan Jing Jie]

The table below shows the comparison of each approach discussed above. The pros and cons of each approach are tabulated in this table. As shown, the accuracy for each classifier is similar, hence the focus should be on each approach novelty to increase the applicable usage range such as working on rainy days. Several novelties in SVM classifiers such as cascade recognizer, tracking and result fusion strategy can be implemented as it reduces computational complexity especially use in mobile application and able to provide stable accuracy.

Table 2.7.3.1 The comparison table for previous works approach

Classifier	SVM			Random Forest	
	Cascade Classifier (Greenhalgh & Mirmehdi, 2012)	Fusion Strategy (Yuan, et al., 2017)	Vanilla (Berkaya, et al., 2015)	Fisher's Criterion (Zaklouta & Stanciulescu, 2012)	Invariant Geometric (Ellahyani, et al., 2016)
Novelty	<ul style="list-style-type: none"> • Cascade of a series of SVM classifiers. 	<ul style="list-style-type: none"> • Use Gaussian weighting for fusion strategy in classifier. • Kalman Filter to track. 	<ul style="list-style-type: none"> • Use detector to enhance classifier 	<ul style="list-style-type: none"> • Fisher Criterion to do space reduction 	<ul style="list-style-type: none"> • Study of descriptor and classifier relationship • Integrate temporal data with Invariant Geometric moments
Detector	<ul style="list-style-type: none"> • MSER 	<ul style="list-style-type: none"> • ACF with HOG • Cascade boosted weak tree classifier 	<ul style="list-style-type: none"> • EDCircle detection • HOG, LBP, Gabor 	<ul style="list-style-type: none"> • SVM with HOG 	<ul style="list-style-type: none"> • HOG, LBP, LSS
Advantages	<ul style="list-style-type: none"> • Resolve lighting issue 	<ul style="list-style-type: none"> • Reduce scope to detect 	<ul style="list-style-type: none"> • Reduce false positive rate by detector. 	<ul style="list-style-type: none"> • Reduce feature space dimension 	<ul style="list-style-type: none"> • Increase detection speed
Weakness	-	<ul style="list-style-type: none"> • Vehicle not in normal driving (Fail to detect) 	-	<ul style="list-style-type: none"> • Only recognize some category of traffic sign 	<ul style="list-style-type: none"> • Failed to work on image under noisy condition
Accuracy	92.1%	97.5%	97.0%	97.2%	97.4%

Future work	<ul style="list-style-type: none"> • Adding more branch to cascade the classifier 	<ul style="list-style-type: none"> • Make use of saliency information as prior knowledge 	<ul style="list-style-type: none"> • Deeper analysis of algorithm combination 	<ul style="list-style-type: none"> • Integrate of temporal data to reduce the limitation to recognize 	<ul style="list-style-type: none"> • Combine current algorithm with neural network
-------------	--	---	--	--	---

Classifier	CNN					CNN Detector (Zhang, et al., 2017)
	Vanilla (Zhu, et al., 2016).	FCN (Zhu, et al., 2016).	SGD (Jin, et al., 2014).	MLP (Cires, et al., 2011)	Vanila (Yang, et al., 2015)	
Novelty	<ul style="list-style-type: none"> • Divide classification according to traffic sign function 	<ul style="list-style-type: none"> • Using FCN • EdgeBox reduce candidate bounding box in short time 	<ul style="list-style-type: none"> • Use Hinge loss SGD to train CNN 	<ul style="list-style-type: none"> • Use of trained multilayer perceptron (MLP) on descriptor 	<ul style="list-style-type: none"> • Filter false detection by SVM 	<ul style="list-style-type: none"> • Study of different neural network
Detector	<ul style="list-style-type: none"> • CNN 	<ul style="list-style-type: none"> • FCN 	-	-	<ul style="list-style-type: none"> • CPM, MSERs, HOG 	<ul style="list-style-type: none"> • Modified YOLOv2
Advantages	<ul style="list-style-type: none"> • Covered large variations in illuminance and weather conditions 	<ul style="list-style-type: none"> • Satisfied object localization accuracy • Noise background removed 	<ul style="list-style-type: none"> • Resolve improper bounding box annotation 	<ul style="list-style-type: none"> • High flexibility • Fast Training period 	<ul style="list-style-type: none"> • High accuracy • Low processing time 	<ul style="list-style-type: none"> • Have higher detector accuracy
Weakness	-	-	<ul style="list-style-type: none"> • Unwell in low resolution image 	-	-	-
Accuracy	88.0%	-	99.65%	99.15	98.24%	-
Future work	<ul style="list-style-type: none"> • Accelerate recognition process to enable real-time 	<ul style="list-style-type: none"> • Make use this algorithm in real-time traffic sign recognition 	<ul style="list-style-type: none"> • Study of reduce the improper bounding box annotation 	-	<ul style="list-style-type: none"> • Use of MLP to reduce falsely classify. 	<ul style="list-style-type: none"> • Accommodate small traffic sign recognition

Classifier	Neural Network		Fourier Descriptors (Larsson & Feslberg, 2011)
	ELM (Huang, et al., 2016)	GAN (Li, et al., 2017),	
Novelty	<ul style="list-style-type: none"> Using ELM reduces computational complexity and reduces computational time in real-time. Reduce cost of training 	<ul style="list-style-type: none"> Use GAN model which is composed of generator network and perceptual network 	<ul style="list-style-type: none"> Fourier descriptors using the correlation-based matching scheme in classification.
Detector	<ul style="list-style-type: none"> HOGv 	-	<ul style="list-style-type: none"> Fourier descriptors
Advantages	<ul style="list-style-type: none"> Satisfied result for variant luminance and weather Balance of accuracy and computation efficiency. 	<ul style="list-style-type: none"> Enhance the poor feature of small object by its structural correlations 	<ul style="list-style-type: none"> Enforcing spatial requirements to have high accuracy.
Weakness	<ul style="list-style-type: none"> Misclassify when the image is too blur 	-	<ul style="list-style-type: none"> Detectors fail to work when there are non-distinct contours
Accuracy	99.5%	-	-
Future work	<ul style="list-style-type: none"> Increase number of hidden nodes Extending ELM classifier in detection 	<ul style="list-style-type: none"> Evaluate detection by using GAN method in detection 	<ul style="list-style-type: none"> Improve the detector when the traffic sign has only single contours. Extend database to include ambient condition

2.7.4 Future Work

[Done By: Jacynth Tham Ming Quan]

This proposed traffic sign recognizing system aims to conquer the main disadvantages faced by the previous systems by implementing the future works of the previous papers. In order to overcome the main issues of the previous works, such as the inability to detect multiple traffic signs and the vulnerability to weather and lighting conditions, the proposed system integrates the use of bilinear interpolation and HSV colour space segmentation into the traffic sign recognition process. The proposed system uses Hu Moments and a variety of classifiers as the main recognition pipeline, including MobileNet (CNN), SVM and Random Forest. At the end of the classification process, the accuracies and recognition times of the different classifiers are compared. Then, the most suitable classifier is then integrated into a mobile application to complete the development of the mobile-based traffic sign recognizer, as proposed in this paper's objectives.

Chapter 3

System Design

3.1 Design Specifications

[Done By: Jacynth Tham Ming Quan]

The processes of the project are categorized into different phases in the development, which includes project pre-development, data pre-processing, model training, fine-tuning, and last but not least, the testing and validation process. In this chapter, the proposed methodology of the project is described in detail, coupled with the design specifications, requirements, and system verification plan.

3.1.1 Methodologies and General Work Procedures

[Done By: Jacynth Tham Ming Quan & Tan Wei Mun]

Figure 3.1.1 below shows the high-level block diagram of the proposed traffic sign recognizer.

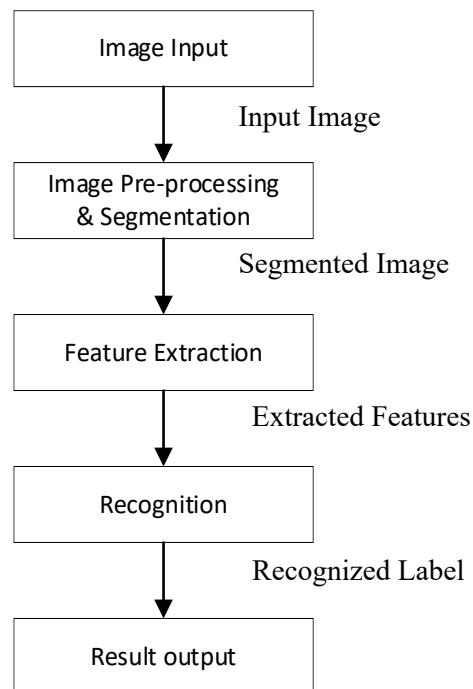


Figure 3.1.1 System Overview Block Diagram

The proposed traffic sign has three main processes between the input and final output – image segmentation, feature extraction and recognition. In the image pre-processing and segmentation stage, the input image is converted to HSV form and undergoes a series of pre-processing techniques such as blurring. Then, the traffic sign is detected using predetermined HSV ranges and segmented out from the input image. This step is crucial to reduce the effects of different lightings and weather conditions in the input image which may affect the final outcome of the recognition process. Then, feature extraction is performed on the input image to extract the most important features of the input image. After the feature extraction process, the features go through a series of classification algorithms to determine which traffic sign is present in the input image. The final result of the proposed traffic sign recognizer is a recognized label of the traffic sign that was classified by the series of classification models. After fine-tuning and testing the classification models chosen in the proposed work, the models will be compared in terms of the accuracies and abilities to deal with different lighting and weather conditions. Then, the top two models chosen will be implemented into video-based desktop applications and mobile applications respectively.

The proposed traffic sign recognizer is realized through the use of a desktop application and a mobile application. On the desktop application, users have the option to select an input image from the file explorer or use an external camera connected to the computer to feed in image frames from a real-time video. The output of the proposed traffic sign recognizer in the desktop application is an image with the recognized traffic signs completed with bounding boxes and classification labels. If users opt for the use of an external camera to provide real-time video input, the proposed recognizer is able to track the recognized traffic signs with bounding boxes until the traffic sign is out of frame. On the mobile application, users are allowed to use images or videos as input to the traffic sign recognizer. Similar to its desktop application counterpart, if a video was used as the input to the proposed traffic sign recognizer, the latter would be able to track the recognized traffic signs with the use of bounding boxes. Additionally, the mobile application would be able to provide voice outputs to the user, indicating the type of traffic sign that has been detected and recognized.

In the following subsections, the system requirements (hardware and software) needed to realize the proposed traffic sign recognizer are described in detail.

3.1.2 Tools to Use

[Done By: Jacynth Tham Ming Quan]

In this section, the hardware and software requirements of the proposed system are discussed in terms of the tools used and the reasons for selection.

In this project, the hardware used in the development process is a laptop and an Android smartphone. Table 3.1.2.1 below shows the minimum and recommended specifications for the hardware used:

Table 3.1.2.1 Hardware Specifications

Device	Specification	Minimum Requirement	Recommended Requirement (Used in this Project)
Laptop	Operating System	Windows 10 (64-bit)	Windows 10 (64-bit)
	Memory	8GB	16GB
	Graphic Card	NVIDIA GeForce 940MX	NVIDIA GeForce 940MX
	Storage	10GB available space	10GB available space
Smartphone	Android Version	Android Oreo	Android 11
	Memory	4GB	6GB
	Storage	1GB available space	3GB available space
	Camera	13MP	20MP

Next, the various software and technologies used in this project and the reason for selection are listed below.

1. Microsoft Visual Studio

Microsoft Visual Studio is an integrated development environment used to develop various types of computer programs. In this project, Visual Studio is chosen as the main IDE due to its compatibility with C++ programming and the use of OpenCV.

2. OpenCV

Open-Source Computer Vision Library (OpenCV) is a public library of programming functions used for real-time computer vision applications, machine learning and image processing. The main programming language used in OpenCV is C and C++. In this project, OpenCV is chosen because of its capabilities to handle object detection, segmentation, recognition, and motion tracking. Moreover, OpenCV also includes a statistical machine learning library that supports the use of neural networks and classifiers such as SVM and Random Forest.

3. GNU Image Manipulation Program (GIMP)

GIMP is a cross-platform, open-source image editor program used mainly for free-form drawing and image manipulation. In this project, GIMP is used to manually analyze the colors of the traffic sign images pixel-by-pixel in order to determine the lower and upper thresholds for the HSV segmentation process.

4. Android Studio

Android Studio is the official IDE for the Android mobile operating system by Google. In this project, Android Studio is used along with the Java programming language for the mobile application development due to its IDE stability and easy-to-use interface. Additionally, Android Studio is also compatible with OpenCV libraries.

5. TensorFlow and TensorFlowLite

TensorFlow is a publicly available artificial intelligence software library developed by Google used to build machine learning models. The classification model used in the proposed project is constructed using TensorFlow libraries because of the seamless library management supported by Google and its capabilities to create large-scale, layered neural networks. TensorFlowLite is a variation of TensorFlow which is much lighter and consumes less processing, thus making it suitable for mobile devices.

6. Keras

Keras is a high-level library built on top of TensorFlow. Written in python, Keras provides a scikit-learn type API to build deep neural networks. Keras is used in the proposed project for the implementation of the MobileNet CNN model to omit the need for working with mathematical tensor algebra, optimization methods and other low-level numerical techniques.

7. Google Colab

Google Colab (short for Colaboratory) is an online, cloud-based Jupyter notebook environment that allows developers to train machine learning models on CPUs, GPUs and TPUs. In the proposed project, Google Colab is used for the development of the mobile-based CNN model.

3.1.3 System Performance Definition

[Done By: Tan Wei Mun]

The proposed traffic sign recognizer aims to improve the detection and recognition accuracy of previous traffic sign recognizers. Since the proposed project uses the vanilla CNN algorithm, the proposed recognizer will be benchmarked against Zhu et al.'s CNN traffic sign recognizer, which had a recognition accuracy of 88%. Hence, the proposed traffic sign recognizer's end target is a detection accuracy of 98% and a recognition accuracy of at least 95%. Moreover, the entire recognition of a single traffic sign should take less than 2 seconds. The system performance requirements are crucial in the proposed project because they ensure that the traffic sign recognition can be implemented in still images and real-time videos.

3.2 System Design / Overview

[Done By: Tan Jing Jie and Ng Jan Hui]

First and foremost, the following system block diagram illustrates the overall system design. The process will start from input of the training dataset images, images pre-processing, feature extraction. These processes will be done for every input image, regardless of it for training or predict purpose.

If the program is training the classifier, the flow will direct to left-hand-side block which a model will be trained. On the other hand, if the purpose is to predict the model, then the flow will be directed to right-hand-side and a classification result will be obtained.

Chapter 3 Image Preprocessing

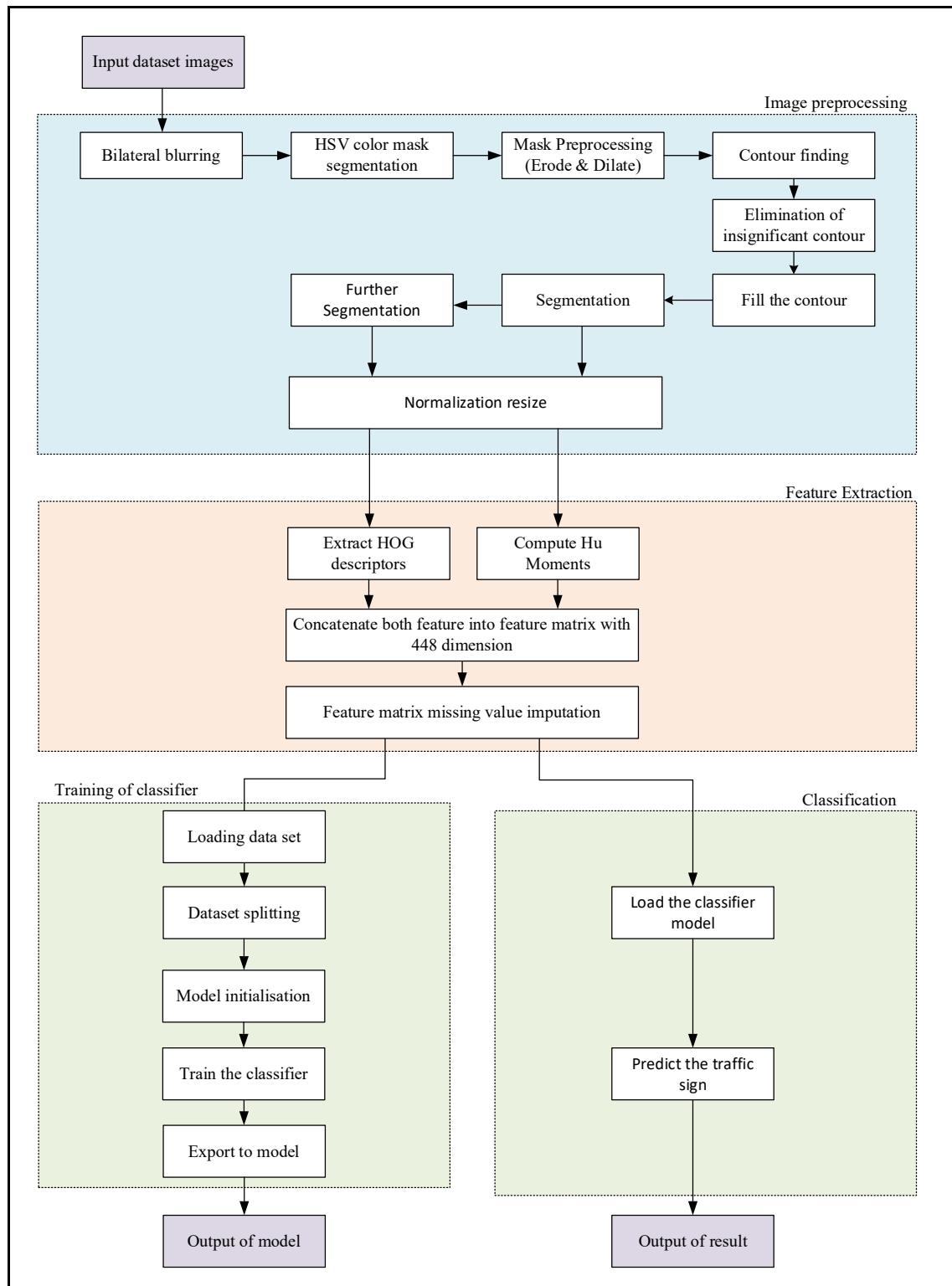


Figure 3.2.1 The Block Diagram for This Traffic Sign Recognition System

3.2.1 System input

[Done By: Tan Jing Jie & Ng Jan Hui & Tan Wei Mun]

Regardless of this system is in training process or in testing process, the input will be obtained by reading a text file. In training process, the training sample (still image) can be obtained by a text file called “Input Dataset/inputSignNames.txt”. Next, in testing process, the user specified still image input is read from a text file called “Input Dataset/testInputs.txt”, while the video paths can be obtained by reading a text file called “Input Dataset/inputSignNamesVideo.txt” which contain the video file locations as shown in Figure 3.2.1.1. The “Input Dataset/” refers to the folder that contain the text file. The system will read the path in the text file one by one while the video will be processed until the end of file. Besides, the system also accepts camera as live stream input.

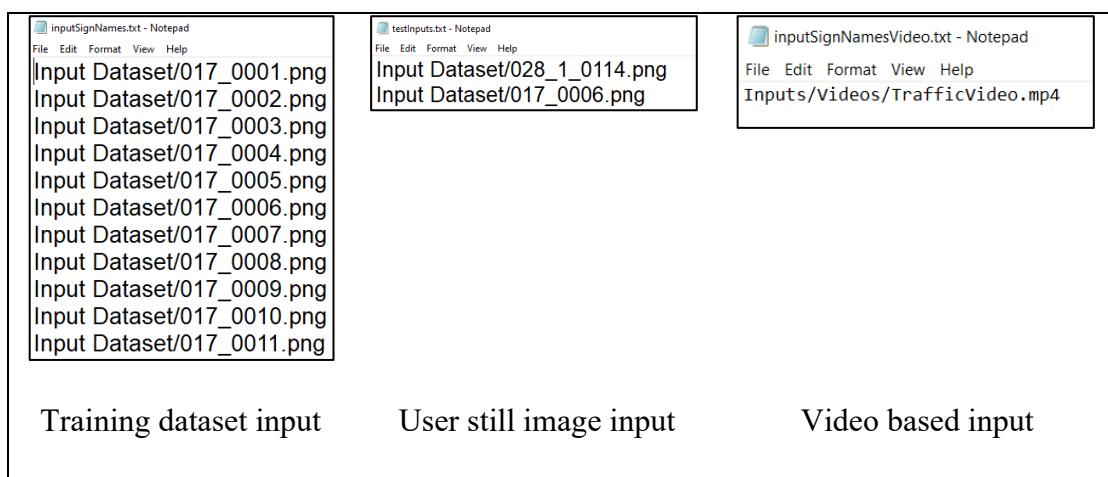


Figure 3.2.1.1 The sample in notepad file

This project has developed the traffic sign recognition system on two platforms –desktop platform and mobile platform. In desktop platforms, the input is obtained by reading a text file or by web camera. The input that is accepted by the text file includes a still image and video file. On the other hand, in mobile platforms, the image input can be obtained from the user's gallery. Besides, the user can use the mobile-integrated camera to provide real-time video stream input.

If it is a video stream input, the video will be retrieved frame by frame, and each frame will be treated by normal image input. In short, after obtaining a single image from input, this image will be sent to the next stage – image pre-processing process.

3.2.2 Bilateral blurring filter

[Done By: Tan Jing Jie]

In order to eliminate the noise from the image background, a bilateral blurring filter needs to be applied. This could be achieved by applying the `bilateralFilter()` function provided in OpenCV. Bilateral blurring is a noise reducing image smoothing filter. This filter rarely affects the area which contains a single colour, but for those areas having different colours, this filter will turn it to its average. Hence, the advantage of this filter is that it preserves most of the contour characteristics compared to surrounding backgrounds which have significantly blurred. The comparison of before and after blurring is shown as the figure below. This blurring step is significant to colour segmentation. Colour segmentation is very sensitive to colour, hence, blurring needs to be used beforehand to eliminate the image's background noise.

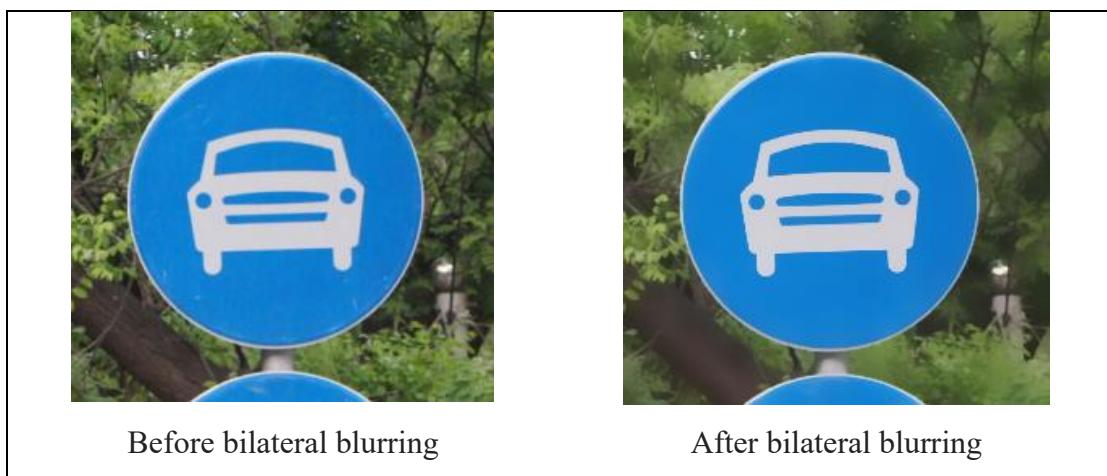


Figure 3.2.2.1 The comparison of traffic sign before and after of bilateral blurring

3.2.3 Colour mask generate using Hue Saturation Value (HSV) colour space

[Done By: Tan Jing Jie]

Before generating the mask, the image needs to convert from Red Green Blue (RGB) colour space to Hue Saturation Value (HSV) based colour scale. The reason to use HSV based image because it is more easily and intuitive to check out the threshold for mask value compare to RGB based image. In this system, HSV colours mask is applied to the input images in order to segment out the traffic sign. OpenCV provided a function – `inRange()`. This function has 4 parameters which are the source image,

output mask, and the lower and upper boundary of HSV value. Hence, by using this function, a total of 3 masks including red, yellow, and blue colour are generated. A bitwise OR is used to concatenate the mask. In short, this HSV colour mask is able to segment out the traffic sign from its background. This step is crucial for the feature extraction process to obtain the representation data of a traffic sign. By using a blue car traffic sign as example, the mask is generated as follows.

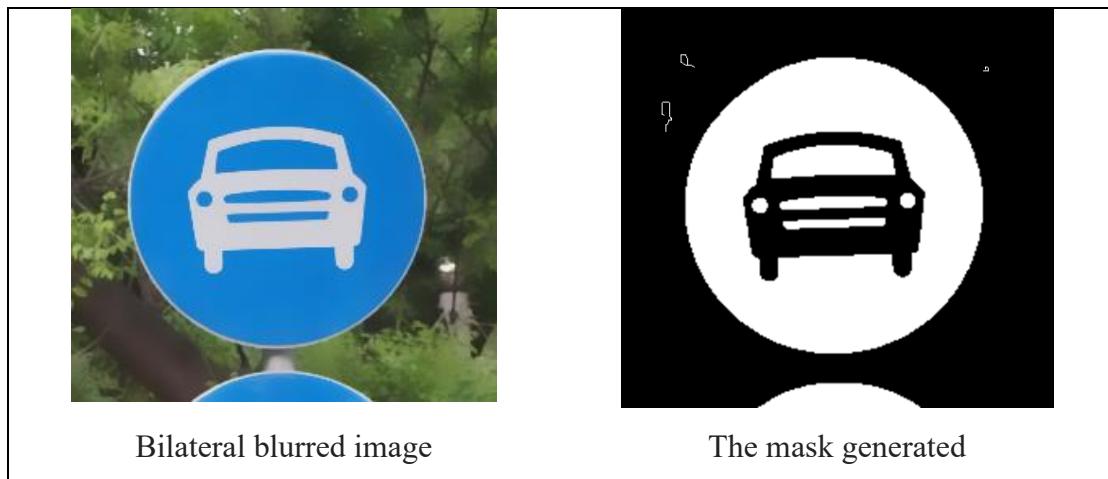


Figure 3.2.3.1 The mask generated from a traffic sign.

3.2.4 Mask processing

[Done By: Tan Jing Jie]

After the HSV colour mask is obtained, the mask needs to undergo a noise elimination step. This can be done by applying the erode and dilate function provided by OpenCV. After the process of erode function to the mask, the overall “thickness” of the mask is reduced. Assuredly, the noise surrounding it is removed too. The depth of erosion can be adjusted by the erosion parameter. After that, the mask will be dilated again by a dilate function with the same depth value used in the erosion parameter. This is to ensure the mask can fit and segment out the traffic sign completely. The figure below shows the noise is removed after erosion and the “thickness” is recovered again after dilation.

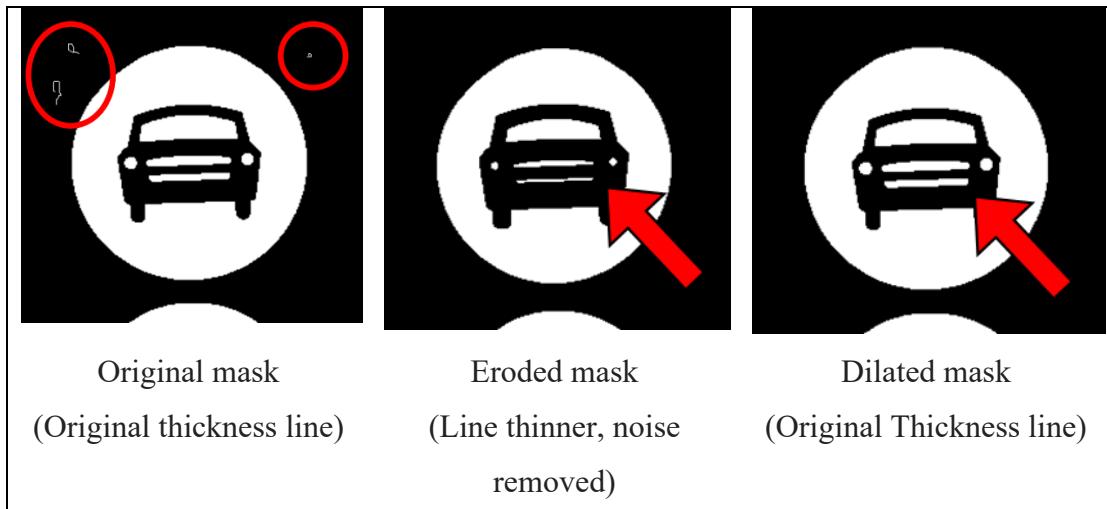


Figure 3.2.4.1 The mask processing flow from original to eroded and dilated

3.2.5 Contour Finding

[Done By: Jacynth Tham Ming Quan & Tan Jing Jie]

After applying the HSV colour space segmentation technique and obtaining the dilated HSV colour mask, all the available contours will be detected using the OpenCV `findContours` function. There are several contour retrieval modes, including `RETR_LIST`, `RETR_EXTERNAL`, `RETR_CCOMP` and etcetera. In this project, the `RETR_EXTERNAL` contour retrieval mode will be used to retrieve only the extreme outer contours. For example, if a parent contour is present with several child contours contained within it, the `findContours` function only returns the outermost contour, which, in this case, should be the outline of the segmented traffic sign. Figure 3.2.5.1 below shows the input and output of the `findContours` function.

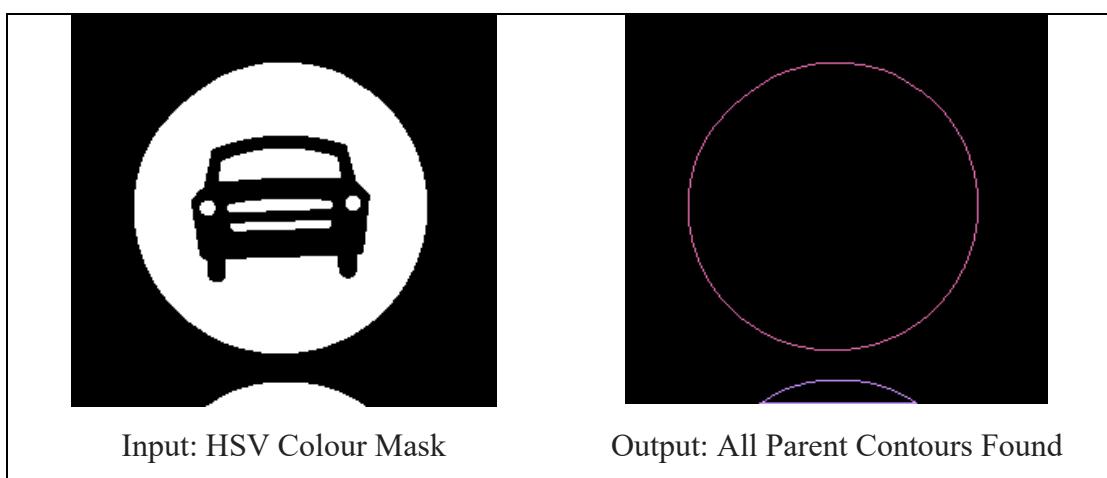


Figure 3.2.5.1 Input and Output of the `FindContours` Function

3.2.6 Elimination of small contour

[Done By: Tan Jing Jie]

An algorithm to remove the unnecessary contour is vital. In this system, the biggest contour finding algorithm is set up in a contours size comparison function, compareContourAreas() is prepared. The function will check the contour area size by using OpenCV function, contourArea(). This comparison function will execute by sort() function to arrange the contour size from smallest to largest. After sorting is completed, a for loop is used to compare each contour with the largest contour. The contour will be removed when it is found 6 times smaller compared to the largest contour. The figure below shows that the not significant contour is removed.

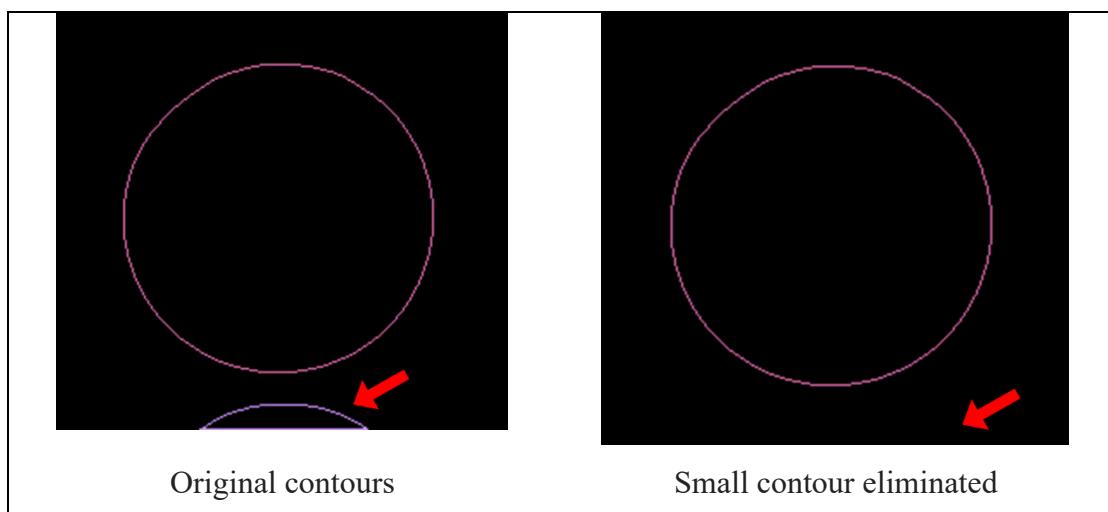


Figure 3.2.6.1 The removal of insignificant contour.

3.2.7 Fill the contour

[Done By: Tan Jing Jie]

After the elimination of the small contour process, the contours will be filled by a customised function, fillHole() which builds on top of floodFill() function that provided by OpenCV. This function algorithm is designed to fill multiple contours. Hence, a final segmentation mask is obtained. In addition, when there is a need to segment out multiple traffic signs from a single image, each of the contours will be extracted out to make the final segmentation mask individually. Later, each of the final segmentation masks will segment out each traffic sign from the original multiple traffic signs image.

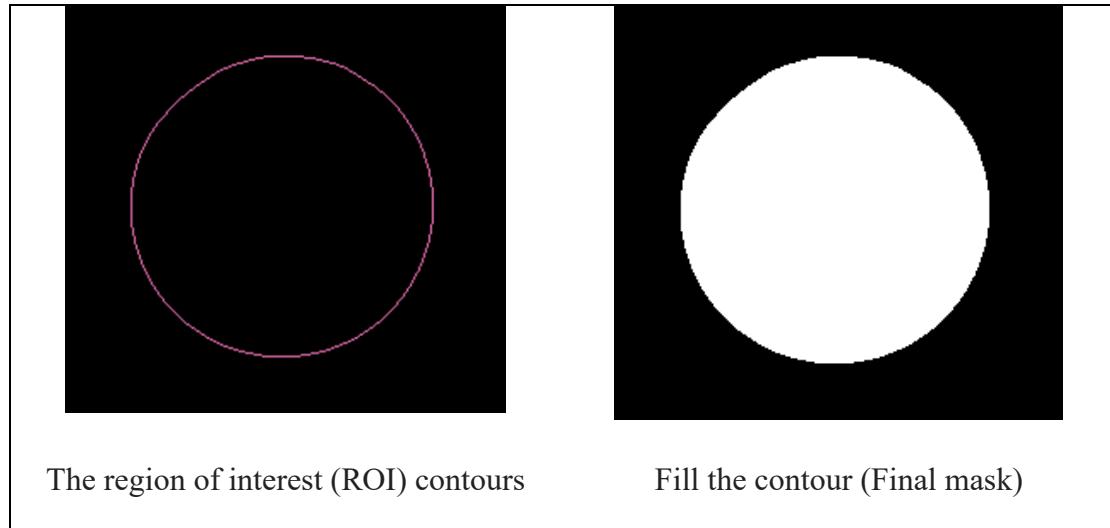


Figure 3.2.7.1 The mask generated from a traffic sign.

3.2.8 Segmentation

[Done By: Tan Jing Jie & Ng Jan Hui]

After the final mask is obtained, the traffic sign can be segment out by applying bitwise AND function to the original input image. Hence, the traffic sign is successfully segmented out from the background.

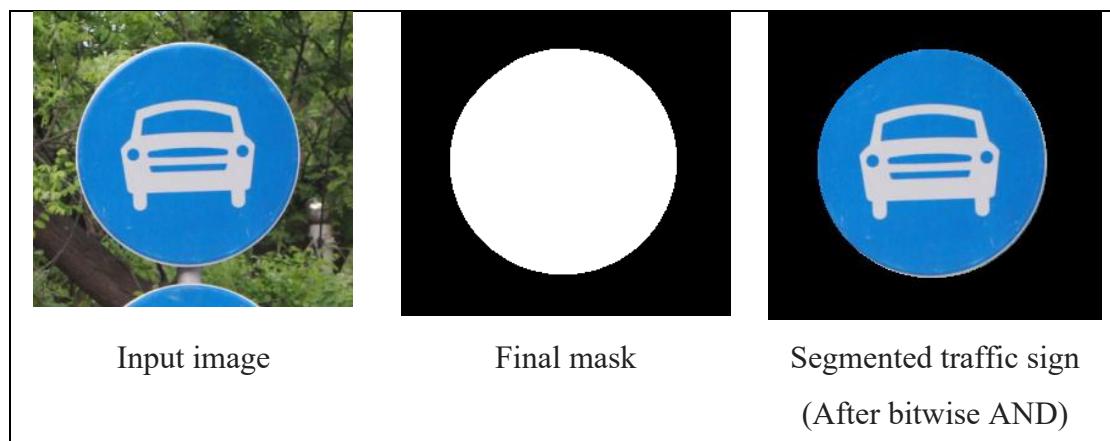


Figure 3.2.2.8 The segmentation process

3.2.9 Further segmentation

[Done By: Tan Jing Jie]

The aim of further segmentation process is to extract out the logo in the traffic sign. Hence, the previous processes (from process 3.2.2 to 3.2.4 & 3.2.7) are repeated to segment out the logo. However, the contour finding, elimination and obtaining mask from the contour process are no longer needed. These processes are significant to extract out the traffic sign from the background. Moreover, during further segmentation, blurring is still needed to remove the lighting noise of the background. In addition, the prior knowledge is one traffic sign only consists of one logo.

There are some changes of parameters to the HSV colour mask. In the first phase of segmentation, red, blue, and yellow colour masks are used. On the other hand, in this further segmentation, white, black and dark blue mask is used. The contours need to be retained completely as they are the main logo in traffic sign which are significant features for classification.

Besides obtaining the logo in the traffic sign, the generated binary colour (pure black and white) mask is also saved. As shown in figure below, the further segmented traffic sign still consists of some blue colour and the logo not in pure white due to the surrounding lighting. However, this information needs to be removed to avoid noise for the feature extraction process. Hence, the binary colour mask is significant for standardisation purposes as it has the form of binary colour of the further segmented traffic sign.

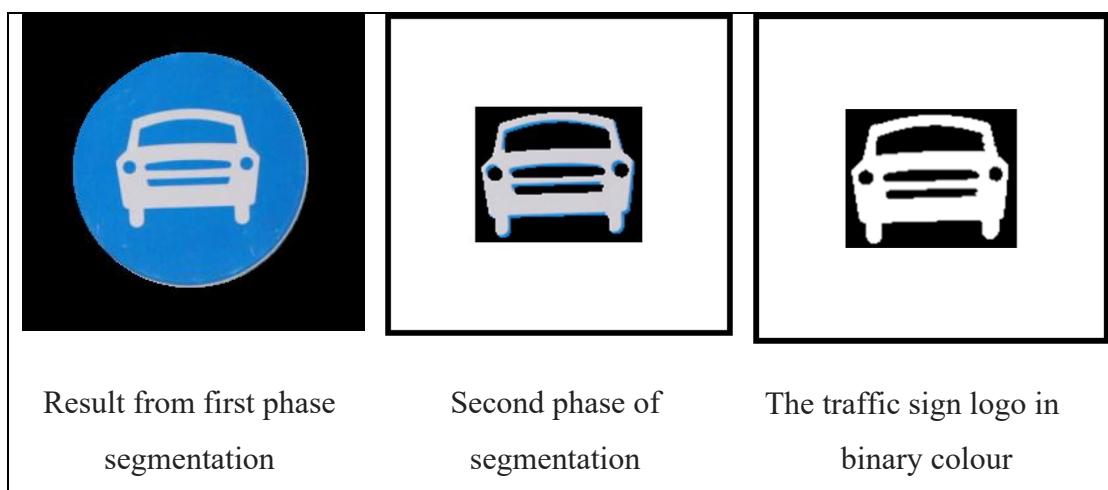


Figure 3.2.9.1 The further segmentation process image

3.2.10 Normalization of image

[Done By: Ng Jan Hui & Tan Jing Jie]

The images within the traffic sign dataset can be of variable sizes. In order to ensure consistent results in feature extraction, standardise the input image sizes is crucial. To achieve this, all image sizes will rescale to 80 x 80 pixels. This particular size ensures that the image quality and features will still be preserved and ensures consistent results during further processing on the pipeline. OpenCV provides the resize() function, and this system use INTER_LINEAR as the interpolation parameter in the resize() to do bilinear interpolation. This normalisation can be used in both the beginning of image segmentation and especially before feature extraction. The figure below shows the process of normalized resize from a further segmentation image.

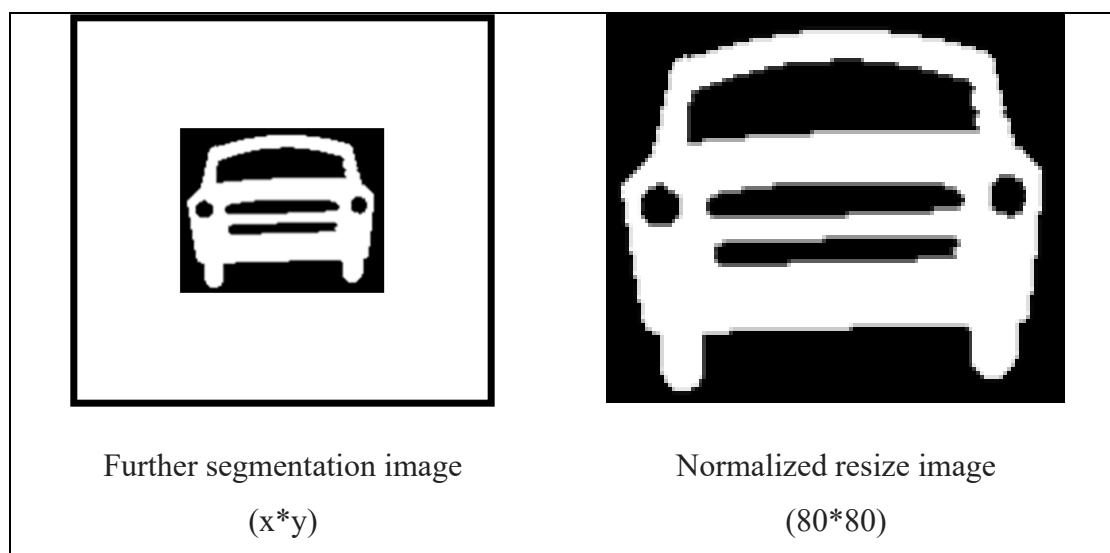


Figure 3.2.10.1 The normalized resize process from further segmentation image

3.2.11 Extracting HoG Descriptors from segmented traffic sign

[Done by: Ng Jan Hui]

After all image preprocessing has been completed, features will be extracted from the initial segments of the traffic sign as well as the internal symbols that are further segmented from the traffic signs.

The feature to be extracted from the initial segment of the traffic signs are the HoG features. HoG or Histogram of oriented gradients are a type of feature descriptor that are tailored for object detection. The object in the case of this project, would be the traffic signs that are to be classified.

HoG descriptors are used because it operates on local cells within the traffic sign image and are invariant to geometric and photometric transformations. This helps the classifier to be able to generalize more to the input images that are placed under various conditions. Besides, HoG features are proven to be highly descriptive and helps to ensure high model prediction accuracy as stated from the works analyzed in the literature review.

OpenCV provides the HoG descriptor extractor object that can be conveniently invoked to extract HoG features from an image. The configurations of the HoG descriptor extractor object will be discussed in detail in Chapter 5 Feature Extraction.

3.2.12 Computing Hu Moments from the internal symbol of traffic sign

[Done by: Ng Jan Hui]

Besides HoG descriptors, the Hu Moments values calculated from the internal symbol that are obtained through further segmentation of the traffic sign, will also be used as another feature type along HoG. The reason of using Hu Moments feature is because the set of 7 Hu Moment values calculate, are also invariant to image transformations, which in turn helps the classifiers to be able to generalize to traffic signs under multiple orientations and conditions.

3.2.13 Concatenating both features into a feature matrix

[Done by: Ng Jan Hui]

After the HoG features and Hu Moments values are generated from the traffic sign samples, both features type will be concatenated. The feature matrix will be of size m by n, where m is the number of features, and n is the number of traffic sign samples. This implies that 1 traffic sign image can be represented using the 7 Hu Moments combined with the HoG features. The corresponding labels for the images will also be encoded to integer values of 1 to 7 and be appended to the last column of the feature matrix.

The feature matrix will then be exported into a CSV file and saved in the same directory. Saving the feature matrix to a CSV file allows the retraining of fine-tuned models, without having the training sample images undergoing segmentation and feature extraction once again. This step can crucially reduce the execution time of the training phase.

3.2.14 Data imputation using Sklearn

[Done by: Ng Jan Hui]

Data imputation will also be performed to fill in the missing values within the CSV file, filling values might occur due to poor segmentation, which results in the hu moments and hog features becoming affected. The imputation strategy employed is filling up the missing values with the median value of each feature column, based on the target label.

The imputed feature dataset will be exported into another CSV file that will be used as the finalized dataset for model training.

3.2.15 Loading dataset and data preparation

[Done by: Ng Jan Hui]

The cleaned feature dataset CSV file will be loaded into the program using the loadCSV() function provided from the OpenCV library, and by specifying the file path. The reason the CSV file is loaded and encapsulated in a TrainData object compared to directly reading using fstream is because, the TrainData class provides data processing functions that will be useful during data splitting and sample retrieval. Moreover, the loadCSV() function will automatically parse the dataset and identifies the last column of the dataset as the labels.

3.2.16 Dataset Splitting

[Done By: Ng Jan Hui]

After the data has been loaded and encapsulated in a TrainData object, the dataset will be split into training and test sets with a ratio of 0.7. This implies that 70% of the data will be used for model training and left out 30% testing set will be as the unseen data to test the trained model's performance.

This data splitting can be achieved by using the function, setTrainTestSplitRatio() from the TrainData class. Afterwards the training set and the labels can be retrieved using the getTrainSamples() and getTrainResponses() function respectively. The training set will be stored in a 2-dimensional Mat while the labels are stored in a 1-dimensional Mat. The similar steps are repeated for the testing set.

3.2.17 Model Initialization

[Done by: Ng Jan Hui]

Three models will be trained to classify the traffic signs. Two standard machine learning classifiers which is the Support Vector Machine, and Random Forest will be used and be trained to recognize traffic signs based on the prepared dataset.

Besides, deep learning will also be explored as a convolutional neural network named MobileNet will be trained to classify the signs. The highlight the deep learning approach is that the MobileNet will be able to learn directly on the sample images without undergoing manual feature extraction.

Multiple classifiers will be used so that a comparison can be made on the performance between the trained classifiers. The hyperparameters that will be set for the models are optimal and were discovered after fine tuning. The choice of hyperparameters will be covered in depth in Chapter 6

3.2.18 Model Training and Testing

[Done by: Ng Jan Hui]

The SVM and Random Forest classifier will then be trained using the prepared training set. The training stop condition would be 1000 Epochs for the SVM and 100 Epochs for the Random Forest. Alternatively, the training would also stop once the accuracy changes between epochs is 0 for the Random Forest and 1e-6 for the SVM.

Once the model training is completed, both models will be saved into an XML file each, to retain the trained model and the weights. Having the trained model allows the direct usage of it during when developing the desktop application.

Then, both models' performance is tested using the left out 30% testing set. The predicted labels for each testing sample by both models are then saved to a 1-dimensional Mat each. The testing accuracy for the models are then printed on the console. The predicted labels by both samples and the ground truth labels will then be exported as CSV files to be analyzed using Python.

3.2.19 Experimental Results Analysis

[Done by: Ng Jan Hui]

After a model is obtain, experimental result analysis will be performed. All exported labels (CSV file) which are from the SVM prediction, Random Forest prediction and the ground truth labels will be read using NumPy. Then, classification metrics such as accuracy, F1 score, precision, recall, and confusion matrix will be computed using the sklearn library. Using the classification metrics, analysis on the models and comparison can be made.

Chapter 4

Image Pre-processing

4.0 Overview

[Done By: Jacynth Tham Ming Quan]

Before the input image can be used for the training or testing process, it has to be pre-processed using techniques such as blurring, HSV colour space segmentation, normalization and etcetera. In this chapter, the image pre-processing techniques used on the input image are described in terms of the methodologies and approaches used, tools and software involved, module requirements, final implementation, verification, and testing.

4.1 Methodology and Tools

[Done By: Jacynth Tham Ming Quan & Por Teong Dean]

The proposed image pre-processing process uses OpenCV libraries to implement pre-processing techniques such as HSV color space segmentation, blurring and normalization to the input images before feeding them into the classification model for training or testing. The image pre-processing process differs slightly from desktop application to mobile application due to the different classification models used. Further design specifications are described in the following subsection.

In the proposed application, the image pre-processing processes for the desktop application and mobile application differed slightly, with only the major components being similar. The tools used to develop this module are as listed below:

1. **Microsoft Visual Studio** – Used as the main IDE for the implementation of the image pre-processing process into the proposed desktop application.
2. **OpenCV** – OpenCV functions such as bilateralFilter, morphologyEx, floodFill, erode, dilate, findContours and etcetera were used for the image pre-processing process

3. **GNU Image Manipulation Program (GIMP)** – Used to explore and identify the colour ranges for red, yellow and blue traffic signs.
4. **Paint** – A secondary program used to explore and identify the colour ranges for red, yellow, and blue traffic signs.
5. **Android Studio** – Used as the main IDE for the implementation of the image pre-processing process into the proposed mobile application. In order to incorporate OpenCV image pre-processing functions into Android Studio, OpenCV Java wrappers were used. Examples of the OpenCV libraries and corresponding functions used include Imgproc (morphologyEx, cvtColor, erode, dilate, Canny, findContours) and Core (bitwise_or, bitwise_and).

4.2 Requirement

[Done By: Jacynth Tham Ming Quan & Ng Jan Hui]

The goal of the image pre-processing stage is to be able to detect red, yellow, and blue traffic signs from the input image, segment them out from the background, extract, and further segment out the regions of interest (the unique icon that is contained within each traffic sign). Moreover, the segmented sign at the last stage of the pre-processing process should be normalized so that all the segmented traffic signs are of the same size. This simplifies the feature extraction stage in the following chapter. The final result of the image pre-processing stage should be a normalized, black-and-white segmented image containing the detected traffic sign's unique icon.

4.3 Design Block Diagram

[Done by: Tan Jing Jie]

The basic pipeline of image pre-processing is tabulated as below. The green-shaded processes are preparing, performing, and processing colour masks. Besides, blue-coloured processes involve the algorithm of performing multiple traffic signs and selecting out the region of interest (ROI). Lastly, the red-coloured processes are segment out the traffic sign or its logo respectively by using the obtained mask. The

size of the segmented image will be standardized for the feature extraction process. The novelty of this image pre-processing is to be able segment out multiple and different colour traffic signs in one input image.

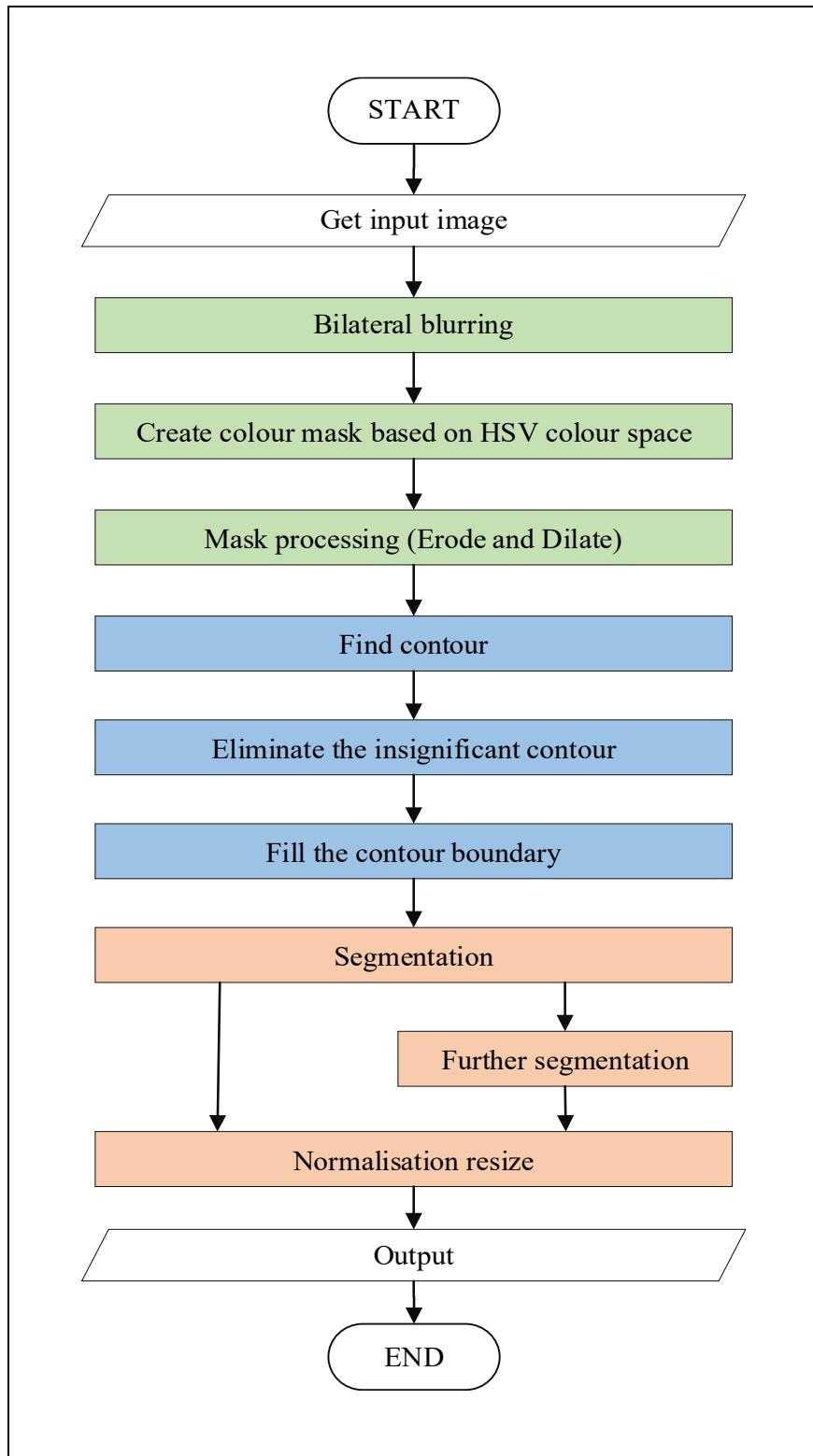


Figure 4.3.1.1 The overall flow of image segmentation.

4.4 Verification Plan

[Done By: Jacynth Tham Ming Quan, Por Teong Dean & Tan Wei Mun]

The proposed test cases for the image pre-processing stage are tabulated below in Table 4.4.1 (desktop application). These test cases were noted of during the implementation of the image pre-processing stage and further tested during the testing and validation stage.

Table 4.4.1 Image Pre-Processing Test Cases (Desktop Application)

Id	Test Case	Expected Results
1	Input image contains a single red traffic sign	Output a segmented image containing only the unique icon within the traffic sign (black-and-white)
2	Input image contains a single yellow traffic sign	Output a segmented image containing only the unique icon within the traffic sign (black-and-white)
3	Input image contains a single blue traffic sign	Output a segmented image containing only the unique icon within the traffic sign (black-and-white)
4	Input image contains traffic signs of varying colours	Output multiple same-sized, segmented images containing the unique icons of each detected traffic sign.
5	Input image contains multiple overlapped traffic signs of varying colours	Output multiple same-sized, segmented images containing the unique icons of each detected traffic sign and not consist of other overlapped traffic sign
6	Input image contain small size traffic sign	Output enlarged detected traffic sign.

The proposed test cases above will be tested in the testing and validation subsection in this chapter.

4.5 Implementation

[Done By: Jacynth Tham Ming Quan & Tan Jing Jie]

In this subsection, the full implementation of the image pre-processing stage is described step by step, starting with blurring the input image, all the way up to HSV colour space segmentation, noise reduction, contours finding and further segmentation. This image pre-processing target to segment out multiple or single traffic signs from a single image. The full flow of the image pre-processing step can be observed in Appendix C.

4.5.1 The Bilateral Blurring filter

[Done By: Tan Jing Jie]

First and foremost, an input of traffic sign images is given. The image will first apply a Bilateral blurring filter. This filter is able to reduce significant noise especially from the traffic sign background. The benefit of using this Bilateral blurring filter is when there are a lot of different colour pixels located in a small area and the blurring only starts to be obvious in this area. Unlike other filters such as Homogeneous blur and Gaussian filter, the entire image would be affected and cause the loss of clear contour. The example shows as the figure below, the blurring caused the traffic sign to lose its contours information, and it led to the deformity of the mask and having noises.

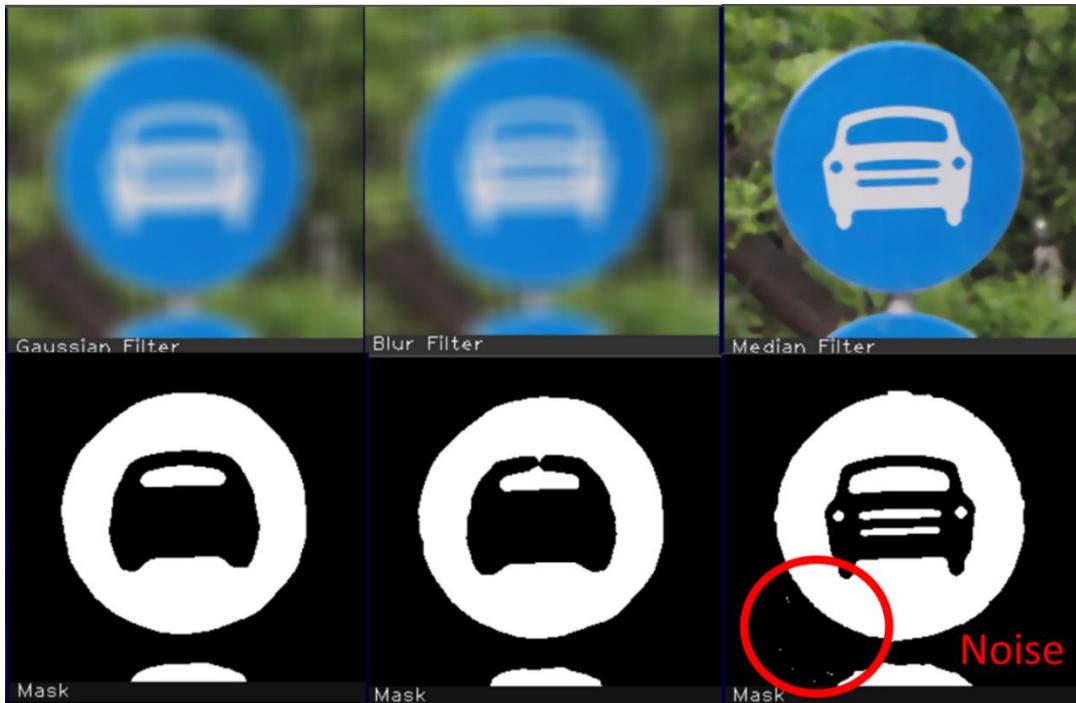


Figure 4.5.1.1 The comparison of other blurring filter result and its generated mask.

As shown as the figure below, bilateral blur only makes the background blur while the traffic sign is almost preserved. The parameter used for sigma colour is 63 while the parameter for sigma space is 7. The sigma colour parameter is used to control the colour space – the larger the parameter value, the farther the colour in the neighbourhood (this is controlled by colour space parameter) will be mixed together. On the other hand, the sigma space parameter controls the neighbourhood size – the larger the parameter value, the larger the influence area by the pixel colour. Hence, as to preserve the contour of the traffic sign, the small sigma space area is needed to avoid the blurring of the contour. In addition, the sigma colour parameter needs to be large for blurring to take place effectively.



Figure 4.5.1.2 The comparison between the before and after applying Bilateral blurring filter to a traffic sign.

4.5.2 The Mask Generated by HSV Colour Space

[Done By: Tan Jing Jie & Jacynth Tham Ming Quan]

Moving on, the traffic sign is converted to Hue Saturation Value (HSV) colour space by using cvtColor() method. The hue range in OpenCV is only 0-180, not the common of 0-255, while saturation and value still remain in the range of 0-255. However, OpenCV does provide a parameter, COLOR_BGR2HSV to assist the conversion from the usual Red Green Blue (RGB) colour space image into HSV colour space.

Next, 10 random traffic sign images of each colour (red, blue, and yellow) were extracted from the traffic sign dataset and examined in GIMP. This step is key to determining the HSV thresholds to be used when extracting the colour mask from the input image. Figure 4.6.2.1 below shows an example view of the HSV values of a traffic sign's yellow pixel:

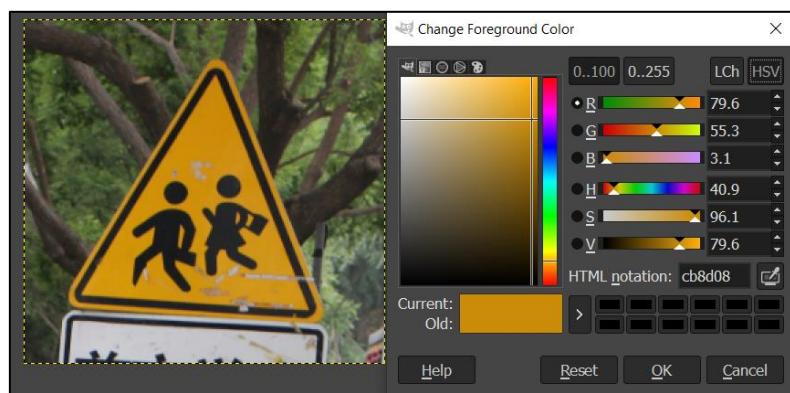


Figure 4.5.2.1 GIMP Color Analysis View on a Yellow Pixel

Upon thorough research, it was discovered that different applications use different HSV scales. For example, GIMP uses H = 0 – 360, S = 0 – 100 and V = 0 – 100 while OpenCV uses H: 0 – 180, S: 0 – 255, V: 0 – 255. Therefore, a custom function named RGB2HSV was declared to handle the conversion of RGB pixels to HSV pixels. In the custom function, the HSV values supported match that of GIMP's HSV range.

The range can be defined in C++ program as using the Scalar data type. The upper and lower HSV thresholds are tabulated in Table 4.2.3.1 below:

Table 4.5.2.1 The colour range defined for each mask

Colour Range	Hue	Saturation	Value
Red mask	150-179	140-255	160-255
	0-3	50-255	50-255
Blue mask	100-128	110-255	100-255
Yellow mask	14-30	100-255	140-255

There are two range of red colour mask, which is 150-179 and 0-3 for Hue, this is because in HSV colour space the start and end is surround in red colour which illustrate as the colour picker in Window Paint program (Noted that the Hue range for Paint program is 0-239).

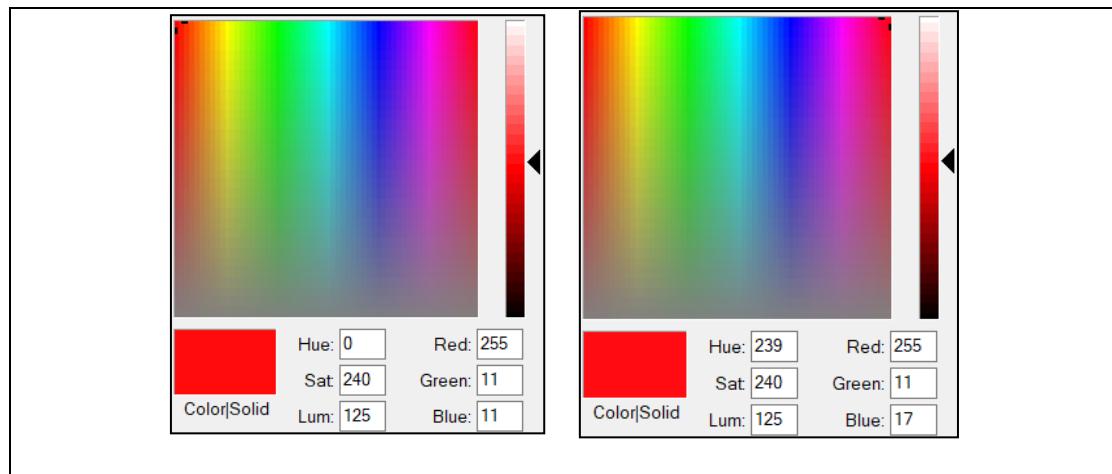


Figure 4.5.2.2 Colour picker show the Hue and Saturation of red colour bound at left and right of the colour space

The importance of combining two red masks is because both ranges are significant. As we can see in the figure below, both colour ranges are important. Hence, it is necessary to combine both colour masks.

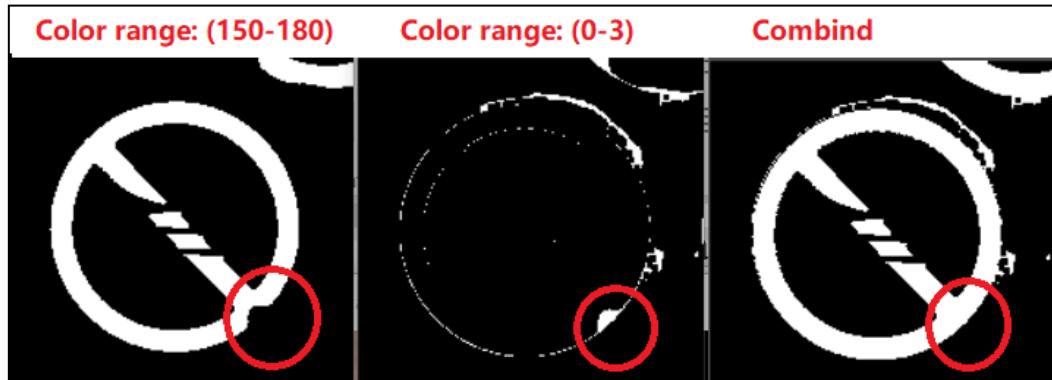


Figure 4.5.3.3 The mask of the hue range of 150-180 and 0-3 is combined to form valid red colour mask

Moving on, a mask result will be created using the `inRange()` function provided by OpenCV. There are 4 parameters in this function including, the HSV image, the minimum and maximum Scalar value and lastly the MAT data type which is used for output destination – the mask. This function is executed 4 times, 2 times for red colour mask, and 1 time for each blue colour mask and yellow colour mask. The output masks are in binary (black and white). Continuously, each of the masks will undergo `morphologyEx()` function which is used to eliminate the noise. The image below shows the result of the blue mask before and after applying `morphologyEx()` with `MORPH_CLOSE` parameter.

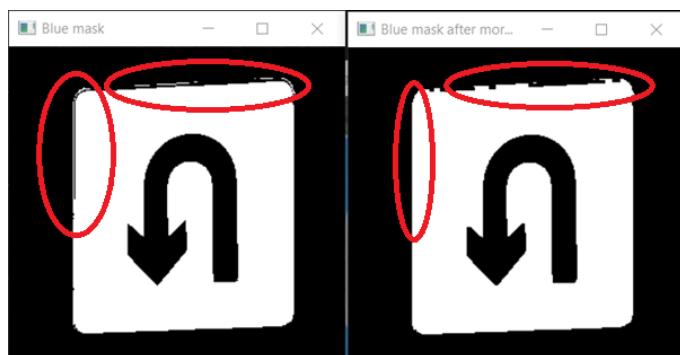


Figure 4.5.3.4 The comparison result of the before and after of the blue mask after applying morphology close function.

The `morphologyEx()` function needs to be done before the combination of masks. This will increase the chance to provide a more fitted mask while reducing the noise. This is important to remove noise before combining the mask due to the noise might be amplified and make it harder to remove.

After that, the four masks are combined by using an “or” logical operator, $mask = redMask1 | redMask2 | blueMask | yellowMask$. This combination is able to let the segmentation system fit the colour mask to each traffic sign with no need to concern about the major boundary colour of the input image. For example, if there is only a red traffic sign inside the image, then blue colour and yellow colour mask will result in only black and which will not affect the final result. Moreover, this step is crucial for doing multiple different colour traffic sign segmentation and detection.

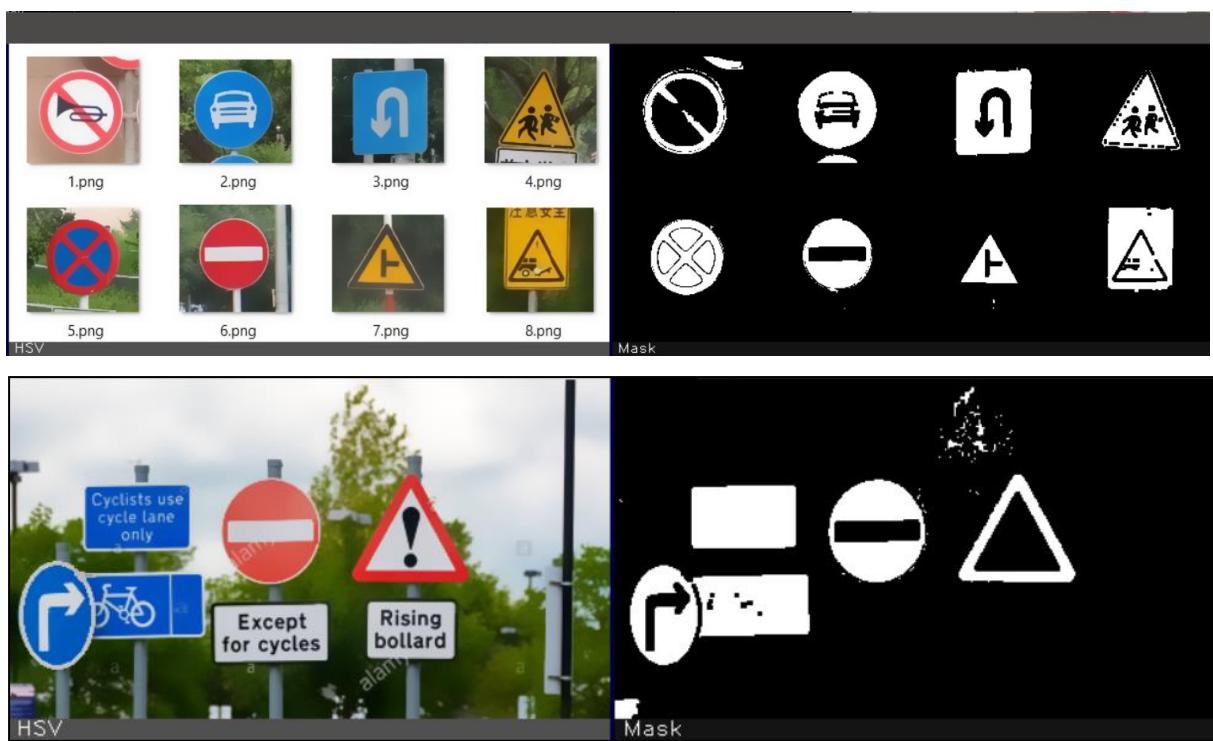


Figure 4.5.3.5 The combination of mask to segment multiple different colour traffic signs

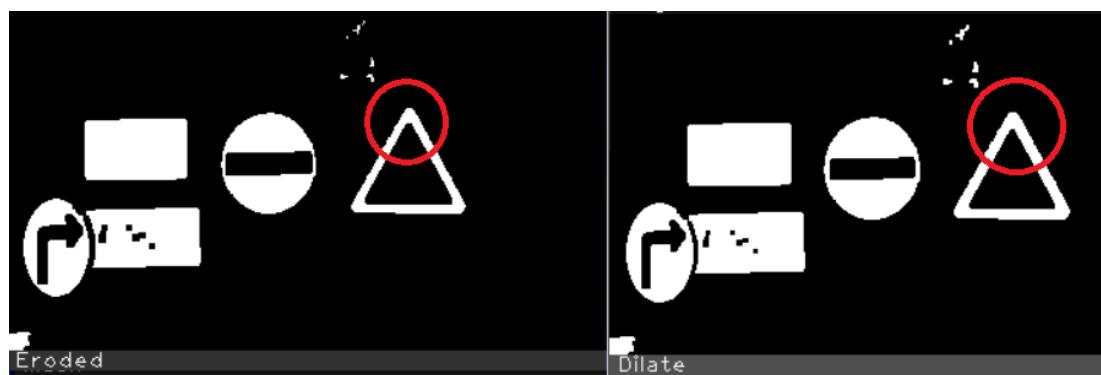
4.5.3 The Mask Processing to Eliminate Noise

[Done By: Tan Jing Jie]

However, the noise is amplified when the background consists of pure blue or pure yellow area. Hence, the erode method and dilate method need to be done. The erode will turn the overall mask shirk, thus able to remove those tiny pixels surrounding the main object. Subsequently, a dilate function is executed to recover back the pixel loss during the shrink stage. Meanwhile, when the pixel is removed in the shrink stage, it will not be enlarged again in dilate function. The figure below shows the mask undergoing the erosion (remove noise) and dilating (recover back the real thickness) process.



Erode of mask, removed the noise.



Dilate the mask, recover back the thickness of contour.

Figure 4.5.3.1 The mask undergo the erode and dilate process.

4.5.4 The Contour Finding

[Done By: Tan Jing Jie & Tan Wei Mun & Por Teong Dean]

The process followed by contour finding. By using the `findContour()` provided by OpenCV, all the mask's contours have been drawn out. The `RETR_EXTERNAL` flag was used to ignore the boundary inside of the boundary (as shown in the figure below, pointed by red colour arrow). In other word, child contour will be ignored, and parent contour will be remained. This `findContours` function only remain the outer most contour. In other words, this flag only obtains the eldest boundary and ignores the child boundary. However, it is noticed that there are few contours which obviously were noise (in red circle), and it will be further processed in subsequent steps.

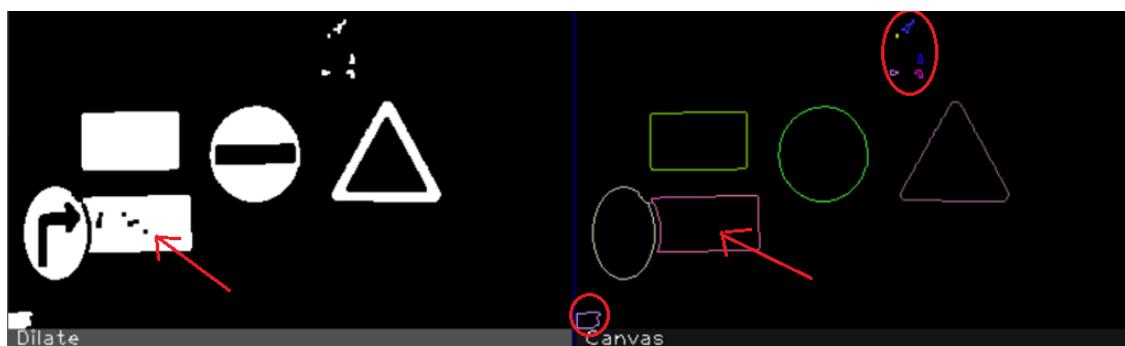


Figure 4.5.4.1 The finding the contours from the masks.

4.5.5 The Elimination of Insignificant Contour

[Done By: Tan Jing Jie]

As mentioned above, there is noise in contours, which is irrelevant to traffic signs. Hence a custom algorithm is developed to eliminate the noise. First, a Boolean function of contours size comparison function, `compareContourAreas()` is prepared. The function will check the area by using OpenCV function, `contourArea()` and return true for `contour1 < contour2`. This function will be executed, and the contours array is sorted.

Moving on, the elimination algorithm will take place. This elimination algorithm is built by comparing each contour size with the largest image in the same picture. In this system, a range of 1/6 times of the biggest contours are considered acceptable contours, otherwise, the contours will be removed. By using this function, it

can eliminate most of the insignificant noise. The only drawback is that the threshold needs to be adjusted to work well for detecting multiple traffic signs with huge size differences. However, this elimination can be considered good as in real life only closed traffic signs are concerned, while far away traffic signs can wait until the pass of the nearest traffic sign.

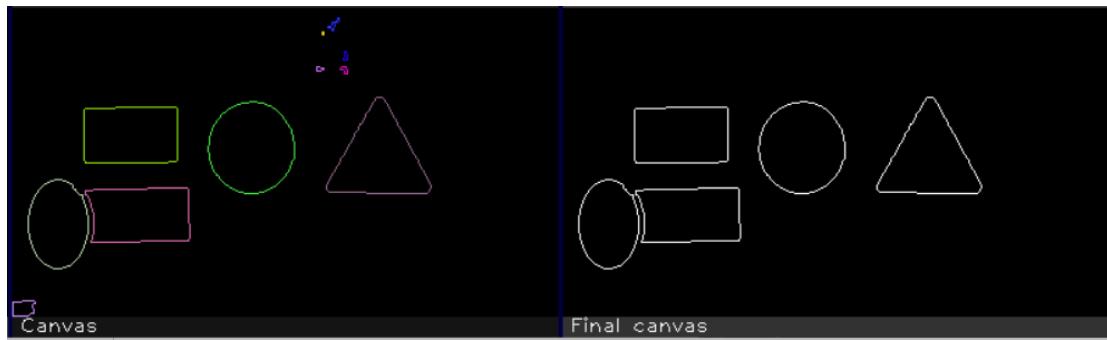


Figure 4.5.4.1 The final canvas contour after eliminating noise

4.5.6 The Filling of The Contour

[Done By: Tan Jing Jie & Jacynth Tham Ming Quan]

Hence, a final canvas mask is produced from the contours. A customized fillHole() function is applied to the contour. In depth, fillHole() function is builded based on the OpenCV floodFill() function. The floodFill() function cannot be directly applied because its nature needs to pinpoint a coordinate which is connected to the field which wants to be filled. Besides, there are multiple contours that need to be filled, and it is tedious to find out the internal point of each contour. Thus, a short algorithm, fillHole() is applied to extract out the final image by using bitwise OR and NOT function. In the fillHole() function, a Mat of zeroes is created following the dimensions of the input image + 2. Then, the Mat of zeroes is converted to the type of the input image, which turns it into a binary image, before undergoing the floodFill() function. The figure below shows the flow of the process of fillHole() function.

Chapter 4 Image Pre-processing

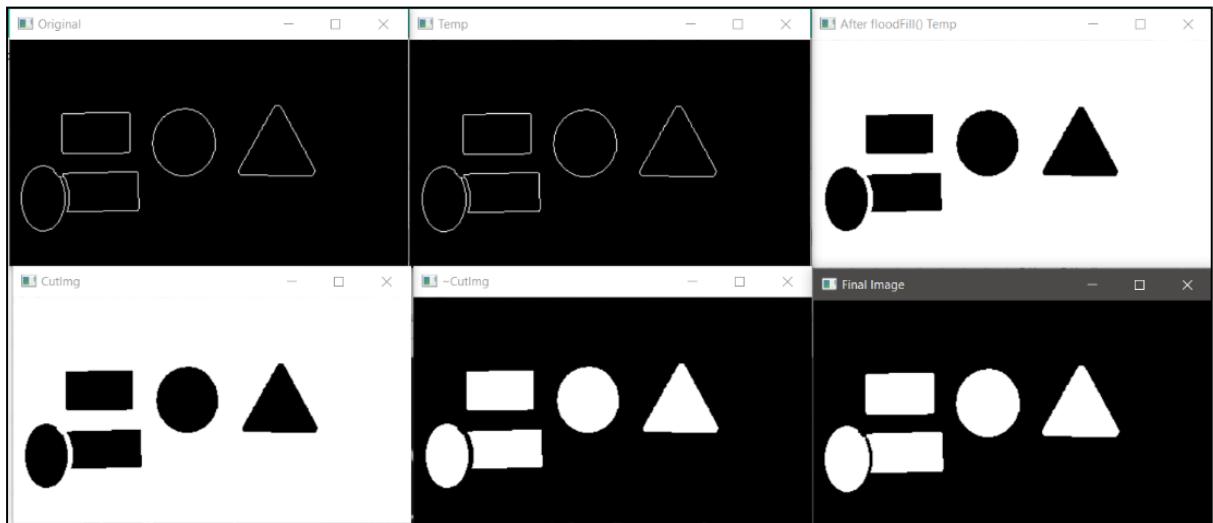


Figure 4.5.6.1 The detail process of fillHole() function

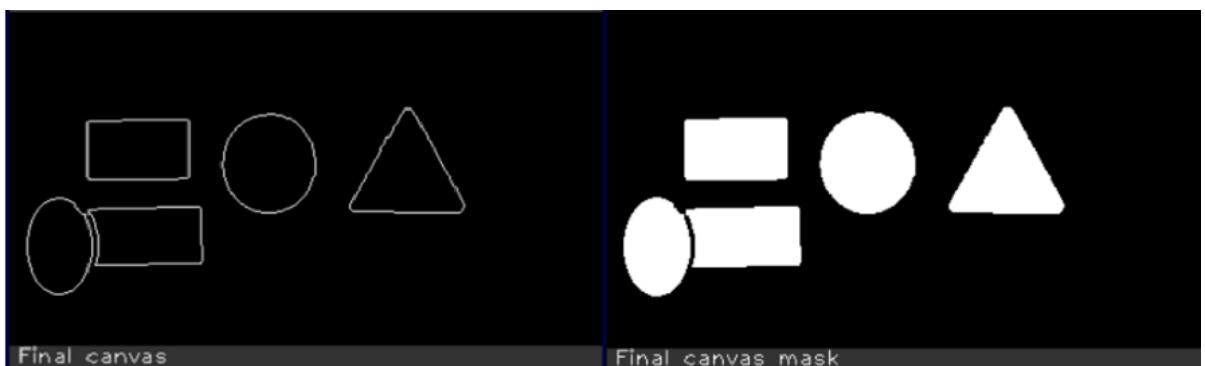


Figure 4.5.6.2 The conversion from final contour to final canvas mask.

4.5.7 The Segmentation by using Canvas Mask

[Done By: Tan Jing Jie & Jacynth Tham Ming Quan]

Moving on, a bitwise AND is performed on both original (source) image and the final canvas mask as *segmented = filledCanvasMask & srcI*. Hence, a segmented result is obtained as shown in figure below.



Figure 4.5.6.3 The conversion from final canvas to final canvas mask.

Next, two functions are used to locate the contours within the input image – the approxPolyDP() function and Rectangle constructor.

1. approxPolyDP() – This function performs contour approximation to each of the contours found. It approximates a contour shape to another shape with a lesser number of vertices specified in the epsilon parameter, which is the third input parameter to this function. Here, an epsilon of 3 is selected, which denotes the maximum distance between the original contour and the approximation made by the function. The last parameter, representing *closed*, is set to true, which means that the function's approximated contours have been closed (last vertex and first vertex are connected). This function is to facilitate the next function – boundingRect().
2. boundingRect() – This function calculates the bounding rectangle of the non-zero pixels of a gray-scale image. The output of approxPolyDP is set as the input parameter of this function, so that the bounding box drawn is based on the contour approximation previously found.

First, approxPolyDP() function aids to find out significant edges for later rectangle construction. There are 2 parameters in this function which are:

1. Epsilon – This parameter controls the maximum range of each point of the contour. The lower the epsilon value, the narrow the range of acceptance.
2. Closed – This parameter defines whether the contours are closed. When this parameter is set to true, which means that the function's approximated contours are closed (first vertex and last vertex are connected).

The following figure show the epsilon value = 9 as to show the significant difference. However, epsilon value = 1 will be used in formal.

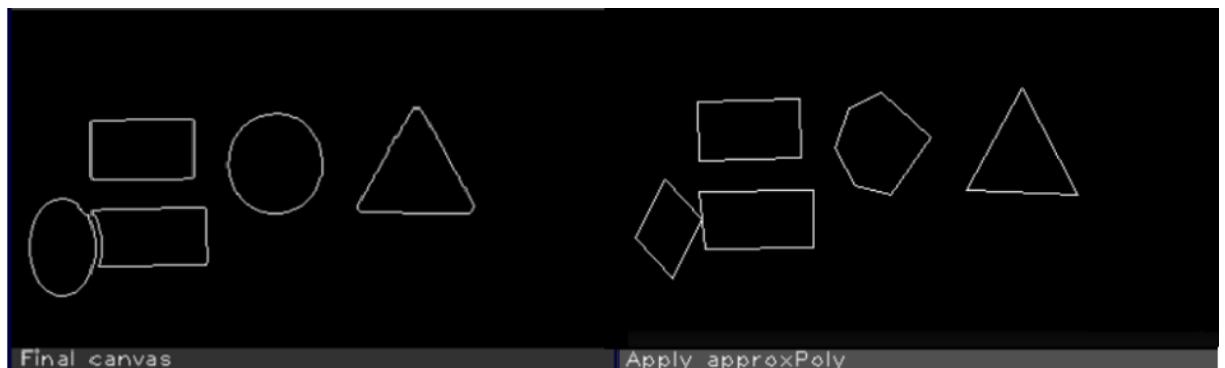


Figure 4.5.6.4 The example approxPolyDP() function applied with epsilon value = 9.

The ultimate purpose of image pre-processing step is to prepare a high-quality image for future classification purpose. Hence, it is important to export out each traffic sign in multiple image segmentation. However, it is not allowed to use bounding boxes as directly cropped out from the segmented images as there is a situation that the traffic sign is curvily overlapped. Hence, a for loop of contours is needed to prepare each mask for each segmented traffic sign. Indeed, a fillHole() function is required for each contour mask. After segmented each traffic sign, each traffic sign can be exported out in the computer directory. The figure below shows the output of each segmented image.

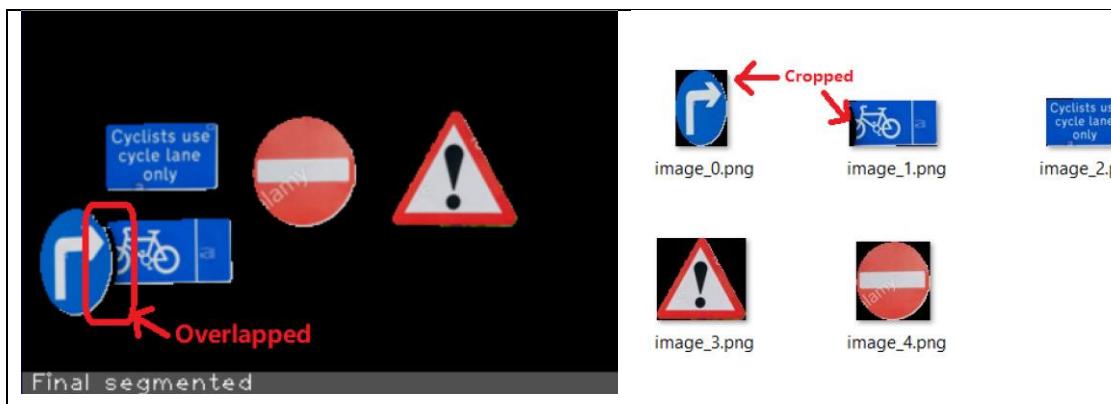


Figure 4.5.7.1 The segmentation of each image and saved into local directory.

4.5.8 The Further segmentation

[Done By: Tan Jing Jie]

After segmenting out the traffic sign from the background, further segmentation process was required to obtain the major information (logo) in the traffic sign. In other words, the background (red, blue or yellow) of the traffic sign logo is removed. This is significant to provide more features for the feature extraction process. The pipeline of the further segmentation process is similar to the initial segmentation step above (subsection 4.5.1 to 4.5.3, and 4.5.7). The difference is that the contour finding and elimination are no longer needed as there is only 1 logo in each traffic sign. Besides, the major difference is that the colour masks are changed to black, white, and blue. The respective colour range parameters are tabulated in the table below:

Table 4.5.8.1 The colour range defined for the further segmented mask

Colour Range	Hue	Saturation	Value
White mask	0-180	0-50	150-255
Black mask	0-180	0-100	0-60
Blue mask	104-110	236-255	0-255

Hence, after doing a bitwise AND operation by using the generated mask above with the original segmented image, the further-segmented traffic sign images (the logo) were obtained. Moving on, the binary image of the traffic sign logo can be obtained by its further segmentation mask. Most of the traffic sign's main logo is in black and white, however, as shown in further segmented traffic signs below, the logos are not in pure white or black due to lighting issues in real time. Hence, binary image is significant to provide standardisation for feature extraction. The figure below shows the result after undergoing the further segmentation.

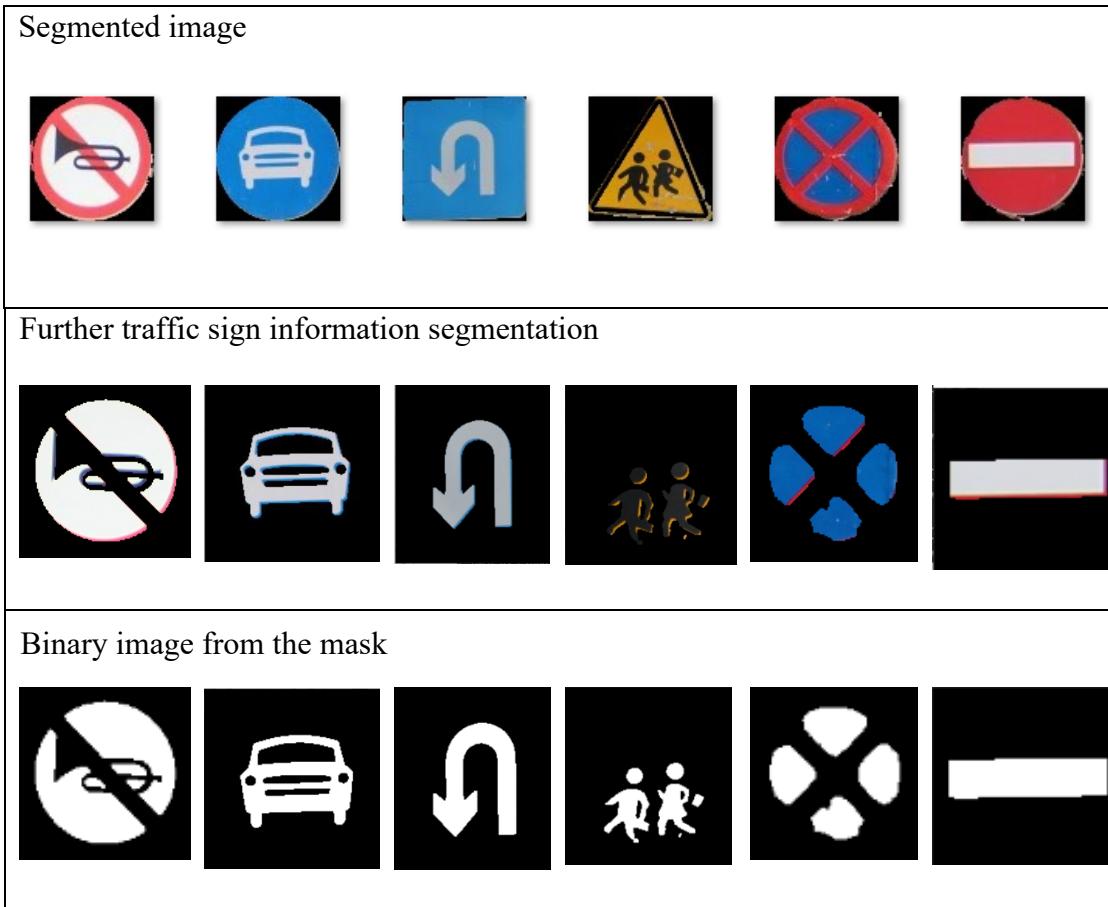


Figure 4.5.8.1 The 6 traffic sign images undergoing further segmentation.

4.5.9 The Normalisation Resize

[Done By: Ng Jan Hui & Tan Jing Jie]

Since the segmented traffic signs will be of varying size, resizing of all images to a size of 80 by 80 pixels will be done using bilinear interpolation. Normalizing all the sizes of the segmented signs and segmented logo, ensures that the feature extraction algorithm will generate consistent features which significant during model training.

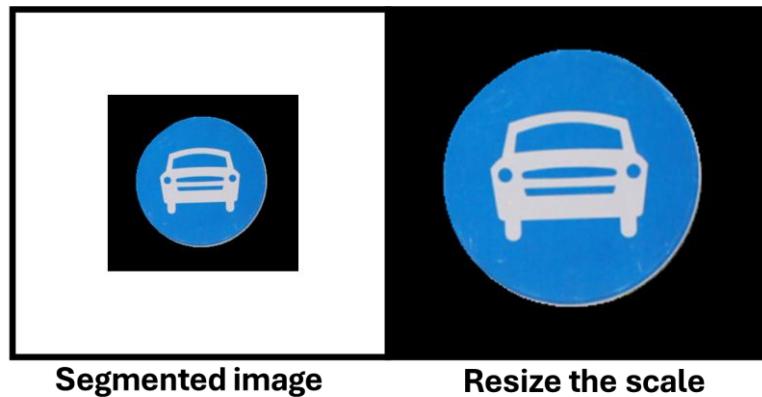


Figure 4.5.9.1 The resize of segmented traffic sign.

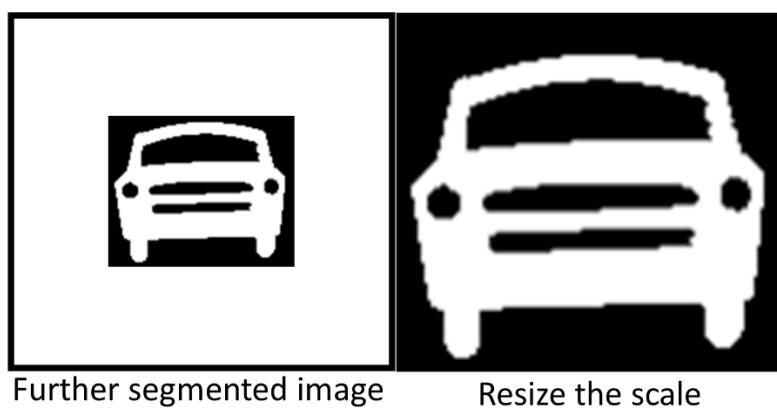


Figure 4.5.9.2 The resize of further segmented binary colour traffic sign (logo).

Lastly, the segmented image will bring to the next stage – feature extraction. In order to show the process of image pre-processing, the output of the flow will be exported into a folder called “result”. The segmented and further segmented result will also save into the respective folder.

D:\ > MiniProject > Project452 > Project452 > Output > Img_1	
Name	Date modified
BinarySegment	9/9/2021 11:56 AM
FurtherSegmented	9/9/2021 11:56 AM
Result	9/9/2021 11:56 AM
Segmented	9/9/2021 11:56 AM

Figure 4.5.9.3 Export the image preprocessing flow and the image preprocessing result.

4.6 Testing

[Done By: Tan Jing Jie & Jacynth Tham Ming Quan & Tan Wei Mun]

Listed below are the system's image pre-processing results when tested on the test cases described in subsection 4.4. The full flow of image pre-processing process can be found in [Appendix B.1](#).

Table 4.6.1 Test Case: Red Traffic Sign Input

Test Case Name	Red traffic sign input
Test Case Description	Input image contains a single red traffic sign
Expected Output	Output a segmented image containing only the binary (black-and-white) unique icon within the traffic sign
Input	
Results	
Status (Pass/Fail)	Pass

Table 4.6.2 Test Case: Blue Traffic Sign Input

Test Case Name	Blue traffic sign input
Test Case Description	Input image contains a single blue traffic sign
Expected Output	Output a segmented image containing only the unique icon within the traffic sign (black-and-white)
Input	
Results	
Status (Pass/Fail)	Pass

Table 4.6.3 Test Case: Yellow Traffic Sign Input

Test Case Name	Yellow traffic sign input
Test Case Description	Input image contains a single yellow traffic sign
Expected Output	Output a segmented image containing only the unique icon within the traffic sign (black-and-white)
Input	
Results	
Status (Pass/Fail)	Pass

Table 4.6.4 Test Case: Multiple Traffic Signs Input of Varying Colours

Test Case Name	Multiple Traffic Signs Input of Varying Colours
Test Case Description	Input image contains multiple traffic signs of varying colours
Expected Output	Output multiple same-sized, segmented images containing the unique icons of each detected traffic sign.
Input	 <p>1.png</p>  <p>2.png</p>  <p>3.png</p>  <p>4.png</p>  <p>5.png</p>  <p>6.png</p>
Results	    
Status (Pass/Fail)	Pass

Table 4.6.5 Test Case: Multiple Overlapped Traffic Signs Input of Varying Colours

Test Case Name	Multiple Overlapped Traffic Signs Input of Varying Colours
Test Case Description	Input image contains multiple overlapped traffic signs.
Expected Output	Output multiple same-sized, segmented images containing the unique icons of each detected traffic sign and not consist other overlapped traffic sign.
Input	
Results	
Status (Pass/Fail)	Pass

Table 4.6.6 Test Case: Small Sized Traffic Signs Input

Test Case Name	Small Sized Traffic Signs Input
Test Case Description	Input image contains small traffic signs
Expected Output	Output enlarged detected traffic sign.
Input	
Results	
Status (Pass/Fail)	Pass

4.7 Summary

[Done By: Jacynth Tham Ming Quan & Tan Jing Jie]

In short, the image pre-processing stage was able to segment out traffic signs using HSV colour space segmentation before removing the noise present, obtaining the contours of the possible traffic signs, and further segmenting the traffic signs. The results of the image pre-processing stage are the unique icons segmented out from each detected traffic sign. The testing and validation process evidenced that the image pre-processing stage works as intended and is able to segment traffic signs varying in colour (red, blue, and yellow) and size. Besides, this image pre-processing works well in segmenting multiple traffic signs in single image. In the next chapter, the prominent features of the segmented traffic signs are extracted to be used in the training and testing of the classification model. The full flow of the image pre-processing step can be observed in Appendix C.

Chapter 5

Feature Extraction

5.0 Overview

[Done by: Ng Jan Hui]

To train the SVM and Random Forest classifier to recognize the traffic signs, it must first learn based on the features of the traffic sign. In this Chapter, the methodologies and flow to extract two feature types from a traffic sign will be covered. The two feature types to be extracted are the object detection HOG features and the shape-based Hu Moment values.

HoG features are used instead of object detection features because HoG features can be extracted directly without performing tedious Bag of Feature transformation that involves k means clustering which are needed for SIFT, SURF and LBP features.

5.1 Methodology and Tools

[Done by: Tan Jing Jie & Jacynth Tham Ming Quan]

The project's feature extraction process uses OpenCV libraries to implement feature extraction techniques. In this system, the feature extraction processes will be applied for both training and testing (application) phase. This stage will extract the significant information from both the segmented traffic sign and segmented logo. The tools used to develop this module are as listed below:

1. **Microsoft Visual Studio** – Used as the main IDE for the implementation of the feature extraction process into the desktop application.
2. **OpenCV** – OpenCV functions such as Hu Moment, HoGDescriptor and etcetera were used for the feature extraction process.

5.2 Requirement

[Done by: Ng Jan Hui & Por Teong Dean]

The outcome of the feature extraction module is to be able to generate the 7 Hu Moments and 441 HoG features from a segmented traffic sign image. The extracted features will be used for training the classifiers. The feature matrix created by concatenating the features vectors for HoG and Hu moments, will also be saved into a CSV file, so that future model retraining can be done without having the images to undergo segmentation and feature extraction once again.

5.3 Design Block Diagram

[Done by: Ng Jan Hui & Tan Wei Mun & Tan Jing Jie]

Figure 5.2.1 depicts the flow of the feature extraction module. HoG features will be extracted from the segmented traffic sign while the Hu Moments will be calculated from the further segmented symbols within the sign. Both the HoG features and Hu Moments will be concatenated into a large feature matrix and be exported as a CSV file to be loaded for training.

The usage of the HoG features is because of the selected six types of traffic signs having varying shapes which are triangle, square, and circle shapes. HoG, otherwise known as Histogram of Gradients, is a descriptor that focuses on the structure and shape of an object. HoG has advantages over edge detection features as it extracts the gradient and the orientation of edges in regions broken down from an image. HoG features will also generate a histogram for each region by using the gradients and orientations for each pixel value. HoG feature descriptor will essentially count the occurrences of gradient orientations in localized portions of images, hence its name.

Hu Moments features are utilized because the shape of the internal symbols of similar traffic signs, which are found under various orientations and scales, can be described using similar set of values. In other words, the Hu Moments values for the internal symbol of a traffic signs are guaranteed to be constant, no matter how it is shifted, rotated, or scaled.

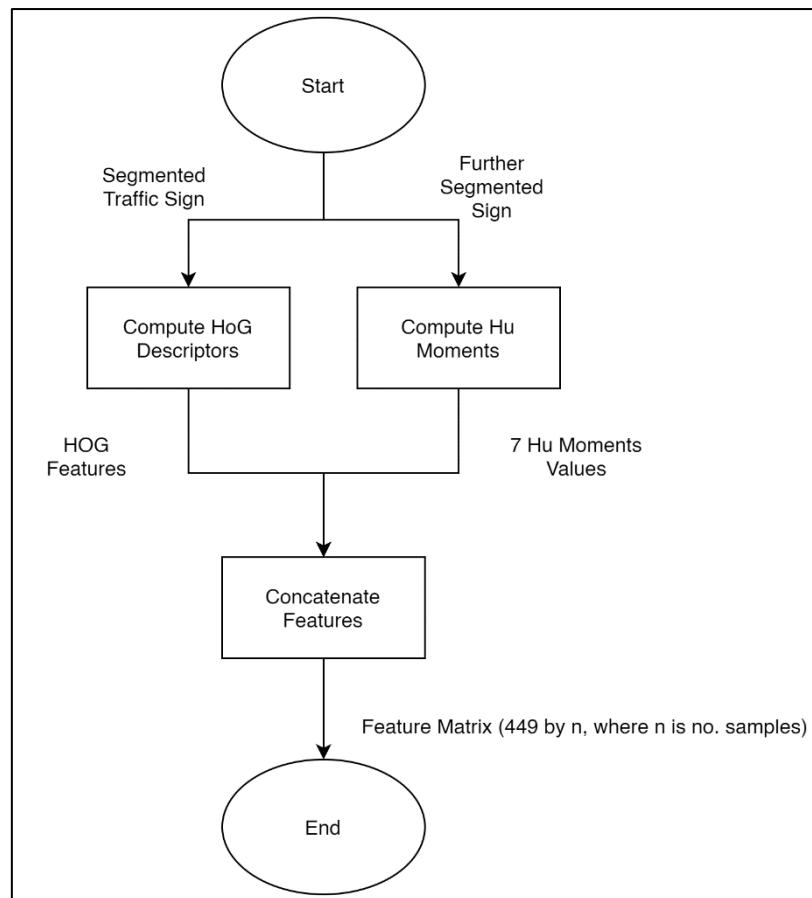


Figure 5.3.1 Feature Extraction Flow

5.4 Verification Plan

[Done by: Ng Jan Hui, Tan Jing Jie & Jacynth Tham Ming Quan]

The test cases for the feature extraction stage are tabulated below in Table 5.4.1 (desktop application). These test cases were noted of during the implementation of the feature extraction stage and further tested during the testing and validation stage.

Table 5.4.1 Feature Extraction Test Cases (Desktop Application)

Id	Test Case	Expected Results
1	Compute HoG features	441 HoG features from a traffic sign segment
2	Compute Hu Moments	7 Hu Moments value from the internal symbol of a traffic sign
3	Concatenate Features	Feature matrix of size, 441 HoG + 7 Hu moments + 1 Label by no. sample
4	Export to CSV	CSV File identical to the feature matrix

The test cases above will be tested in the testing and validation subsection in this chapter.

5.5 Implementation

5.5.1 Generating HoG and Hu moments features

[Done by: Ng Jan Hui]

Firstly, the initial segments and further segments of input traffic signs are stored in two separate vectors respectively after the segmentation phase. Then, a self-defined function named extractFeatureFromTrafficSign() is written, the function takes in the initial segment vector and further segment vector as inputs. The function returns by reference, a vector of double vectors called huMomentFeatures and a vector of float vectors called HoGfeatures.

Starting off, the function will loop over the furtherSegments vector and computes the seven hu moment values of the internal symbols segmented from each traffic signs. The computed hu moments are then log transformed and are pushed into the huMomentFeatures variable in a row-wise manner.

The function then continues to loop through the initial segments vector. Afterwards, a HoG object provided from the OpenCV object detection library is instantiated. The configurations of the HoG object are set to window size of 80 by 80, cell size of 32 by 32, block size of 32 by 32 and block stride of 8 by 8. Using these configurations will generate 441 HoG features from a traffic sign of size 80 by 80. Each initial segment Mat within the vector is resized into size 80 by 80 using bilinear interpolation.

The extracted HoG descriptors are then stored pushed into the HoGfeatures variable also in a row-wise manner. The output of the function is depicted in figure 5.5.1

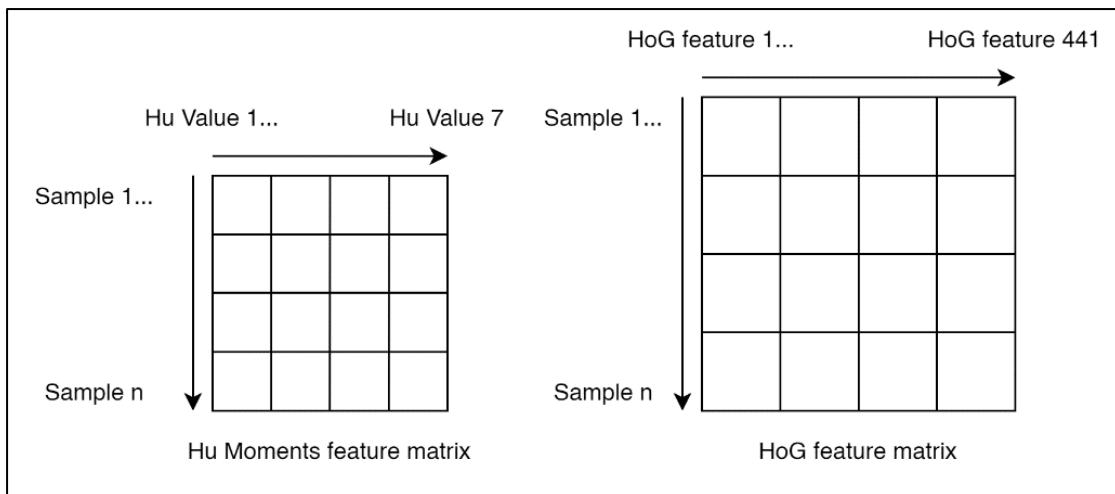


Figure 5.5.1 Feature matrices generated from traffic signs samples

Both feature matrices will then be concatenated into one feature matrix that is of size $(441 + 7)$ by n , where n is the number of samples, and 441 being the HoG features, 7 being the seven hu moment values.

5.5.2 Write feature matrix to CSV

[Done by: Ng Jan Hui & Tan Wei Mun]

To reduce the program execution time, the feature matrix is then exported into a CSV file that can be read in a later date. Converting the feature matrix into a CSV file ensures that the segmentation process is not required to take place every time the program runs. To achieve it, another self-defined function named `generateFeatureMatrix()` was written. The function takes in the 2 feature matrices as well as an additional string vector parameter that stores all the labels for the traffic sign samples.

Firstly, the function will open a new file with CSV extension using `ofstream`. Then each element of both feature vectors will be written into the CSV. For the last column of each row, the corresponding labels will be written in as well. The writing to CSV process, ensures that the program execution time can be reduced as, the CSV file can be directly loaded into the program as a dataset and be processed for model training, without having the traffic sign images going through the segmentation and feature extraction phases once more. Figure 5.5.2.1 depicts the structure of the concatenated feature matrix.

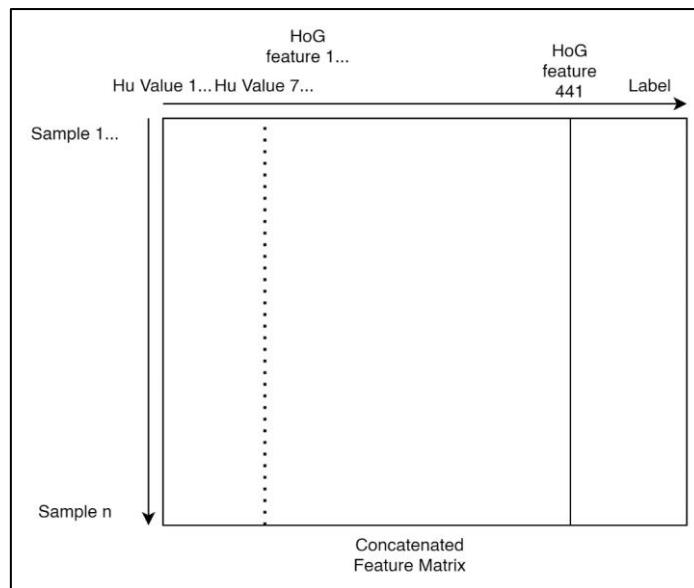


Figure 5.5.2.1 Concatenated feature matrix of Hu Moments, HoG features and labels

5.6 Testing

[Done By: Ng Jan Hui & Tan Jing Jie]

Table 5.6.1 Test case of Compute HoG features

Test Case Name	Compute HoG features
Test Case Description	Input the initial segment vector to the feature extraction algorithm
Expected Output	441 HoG descriptors for 1 sample
Input	Mat type vector storing the initial segments of traffic signs 
Results	Vector of float vectors, with dimension 441 by no. of samples
Status (Pass/Fail)	Pass

Table 5.6.2 Test case of Compute Hu Moments

Test Case Name	Compute Hu moments
Test Case Description	Input the further segment vector to the feature extraction algorithm
Expected Output	7 Hu Moment values for 1 internal symbol
Input	Mat type vector storing the internal symbols of traffic signs 
Results	Vector of double vectors, with dimension 7 by no. of samples
Status (Pass/Fail)	Pass

Table 5.6.3 Test Case of Concatenate Feature

Test Case Name	Concatenate Features
Test Case Description	Combine both feature vectors to 1 feature matrix
Expected Output	Feature matrix of size 441 (HoG) + 7 (Hu) + 1 (Label) by no. of samples
Input	Hu Moments feature vector HoG feature vector
Results	Feature Matrix of size 449 by no. of samples (Example image shown is incomplete as the number of feature per row is too large to be displayed) <pre>0.53422618, 1.9924245, 2.9506991, 4.6621408, 8.8035126, -5.7814589, -8 0.49861723, 0.49861723, 0.49861723, 0.12890698, 0.11380678, 0.02527906 85512, 0.10159252, 0.48967552, 0.45127183, 0.41053042, 0.39583945, 0.0 0.083737716, 0.072367013, 0.48828194, 0.48828194, 0.48828194, 0.488151 225, 0.01040808, 0.010052720, 0.20777650, 0.26014528, 0.56282502, 0.56</pre>
Status (Pass/Fail)	Pass

Table 5.6.4 Test Case of CSV

Test Case Name	Export to CSV
Test Case Description	Export the feature matrix to CSV
Expected Output	CSV file identical to the feature matrix generated
Input	Feature Matrix
Results	<p>CSV file generated with name “traffic_sign_concat.csv”</p>
Status (Pass/Fail)	Pass

5.7 Summary

[Done By: Ng Jan Hui]

In short, the feature extraction module is a crucial step to be taken before model training, as the classifiers require to learn from the underlying features that represent the traffic sign images. The 2 features extracted which are the Hu Moments and HoG descriptors will give 448 features in total, which are sufficient for model training.

Chapter 6

Classification

6.0 Overview

[Done By: Jacynth Tham Ming Quan]

After the image pre-processing and feature extraction stages, the next stage is to train the classification model. In this chapter, three different models were trained – SVM, Random Forest and MobileNet (CNN). The subsections in this chapter discuss the methodology and tools used in the training of the three classification models, the classification requirements, the implementation processes of each classification model and last but not least, the results of the verification and testing process.

6.1 Methodology and Tools

[Done By: Jacynth Tham Ming Quan, Tan Jing Jie, Ng Jan Hui & Tan Wei Mun]

The three selected models – SVM, Random Forest and MobileNet (CNN) – were trained and tested on different platforms as they require different external libraries and functions. The SVM and Random Forest classifiers were trained in a C++ environment using OpenCV libraries. On the other hand, the MobileNet CNN model was trained in a Python environment using TensorFlow, TensorFlow Lite and Keras libraries.

Furthermore, the datasets used for the training and testing process of the three classifiers were different. The SVM and Random Forest used the Chinese Traffic Sign Database (TSRD) while the MobileNet used the German Traffic Sign Recognition Database (GTSRB). The reason for selection is because the nature of MobileNet classifier, unlike the other two classifiers, requires more training data in order to achieve better accuracy.

The tools used to develop this module are as listed below:

1. **Microsoft Visual Studio** – Used as the main IDE for training and testing the Support Vector Machine and Random Forest classifiers along with OpenCV libraries.
2. **OpenCV** – OpenCV modules such as DNN (Deep Neural Networks), Objdetect (Object Detection) and ML (Machine Learning) were used for the model training process in this chapter.
3. **Google Colab** – Used as the main web IDE for training the MobileNet (CNN) classifier along with TensorFlow, TensorFlowLite and Keras libraries.
4. **TensorFlow, TensorFlowHub, TensorFlowLite & Keras** – External libraries that provided the functions to support the training, testing and exporting of the MobileNet classifier.
5. **Numpy, Pyplot, Sklearn & Pandas** – Python-based libraries with functions such as fit, summary, figure, plot and etcetera to support the training and visualization of the MobileNet classifier.

6.2 Requirement

[Done By: Jacynth Tham Ming Quan]

The goal of the classification stage is to be able to use the features extracted in the previous stage to classify traffic signs with an accuracy of at least 80% on bright and distinct traffic sign input images. More leniency was given to the classification of dark and blur traffic sign images during the training and testing of the SVM and Random Forest classifiers due to the scarceness of these images in the given dataset. On the other hand, the MobileNet classifier was trained on the GTSRB (German Traffic Sign Recognition Dataset), which contained more than one -third of dark, blur and weather-affected traffic sign images. The final results of the classification stage are the testing accuracies of the three trained classifiers.

6.3 Design block diagram

[Done By: Ng Jan Hui, Tan Wei Mun & Tan Jing Jie]

The block diagram below depicts the overall flow of the classification processes. The dataset is initially loaded and split into training and testing data sets according to the predefined train test ratio first before the classifier is trained. Next, the class labels of the training and testing data then is retrieved and utilized in the model training and verification stage later. The 3 classification algorithms used are Random Forest, Support Vector Machine and Mobile net. Furthermore, the classifiers will be initialised with the specified hyperparameter and trained using the training data set. Finally, the trained model will be saved, exported, and validated against the testing data set to determine the classifier's testing accuracy.

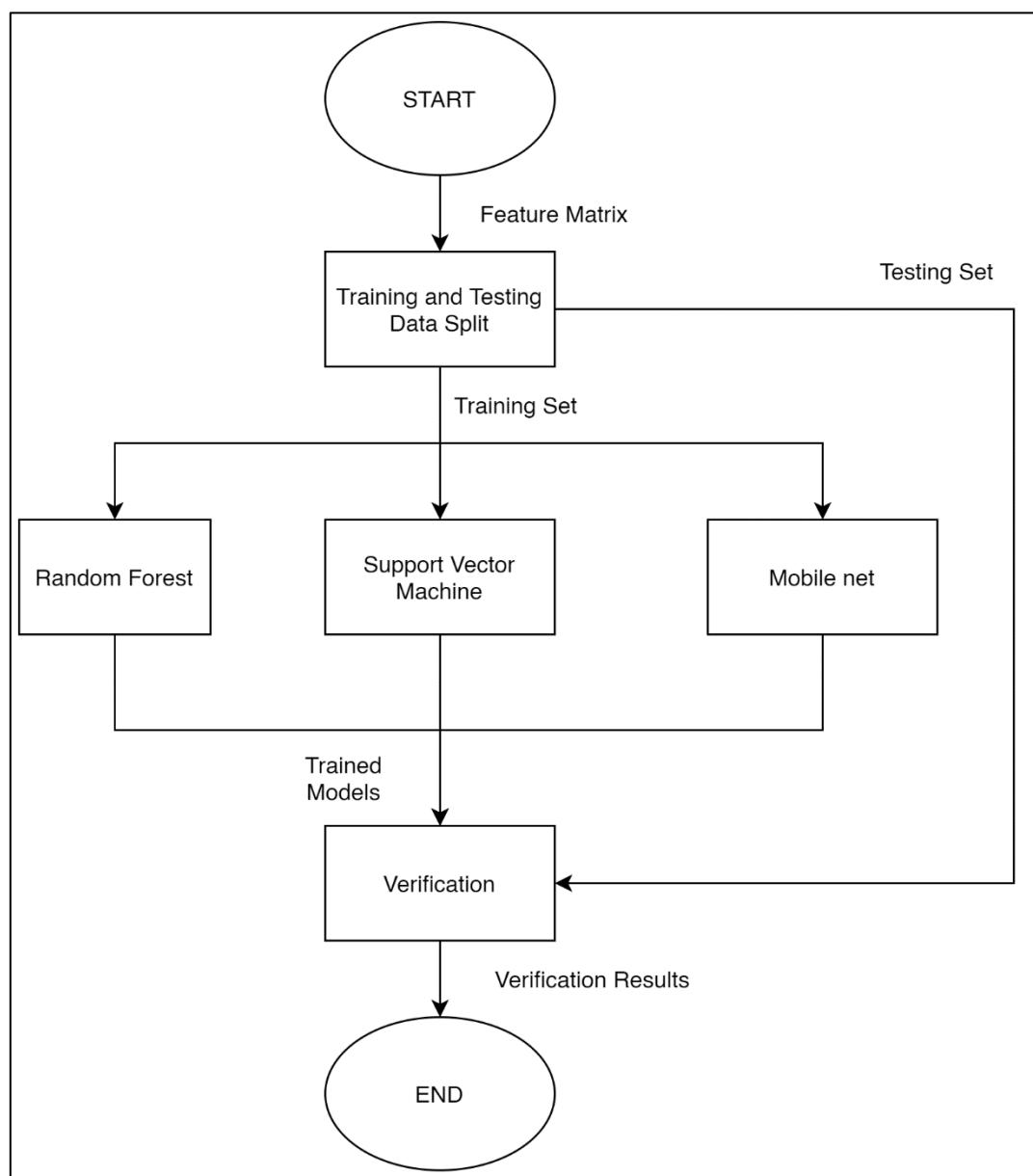


Figure 6.3.1 The overall flow of classification

6.4 Verification plan

[Done By: Tan Wei Mun, Por Teong Dean & Ng Jan Hui]

The test cases for the classification stage are tabulated below in Table 6.4.1 (SVM and Random Forest) and Table 6.4.2 (MobileNet). These test cases were noted of during the implementation of the classification stage and further tested during the testing and validation stage.

Table 6.4.1 Classification Test Cases (SVM & Random Forest)

Id	Test Case	Expected Results
1	Model has successfully completed training and testing process	Output model's final testing accuracy in console
2	Model had missing input files	Output error message in console and terminate program
3	Model has been exported successfully	View exported model in the specified output directory

[Done by: Tan Jing Jie & Jacynth Tham Ming Quan]

Table 6.4.2 Classification Test Cases (MobileNet)

Id	Test Case	Expected Results
1	Model completed the training and testing process	Output model's final testing accuracy in console
2	Export of trained model	A tflite model is created

The test cases above will be tested in the Testing subsection in this chapter.

6.5 Chinese Traffic Sign Feature Dataset Preparation

6.5.1 Dataset loading and splitting

[Done By: Ng Jan Hui]

Before training both the SVM and Random Forest classifiers using the features generated in Chapter 5, the feature CSV file must be loaded into the code using the OpenCV training data class. The OpenCV training data class provides the function `loadFromCSV()` to read in a CSV file storing all the features and the labels generated from each traffic sign.

After reading in the CSV, the `setTrainTestSplitRatio()` function was called, and 0.70 was passed into the function as the argument. 0.70 indicates that the function will split the feature matrix into 70% training set and 30% test set. Adhering to the leave one out cross validation technique, the 30% test set will be used as the unseen data to evaluate the model accuracy after the training phase.

Then, the training set and its corresponding labels can be retrieved and stored into a `Mat` object by calling the `getTrainSamples()` and `getTrainResponses()` functions respectively. Similarly, the testing set and its corresponding labels can also be retrieved by calling the `getTestSamples()` and `getTestResponses()` functions respectively.

Training Data Matrix							
	Hu Moment 1	Hu Moment 2	Hu Moment 3	Hu Moment 4	Hu Moment 5	Hu Moment 6	Hu Moment 7
Training Sample 1	0.493552	1.94784	2.87186	4.28631	7.18004	-5.28527	5.69741
Training Sample 2	0.477475	2.24053	2.3692	4.02962	-7.66918	-5.22815	7.25971
Training Sample 3	0.493552	1.94784	2.87186	4.28631	7.18004	-5.28527	5.69741
Training Sample 4	0.493552	1.94784	2.87186	4.28631	7.18004	-5.28527	5.69741
Training Sample 5	0.0614379	0.148647	1.59189	2.54423	-4.67195	-2.65006	4.92198
Training Sample 6	0.493552	1.94784	2.87186	4.28631	7.18004	-5.28527	5.69741
Training Sample 7	0.541904	2.3173	2.89815	4.32345	8.00816	-5.48475	8.2042
Training Sample 8	0.620738	3.14299	3.92975	4.37191	-8.67028	6.02523	-8.67626
Training Sample 9	0.530238	2.26774	2.83923	4.17065	7.70797	-5.30476	8.10488
Training Sample 10	0.661319	3.33568	5.24118	5.51835	11.4037	7.2914	-10.9204

Figure 6.5.1.1 Example of first 10 training samples

Test Data Matrix							
	Hu Moment 1	Hu Moment 2	Hu Moment 3	Hu Moment 4	Hu Moment 5	Hu Moment 6	Hu Moment 7
Testing Sample 1	0.793764	2.53739	3.5176	5.09196	9.60002	6.46783	9.50482
Testing Sample 2	0.493552	1.94784	2.87186	4.28631	7.18004	-5.28527	5.69741
Testing Sample 3	0.493552	1.94784	2.87186	4.28631	7.18004	-5.28527	5.69741
Testing Sample 4	-1.57645	-3.1407	-4.37735	-4.35973	-8.72827	-5.92958	-6.20321
Testing Sample 5	0.62689	3.36307	5.32412	5.0428	-11.0133	7.0063	-10.2321
Testing Sample 6	0.375204	0.82383	6.45402	7.02361	13.8013	7.83154	-14.155
Testing Sample 7	0.409505	0.913243	3.90472	4.16212	8.19667	4.89182	-9.33679
Testing Sample 8	0.730399	2.29895	2.87043	3.95125	7.36236	5.10238	-8.81103
Testing Sample 9	0.90309	1.80618	2.87186	4.28631	7.18004	-5.28527	5.69741
Testing Sample 10	-0.121841	-0.119316	0.0603433	0.800206	1.46835	-1.84241	1.31887

Figure 6.5.1.2 Example of first 10 testing samples

6.5.2 Training and testing labels type conversion

[Done By: Ng Jan Hui]

According to the OpenCV documentation, the encoded labels to be passed into the SVM model for training, needs to be first converted into integer type. To comply with that requirement, the label vector for the training set is converted into a label vector of integer type using the `convertTo()` function, and passing `CV_32S` as the `rtype`. The converted label vector is then stored into a `Mat`. The testing label vector undergoes the similarly conversion process.

6.5.3 Dataset Standardization

[Done By: Ng Jan Hui]

After the data has been prepared, the prior step before model training would be data scaling. At first, standardization was applied to the training set by using the mean and standard deviation of the training set. The scaled data was then fed as input data for model training.

But the addition of standardization made the classification accuracy unsatisfactory as the models became heavily biased against traffic sign types with more samples. In other words, the skewed dataset caused the model to not able to generalize to other inputs. After further contemplating, a decision was made to remove standardization as the model trained using the raw data, showed a higher testing accuracy overall, when compared to the model trained using the standardized dataset.

[Done By: Tan Wei Mun]

Moreover, it has been discovered that this standardization is only applicable to datasets or a batch of pictures that can be processed together. When each picture is processed individually as switching to real-time recognition, the standardization warped the features and resulted in failed recognition. This is because the mean and standard deviation used to standardize input images or frames are computed using only one record. While exporting all the mean and standard deviation of 448 columns (HOG and Hu moment) to be used to process the new input is difficult. As corresponding to our objectives of delivering a real-time recognition model, the standardization step of the dataset is skipped during the classification testing phase.

6.6 GTRSB Dataset Preparation

6.6.1 Dataset Exploration

[Done By: Tan Jing Jie & Jacynth Tham Ming Quan]

Unlike the desktop-based traffic sign recognizer described in the earlier chapters, the training of the MobileNet model in this chapter used the GTSRB dataset, which stands for the German Traffic Sign Recognition Benchmark.

The GTSRB dataset comes with 43 classes of traffic signs with a total of 51839 traffic sign images, varying in lighting, clarity and weather conditions. All of the images from the GTSRB dataset comes with the .ppm extension. Figure 6.6.1.1 below shows a sample batch randomly selected from the dataset.

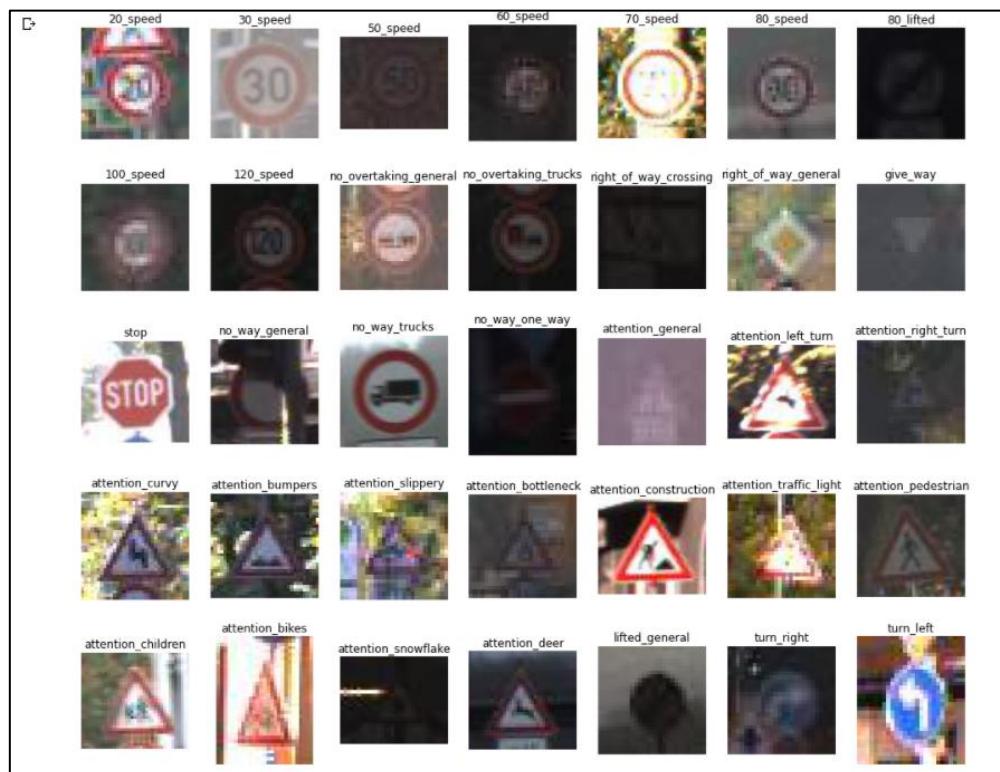


Figure 6.6.1.1 Sample Batch of Images from GTSRB Dataset

As seen from the figure above, the dataset consists of traffic sign images that are dark, blur, stretched, over-exposed and etcetera. This allows the MobileNet model trained afterwards to be able to recognize traffic signs in all kinds of different lighting conditions. Figure 6.6.1.2 below shows the traffic sign images of only the stop sign class.



Figure 6.6.1.2 Sample Batch of Images of Stop Sign Class

The figure above shows the lighting and angle variations in each of the traffic sign images clearly.

Furthermore, each of the images in the dataset comes with a numerical code that represents the traffic sign label in the image. For example, 0 represents “20_speed”, 1 represents “30_speed” and etcetera. These numerical labels were understandable by the model but not humans. Therefore, a text file containing readable traffic sign labels was created in the output folder. Figure 6.6.1.3 below shows a portion of the readable labels in the text file.

```

20_speed
30_speed
50_speed
60_speed
70_speed
80_speed
80_lifted
100_speed
120_speed
no_overtaking_general
no_overtaking_trucks
right_of_way_crossing
right_of_way_general
give_way
stop
no_way_general
no_way_trucks
no_way_one_way
attention_general
attention_left_turn
attention_right_turn
attention_curvy
attention_bumpers
attention_slippery
attention_bottleneck
attention_construction
attention_traffic_light

```

Figure 6.6.1.3 Contents of mobilenet_label_readable.txt

6.6.2 Dataset Splitting

[Done By: Tan Jing Jie & Jacynth Tham Ming Quan]

The training set and testing set contain 39209 images and 12630 images respectively. The ratio between training and testing set is 3:1. The exploration of dataset shown in Figure 6.6.2.1 below:

```

[19] number_training_data = len(training_data)
     print("The number of training data: " + str(number_training_data))
     The number of training data: 39209

[23] number_testing_data = len(testing_data)
     print("The number of testing data: " + str(number_testing_data))
     The number of testing data: 12630

```

Figure 6.6.2.1 Number of Images In The Training Set and Testing Set

6.5.3 Dataset Preprocessing

[Done By: Jacynth Tham Ming Quan & Tan Jing Jie]

Moreover, the training images and testing images and their respective labels were sorted and stored in two separate folders to be used in the training and testing process respectively. Moving on, all the images were converted from the ppm file format to jpg. This is because the .ppm files were not supported by TensorFlow by default.

Furthermore, as the TensorFlow Hub's image modules require float input between 0 and 1 inclusive, the Keras ImageDataGenerator's rescale function to generate the float inputs needed. The input images were scaled down by a factor of 255 before training the model. The reason for selecting the number 225 is because the input images come in RBG forms, with each pixel in the range of 0 to 255. In other words, the rescale function transforms every pixel value from the range of [0, 255] to [0, 1]. The advantage of doing so is that images with high pixel ranges and low pixel ranges can share the same machine learning model with the same weights and learning rate. Without scaling the pixel values into the range of [0,1], the high pixel range images will have a higher chance of updating the model training weights compared to the low pixel range images. Thus, scaling the input images ensures that the model training process is fair for all input images.

6.7 SVM Model Classifier

[Done By: Tan Wei Mun]

SVM is a machine learning algorithm that employs kernel trick, a nuclear function that supports vector machines, to convert data and then determines an ideal boundary or super plane in the potential output based on this conversion information. Briefly described, it does some extremely complicated data transformations before segregating the user's data using preset labels.

There are three advantages of using the SVM model. Firstly, it requires only small samples to train, not that the absolute number of samples is small, but that the SVM algorithm requires a relatively small number of samples compared to the complexity of the problem. Next, the error between the approximation of the problem's actual model and the problem's real solution is minimized. Lastly, the SVM is nonlinear, which refers to SVM as good at dealing with the linear indiscipline of sample data, mainly through relaxation variables and nuclear function technology. This part is the essence of SVM.

Therefore, the SVM is used as one of our project's classifiers to recognize the traffic signs.

6.7.1 SVM Model Initialization

[Done By: Tan Wei Mun]

From the OpenCV machine learning library, an SVM classifier was created and initialized. The hyperparameters set to the SVM classifier are listed in the table below.

Table 6.7.1.1 SVM classifier hyperparameters

Hyperparameter	Value
Type	C_SVC
Kernel	LINEAR
Termination Criteria	MAX_ITER, 1000 Epochs or 1-e6 difference

For the SVM classifier, there are three hyperparameters to set up, and they are all finalized after a series of evaluations and testing. First and foremost, the C SVC type is selected for usage in multi-class classification. This type is notable for poor class separation, mainly when the training data is non-linearly separable. The linear kernel is the next hyperparameter. It has a linear core, which means there is no orientation mapping to high-dimensional space, and linear differentiation or regression is completed in the original characteristic space, the quickest option. Finally, the MAX_ITER is chosen in the termination criteria, so it will loop the training until the maximum iteration while the maximum iteration is set to 1000 and the epsilon is set 1-e6. So it is expected to loop the training iteration 1000 times, and the minimum accuracy changes should be 1-e6 between two iterations when it stops.

6.7.2 SVM Model Training

[Done By: Tan Wei Mun]

Next, the classifier's train() function was used. The training set and training labels were used as parameters in the SVM classifier. The SVM classifier's training took 188.045 milliseconds, which is pretty quick considering a large number of features and data.

The model is then serialized into an XML file using the save() functions to guarantee that the trained weights for the SVM classifier are maintained. The model may be reused directly for recognizing signs without retraining the model repeatedly. The XML file contains all of the trained SVM model's settings. The model may then be imported into another application for classification using the SVM classifier object's load() method.

```

1  <?xml version="1.0"?>
2  <opencv_storage>
3  <opencv_ml_svm>
4  <format>3</format>
5  <svmType>C_SVC</svmType>
6  <kernel>
7  <type>LINEAR</type></kernel>
8  <C>1.</C>
9  <term_criteria><iterations>1000</iterations></term_criteria>
10 <var_count>448</var_count>
11 <class_count>6</class_count>
12 <class_labels type_id="opencv-matrix">
13 <rows>6</rows>
14 <cols>1</cols>
15 <dt>i</dt>
16 <data>
17   1 2 3 4 5 6</data></class_labels>
18 <sv_total>15</sv_total>
19 <support_vectors>
20 <_>
21   -6.46193996e-02 -3.40400159e-01 2.68761784e-01 6.72850087e-02
22   -1.29463030e-02 2.21065320e-02 -1.75577942e-02 3.81560735e-02
23   -2.74398267e-01 -2.66658604e-01 -2.35376760e-01 6.21715076e-02
24   5.47404177e-02 2.02973649e-01 1.79592341e-01 2.13125274e-01
25   1.17852334e-02 -1.90087929e-01 -2.10779384e-01 -1.64835379e-01
26   4.60601458e-03 7.05147982e-02 2.60118812e-01 1.84395909e-01
27   1.48160547e-01 4.01229747e-02 -1.54555127e-01 -1.96107626e-01
28   -6.99873269e-02 6.37205318e-02 5.39658628e-02 2.31206417e-01
29   1.36319771e-01 5.91849461e-02 -4.93017733e-02 -1.79321378e-01

```

Figure 6.7.2.1 XML File that holds all configurations for the SVM classifier

6.7.3 SVM Model Evaluation

[Done By: Tan Wei Mun]

The performance of the SVM classifier was evaluated using the left-out test set after training was completed. The testing accuracy of the SVM classifier was 0.9744, as seen in figure 6.7.3.1 below. The SVM classifier's performance was good considering the relatively small training dataset and short training time.

SVM Classification accuracy: 0.974441

Figure 6.7.3.1 SVM Classification Accuracy

Further classification analyses, such as computing classification metrics such as Recall, Precision, and F1 Score, as well as visualizing the confusion matrix, are undoubtedly difficult to do in the C++ environment. As a result, the SVM classifier's predicted labels and the ground truth labels were saved as separate CSV files. The CSV data were then imported into a Python notebook file using the NumPy library, and the sklearn module was used to do classification analysis. Chapter 9 will go into the specifics of the analysis.

6.8 Random Forest Classifier

[Done By: Ng Jan Hui]

Apart from using Support Vector Machines as the classifier, a Random Forest classifier was also used. The reasoning behind the usage of a Random Forest classifier was to compare the performance between both classifiers, since Random Forest classifiers are more intrinsically attuned for multiclass classification, while Support Vector Machines are intrinsically suited for two-class classification problems.

Besides that, Random Forests can also handle features of varying scales and allows the usage of the training data as it is. This implies that the Random Forest can train directly from the raw training set without affecting its trained performance. Additionally, the Random Forest classifier is a probabilistic classifier that allows the retrieval of votes for the labels.

6.8.1 Random Forest Model Initialization

[Done By: Ng Jan Hui]

From the OpenCV machine learning library, a Random Forest classifier was created and initialized. The hyperparameters set to the Random Forest classifier is listed in the table below.

Table 6.8.1.1 The hyperparameter list of Random Forest Model Initialization

Hyperparameter	Value
Max Depth	10
Minimum Sample Count	2
Regression Accuracy	0
Use of Surrogates	False
Max Categories	16
Priors	None
Calculate Var Importance	True

Active Var Count	0
Termination Criteria	100 Epochs or 0 difference

The reasoning of several selected hyperparameter is as follows. The max depth of the Random Forest is limited to 10. By default, there is no limit for the depth of the Random Forest. Limiting the depth to 10 ensures that the growth of the Random Forest can be regularized and prevent overfitting from happening. The use of surrogates has been set to false since all missing data had been imputed and feature importance is equal. The training termination criteria is set to 100 epochs, or until the accuracy changes between two consecutive epochs is 0.

The hyperparameters set for Decision Trees can also be used in Random Forests. For example, minimum sample count and max categories are set as the hyperparameters for the individual Decision Trees in the Random Forest.

6.8.2 Random Forest Model Training

[Done By: Ng Jan Hui]

Then, the train() function of the classifier was called. The training set and the training labels were passed into the Random Forest classifier as the parameters. Compared to the SVM, the training for the Random Forest classifier took approximately 1-2 minutes to complete.

To ensure that the trained weights for the Random Forest classifier are retained, and the model can be used to directly classify signs without retraining, the model is serialized into an XML file using the save() functions. The XML file holds all the configurations of the trained Random Forest model. The model can then be loaded into another program for classification using the load() function of the Random Forest classifier object. Figure 6.8.2.1 shows a preview of the XML file.

Figure 6.8.2.1 XML File that holds all configurations for the RF classifier

6.8.3 Random Forest Model Evaluation

[Done By: Ng Jan Hui]

Upon completion of training, the performance of the Random Forest classifier was evaluated using the left-out test set. The Random Forest classifier had attained a testing accuracy of 0.97. The performance of the Random Forest classifier was satisfactory as seen in figure 6.8.3.1 below.

Rtree classification accuracy: 0.971246

Figure 6.8.3.1 Random Forest Classification Accuracy

Performing further classification analysis such as computing classification metrics (Recall, Precision, F1 Score) and plotting of confusion matrix are not available in the C++ environment. Hence, the predicted labels by the Random Forest classifier and the ground truth labels were saved into individual CSV files respectively.

The CSV files were then loaded into a Python notebook file using the NumPy library and will be used for classification analysis using the sklearn library. The details of analysis will be covered in Chapter 9.

6.9 Mobile Net Convolutional Neural Network (CNN)

[Done By: Tan Jing Jie & Jacynth Tham Ming Quan]

The MobileNet CNN model was first trained and tested in Google Colab before exporting it to a TensorFlowLite model which needs to be included in the mobile application. This model plays the role of classifying the traffic sign in mobile applications. The following subsections describe how the trained MobileNet model was obtained. The related code can find in [Appendix B.5](#).

There are 2 reasons to use this mobile net convolutional neural network as a classifier. First, it generates a lightweight classifier, around 12 Megabyte model. This model only needs a simple input, and it can compute the output immediately. Second, the time taken for the model to classify a traffic sign only consumes less than 0.1 seconds which is very suitable in real time use. All of these advantages make it very suitable for use in mobile devices.

6.9.1 Google Colab Project Setup

[Done By: Jacynth Tham Ming Quan & Tan Jing Jie]

Before the MobileNet model can be trained, the Google Colab environment was set up. First, the Tensorflow Hub dependency has to be added. This external library contains pretrained, reusable machine learning models and component such as the MobileNet CNN model. Besides this, other python libraries such as tensorflow, numpy, matplotlib, sklearn.metrics and pandas were imported into the project.

Next, the output directories had to be configured. The main folder to store the output model and retrained labels text file was named as “output”.

6.9.2 Model Configuration

[Done By: Jacynth Tham Ming Quan & Tan Jing Jie]

MobileNets is a family of mobile-first computer vision TensorFlow models, designed to maximize classification accuracies while minimizing processor and resources usage. The MobileNet network comes with various configurations for its MobileNet models. In this project, the model with an input image size of 224 X 224

pixels was selected. Generally, smaller input image sizes led to faster models, but a smaller input size would also lead to worse accuracy. Therefore, the selection of the 224 X 224 pixels was the most adequate trade-off between the recognition time and accuracy.

6.9.3 Training the Model Using Transfer Learning

[Done By: Tan Jing Jie & Jacynth Tham Ming Quan]

Moving on, a headless MobileNet model was downloaded from the official TensorFlow Hub developer website. As the downloaded MobileNet model had been previously trained to handle general on-device computer vision classification problems, its feature extraction layer would already have existing features present. Since this project's aim is to retrain the MobileNet model to classify only traffic signs, the variables in the feature extraction layer is frozen by setting the feature_extractor_layer.trainable variable to false. This way, the MobileNet model trained later on will only modify the classifier layer. This method is known as Transfer Learning. Transfer learning is a technique used to retrain pretrained machine learning models so that they are able to use pre-learnt features from one problem to solve a completely new problem. In this case, only the classifier layer of the MobileNet model requires retraining through the use of a custom dataset.

Next, a model head was defined using tf.keras.Sequential. A Sequential model is appropriate in this classification problem because each layer in the CNN model only has one input and one output. Moving on, the number of dense layers in the head was set as the number of traffic sign classes – 43. Figure 6.9.3.1 below shows the summary of the model after adding the newly defined head.

```
model.summary()

Model: "sequential"
+-----+
Layer (type)        Output Shape       Param #
+=====+
keras_layer (KerasLayer)    (None, 1280)      2257984
+=====+
dense (Dense)        (None, 43)          55083
+=====+
Total params: 2,313,067
Trainable params: 55,083
Non-trainable params: 2,257,984
+=====+
```

Figure 6.9.3.1 Summary of Newly Defined MobileNet Model

Next, the MobileNet model was trained in several training iterations, called epochs. In each epoch, the model is trained by iterating over all the training samples. In order to achieve the highest accuracy with minimal training loss, a total of 20 epochs were used to train the model. In each of the epoch, the accuracies and losses were displayed. The output of the training process is shown below in Figure 6.9.3.2.

```

↳ Epoch 1/20
1226/1226 [=====] - 105s 82ms/step - loss: 0.8385 - acc: 0.8125
Epoch 2/20
1226/1226 [=====] - 101s 82ms/step - loss: 0.5109 - acc: 0.8750
Epoch 3/20
1226/1226 [=====] - 100s 82ms/step - loss: 0.4904 - acc: 0.9375
Epoch 4/20
1226/1226 [=====] - 100s 82ms/step - loss: 0.3945 - acc: 0.8750
Epoch 5/20
1226/1226 [=====] - 101s 82ms/step - loss: 0.3489 - acc: 0.9375
Epoch 6/20
1226/1226 [=====] - 101s 82ms/step - loss: 0.6097 - acc: 0.8438
Epoch 7/20
1226/1226 [=====] - 101s 82ms/step - loss: 0.2572 - acc: 0.9062
Epoch 8/20
1226/1226 [=====] - 101s 82ms/step - loss: 0.2383 - acc: 0.9375
Epoch 9/20
1226/1226 [=====] - 100s 82ms/step - loss: 0.3096 - acc: 0.9062
Epoch 10/20
1226/1226 [=====] - 100s 82ms/step - loss: 0.1650 - acc: 1.0000
Epoch 11/20
1226/1226 [=====] - 101s 82ms/step - loss: 0.2704 - acc: 0.9375
Epoch 12/20
1226/1226 [=====] - 101s 83ms/step - loss: 0.3060 - acc: 0.9375
Epoch 13/20
1226/1226 [=====] - 101s 83ms/step - loss: 0.1974 - acc: 0.9375
Epoch 14/20
1226/1226 [=====] - 102s 83ms/step - loss: 0.3539 - acc: 0.8750
Epoch 15/20
1226/1226 [=====] - 101s 83ms/step - loss: 0.3042 - acc: 0.8750
Epoch 16/20
1226/1226 [=====] - 102s 83ms/step - loss: 0.2089 - acc: 0.9375
Epoch 17/20
1226/1226 [=====] - 101s 83ms/step - loss: 0.1436 - acc: 0.9688
Epoch 18/20
1226/1226 [=====] - 101s 82ms/step - loss: 0.1997 - acc: 0.9375
Epoch 19/20
1226/1226 [=====] - 101s 82ms/step - loss: 0.0936 - acc: 0.9688
Epoch 20/20
1226/1226 [=====] - 102s 83ms/step - loss: 0.0986 - acc: 0.9688

```

Figure 6.9.3.2 Output of Model Training Process

After the model has been trained, the graph of model loss against training steps was plotted. This graph is shown below in Figure 6.9.3.3.

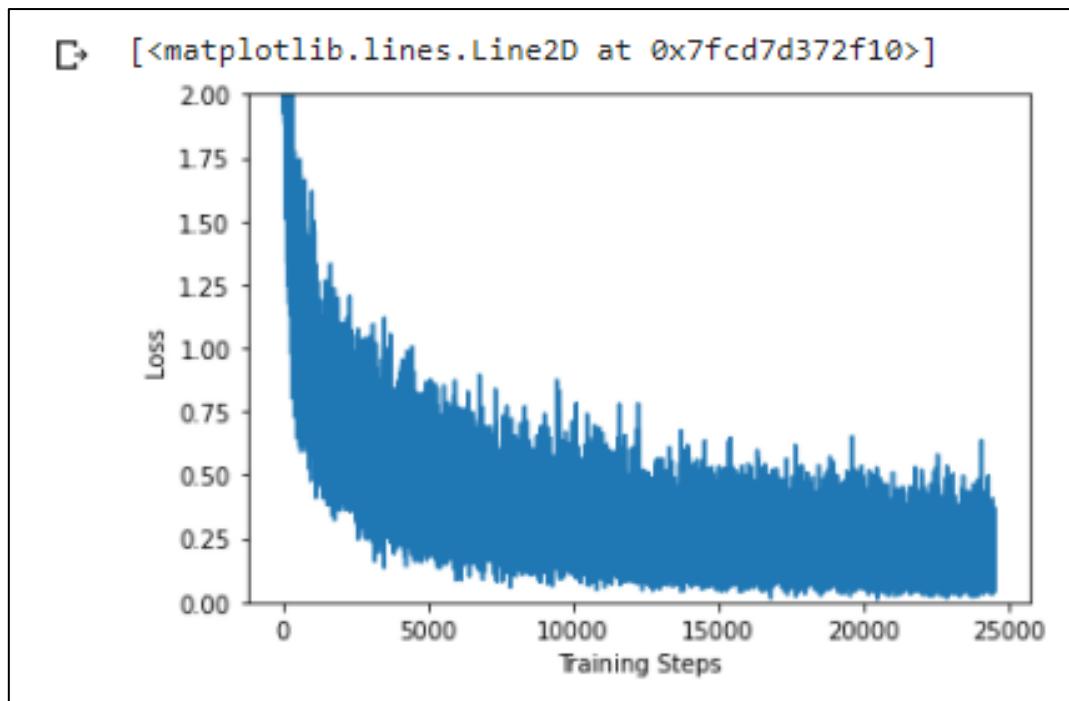


Figure 6.9.3.3 Graph of Model Loss Against Training Steps

As seen from the graph above, as the training steps increase, the model loss decreases gradually, which is the general trend for majority of the classification models. Furthermore, in order to verify the accuracy of the trained model during the training process, a graph of model accuracy against training steps was plotted. This graph is shown below in Figure 6.9.3.4.

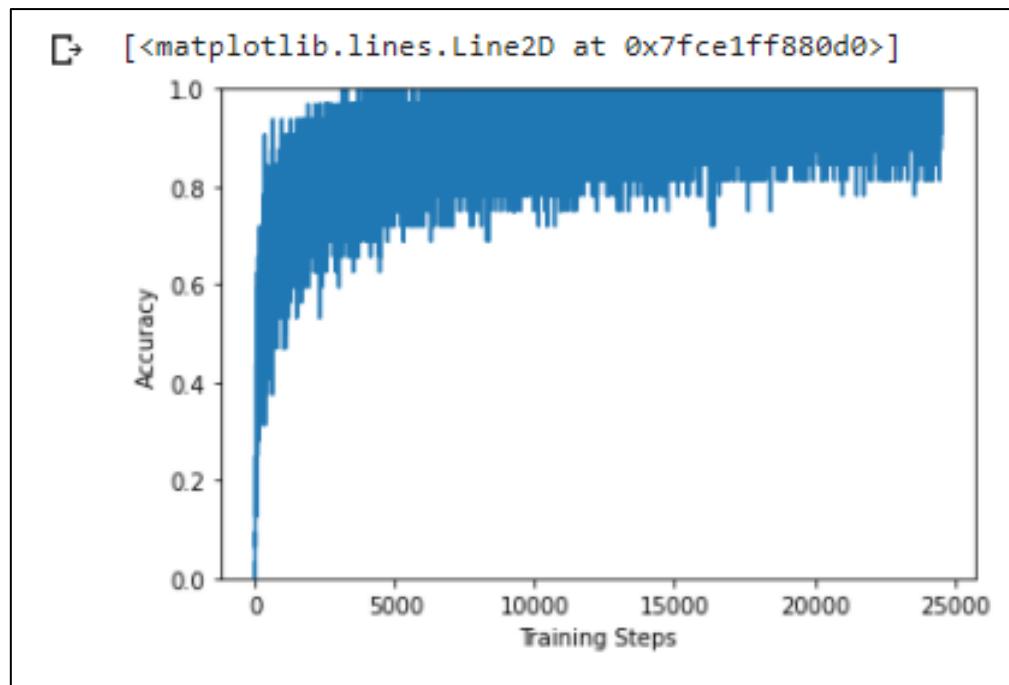


Figure 6.9.3.4 Graph of Model Accuracy Against Training Steps

As seen in the graph above, the accuracy of the trained model increases with the training step, which confirms that the model training process did not meet any errors.

6.9.4 Testing the Model

[Done By: Tan Jing Jie & Jacynth Tham Ming Quan]

The next step is to test the newly trained MobileNet model on a random batch of images. The random batch contains of 32 traffic sign images, 26 of which the trained model predicted correctly. This yielded an accuracy of 0.8125. Figure 6.9.4.1 below shows the testing results of the model.



Figure 6.9.4.1 Graph of Model Accuracy Against Training Steps

From the image above, it is observed that 50% of the false predictions by the trained model was due to poor lighting in the input images. This evidenced that even though the model performed well, it still faced minor difficulties classifying signs in poor lighting conditions.

6.9.5 Converting and Exporting the Model

[Done By: Tan Jing Jie & Jacynth Tham Ming Quan]

The final step is to convert the model from TensorFlow to TensorFlowLite. This is done using the TFLiteConverter function provided by the TensorFlow library, TensorFlowLite models are lighter and require less processing power, which makes it perfect for on-device traffic sign recognition. Once the model has been converted and exported, it is stored under the output folder that was created at the start of the project. Figure 6.9.5.2 below shows the contents of the output folder after exporting the model.

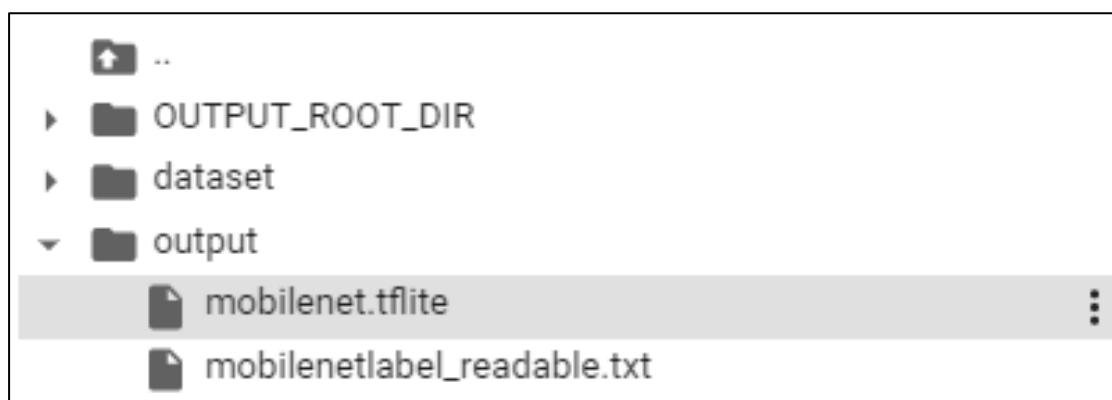


Figure 6.9.5.2 Contents of Output Folder After Exporting Model

After the model was converted and exported, the TensorFlowLite model was imported back into the process and tested to ensure that the accuracy remains the same. Several results of the testing process are shown below in Figure 6.9.5.3.

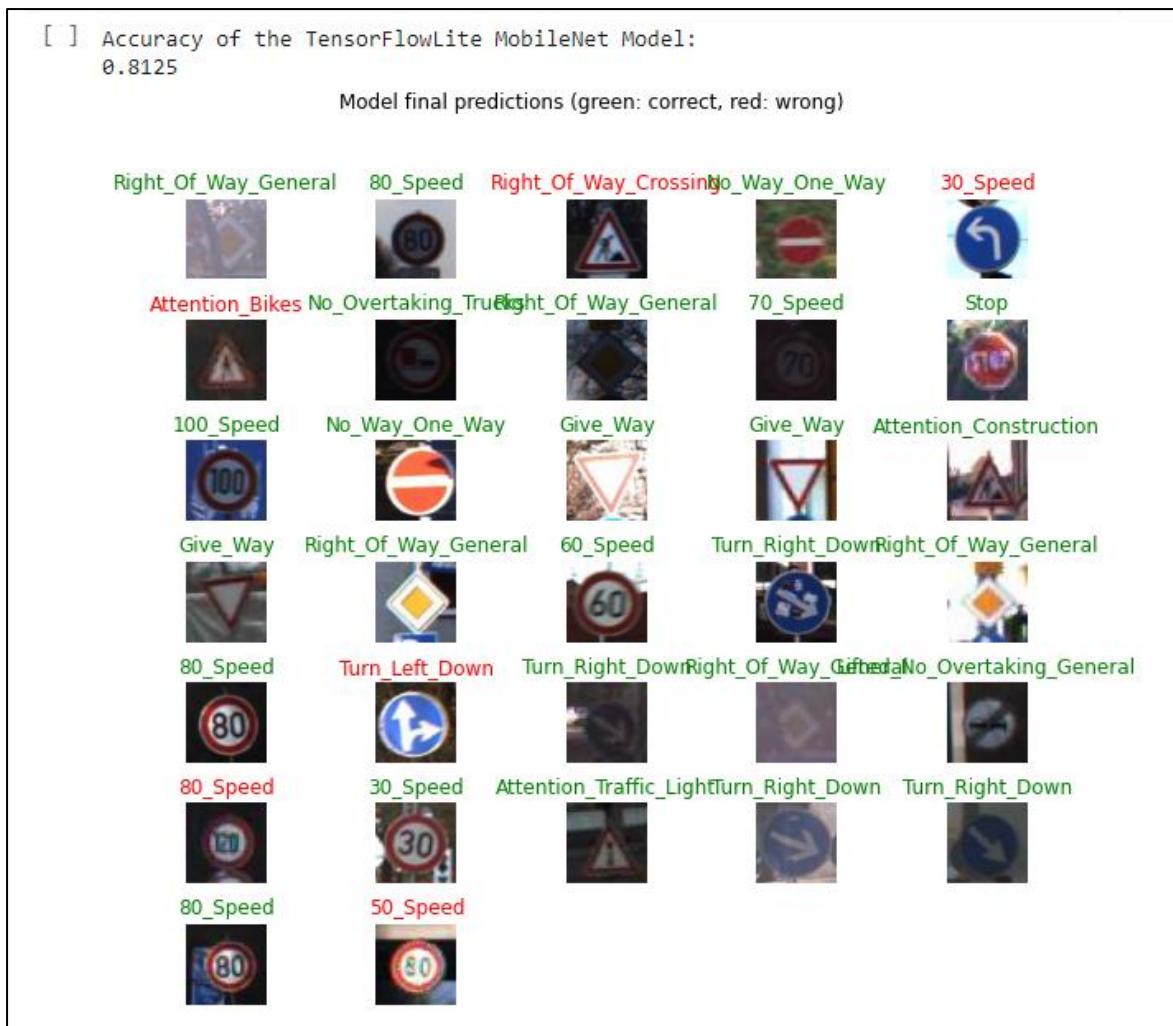


Figure 6.9.5.3 Contents of Output Folder After Exporting Model

As seen in the figure above, the accuracy of the TensorFlowLite is exactly the same as the TensorFlow model before exporting. Thus, with a final accuracy of 0.8125, the retrained MobileNet model is ready to be integrated into the Android application. The model will accept an image as input, and it will output a float array of confidence levels of each class of traffic sign. The validity of traffic signs can be controlled by setting a threshold value to the confidence level. The detail of implementation of this model in mobile applications will show in [Chapter 8](#).

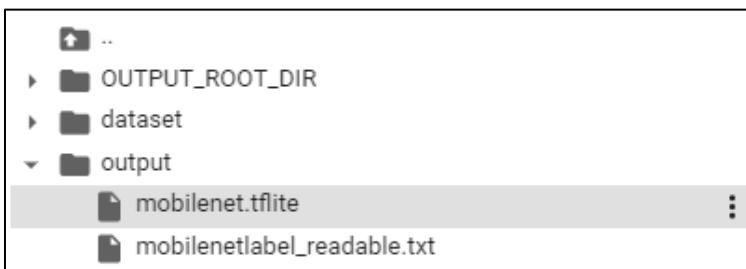
6.10 Testing

[Done By: Tan Jing Jie]

Table 6.10.1 Test case of model completed the training and testing process

Test Case Name	Model completed the training and testing process
Test Case Description	Check the model completed the training process and test the accuracy
Expected Output	Output model's final testing accuracy in console
Input	Training dataset
Results	<p>The accuracy is shown.</p> <pre>[35] #Calculate evaluated loss and accuracy of model score = model.evaluate(x=image_val_data, batch_size=image_val_data) print("Loss: ", score[0], "Accuracy: ", score[1]) 394/394 [=====] - 1 - 34s 84ms/step - 1 Loss: 1.3311518430709839 Accuracy: 0.8125</pre>
Status (Pass/Fail)	Pass

Table 6.10.2 Test case of export of trained model

Test Case Name	Export of trained model
Test Case Description	Export of model after completing training and testing
Expected Output	A TensorFlowLite model is created
Input	Training dataset
Results	<p>A TensorFlowLite model, “mobilenet.tflite” is created.</p>  <pre> ... ▶️ OUTPUT_ROOT_DIR ▶️ dataset ▶️ output └─ mobilenet.tflite └─ mobilenetlabel_readable.txt </pre>
Status (Pass/Fail)	Pass

[Done By: Ng Jan Hui]

Table 6.10.3 Test case of SVM and RF Training and Testing

Test Case Name	SVM and RF Training & Testing
Test Case Description	Test the SVM and Random Forest's accuracy after training
Expected Output	Testing accuracies for both models and completion message
Input	Testing Set
Results	<p>Console output “Done Model Training – SVM + RF” & prints the accuracy</p> <pre>Done Model Training - SVM + RF Experimental Results - Accuracy of Classifier Rtree Classification accuracy: 0.971246 SVM Classification accuracy: 0.974441</pre>
Status (Pass/Fail)	Pass

Table 6.10.4 Test case of SVM and RF Model Save

Test Case Name	Save SVM and RF Models																																																																																												
Test Case Description	Serialize the SVM and RF models to an XML file each																																																																																												
Expected Output	2 XML files named T6G1_SVM.xml and T6G1_rtree.xml																																																																																												
Input	Trained SVM Model and RF Model																																																																																												
Results	<p>2 XML files generated within the same directory</p> <table border="1"> <thead> <tr> <th>File/Folder</th> <th>Last Modified</th> <th>Type</th> <th>Size</th> </tr> </thead> <tbody> <tr><td>x64</td><td>2021-06-29 12:06 PM</td><td>File folder</td><td></td></tr> <tr><td>Models</td><td>2021-09-07 7:18 PM</td><td>File folder</td><td></td></tr> <tr><td>Inputs</td><td>2021-07-28 9:18 AM</td><td>File folder</td><td></td></tr> <tr><td>Input Dataset</td><td>2021-09-09 11:41 AM</td><td>File folder</td><td></td></tr> <tr><td>Feature CSV</td><td>2021-09-07 10:07 PM</td><td>File folder</td><td></td></tr> <tr><td>Debug</td><td>2021-06-29 12:06 PM</td><td>File folder</td><td></td></tr> <tr><td>backup</td><td>2021-09-02 5:22 PM</td><td>File folder</td><td></td></tr> <tr><td>T6G1_SVM.xml</td><td>2021-09-10 3:10 PM</td><td>XML Document</td><td>1,519 KB</td></tr> <tr><td>T6G1_rtree.xml</td><td>2021-09-10 3:10 PM</td><td>XML Document</td><td>1,220 KB</td></tr> <tr><td>Input Dataset.zip</td><td>2021-09-07 11:53 PM</td><td>WinRAR ZIP archive</td><td>40,850 KB</td></tr> <tr><td>Project452.sln</td><td>2021-06-29 12:13 PM</td><td>Visual Studio Solu...</td><td>2 KB</td></tr> <tr><td>Project452.vcxproj.filters</td><td>2021-09-10 2:59 PM</td><td>VC++ Project Filte...</td><td>2 KB</td></tr> <tr><td>Project452.vcxproj</td><td>2021-09-10 2:59 PM</td><td>VC++ Project</td><td>7 KB</td></tr> <tr><td>Project452.vcxproj.user</td><td>2021-09-10 2:33 PM</td><td>Per-User Project O...</td><td>1 KB</td></tr> <tr><td>traffic_sign_concat_imputed.csv</td><td>2021-09-04 12:25 AM</td><td>Microsoft Excel C...</td><td>4,169 KB</td></tr> <tr><td>testLabels.csv</td><td>2021-09-10 3:10 PM</td><td>Microsoft Excel C...</td><td>1 KB</td></tr> <tr><td>SVM_res.csv</td><td>2021-09-10 3:10 PM</td><td>Microsoft Excel C...</td><td>1 KB</td></tr> <tr><td>Random_Forest_res.csv</td><td>2021-09-10 3:10 PM</td><td>Microsoft Excel C...</td><td>1 KB</td></tr> <tr><td>user_main.cpp</td><td>2021-09-10 2:34 PM</td><td>C++ Source File</td><td>16 KB</td></tr> <tr><td>supp.cpp</td><td>2021-08-31 12:36 PM</td><td>C++ Source File</td><td>4 KB</td></tr> <tr><td>main.cpp</td><td>2021-09-08 6:00 PM</td><td>C++ Source File</td><td>17 KB</td></tr> <tr><td>Supp.h</td><td>2021-06-22 5:27 AM</td><td>C/C++ Header</td><td>1 KB</td></tr> </tbody> </table>	File/Folder	Last Modified	Type	Size	x64	2021-06-29 12:06 PM	File folder		Models	2021-09-07 7:18 PM	File folder		Inputs	2021-07-28 9:18 AM	File folder		Input Dataset	2021-09-09 11:41 AM	File folder		Feature CSV	2021-09-07 10:07 PM	File folder		Debug	2021-06-29 12:06 PM	File folder		backup	2021-09-02 5:22 PM	File folder		T6G1_SVM.xml	2021-09-10 3:10 PM	XML Document	1,519 KB	T6G1_rtree.xml	2021-09-10 3:10 PM	XML Document	1,220 KB	Input Dataset.zip	2021-09-07 11:53 PM	WinRAR ZIP archive	40,850 KB	Project452.sln	2021-06-29 12:13 PM	Visual Studio Solu...	2 KB	Project452.vcxproj.filters	2021-09-10 2:59 PM	VC++ Project Filte...	2 KB	Project452.vcxproj	2021-09-10 2:59 PM	VC++ Project	7 KB	Project452.vcxproj.user	2021-09-10 2:33 PM	Per-User Project O...	1 KB	traffic_sign_concat_imputed.csv	2021-09-04 12:25 AM	Microsoft Excel C...	4,169 KB	testLabels.csv	2021-09-10 3:10 PM	Microsoft Excel C...	1 KB	SVM_res.csv	2021-09-10 3:10 PM	Microsoft Excel C...	1 KB	Random_Forest_res.csv	2021-09-10 3:10 PM	Microsoft Excel C...	1 KB	user_main.cpp	2021-09-10 2:34 PM	C++ Source File	16 KB	supp.cpp	2021-08-31 12:36 PM	C++ Source File	4 KB	main.cpp	2021-09-08 6:00 PM	C++ Source File	17 KB	Supp.h	2021-06-22 5:27 AM	C/C++ Header	1 KB
File/Folder	Last Modified	Type	Size																																																																																										
x64	2021-06-29 12:06 PM	File folder																																																																																											
Models	2021-09-07 7:18 PM	File folder																																																																																											
Inputs	2021-07-28 9:18 AM	File folder																																																																																											
Input Dataset	2021-09-09 11:41 AM	File folder																																																																																											
Feature CSV	2021-09-07 10:07 PM	File folder																																																																																											
Debug	2021-06-29 12:06 PM	File folder																																																																																											
backup	2021-09-02 5:22 PM	File folder																																																																																											
T6G1_SVM.xml	2021-09-10 3:10 PM	XML Document	1,519 KB																																																																																										
T6G1_rtree.xml	2021-09-10 3:10 PM	XML Document	1,220 KB																																																																																										
Input Dataset.zip	2021-09-07 11:53 PM	WinRAR ZIP archive	40,850 KB																																																																																										
Project452.sln	2021-06-29 12:13 PM	Visual Studio Solu...	2 KB																																																																																										
Project452.vcxproj.filters	2021-09-10 2:59 PM	VC++ Project Filte...	2 KB																																																																																										
Project452.vcxproj	2021-09-10 2:59 PM	VC++ Project	7 KB																																																																																										
Project452.vcxproj.user	2021-09-10 2:33 PM	Per-User Project O...	1 KB																																																																																										
traffic_sign_concat_imputed.csv	2021-09-04 12:25 AM	Microsoft Excel C...	4,169 KB																																																																																										
testLabels.csv	2021-09-10 3:10 PM	Microsoft Excel C...	1 KB																																																																																										
SVM_res.csv	2021-09-10 3:10 PM	Microsoft Excel C...	1 KB																																																																																										
Random_Forest_res.csv	2021-09-10 3:10 PM	Microsoft Excel C...	1 KB																																																																																										
user_main.cpp	2021-09-10 2:34 PM	C++ Source File	16 KB																																																																																										
supp.cpp	2021-08-31 12:36 PM	C++ Source File	4 KB																																																																																										
main.cpp	2021-09-08 6:00 PM	C++ Source File	17 KB																																																																																										
Supp.h	2021-06-22 5:27 AM	C/C++ Header	1 KB																																																																																										
Status (Pass/Fail)	Pass																																																																																												

6.11 Summary

[Done by: Ng Jan Hui]

To conclude this chapter, all models were trained using the prepared feature dataset extracted the traffic signs. All models had been tuned according to the optimal hyperparameters that gives the best prediction accuracy. The models were tested using the left-out 30% testing set to evaluate the performance of the models.

To apply the trained models in the desktop application and mobile application, the models were then serialized and saved as XML files for the SVM and Random Forest classifier, and a tflite file for the MobileNet model. The trained models can be loaded in another program to classify user specified traffic signs, provided that the traffic signs pass through the same segmentation and feature extraction pipeline.

Chapter 7

Desktop application

7.0 Overview

[Done By: Jacynth Tham Ming Quan]

In this chapter, the image pre-processing, feature extraction and classification stages are implemented into three different desktop applications – image-based, video-based and webcam-based. The two classifiers – SVM and Random Forest – that have been exported from the previous chapter are used in the desktop applications implemented in this chapter. In the following subsections, the methodology and tools used are discussed, followed by the block diagram, verification plan, implementation, and testing of the desktop applications. As the testing process is different for all three desktop applications, the testing phase is merged into the implementation of each desktop application.

7.1 Methodology and Tools

[Done by: Tan Jing Jie, Jacynth Tham Ming Quan]

Having explored, trained, and tested the traffic sign segmentation process all the way up to the classification process, the exported classifier is implemented into a Desktop application. The desktop TSR system is coded in full C++ with support of the training model. Furthermore, users are given the option to select still-image input, video stream input and real-time video stream input, all of them share the same image pre-processing, segmentation, and classification steps.

The tools used to develop this module are as listed below:

1. **Microsoft Visual Studio** – Used as the main IDE for training and testing the Support Vector Machine and Random Forest classifiers along with OpenCV libraries.

2. **OpenCV** – OpenCV modules such as DNN (Deep Neural Networks), Objdetect (Object Detection) and ML (Machine Learning) were used for the recognition process in this chapter.

7.2 Requirement

[Done by: Jacynth Tham Ming Quan, Tan Jing Jie]

Upon completion of the three desktop-based applications, users will be able to recognize traffic signs from their desktops using still-images, video-based input and webcam-based input. The traffic signs in the video-based and webcam-based inputs will be segmented and classified in real-time, with the detected traffic sign bounded by a bounding box in the current camera frame. Furthermore, the classification results will be shown in real-time in the console.

7.3 Block Diagram

[Done By: Ng Jan Hui & Tan Wei Mun]

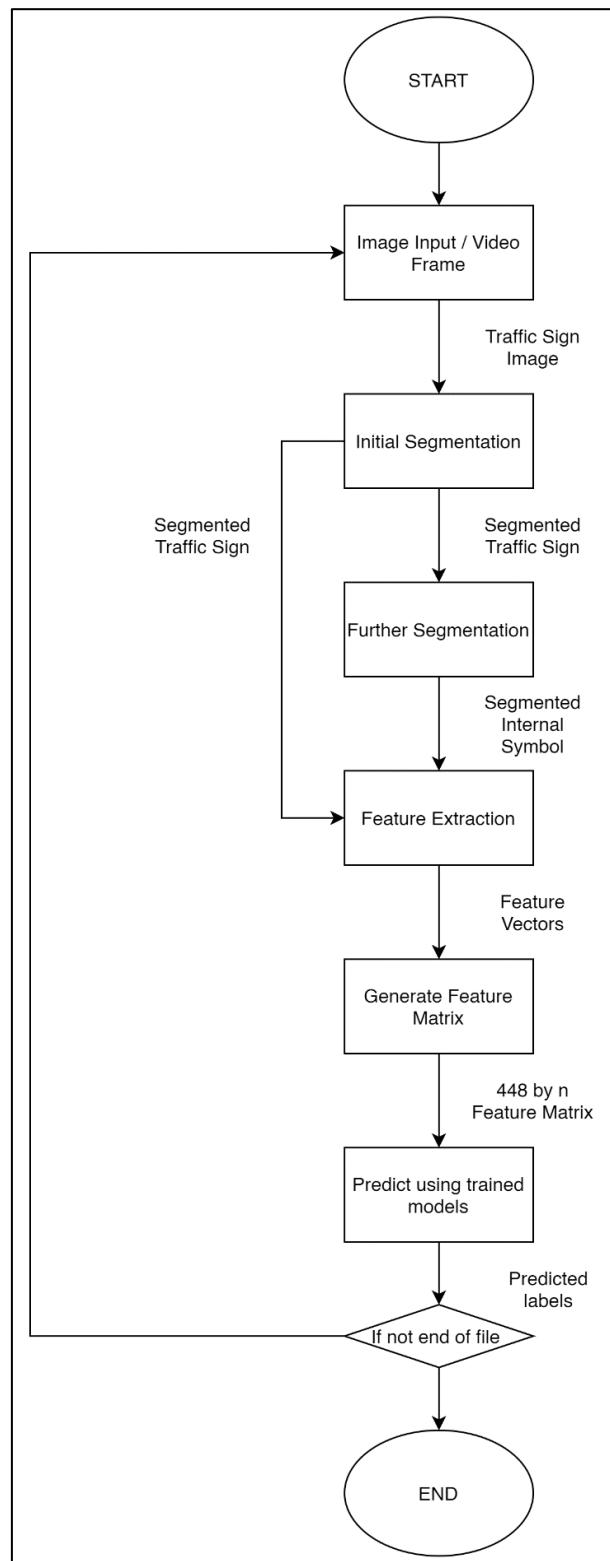


Figure 7.3.1 Block Diagram of Desktop Application

The block diagram above depicts the flow from user input to output. Still pictures, video files, and webcams are the three types of input. Even though the input type and method change, the overall flow remains the same since the video and web camera are both processed frame by frame. In addition, to classify the inputs, the trained SVM and Random Forest classifiers will be imported into the application. Following the acquisition of an image or frame, it will be segmented, feature extracted and finally classified. The more detailed flow in depicted in Figure 7.3.2. For still pictures, it will process all of the inputs before showing the input images on the window and the results on the console, each with the predicted results and confidence value. Video-based recognition for video file input or web camera input, on the other hand, is identical. A bounding box on the frame will show the location of the traffic signs, and the classification results for each frame will be displayed on the console. When the processing frame is displayed, the classification result will be displayed on the console simultaneously.

[Done By: Ng Jan Hui]

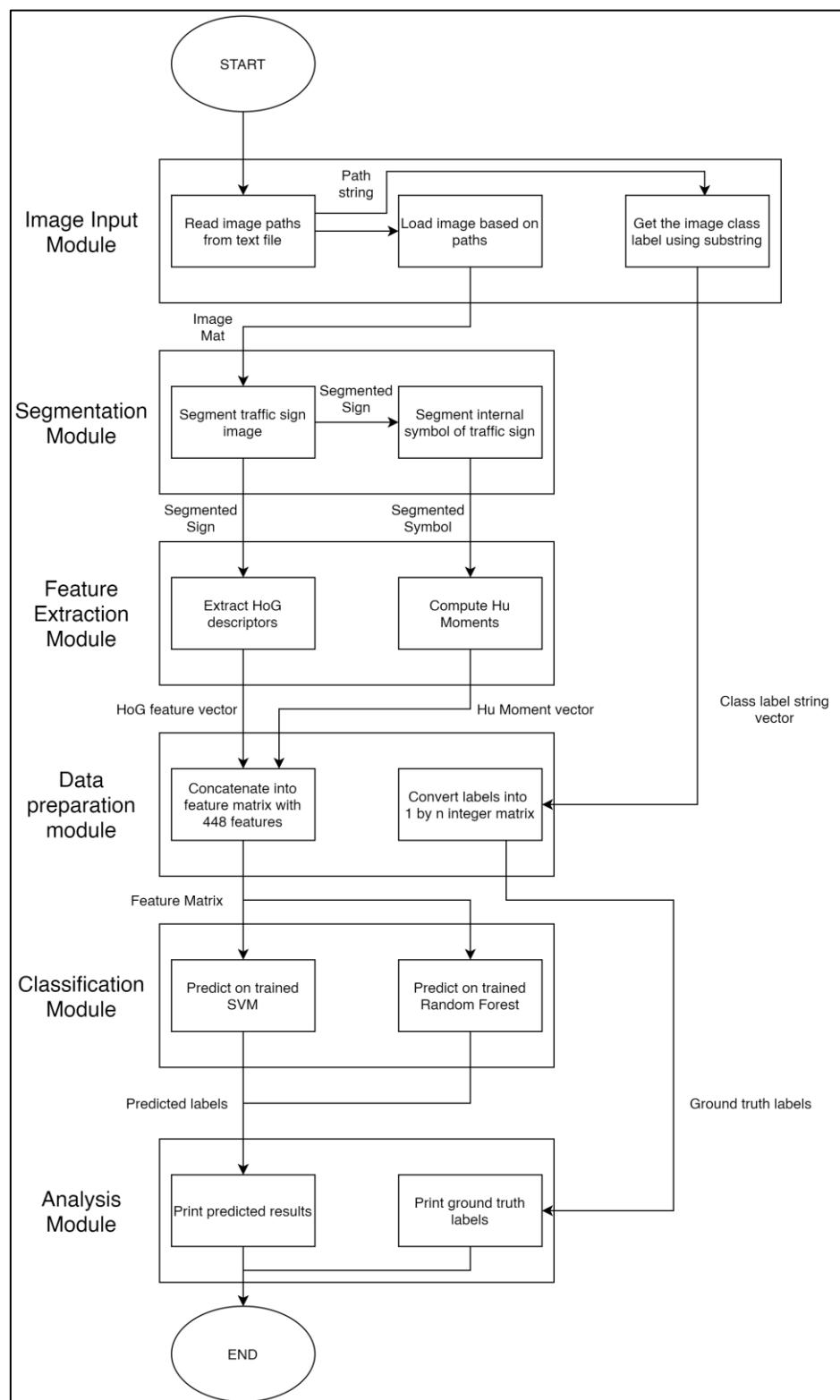


Figure 3.2.2 Block Diagram for Traffic Signs Recognizer System (Desktop Application)

7.4 Verification Plan

[Done by: Tan Wei Mun & Tan Jing Jie & Jacynth Tham Ming Quan]

The test cases for the developed traffic sign recognizing desktop application are tabulated below in Table 7.4.1 (image-based input) and Table 7.4.2 (video-based input). These test cases were noted during the desktop application development and are then evaluated together in the implementation stage.

Table 7.4.1 Image-Based Input Activity Test Cases

Id	Test Case	Expected Results
1	Users select an input image consisting of a red traffic sign	Display the names and confidence percentage of the classification result.
2	Users select an input image consisting of a blue traffic sign	Display the names and confidence percentage of the classification result.
3	Users select an input image consisting of a yellow traffic sign	Display the names and confidence percentage of the classification result.
4	Users select multiple input images containing traffic signs of varying colours	Display the names and confidence percentage of the classification result.
5	User selects one or more input images that are not trained by the model	Display invalid type sign detected and confidence

Table 7.4.2 Video-Based Input (Video file and Webcam) Activity Test Cases

Id	Test Case	Expected Results
1	Users select an input image consisting of a red traffic sign	Display the names and confidence percentage of the classification result.
2	Users select an input image consisting of a blue traffic sign	Display the names and confidence percentage of the classification result.
3	Users select an input image consisting of a yellow traffic sign	Display the names and confidence percentage of the classification result.
4	User selects one or more input images with no traffic signs	Display message that no signs are detected

The test cases above will be tested in the testing and validation subsection in this chapter.

7.5 Implementation and Testing

[Done by: Tan Wei Mun, Ng Jan Hui, Por Teong Dean]

In this subchapter, the trained SVM and random forest classifiers are implemented in the visual studio with different inputs. The test cases will be carried out in the implementation as well to test if the models meet the performance requirement.

7.5.1 User input static image

[Done by: Ng Jan Hui & Tan Jing Jie]

After training and saving the Random Forest and SVM classifiers. The models were used on an application to test on user inputs. The user may specify the path of all the traffic sign images they wish to be classified in an input text file. The program will then read from the input text file and load all the images based on the paths.

The images will then undergo the similar segmentation, feature extraction and feature matrix generation stages. Then the trained models will be loaded into the program using the provided OpenCV Algorithm::load<ml::MODEL>() function. Figure 7.4.1.1 shows the processing steps of user input images. In addition, a checking condition was implemented as an extension to the segmentation algorithm to detect whether the user inputted image can be segmented. As seen on image no.9, a segmentation failed message followed by bad lighting condition reason is shown.

Additionally, the user may input totally unseen images from the training and testing set to test the generalizability of the models. The models could also detect inputs of invalid types as per the project work specifications. Figure 7.4.1.2 shows the classification results by both models based on user specified input traffic signs.

```

Begin Segmentation
1 Input Dataset/028_1_0114.png      Segmented Successfully
2 Input Dataset/028_1_0115.png      Segmented Successfully
3 Input Dataset/017_0006.png        Segmented Successfully
4 Input Dataset/017_1_0017.png      Segmented Successfully
5 Input Dataset/037_0002.png        Segmented Successfully
6 Input Dataset/037_0005.png        Segmented Successfully
7 Input Dataset/054_1_0010.png      Segmented Successfully
8 Input Dataset/054_1_0011.png      Segmented Successfully
9 Input Dataset/055_1_0039.png      Segmented Successfully
10 Input Dataset/055_1_0040.png     Segmentation Failed (Bad Lighting Condition)
11 Input Dataset/031_1_0008.png     Segmented Successfully
12 Input Dataset/031_1_0009.png     Segmented Successfully
13 Input Dataset/000_1_0048.png     Segmented Successfully
14 Input Dataset/017_0099.png       Segmented Successfully
15 Input Dataset/031_0021.png       Segmented Successfully
16 Input Dataset/037_02_0001.png     Segmented Successfully

Begin Feature Extraction      ....Done
Generating feature matrix     ....Done
Begin prediction             ....Done

```

Figure 7.5.1.1 User Input Image Preprocessing

```

C:\Users\seann\source\repos\Project452\x64\Debug\Project452.exe
Begin Segmentation
1 Input Dataset/028_1_0114.png      Segmented Successfully
2 Input Dataset/017_0006.png        Segmented successfully
3 Input Dataset/037_0002.png        Segmented successfully
4 Input Dataset/054_1_0010.png      Segmented Successfully
5 Input Dataset/055_1_0039.png      Segmented Successfully
6 Input Dataset/031_1_0009.png     Segmented successfully
7 Input Dataset/000_1_0048.png     Segmented successfully
8 Input Dataset/017_0099.png       Segmented Successfully
9 Input Dataset/037_02_0001.png     Segmented Successfully

Begin Feature Extraction      ....Done
Generating feature matrix     ....Done
Begin prediction             ....Done

=====
Predicted signs based on user input
=====

Sign Input Name          Actual Label   SVM Prediction   Random Forest Prediction (Confidence)
Input Dataset/028_1_0114.png: Car           Car            Car (100.00%)
Input Dataset/017_0006.png: No horn        No horn        No horn (99.00%)
Input Dataset/037_0002.png: People crossing People crossing People crossing (90.00%)
Input Dataset/054_1_0010.png: No stopping   No stopping   No stopping (100.00%)
Input Dataset/055_1_0039.png: No entry      No entry      No entry (96.00%)
Input Dataset/031_1_0009.png: Uturn         Uturn         Uturn (93.00%)
Input Dataset/000_1_0048.png: Invalid Type  Car           Car (38.00%)
Input Dataset/017_0099.png: No horn        No horn      No stopping (30.00%)
Input Dataset/037_02_0001.png: People crossing People crossing People crossing (37.00%)

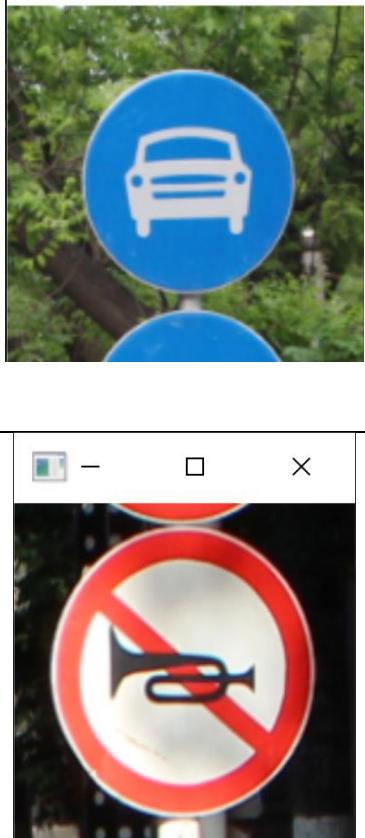
```

Figure 7.5.1.2 Classification results based on user input

[Done by: Ng Jan Hui]

The program will then output the predicted labels by the SVM and Random Forest for each input sign. Since the Random Forest is a probabilistic classifier, the sign label and its confidence value can also be obtained and be printed out on the console. Table 7.5.1.1 shows the classification results based on user inputs.

Table 7.5.1.1 Classification results

User Specified Input Images	Predicted Labels
	<p>1st Image – Actual Label</p> <p>2nd Image – SVM Prediction</p> <p>3rd Image – RF Prediction (With Confidence)</p>
	<p>Actual Label Car</p> <p>SVM Prediction Car</p> <p>Random Forest Prediction (Confidence) Car (100.00%)</p>
	*All models predicted correctly
	<p>Actual Label: No horn</p> <p>SVM Prediction: No horn</p> <p>RF Prediction: No horn (99.00%)</p>
	*All models predicted correctly

	<p>Actual Label: People crossing</p> <p>SVM Prediction: People crossing</p> <p>RF Prediction: People crossing (90.00%)</p> <p>*All models predicted correctly</p>
	<p>Actual Label: No stopping</p> <p>SVM Prediction: No stopping</p> <p>RF Prediction: No stopping (100.00%)</p> <p>*All models predicted correctly</p>
	<p>Actual Label: No entry</p> <p>SVM Prediction: No entry</p> <p>RF Prediction: No entry (96.00%)</p>

	*All models predicted correctly
	<p>Actual Label: Uturn</p> <p>SVM Prediction: Uturn</p> <p>RF Prediction: Uturn (93.00%)</p> <p>*All models predicted correctly</p>
 <p>Unseen Image (Not in Training set) Expected Label: No Horn</p>	<p>Actual Label: No horn</p> <p>SVM Prediction: No horn</p> <p>RF Prediction: No stopping (30.00%)</p> <p>*SVM Predicted Correctly</p>

 <p>Unseen Image (Not in Training set)</p> <p>Expected Label: People Crossing</p>	<p>Actual Label:</p> <p>People crossing</p> <p>SVM Prediction:</p> <p>People crossing</p> <p>RF Prediction:</p> <p>People crossing (37.00%)</p> <p>*All models predicted correctly</p>
 <p>Invalid Type (Not Trained) (As requested in project work specification)</p>	<p>Actual Label:</p> <p>Invalid Type</p> <p>SVM Prediction:</p> <p>Car</p> <p>RF Prediction:</p> <p>Car (38.00%)</p> <p>*All models predicted wrongly on invalid type</p>

The performance of the SVM and Random Forest Classifier was satisfactory as it was able to predict the user inputted signs accurately and with high confidence. In addition to that, the models were also able to predict signs that were randomly picked from Google for testing. This indicates that the segmentation process and the features extracted were powerful enough for the model to still pick up totally unfamiliar sign and classify it correctly. The code for this section can be found in [Appendix B6](#).

As requested in the project work specifications file, 10 samples for the six selected types of traffic signs were tested on the trained models. Note that images that start with 017 are No horn, 028 are Car, 031 are U-turn, 037 are People crossing, 054 are No stopping, 055 are No entry, and everything else are considered Invalid types. As demonstrated above, the 60 inputs images can be loaded into program just by specifying the path in the input text file, without having to change the structure of the code.

The results have been shown in the series of images below.

C:\Users\seann\source\repos\Project452\x64\Debug\Project452.exe					
Begin Segmentation					
1	Input	Dataset\017_0001.png	Segmented	Succesfully	
2	Input	Dataset\017_1_0001.png	Segmented	Succesfully	
3	Input	Dataset\017_1_0002.png	Segmented	Succesfully	
4	Input	Dataset\017_1_0003.png	Segmented	Succesfully	
5	Input	Dataset\017_1_0004.png	Segmented	Succesfully	
6	Input	Dataset\017_1_0005.png	Segmented	Succesfully	
7	Input	Dataset\017_1_0006.png	Segmented	Succesfully	
8	Input	Dataset\017_1_0007.png	Segmented	Succesfully	
9	Input	Dataset\017_1_0008.png	Segmented	Succesfully	
10	Input	Dataset\017_1_0009.png	Segmented	Succesfully	
11	Input	Dataset\028_0001.png	Segmented	Succesfully	
12	Input	Dataset\028_1_0001.png	Segmented	Succesfully	
13	Input	Dataset\028_1_0004.png	Segmented	Succesfully	
14	Input	Dataset\028_1_0005.png	Segmented	Succesfully	
15	Input	Dataset\028_1_0006.png	Segmented	Succesfully	
16	Input	Dataset\028_1_0114.png	Segmented	Succesfully	
17	Input	Dataset\028_1_0115.png	Segmented	Succesfully	
18	Input	Dataset\028_1_0116.png	Segmented	Succesfully	
19	Input	Dataset\028_1_0117.png	Segmented	Succesfully	
20	Input	Dataset\028_1_0118.png	Segmented	Succesfully	
21	Input	Dataset\031_0001.png	Segmented	Succesfully	
22	Input	Dataset\031_1_0001.png	Segmented	Succesfully	
23	Input	Dataset\031_1_0002.png	Segmented	Succesfully	
24	Input	Dataset\031_1_0003.png	Segmented	Succesfully	
25	Input	Dataset\031_1_0004.png	Segmented	Succesfully	
26	Input	Dataset\031_1_0005.png	Segmented	Succesfully	
27	Input	Dataset\031_1_0006.png	Segmented	Succesfully	
28	Input	Dataset\031_1_0007.png	Segmented	Succesfully	
29	Input	Dataset\031_1_0008.png	Segmented	Succesfully	
30	Input	Dataset\031_1_0009.png	Segmented	Succesfully	
31	Input	Dataset\037_0001.png	Segmented	Succesfully	
32	Input	Dataset\037_0002.png	Segmented	Succesfully	
33	Input	Dataset\037_1_0001.png	Segmented	Succesfully	
34	Input	Dataset\037_1_0002.png	Segmented	Succesfully	
35	Input	Dataset\037_1_0003.png	Segmented	Succesfully	
36	Input	Dataset\037_1_0004.png	Segmented	Succesfully	
37	Input	Dataset\037_1_0005.png	Segmented	Succesfully	
38	Input	Dataset\037_1_0006.png	Segmented	Succesfully	
39	Input	Dataset\037_1_0009.png	Segmented	Succesfully	
40	Input	Dataset\037_1_0010.png	Segmented	Succesfully	
41	Input	Dataset\054_1_0006.png	Segmented	Succesfully	
42	Input	Dataset\054_1_0007.png	Segmented	Succesfully	
43	Input	Dataset\054_1_0008.png	Segmented	Succesfully	
44	Input	Dataset\054_1_0009.png	Segmented	Succesfully	
45	Input	Dataset\054_1_0010.png	Segmented	Succesfully	
46	Input	Dataset\054_1_0011.png	Segmented	Succesfully	
47	Input	Dataset\054_1_0012.png	Segmented	Succesfully	
48	Input	Dataset\054_1_0021.png	Segmented	Succesfully	
49	Input	Dataset\054_1_0036.png	Segmented	Succesfully	
50	Input	Dataset\054_1_0037.png	Segmented	Succesfully	
51	Input	Dataset\055_0012.png	Segmented	Succesfully	
52	Input	Dataset\055_0013.png	Segmented	Succesfully	
53	Input	Dataset\055_1_0009.png	Segmented	Succesfully	
54	Input	Dataset\055_1_0012.png	Segmented	Succesfully	
55	Input	Dataset\055_1_0013.png	Segmented	Succesfully	
56	Input	Dataset\055_1_0039.png	Segmented	Succesfully	
57	Input	Dataset\055_1_0053.png	Segmented	Succesfully	
58	Input	Dataset\055_1_0054.png	Segmented	Succesfully	
59	Input	Dataset\055_1_0055.png	Segmented	Succesfully	
60	Input	Dataset\055_1_0065.png	Segmented	Succesfully	

Figure 7.5.1.3 Sending 60 images for segmentation

Chapter 7 Desktop application

Predicted signs based on user input				
Sign Input Name	Actual Label	SVM Prediction	Random Forest Prediction (Confidence)	
Input Dataset/017_0001.png:	No horn	No horn	No horn (96.00%)	
Input Dataset/017_1_0001.png:	No horn	No horn	No horn (96.00%)	
Input Dataset/017_1_0002.png:	No horn	No horn	No horn (98.00%)	
Input Dataset/017_1_0003.png:	No horn	No horn	No horn (94.00%)	
Input Dataset/017_1_0004.png:	No horn	No horn	No horn (99.00%)	
Input Dataset/017_1_0005.png:	No horn	No horn	No horn (98.00%)	
Input Dataset/017_1_0006.png:	No horn	No horn	No horn (99.00%)	
Input Dataset/017_1_0007.png:	No horn	No horn	No horn (92.00%)	
Input Dataset/017_1_0008.png:	No horn	No horn	No horn (97.00%)	
Input Dataset/017_1_0009.png:	No horn	No horn	No horn (97.00%)	
Input Dataset/028_0001.png:	Car	No horn	car (87.00%)	
Input Dataset/028_1_0001.png:	Car	No horn	car (87.00%)	
Input Dataset/028_1_0004.png:	Car	No horn	car (89.00%)	
Input Dataset/028_1_0005.png:	Car	No horn	car (96.00%)	
Input Dataset/028_1_0006.png:	Car	No horn	car (96.00%)	
Input Dataset/028_1_0114.png:	Car	Car	Car (100.00%)	
Input Dataset/028_1_0115.png:	Car	Car	Car (100.00%)	
Input Dataset/028_1_0116.png:	Car	Car	Car (100.00%)	
Input Dataset/028_1_0117.png:	Car	Car	Car (100.00%)	
Input Dataset/028_1_0118.png:	Car	Car	Car (100.00%)	
Input Dataset/031_0001.png:	Uturn	Uturn	uturn (97.00%)	
Input Dataset/031_1_0001.png:	Uturn	Uturn	uturn (97.00%)	
Input Dataset/031_1_0002.png:	Uturn	Uturn	uturn (99.00%)	
Input Dataset/031_1_0003.png:	Uturn	Uturn	uturn (97.00%)	
Input Dataset/031_1_0004.png:	Uturn	Uturn	uturn (94.00%)	

Figure 7.5.1.4 Predictions (1/3)

Input Dataset/031_1_0005.png:	Uturn	Uturn	Uturn (95.00%)
Input Dataset/031_1_0006.png:	Uturn	Uturn	Uturn (97.00%)
Input Dataset/031_1_0007.png:	Uturn	Uturn	Uturn (90.00%)
Input Dataset/031_1_0008.png:	Uturn	Uturn	Uturn (97.00%)
Input Dataset/031_1_0009.png:	Uturn	Uturn	Uturn (93.00%)
Input Dataset/037_0001.png:	People crossing	People crossing	People crossing (89.00%)
Input Dataset/037_0002.png:	People crossing	People crossing	People crossing (90.00%)
Input Dataset/037_1_0001.png:	People crossing	People crossing	People crossing (89.00%)
Input Dataset/037_1_0002.png:	People crossing	People crossing	People crossing (90.00%)
Input Dataset/037_1_0003.png:	People crossing	People crossing	People crossing (87.00%)
Input Dataset/037_1_0004.png:	People crossing	People crossing	People crossing (93.00%)
Input Dataset/037_1_0005.png:	People crossing	People crossing	People crossing (91.00%)
Input Dataset/037_1_0006.png:	People crossing	People crossing	People crossing (87.00%)
Input Dataset/037_1_0009.png:	People crossing	People crossing	People crossing (92.00%)
Input Dataset/037_1_0010.png:	People crossing	No horn	People crossing (80.00%)
Input Dataset/054_1_0006.png:	No stopping	No horn	No stopping (96.00%)
Input Dataset/054_1_0007.png:	No stopping	No stopping	No stopping (100.00%)
Input Dataset/054_1_0008.png:	No stopping	No horn	No stopping (99.00%)
Input Dataset/054_1_0009.png:	No stopping	No horn	No stopping (99.00%)
Input Dataset/054_1_0010.png:	No stopping	No stopping	No stopping (100.00%)
Input Dataset/054_1_0011.png:	No stopping	No stopping	No stopping (100.00%)
Input Dataset/054_1_0012.png:	No stopping	No stopping	No stopping (100.00%)
Input Dataset/054_1_0021.png:	No stopping	No horn	No stopping (94.00%)
Input Dataset/054_1_0036.png:	No stopping	No horn	No stopping (98.00%)
Input Dataset/054_1_0037.png:	No stopping	No horn	No stopping (98.00%)
Input Dataset/055_0012.png:	No entry	No horn	No entry (60.00%)
Input Dataset/055_0013.png:	No entry	No entry	No entry (92.00%)
Input Dataset/055_1_0009.png:	No entry	No horn	No entry (71.00%)
Input Dataset/055_1_0012.png:	No entry	No horn	No entry (60.00%)
Input Dataset/055_1_0013.png:	No entry	No entry	No entry (92.00%)
Input Dataset/055_1_0039.png:	No entry	No entry	No entry (96.00%)
Input Dataset/055_1_0053.png:	No entry	No entry	No entry (97.00%)

Figure 7.5.1.5 Predictions (2/3)

Input Dataset/055_1_0054.png:	No entry	No entry	No entry (96.00%)
Input Dataset/055_1_0055.png:	No entry	No entry	No entry (100.00%)
Input Dataset/055_1_0065.png:	No entry	No horn	No entry (52.00%)

Figure 7.5.1.6 Predictions (3/3)

The trained models performed quite well when traffic sign images were spontaneously inputted for prediction. Although there were still some errors in the SVM prediction (indicated in red box), however the Random Forest classifier was able to predict 100% correct for all the 60 images.

7.5.2 Video-Based Recognition

[Done by: Tan Wei Mun]

Besides, the trained SVM and Random Forest classifiers were integrated into a video-based desktop application. First, the trained models are loaded using the Load() function provided in the OpenCV ml library. Next, the desktop application takes in the video frame by frame. The video frames are next being pre-processed, segmented, and its features are then being extracted and converted to a feature matrix, and lastly, the produced matrix is classified to obtain the classification results. The output of the developed video-based desktop application has two parts – the detected traffic sign in the camera frame was bounded by a white bounding box.

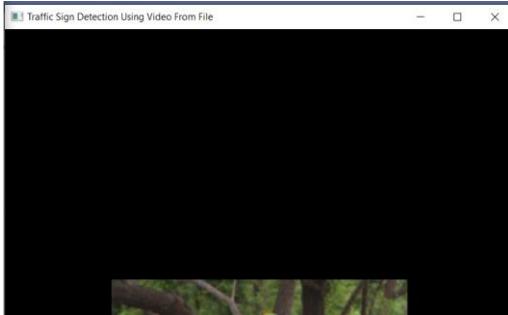
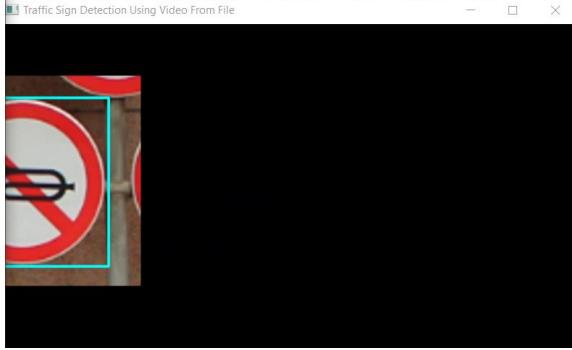
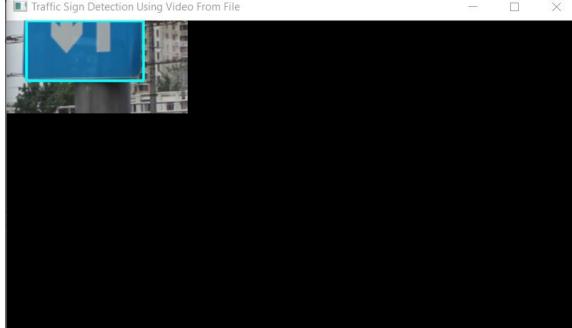
The bounding box is used in tracking the traffic signs as it moves around in the camera frame. The bounding box will be generated on each traffic signs in the frame while segmenting the traffic signs. The purpose of the bounding box is to show the detected traffic signs and the region of interest with a square box before inputting the image into the classification model. Each of the bounding box has the following attributes:

- X-axis coordinates corresponding to the center of the bounding box
- Y-axis coordinates corresponding to the center of the bounding box
- Width of the bounding box
- Height of the bounding box
- Traffic sign's class label (which will appear after the final classification process)

On the other hand, the classification results were shown in the console log. The predictions of the SVM model were listed on the left, and the predictions of the Random Forest model were listed on the right-hand side. When the application segment zero images, it assumes there is no traffic in the current frame and thus alert no traffic signs detected.

A test run of the video-based implementation and testing of expected test cases are shown in table 7.5.2.1.

Table 7.5.2.1 Video-based implementation testing

Video frames	Predicted Labels and Analysis
 	<p>No signs detected</p>
	<p>Case: No traffic signs in the frames</p> <p>Output: It will output the message that no traffic signs are detected.</p>
 	<p>Classifiers SVM RF (Confidence) Prediction: Utturn No horn (22.00%)</p> <p>Classifiers SVM RF (Confidence) Prediction: Utturn No entry (32.00%)</p> <p>Classifiers SVM RF (Confidence) Prediction: No horn No entry (29.00%)</p> <p>Classifiers SVM RF (Confidence) Prediction: No horn No entry (38.00%)</p> <p>Classifiers SVM RF (Confidence) Prediction: No horn No entry (36.00%)</p> <p>Classifiers SVM RF (Confidence) Prediction: No horn No entry (34.00%)</p> <p>Classifiers SVM RF (Confidence) Prediction: No horn No entry (31.00%)</p> <p>Classifiers SVM RF (Confidence) Prediction: No horn No entry (35.00%)</p>

	<p>Classifiers SVM RF (Confidence) Prediction: No horn No entry (35.00%)</p> <p>Classifiers SVM RF (Confidence) Prediction: No horn No entry (35.00%)</p> <p>Classifiers SVM RF (Confidence) Prediction: No horn No entry (36.00%)</p> <p>Classifiers SVM RF (Confidence) Prediction: No horn No entry (39.00%)</p>
	<p>Classifiers SVM RF (Confidence) Prediction: People crossing No stopping (29.00%)</p> <p>Classifiers SVM RF (Confidence) Prediction: People crossing No stopping (35.00%)</p> <p>Classifiers SVM RF (Confidence) Prediction: People crossing No entry (27.00%)</p> <p>Classifiers SVM RF (Confidence) Prediction: No horn No stopping (28.00%)</p> <p>Classifiers SVM RF (Confidence) Prediction: No horn No horn (28.00%)</p>

Case: incomplete signs.

Output: It will still be recognized as other classes signs as well. SVM will recognize incomplete as no horn sign, whereas Random Forest recognizes as no entry signs.

Classifiers SVM RF (Confidence)
Prediction: People crossing No stopping (29.00%)

Classifiers SVM RF (Confidence)
Prediction: People crossing No stopping (35.00%)

Classifiers SVM RF (Confidence)
Prediction: People crossing No entry (27.00%)

Classifiers SVM RF (Confidence)
Prediction: No horn No stopping (28.00%)

Classifiers SVM RF (Confidence)
Prediction: No horn No horn (28.00%)

Case: No horn signs

Output: Both classifiers can detect “no horn” signs, but they will still be wrong sometimes.

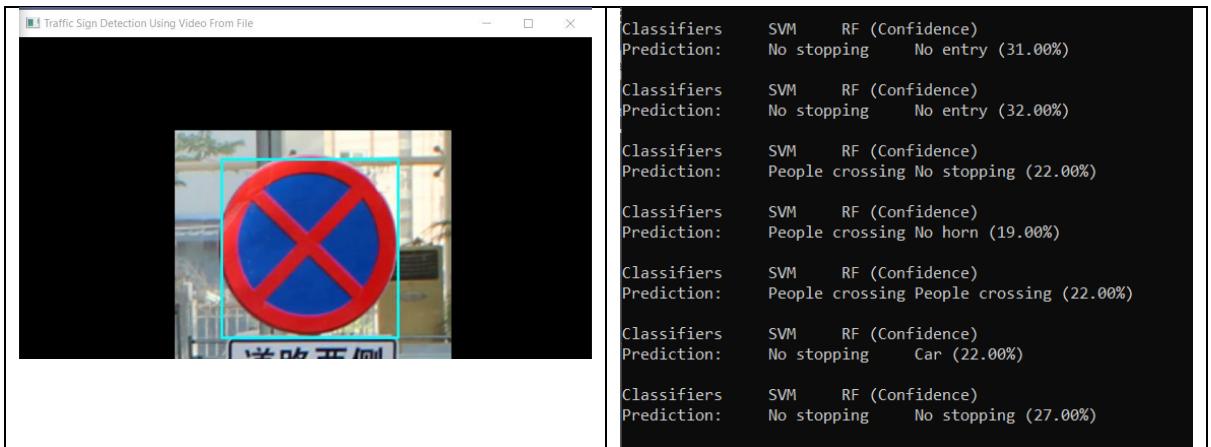
	<table border="1"> <thead> <tr> <th>Classifiers</th><th>SVM</th><th>RF (Confidence)</th></tr> </thead> <tbody> <tr> <td>Prediction:</td><td>Uturn</td><td>No entry (38.00%)</td></tr> <tr> <td>Prediction:</td><td>Uturn</td><td>No entry (41.00%)</td></tr> <tr> <td>Prediction:</td><td>Uturn</td><td>No entry (38.00%)</td></tr> <tr> <td>Prediction:</td><td>Uturn</td><td>No entry (29.00%)</td></tr> <tr> <td>Prediction:</td><td>Uturn</td><td>RF (Confidence)</td></tr> <tr> <td>Prediction:</td><td>Uturn</td><td>Uturn (44.00%)</td></tr> <tr> <td>Prediction:</td><td>Uturn</td><td>RF (Confidence)</td></tr> <tr> <td>Prediction:</td><td>Uturn</td><td>Uturn (47.00%)</td></tr> <tr> <td>Prediction:</td><td>Uturn</td><td>RF (Confidence)</td></tr> <tr> <td>Prediction:</td><td>Uturn</td><td>Uturn (43.00%)</td></tr> </tbody> </table>	Classifiers	SVM	RF (Confidence)	Prediction:	Uturn	No entry (38.00%)	Prediction:	Uturn	No entry (41.00%)	Prediction:	Uturn	No entry (38.00%)	Prediction:	Uturn	No entry (29.00%)	Prediction:	Uturn	RF (Confidence)	Prediction:	Uturn	Uturn (44.00%)	Prediction:	Uturn	RF (Confidence)	Prediction:	Uturn	Uturn (47.00%)	Prediction:	Uturn	RF (Confidence)	Prediction:	Uturn	Uturn (43.00%)									
Classifiers	SVM	RF (Confidence)																																									
Prediction:	Uturn	No entry (38.00%)																																									
Prediction:	Uturn	No entry (41.00%)																																									
Prediction:	Uturn	No entry (38.00%)																																									
Prediction:	Uturn	No entry (29.00%)																																									
Prediction:	Uturn	RF (Confidence)																																									
Prediction:	Uturn	Uturn (44.00%)																																									
Prediction:	Uturn	RF (Confidence)																																									
Prediction:	Uturn	Uturn (47.00%)																																									
Prediction:	Uturn	RF (Confidence)																																									
Prediction:	Uturn	Uturn (43.00%)																																									
	<table border="1"> <thead> <tr> <th>Classifiers</th><th>SVM</th><th>RF (Confidence)</th></tr> </thead> <tbody> <tr> <td>Prediction:</td><td>People crossing</td><td>No entry (30.00%)</td></tr> <tr> <td>Prediction:</td><td>SVM</td><td>RF (Confidence)</td></tr> <tr> <td>Prediction:</td><td>People crossing</td><td>People crossing (39.00%)</td></tr> <tr> <td>Prediction:</td><td>SVM</td><td>RF (Confidence)</td></tr> <tr> <td>Prediction:</td><td>People crossing</td><td>People crossing (58.00%)</td></tr> <tr> <td>Prediction:</td><td>SVM</td><td>RF (Confidence)</td></tr> <tr> <td>Prediction:</td><td>People crossing</td><td>People crossing (72.00%)</td></tr> <tr> <td>Prediction:</td><td>SVM</td><td>RF (Confidence)</td></tr> <tr> <td>Prediction:</td><td>People crossing</td><td>People crossing (66.00%)</td></tr> <tr> <td>Prediction:</td><td>SVM</td><td>RF (Confidence)</td></tr> <tr> <td>Prediction:</td><td>People crossing</td><td>People crossing (54.00%)</td></tr> <tr> <td>Prediction:</td><td>SVM</td><td>RF (Confidence)</td></tr> <tr> <td>Prediction:</td><td>People crossing</td><td>People crossing (47.00%)</td></tr> </tbody> </table>	Classifiers	SVM	RF (Confidence)	Prediction:	People crossing	No entry (30.00%)	Prediction:	SVM	RF (Confidence)	Prediction:	People crossing	People crossing (39.00%)	Prediction:	SVM	RF (Confidence)	Prediction:	People crossing	People crossing (58.00%)	Prediction:	SVM	RF (Confidence)	Prediction:	People crossing	People crossing (72.00%)	Prediction:	SVM	RF (Confidence)	Prediction:	People crossing	People crossing (66.00%)	Prediction:	SVM	RF (Confidence)	Prediction:	People crossing	People crossing (54.00%)	Prediction:	SVM	RF (Confidence)	Prediction:	People crossing	People crossing (47.00%)
Classifiers	SVM	RF (Confidence)																																									
Prediction:	People crossing	No entry (30.00%)																																									
Prediction:	SVM	RF (Confidence)																																									
Prediction:	People crossing	People crossing (39.00%)																																									
Prediction:	SVM	RF (Confidence)																																									
Prediction:	People crossing	People crossing (58.00%)																																									
Prediction:	SVM	RF (Confidence)																																									
Prediction:	People crossing	People crossing (72.00%)																																									
Prediction:	SVM	RF (Confidence)																																									
Prediction:	People crossing	People crossing (66.00%)																																									
Prediction:	SVM	RF (Confidence)																																									
Prediction:	People crossing	People crossing (54.00%)																																									
Prediction:	SVM	RF (Confidence)																																									
Prediction:	People crossing	People crossing (47.00%)																																									

Case: U-turn signs.

Output: Although both classifiers can recognize this sign, the SVM is slightly more accurate than Random Forest.

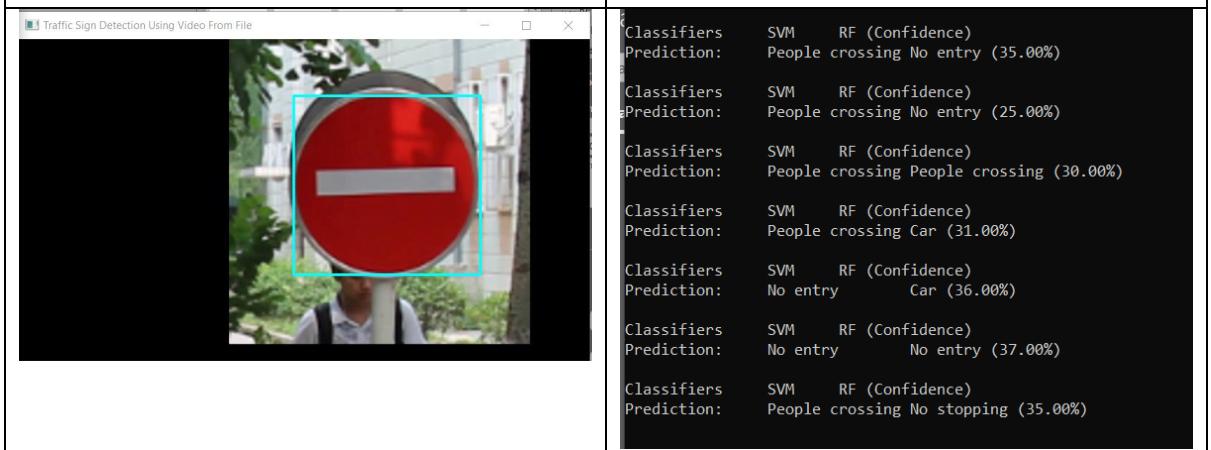
Case: People crossing sign

Output: Both classifiers can also recognize this sign.



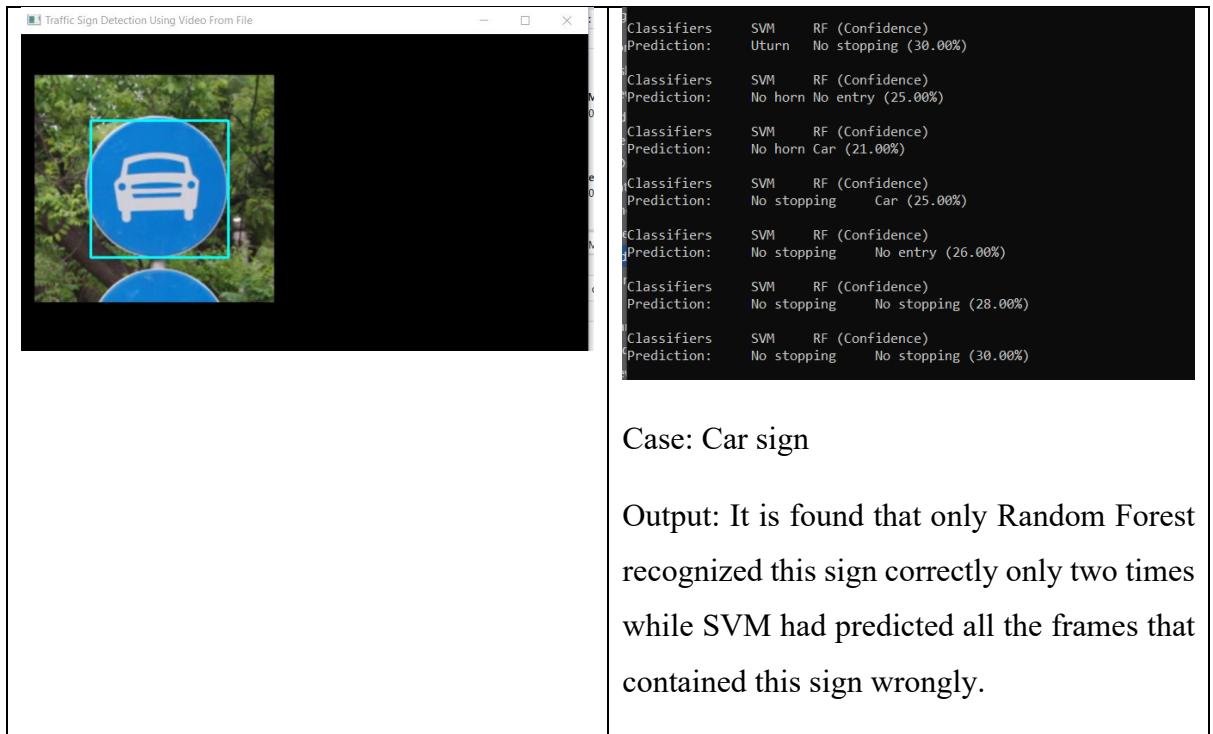
Case: No stopping signs

Output: Both classifiers still can detect the sign, but the accuracy is slightly dropped.



Case: No entry signs

Output: Even both classifiers can still recognize the sign correctly, but they recognize no entry signs mostly as people crossing.



Case: Car sign

Output: It is found that only Random Forest recognized this sign correctly only two times while SVM had predicted all the frames that contained this sign wrongly.

From the test cases above, the SVM classifier had classified more signs correctly than the Random Forest classifier, which met the accuracies calculated in the classification chapter (SVM achieved higher testing accuracy than Random Forest using the same testing data). While when the incomplete signs are segmented and classified by the classifiers, the SVM classifier usually classifies all the incomplete as no horn signs while Random Forest classifiers are predicted as no entry sign. This could be improved in the future by adding the shape recognition with the colour segmentation to drop the segmented images that have no completed circle, triangle and square.

Without further ado, the performance for video-based recognition is slightly dropped compared to still image recognition. The traffic signs captured in the video might be distorted when the signs are moving, thus leading to unsatisfactory accuracy for some traffic signs. Besides, it is found that the feature extraction step of the traffic signs took time, making real-time recognition of the video file a bit lag in which the output of the traffic signs bounding box on the frame is a bit slow compared to the original video speed.

Last but not least, the whole video-based recognition coding could be found in [Appendix B6](#).

7.5.3 Camera Based Recognition

[Done By: Por Teong Dean]

Next, the trained SVM and Random Forest classifiers were integrated into a real-time, webcam-based input desktop application. The desktop application takes in each camera frame, pre-processes, segments, and classifies the frames to obtain the classification results. The output of the developed webcam-based desktop application has two parts – the detected traffic sign in the camera frame was bounded by a white bounding box. This bounding box tracks the traffic signs as it moves around in the camera frame. On the other hand, the classification results were shown in the console log. The predictions of the SVM model were listed on the left and the predictions of the Random Forest model were listed on the right-hand side. Table 7.5.3.1 below tabulates the results of the desktop application.

Table 7.5.3.1 Webcam-Based Traffic Sign Recognizer Results

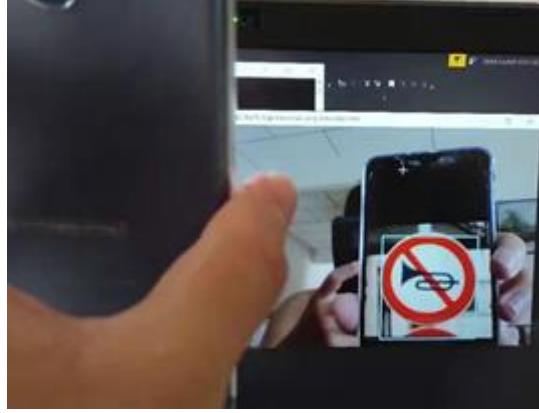
Test Case Name	Red color No horn traffic sign																														
Test Case Description	Test the traffic sign by using own laptop camera																														
Expected Output	Display the result of the traffic sign name after classified by SVM and Random Forest Classifiers																														
Input																															
Results	<table border="0"> <tr> <td>Classifiers</td> <td>SVM</td> <td>RF</td> </tr> <tr> <td>Prediction:</td> <td>No horn</td> <td>No horn</td> </tr> <tr> <td>Prediction:</td> <td>No stopping</td> <td>Car</td> </tr> <tr> <td>Prediction:</td> <td>No horn</td> <td>No horn</td> </tr> <tr> <td>Prediction:</td> <td>No horn</td> <td>No horn</td> </tr> <tr> <td>Prediction:</td> <td>People crossing</td> <td>Car</td> </tr> <tr> <td>Prediction:</td> <td>No horn</td> <td>Car</td> </tr> </table>	Classifiers	SVM	RF	Prediction:	No horn	No horn	Prediction:	No stopping	Car	Prediction:	No horn	No horn	Prediction:	No horn	No horn	Prediction:	People crossing	Car	Prediction:	No horn	Car									
Classifiers	SVM	RF																													
Prediction:	No horn	No horn																													
Prediction:	No horn	No horn																													
Prediction:	No horn	No horn																													
Prediction:	No horn	No horn																													
Prediction:	No stopping	Car																													
Prediction:	No horn	No horn																													
Prediction:	No horn	No horn																													
Prediction:	People crossing	Car																													
Prediction:	No horn	Car																													
Description	Both classifiers can detect “no horn” signs but they will still be wrong sometimes.																														

Table 7.5.3.2 Webcam-Based Traffic Sign Recognizer Results

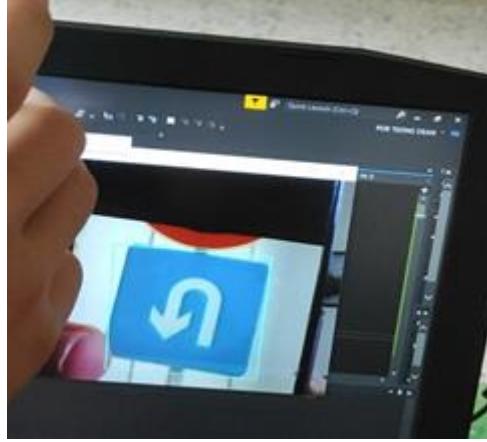
Test Case Name	Blue color Uturn traffic sign
Test Case Description	Test the traffic sign by using own laptop camera
Expected Output	Display the result of the traffic sign name after classified by SVM and Random Forest Classifiers
Input	
Results	Prediction: Uturn Car Prediction: Uturn Uturn Prediction: Uturn Uturn Prediction: Uturn Car Prediction: Uturn Uturn
Description	Both classifiers can detect “Uturn” signs but they will still be wrong sometimes. The prediction of SVM classifier is better than the Random Forest classifier.

Table 7.5.3.3 Webcam-Based Traffic Sign Recognizer Results

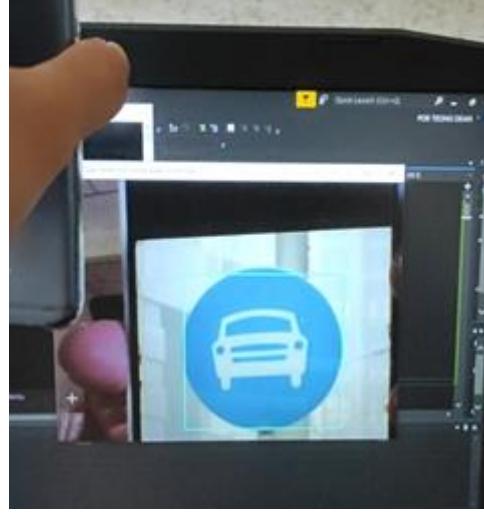
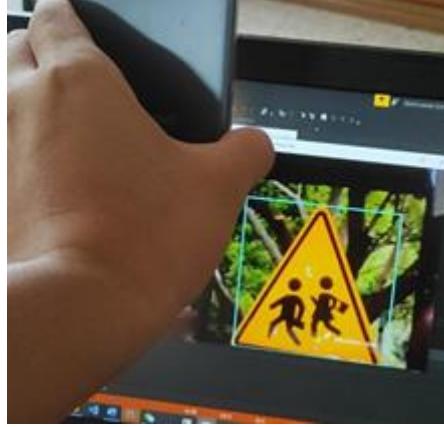
Test Case Name	Blue color Car traffic sign																								
Test Case Description	Test the traffic sign by using own laptop camera																								
Expected Output	Display the result of the traffic sign name after classified by SVM and Random Forest Classifiers																								
Input																									
Results	<table border="1"> <tr><td>Prediction:</td><td>Uturn</td><td>Car</td></tr> <tr><td>Prediction:</td><td>Car</td><td>Car</td></tr> <tr><td>Prediction:</td><td>Car</td><td>Car</td></tr> <tr><td>Prediction:</td><td>Car</td><td>Car</td></tr> <tr><td>Prediction:</td><td>Car</td><td>Car</td></tr> <tr><td>Prediction:</td><td>Uturn</td><td>Car</td></tr> <tr><td>Prediction:</td><td>Uturn</td><td>Car</td></tr> <tr><td>Prediction:</td><td>Uturn</td><td>Car</td></tr> </table>	Prediction:	Uturn	Car	Prediction:	Car	Car	Prediction:	Uturn	Car	Prediction:	Uturn	Car	Prediction:	Uturn	Car									
Prediction:	Uturn	Car																							
Prediction:	Car	Car																							
Prediction:	Car	Car																							
Prediction:	Car	Car																							
Prediction:	Car	Car																							
Prediction:	Uturn	Car																							
Prediction:	Uturn	Car																							
Prediction:	Uturn	Car																							
Description	Both classifiers can detect “car” sign but the prediction result of the Random Forest is much better than the SVM classifier.																								

Table 7.5.3.4 Webcam-Based Traffic Sign Recognizer Results

Test Case Name	Yellow color people crossing traffic sign
Test Case Description	Test the traffic sign by using own laptop camera
Expected Output	Display the result of the traffic sign name after classified by SVM and Random Forest Classifiers
Input	
Results	<pre> Prediction: People crossing No stopping Prediction: People crossing People crossing Prediction: People crossing Car </pre>
Description	Both classifiers can detect “people crossing” sign but the prediction result of the SVM is much better than the Random Forest classifier

Both of the SVM and Random Forest classifiers can detect well for the No horn sign. For both the Uturn and People Crossing Sign, the prediction performance of SVM is better than the Random Forest Classifier and for the Car Sign, Random Forrest predicts well compare with the SVM classifier.

This proposed camera-based traffic sign classification still has a lot of barriers that need to conquer such as the classification result will easily be affected by the lighting condition of the camera area. Furthermore, when dealing with blur images, the prediction result for both of the classifiers will be unstable.

In a summary, this proposed camera-based traffic sign classification needs to be further improved to get a stable, and highly accurate result.

7.6 Summary

[Done By: Jacynth Tham Ming Quan & Tan Jing Jie]

In short, the 3 desktop applications – image-based, video-based and webcam-based – were able to recognize traffic signs varying in sizes and colors (red, blue and yellow). All three of the developed desktop applications used the image pre-processing, feature extraction and classification models (SVM and Random Forest) that had been explored and tested in the earlier chapters. Furthermore, the testing stage of the three desktop applications evidenced that the accuracies obtained from Chapter 6 were still adhered to, since majority of the test cases could be predicted correctly.

Chapter 8

Android application

8.0 Overview

[Done By: Jacynth Tham Ming Quan & Tan Jing Jie]

In this chapter, the trained MobileNet TensorFlowLite model that was tested and exported previously is integrated into an Android application through the use of Android Studio and OpenCV Java wrappers. The developed mobile application is completed with a comprehensive user interface and the options to use file-based image inputs of real-time video stream inputs for the recognition process. The outputs of the mobile application for both image-based and video-based inputs include the class label of the recognized traffic sign as well as voice outputs. Last but not least, in order to demonstrate the developed application's capabilities to be integrated into smart cars, various test cases were applied to verify that the developed mobile application works as intended.

8.1 Methodology and Tools

[Done By: Tan Jing Jie, Jacynth Tham Ming Quan & Por Teong Dean]

Having explored, trained, and tested the traffic sign segmentation process all the way up to the classification process, the exported MobileNet classifier is implemented into an Android application with voice outputs. The developed Android application is coded in full Java and is only executable on Android smartphones. Furthermore, users are given the option to select still-image input or real-time video stream input, both of which share the same image pre-processing, segmentation, and classification steps.

The tool used in the development of this module is Android Studio. The developments of the UI, camera permissions handling and OpenCV integrations were handled by the libraries provided by Android Studio dependencies such as RXPermissions. As the MobileNet model had already been converted into a TensorFlowLite file, the TensorFlow libraries do not have to be tampered with in the

development of the Android application. However, interpreter library, TensorFlowLite dependency needs to be included in the project gradle.

The tools used to develop this module are as listed below:

1. **Android Studio** – The main IDE used to develop the user interface of the mobile application.
2. **RxPermission** – An external library used to prompt camera permissions from the user.
3. **TensorFlowLite** – An external library used to support the imported MobileNet model for the classification process.
4. **OpenCV Library** – An external library used to provide image preprocessing method such as bilateralFilter and et cetera.

8.2 Requirement

[Done By: Jacynth Tham Ming Quan & Ng Jan Hui]

Upon the completion of the Android application, users will be able to recognize traffic signs from their mobile devices through still-images or real-time video streams input. In the file-based image input activity, users should be able to export the segmented traffic sign to a folder in their gallery. On the other hand, for the real-time video-based activity, users will be able to see the detected traffic sign bounded with a white bounding box. Both activities will be able to notify the user if no traffic sign is recognized in the current image or camera frame. Moreover, the Android application should be able to output the top few classification results with the classification confidence of each result coupled with voice outputs to benefit the users on the road.

8.3 Block Diagram

[Done By: Tan Jing Jie]

The block diagram below shows the overall flow from user input to output. In the very beginning, users must select one of the input method option.

If the “From File” action is selected, the user will be prompted to select an image from his cell phone gallery. After an image is obtained, it will undergo the process of segmentation and classification. On the other hand, if the “From Camera” option is selected, the user will be directed to a preview screen which will show a bounding box to the segmented traffic sign. Users may press the “output” button to get the classification result. classification.

After obtaining the result, both input methods (“From File” or “From Camera”) will show the result in a text box. For the “from camera” input option, the result will be spoken out immediately, while for “from file” the output will be spoken out when the user clicks the text box.

Lastly, there is an export function for the “From Camera” input option. When the button is clicked, the segmented traffic sign will save into the phone local directory with the name of the traffic sign classification result.

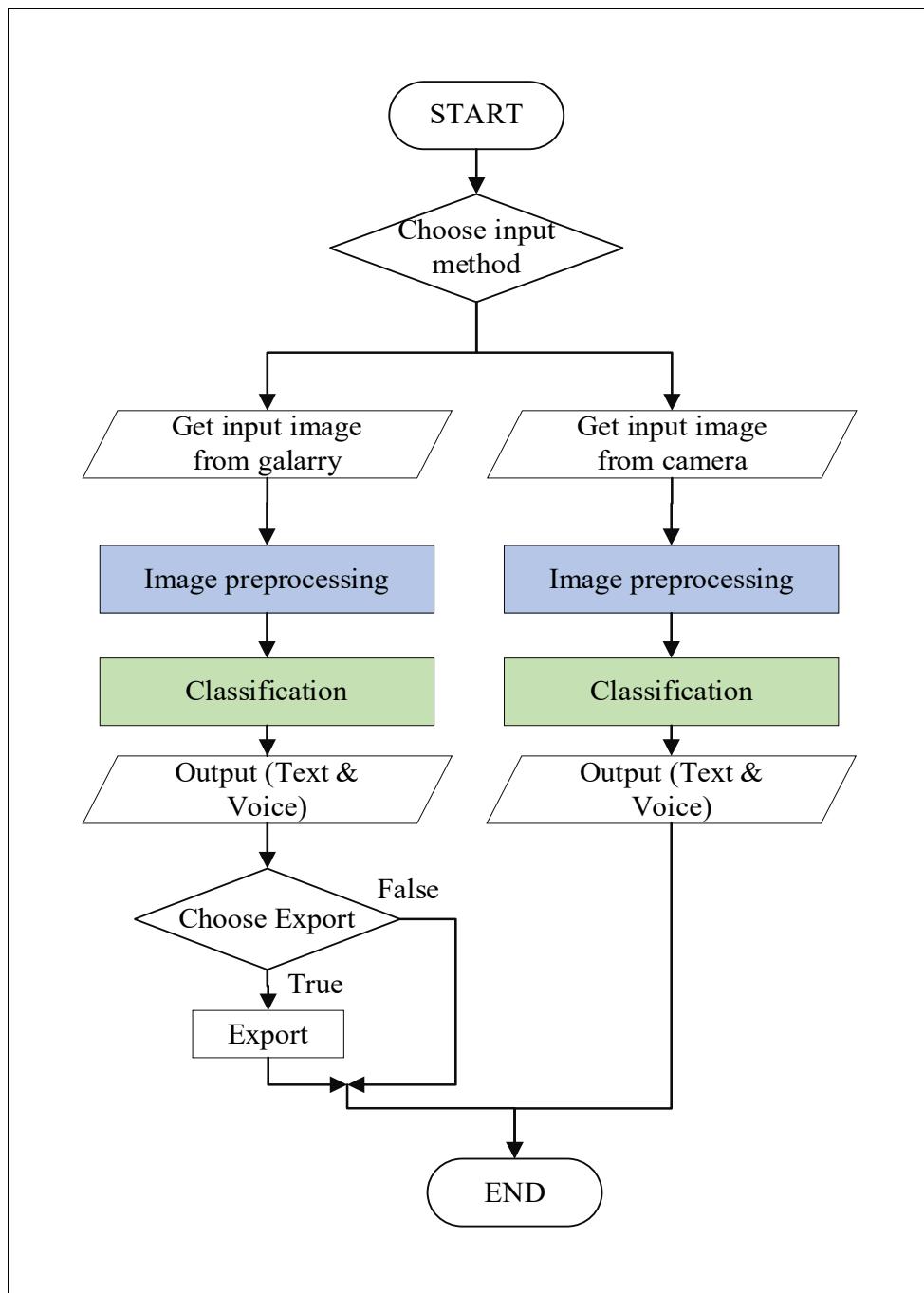


Figure 8.3.1 The Overall Flow of Android Mobile Application

8.4 Verification Plan

[Done By: Tan Jing Jie & Jacynth Tham Ming Quan & Ng Jan Hui]

The test cases for the developed traffic sign recognizing mobile application are tabulated below in Table 8.4.1 (image-based input) and Table 8.4.2 (video-based input). These test cases were noted of during the development of the mobile application and further tested during the testing and validation stage.

Table 8.4.1 Image-Based Input Activity Test Cases

Id	Test Case	Expected Results
1	Users select an input image consisting of a red traffic sign	Display the names and confidence percentage of the classification result.
2	Users select an input image consisting of a blue traffic sign	Display the names and confidence percentage of the classification result.
3	Users select an input image consisting of a yellow traffic sign	Display the names and confidence percentage of the classification result.
4	Users select multiple input images containing traffic signs of different lighting condition	Display the names and confidence percentage of the classification result in view pager.
5	Users select multiple input images containing traffic signs of varying colours	Display the names and confidence percentage of the classification result in view pager.
6	Users press on the output textbox	Read the classification output out loud (voice output)
7	User selects one or more input images with no traffic signs	Display “No valid traffic sign”
8	Users click on the export button	Export individual segmented traffic sign(s) with classification label as name to phone gallery.

Table 8.4.2 Video-Based Input Activity Test Cases

Id	Test Case	Expected Results
1	Current camera frame consists of a traffic sign	Track the detected traffic sign with white bounding box
2	Users click on the “Output” button with a red traffic sign in frame	Display the names and confidence percentage of the classification result.
3	Users click on the “Output” button with a yellow traffic sign in frame	Display the names and confidence percentage of the classification result.
4	Users click on the “Output” button with a blue traffic sign in frame	Display the names and confidence percentage of the classification result.
5	Users click on the “Output” button with no valid traffic sign in frame	Display “No valid traffic sign”
6	Users click on the “File Input” button	Redirect users to the image-based input activity

The test cases above will be tested in the testing and validation subsection in this chapter.

8.5 Implementation

[Done By: Tan Jing Jie & Jacynth Tham Ming Quan & Tan Wei Mun]

In this subsection, the full implementation of the still-image and real-time video-based classification process is described in detail from the moment the developed application receives the input to the point that the developed application outputs a voice message and shows the classification results to the user. An export function was built for user to save the segmented traffic sign with their classified label. The

implementation of this project follows the flow illustrated in Subsection 8.3 Overall Block Diagram

8.5.1 Project Setup (Android Studio)

[Done By: Tan Jing Jie & Jacynth Tham Ming Quan & Tan Wei Mun]

Before the integration of C++ OpenCV functions into Android Studio was explored, the project environment has to be set up. This step involved importing OpenCV libraries, OpenCV-compatible android software development kits (SDKs) and modifying the project's gradle properties. Moreover, the OpenCV dependencies had to be configured. After thorough research, it was discovered that there are several methods for Android Studio to be able to access the OpenCV functions, such as through Android Studio's native API or Java Wrapper APIs. In this mobile application, OpenCV Java wrappers are used to integrate the C++ codes into Android Studio because this method allows the team to work with a more familiar programming language. The Java wrappers uses the Java Native Interface (JNI) to channel functions calls to the native OpenCV libraries.

8.5.2 Application Initialization and Permissions

[Done By: Tan Jing Jie & Jacynth Tham Ming Quan & Tan Wei Mun]

Upon entering the mobile application for the first time, users are prompted to grant camera permissions for the mobile application. The granting of permissions is handled by an external library named RxPermissions. This library is included into the dependencies of the mobile application in order to simplify the permission granting process. The basic user interface of the mobile application consists of a full-screen camera view in landscape mode. Figure 4.6.2.1 below illustrates the mobile application's animated splash screen.



Figure 4.7.2.1 Mobile Application's Splash Screen

8.5.3 Input

[Done By: Jacynth Tham Ming Quan & Tan Jing Jie]

After bypassing the application's splash screen, users are presented with two input options – file-based image input and real-time video stream input. Figure 8.5.3.1 below shows the user interface of the two options.



Figure 8.5.3.1 Input Option User Interface

From the image above, if users click on the “From File” button, they will be redirected to the file-based image input activity, whose user interface is shown below in Figure 8.5.3.2.

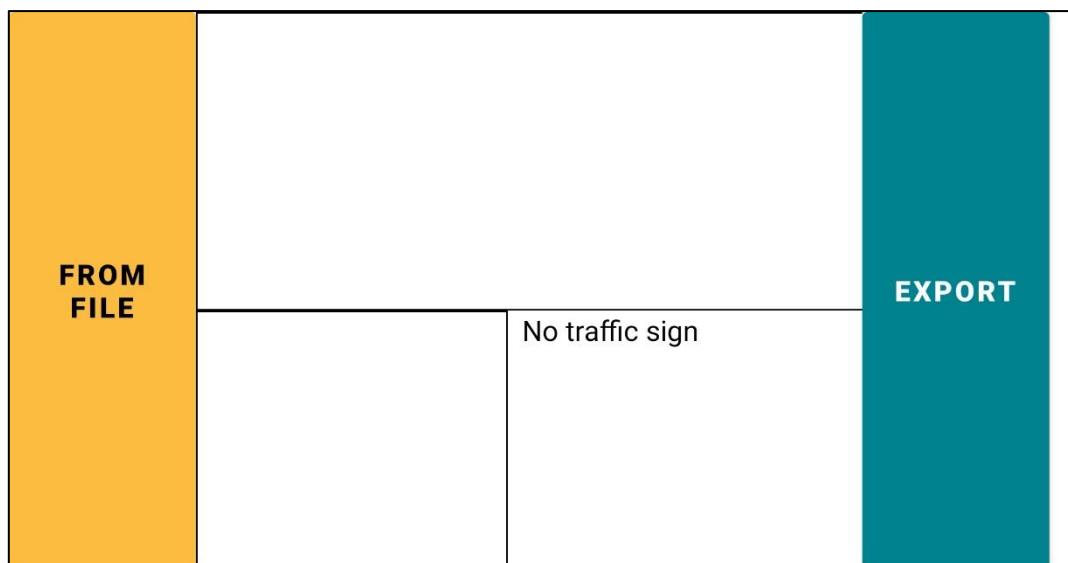


Figure 8.5.3.2 File-Based Image Input Activity

In the figure above, users are allowed to select image from their phone's gallery or file manager by clicking on the “From File” button. Furthermore, the “Export” button allows users to export the segmented traffic sign to their gallery. The outputs of this activity will be described in following subsections.

On the other hand, if users select the “From Camera” button from the options menu, then the users will be directed to the camera view as shown below in Figure 8.5.3.3.

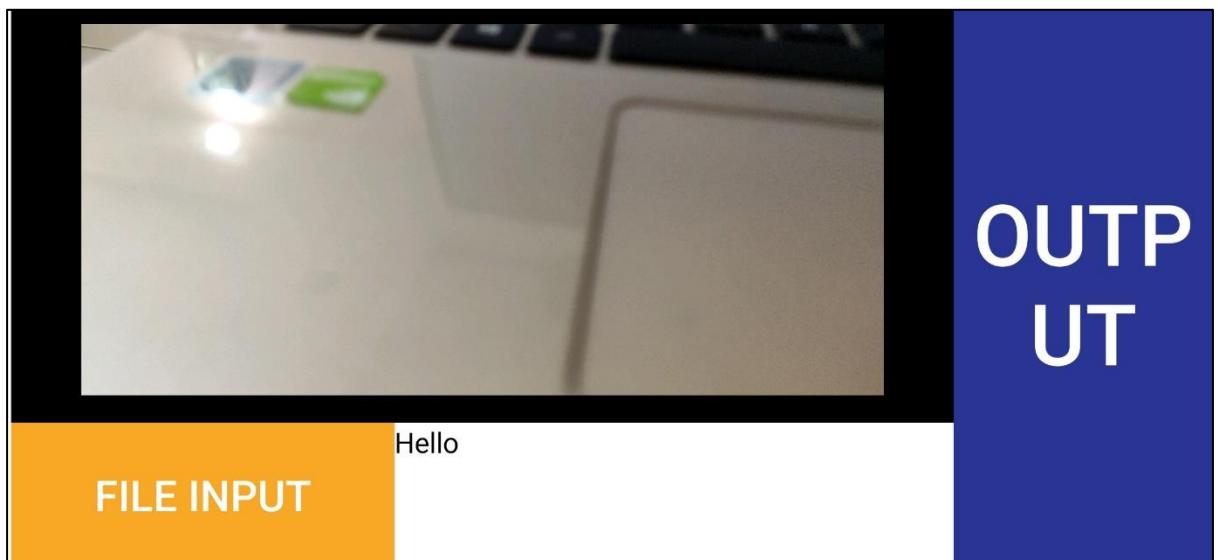


Figure 8.5.3.3 Real-Time Video Stream Input Activity

From the image above, users are able to navigate to the file-based image input activity by clicking on the “File Input” button. If users wish to view the recognition results of a particular traffic sign, they can click on the “Output” button. The outputs of this activity will be explained in the upcoming subsections.

8.5.4 Image preprocessing – Segmentation of traffic sign

[Done By: Tan Jing Jie & Jacynth Tham Ming Quan & Tan Wei Mun]

Next, colour detection was explored in Android Studio. In this mobile application, the HSV ranges utilized are tabulated in Table 8.5.4.1 below:

Table 8.5.4.1 The HSV Range used

Color Range	Hue (H)	Saturation (S)	Value (V)
Red	150 – 180	140 – 255	160 – 255
	0 – 3	50-255	50-255
Yellow	14 – 30	100 – 255	140 – 255
Blue	100 – 128	150 – 255	100 – 255

The three-colour masks were obtained separately, all using the current video frame as the input image. Then, a bitwise OR function was used to merge all three of the colour masks. This was performed on every frame of the real-time video stream. Figure 4.6.3.1 below shows an example output of the binary colour mask:



Figure 4.6.3.1 Colour Mask Output in User Interface

The segmentation framework is implemented after the mask is obtained. As the use of OpenCV in Android Studio differs from that of Visual Studio, the mask needs to be turned into RGB colour space before output to preview layout.

The method to do the segmentation is similar to the approach in C++ as stated in Chapter 4. First, after obtaining the HSV colour mask, a `findContour()` function is used to plot out all the related contours. A sort of function is implemented to arrange according to each contour size. Moving on, the largest contour will be selected to build the bounding box. The bounding box will be merged into the source image and return the frame for displaying in preview layout.

8.5.5 Classification of Traffic Signs

[Done By: Tan Jing Jie & Jacynth Tham Ming Quan]

Figure 8.5.5.1 below shows the flowchart of the classification process. The step starts from input model, get the input image in Mat type, and convert to model-acceptable data type. After the interpreter runs completely, the result obtained will be processed and sorted according to the confidence level. Lastly, the result will save to a list of classification result objects.

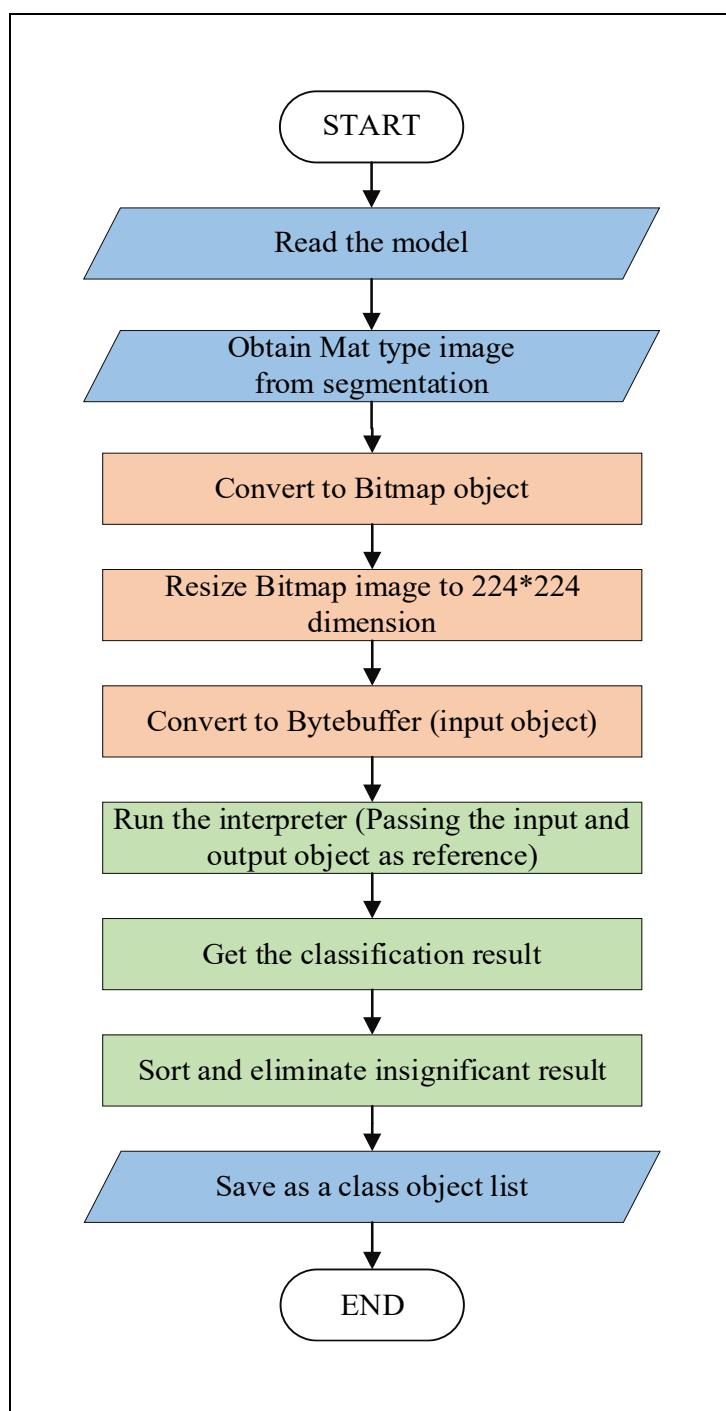


Figure 8.5.5.1 Classification Process Flowchart

In order to run the classification program in Java, a class needs to prepare to ease the process of output. The class is called “Classification” which contains a string and float value – the label and its confidence level.

Next, it is needed to ensure the TensorFlow Lite library is configured in gradle. An Interpreter object is created and the model which is located in the asset is read and saved into it. Moving on, two objects – input and output, need to be created. In this mobile net classification model, the input is an image, so a ByteBuffer object is prepared for input. Next, the output is the confidence level to each of the traffic sign labels, so a float array object is prepared to store the output.

Having configured the project for the classification, the first step is to load the MobileNet model into the developed application. Once the model has been loaded and verified to ensure that it is not corrupted, the classification process begins. Upon each input image received or each camera frame sent for output, the developed application obtains a Mat-typed image from the final segmentation process. This Mat object is then converted to a Bitmap object in order to work with the MobileNet model. Next, the Bitmap object is resized to 224 * 224 pixels as this was the specified input size of the MobileNet model. After resizing, the Bitmap object is converted to a ByteBuffer object, which is then fed into the Interpreter by passing the input and output objects as references.

After invoking the run of the interpreter method, then the classification is started. After the classification is complete, we will get the confidence level through the output object. Continuously, the output will be sorted according to high to low. This is because a high confidence level traffic sign label indicates a higher chance of the correct label. If there is a case that the confidence level does not reach 10% the label will be eliminated. If there is a case there have no traffic sign reach 10% confidence level, then the output will show invalid traffic sign. By using the class that was created earlier, the output is stored into the class object (The label with its confidence level). In the end, a list of valid results can be obtained. Lastly, the valid classification results are saved into a class object list, which is passed back as reference for the output process.

8.5.6 Output (Image-Based)

[Done By: Tan Jing Jie & Jacynth Tham Ming Quan]

In the file input option, the output will be triggered automatically when the user selects an image from the gallery. By using the segmentation and classification techniques mentioned in subsections 8.5.4 and 8.5.5, the sorted classification result is obtained. The output shows the figure below. As shown in figure below, the system displays few possible labels when the confidence level is passed the threshold.

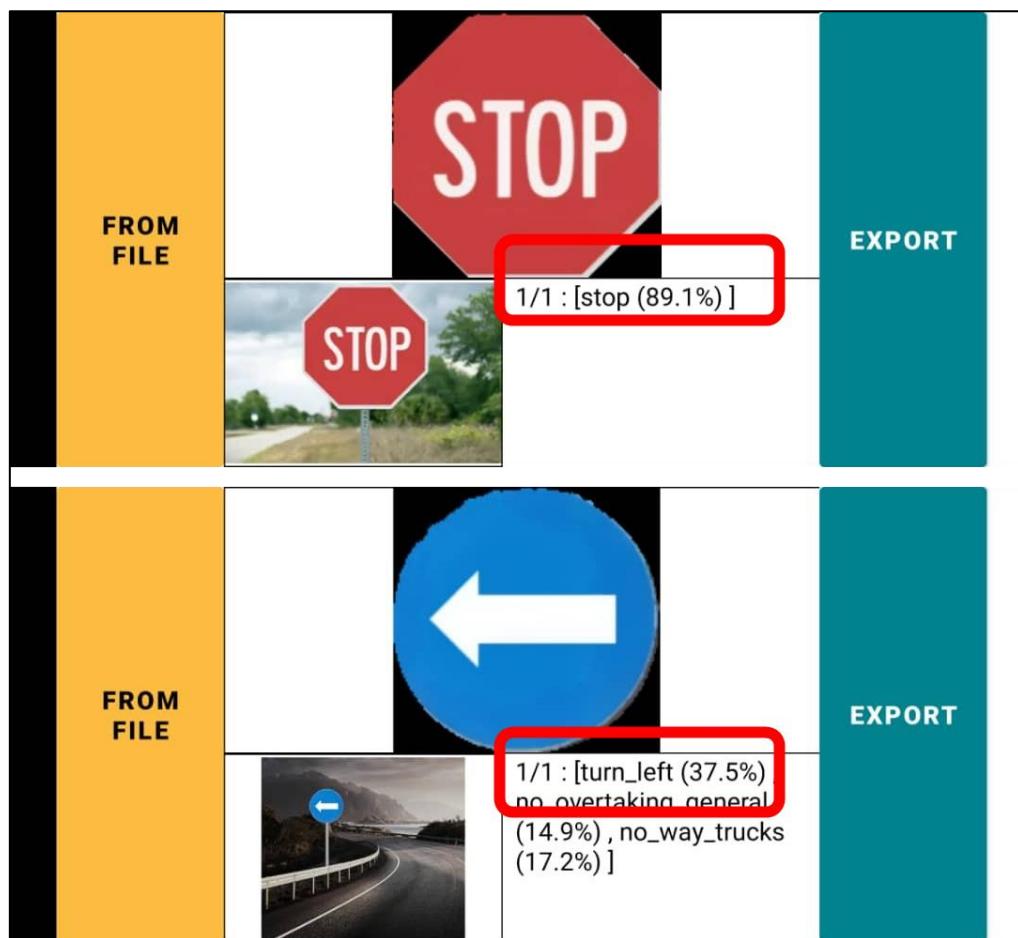


Figure 8.5.6.1 The output after classification by using single traffic sign

On the other hand, this android traffic sign recognition system is able to differentiate out the traffic sign from a multiple traffic signs input image. As shown in the example below, two traffic signs are segmented out. Users can swipe the pager to left or right to see the traffic sign classify result.

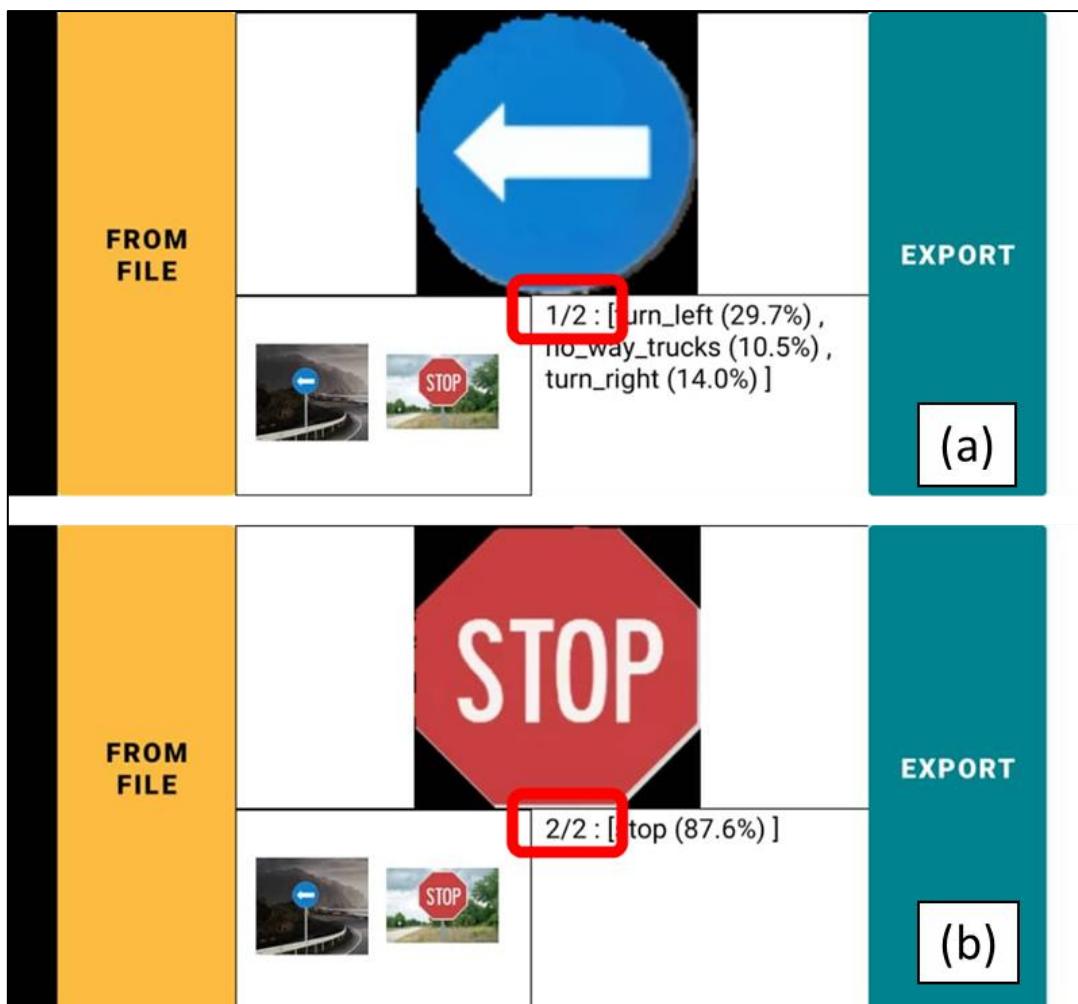


Figure 8.5.6.2 The output after classification by using single traffic sign

This application uses the android Text to Speech library. When the user clicks the text output area, the output will be spoken out. By using the example in Figure 8.5.5.2 (a), although the text box has shown a lot of possible labels, the output will only read out the highest confidence level label, that is, “One, Turn left”. Same goes to Figure 8.5.5.2 (b), the output is, “Two, Stop”.

On the other hand, if no traffic signs were detected in the image, then the mobile application shows the output as seen below in Figure 8.5.6.3.

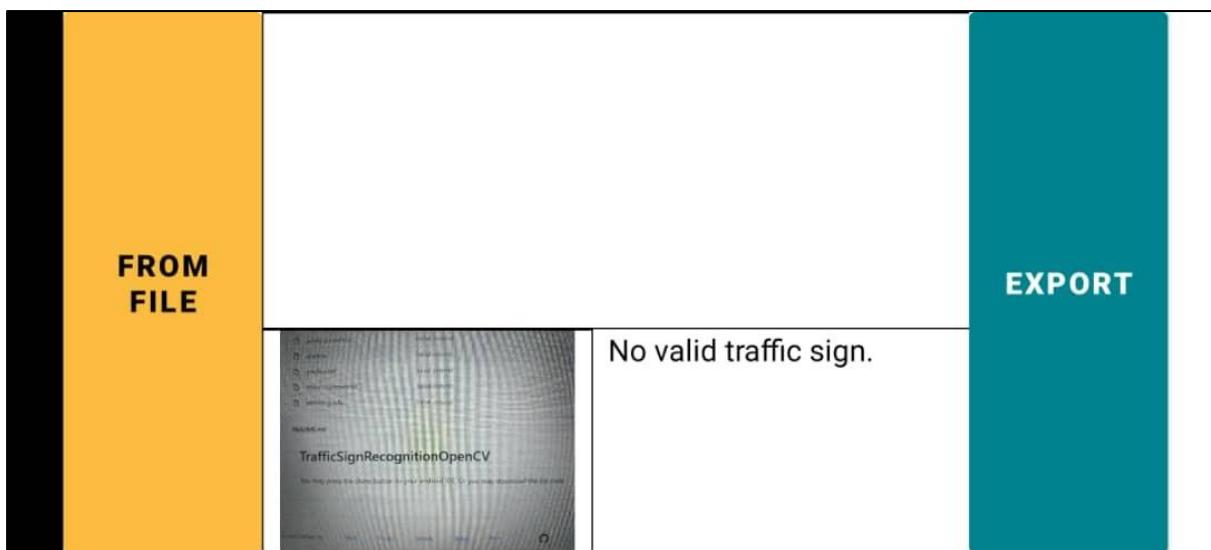


Figure 8.5.6.3 No Traffic Sign Detected Output in File-Based Image Input Activity

Last but not least, an export function is provided to save the segmented traffic signs and it classifies the result. A folder will be made with the concatenation of “openCV_” and the current date time value, for example, “openCV_1630598220980”. The content in the folder is the segmented image. The image name is the classify result such as “stop.png”. The example of directory is shown below.

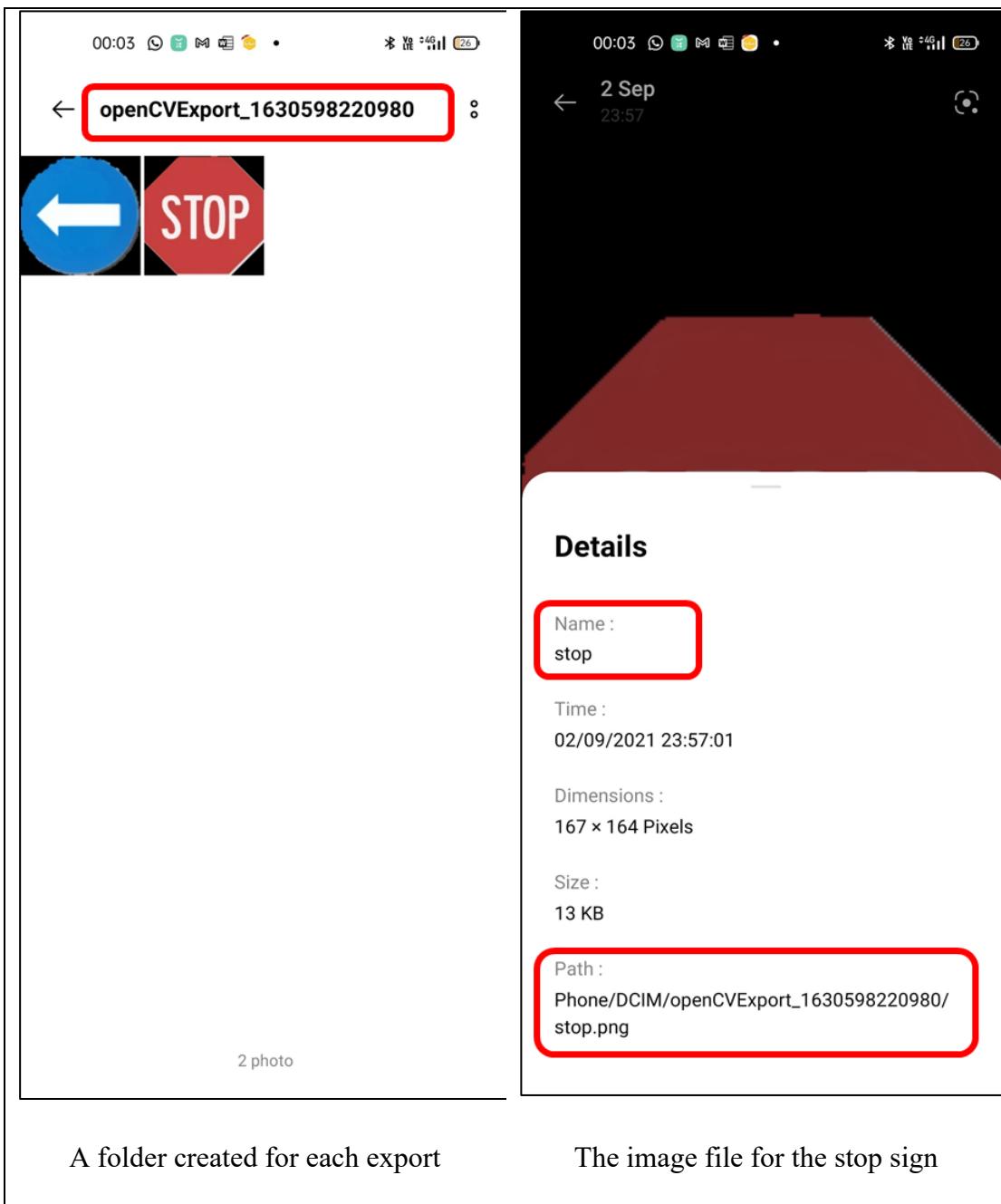


Figure 8.5.6.3 The output after classification by using single traffic sign

8.5.7 Output (Video-Based)

[Done By: Jacynth Tham Ming Quan & Tan Jing Jie]

Using the exact same segmentation and classification techniques mentioned in subsections 8.5.4 and 8.5.5, the outputs of the real-time video-based traffic sign recognition activity is shown below in Figure 8.5.7.1.

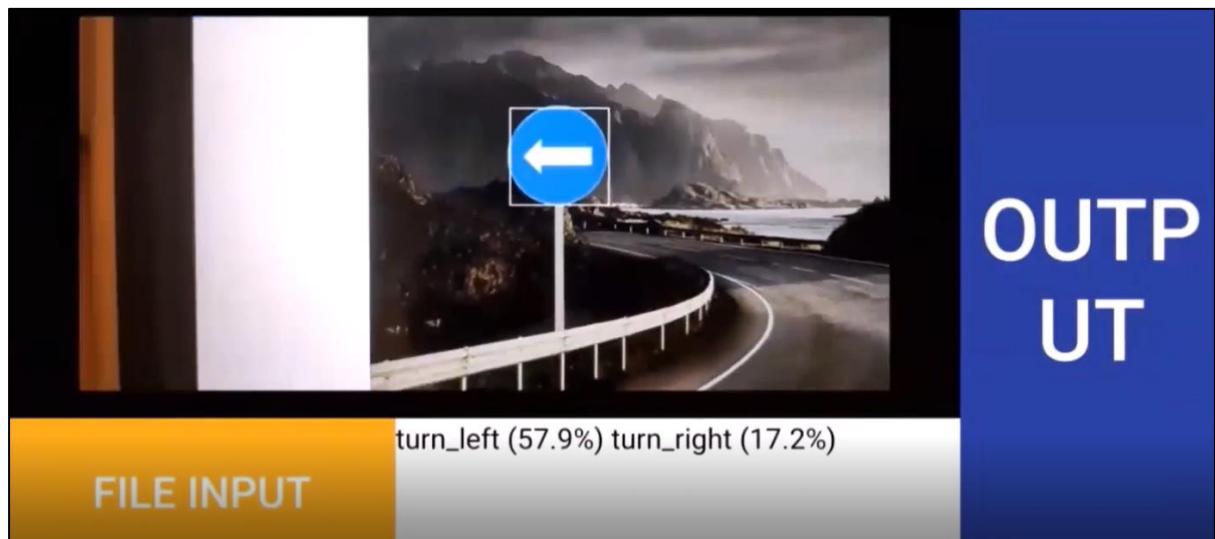


Figure 8.5.7.1 Output of Real-Time Video-Based Input Activity

As seen in the figure above, the detected traffic sign in the camera frame was surrounded by a white bounding box. The detections of traffic signs are the results of the HSV colour space segmentation performed on every camera frame. Behind the scenes, this bounded traffic sign is segmented from the full image and fed into the MobileNet model for classification. The model returns a list of possible classifications, which is then sorted from the highest confidence percentage to the lowest confidence percentage.

In the output text box next to the “File Input” button, the top three classifications are shown with the confidence percentages next to each classification result. In the figure above, the “turn_left” class had the highest confidence percentage, which evidenced that the developed traffic sign recognizer was able to correctly classify traffic signs from video-streams.

Moreover, upon obtaining the classification results, the developed application is capable of reading the results out loud due to the implementation of voice output. In

real-life scenarios, the developed application is able to warn drivers and smart car users when approaching a traffic sign.

Figure 8.5.7.2 below shows another classified output of a red traffic sign by the developed application.

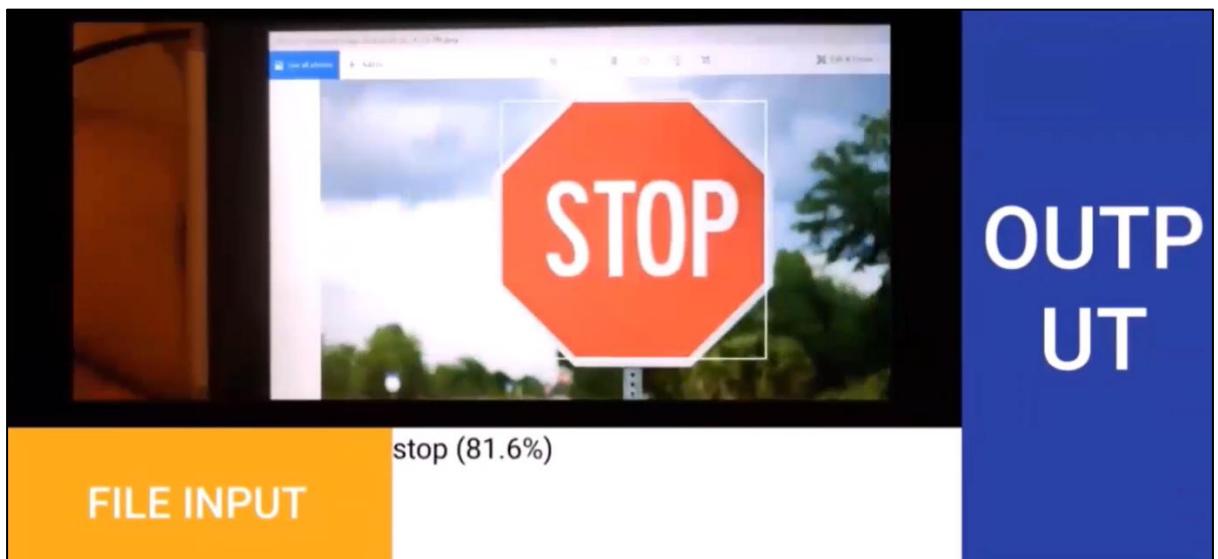


Figure 8.5.7.2 Second Output of Real-Time Video-Based Input Activity

On the other hand, if no traffic signs were recognized in the current camera frame, then the mobile application shows the output as seen below in Figure 8.5.7.3.

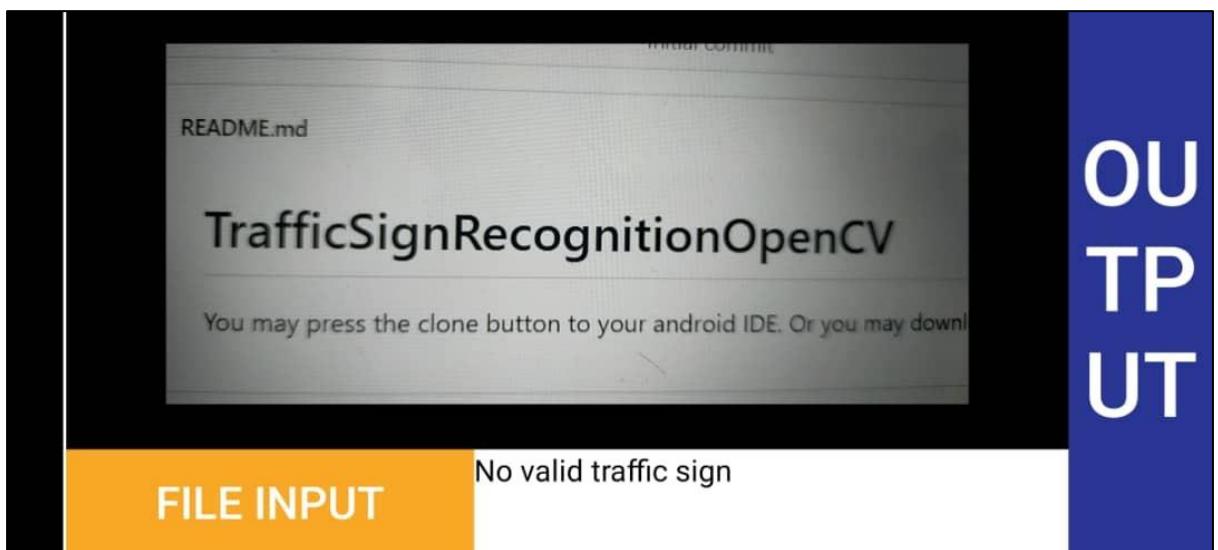


Figure 8.5.7.3 No Traffic Sign Detected Output in Video-Based Input Activity

In the upcoming subsection, various test cases were performed on the developed application to ensure that the system's performance and accuracy works as intended.

8.6 Testing

[Done By: Tan Jing Jie & Jacynth Tham Ming Quan & Ng Jan Hui]

Listed below are the developed application's results and outputs when tested on the test cases described in subsection 8.4.

Table 8.6.1 Test Case: Red Traffic Sign Input (Image-Based)

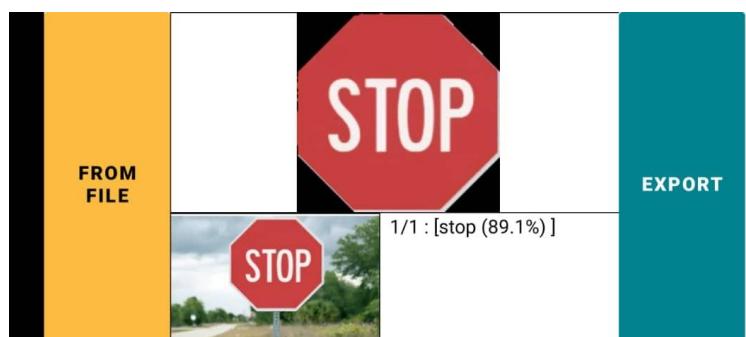
Test Case Name	Red Traffic Sign Input (Image-Based)				
Test Case Description	Users select an input image consisting of a red traffic sign				
Expected Output	Display the names and confidence percentage of the classification result.				
Input					
Results	 1/1 : [stop (89.1%)]				
Status (Pass/Fail)	Pass				

Table 8.6.2 Test Case: Blue Traffic Sign Input (Image-Based)

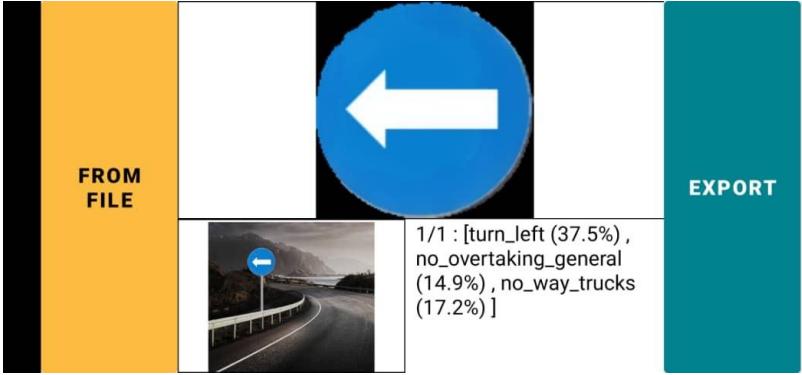
Test Case Name	Blue Traffic Sign Input (Image-Based)			
Test Case Description	Users select an input image consisting of a blue traffic sign			
Expected Output	Display the names and confidence percentage of the classification result.			
Input				
Results	 <p>1/1 : [turn_left (37.5%), no_overtaking_general (14.9%), no_way_trucks (17.2%)]</p>			
Status (Pass/Fail)	Pass			

Table 8.6.3 Test Case: Yellow Traffic Sign Input (Image-Based)

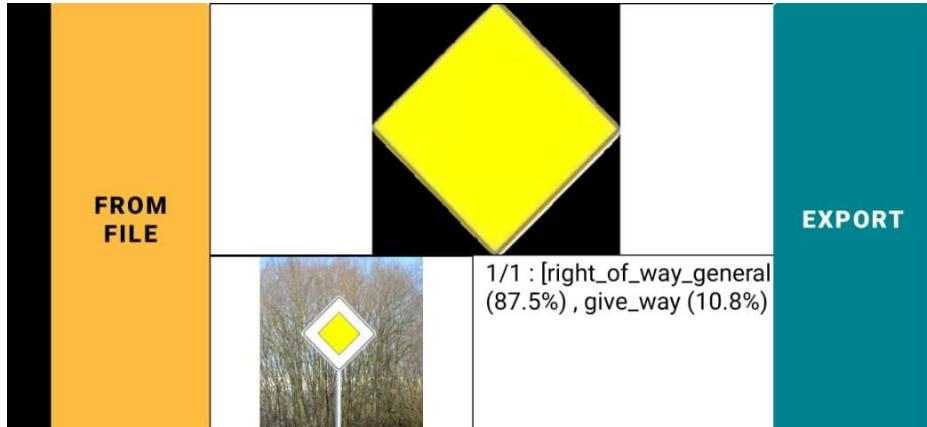
Test Case Name	Yellow Traffic Sign Input (Image-Based)				
Test Case Description	Users select an input image consisting of a yellow traffic sign				
Expected Output	Display the names and confidence percentage of the classification result.				
Input					
Results	 <p>1/1 : [right_of_way_general (87.5%) , give_way (10.8%)]</p>				
Status (Pass/Fail)	Pass				

Table 8.6.4 Test Case: Multiple Traffic Sign with Various Lighting Input (Image-Based)

Test Case Name	Multiple Traffic Sign with Various Lighting Input (Image-Based)											
Test Case Description	Users select multiple input images containing traffic signs of different lighting condition											
Expected Output	Display the names and confidence percentage of the classification result in view pager.											
Input												
Results	<table border="1" style="width: 100%; height: 100%;"> <tr> <td style="background-color: yellow; width: 25%; vertical-align: top;"> FROM FILE </td> <td style="width: 25%; text-align: center;">   </td> <td style="width: 25%; text-align: center;"> <div style="background-color: black; color: white; padding: 5px; border-radius: 50%; width: 100px; height: 100px; display: flex; align-items: center; justify-content: center;">  </div> 1/2 : [turn_left (18.3%) , turn_straight (16.6%) , no_overtaking_general (13.5%) , right_of_way_crossing (11.9%)] </td> <td style="background-color: teal; width: 25%; vertical-align: top;"> EXPORT </td> </tr> </table> <table border="1" style="width: 100%; height: 100%;"> <tr> <td style="background-color: yellow; width: 25%; vertical-align: top;"> FROM FILE </td> <td style="width: 25%; text-align: center;">   </td> <td style="width: 25%; text-align: center;"> <div style="background-color: black; color: white; padding: 5px; border-radius: 50%; width: 100px; height: 100px; display: flex; align-items: center; justify-content: center;">  </div> 2/2 : [turn_left (20.2%) , no_overtaking_general (12.2%) , no_way_trucks (14.5%)] </td> <td style="background-color: teal; width: 25%; vertical-align: top;"> EXPORT </td> </tr> </table>				FROM FILE	 	<div style="background-color: black; color: white; padding: 5px; border-radius: 50%; width: 100px; height: 100px; display: flex; align-items: center; justify-content: center;">  </div> 1/2 : [turn_left (18.3%) , turn_straight (16.6%) , no_overtaking_general (13.5%) , right_of_way_crossing (11.9%)]	EXPORT	FROM FILE	 	<div style="background-color: black; color: white; padding: 5px; border-radius: 50%; width: 100px; height: 100px; display: flex; align-items: center; justify-content: center;">  </div> 2/2 : [turn_left (20.2%) , no_overtaking_general (12.2%) , no_way_trucks (14.5%)]	EXPORT
FROM FILE	 	<div style="background-color: black; color: white; padding: 5px; border-radius: 50%; width: 100px; height: 100px; display: flex; align-items: center; justify-content: center;">  </div> 1/2 : [turn_left (18.3%) , turn_straight (16.6%) , no_overtaking_general (13.5%) , right_of_way_crossing (11.9%)]	EXPORT									
FROM FILE	 	<div style="background-color: black; color: white; padding: 5px; border-radius: 50%; width: 100px; height: 100px; display: flex; align-items: center; justify-content: center;">  </div> 2/2 : [turn_left (20.2%) , no_overtaking_general (12.2%) , no_way_trucks (14.5%)]	EXPORT									

Status (Pass/Fail)	Pass
-------------------------------	------

Table 8.6.5 Test Case: Multiple Input Images Containing Traffic Signs of Varying Color (Image-Based)

Test Case Name	Multiple Input Images Containing Traffic Signs of Varying Color (Image-Based)
Test Case Description	Users select multiple input images containing traffic signs of varying colours.
Expected Output	Display the names and confidence percentage of the classification result in view pager.
Input	 
Results	<p>1/2 : [turn_left (29.7%) , no_way_trucks (10.5%) , turn_right (14.0%)]</p>

Status (Pass/Fail)	Pass

Table 8.6.6 Test Case: Speak out traffic sign (Image-Based)

Test Case Name	Yellow Traffic Sign Input (Image-Based)
Test Case Description	Users press on the output textbox
Expected Output	Read the classification output out loud (voice output)
Input	
Results	The text to speech service speaks “One, right of way general”
Status (Pass/Fail)	Pass

Table 8.6.7 Test Case: No Valid Traffic Sign (Image-Based)

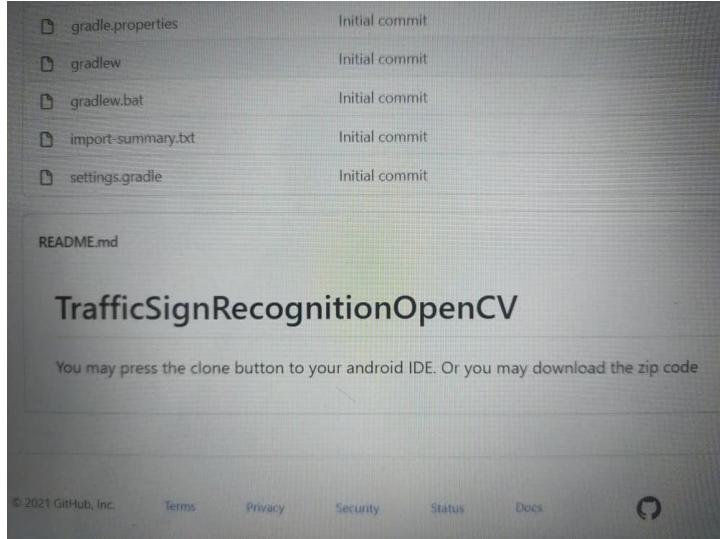
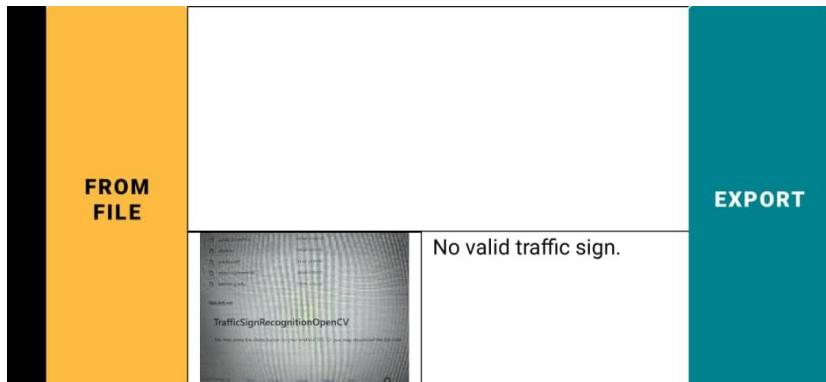
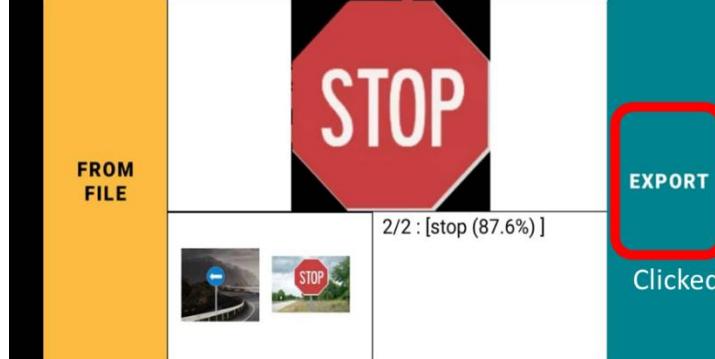
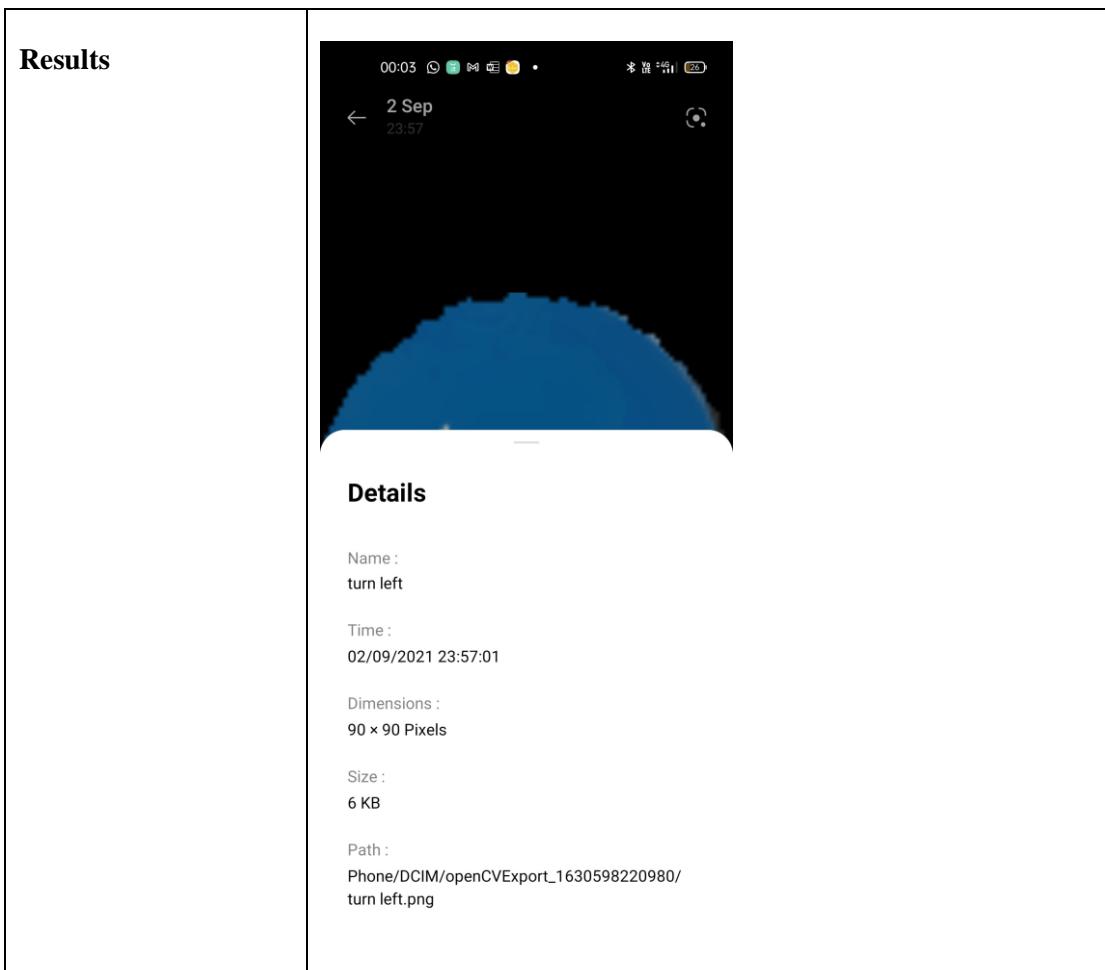
Test Case Name	No Valid Traffic Sign (Image-Based)
Test Case Description	User selects image with no traffic signs
Expected Output	Display “No valid traffic sign”
Input	
Results	
Status (Pass/Fail)	Pass

Table 8.6.8 Test Case: Export Each Segmented Traffic Sign (Image-Based)

Test Case Name	Export each segmented traffic sign (image-based)
Test Case Description	Users click on the export button.
Expected Output	Export individual segmented traffic sign with classification label as name to phone gallery.
Input	



	<p>Details</p> <p>Name : stop</p> <p>Time : 02/09/2021 23:57:01</p> <p>Dimensions : 167 x 164 Pixels</p> <p>Size : 13 KB</p> <p>Path : Phone/DCIM/openCVExport_1630598220980/ stop.png</p>
Status (Pass/Fail)	Pass

Table 8.6.9 Test Case: Red Traffic Sign Input (Video-Based)

Test Case Name	Red Traffic Sign Input (Video-Based)
Test Case Description	Users click on the “Output” button with a red traffic sign in frame
Expected Output	Display the names and confidence percentage of the classification result.
Input	

Results	 <p>The screenshot shows a mobile application interface. At the top, there is a camera viewfinder showing a red octagonal 'STOP' sign against a blue sky with clouds. Below the camera view is a white rectangular area containing the text 'stop (81.6%)'. To the left of this area is a yellow button labeled 'FILE INPUT'. To the right is a blue vertical bar with the text 'OUTP UT' in white. The overall interface is dark-themed.</p>
Status (Pass/Fail)	Pass

Table 8.6.10 Test Case: Blue Traffic Sign Input (Video-Based)

Test Case Name	Blue Traffic Sign Input (Video-Based)
Test Case Description	Users click on the “Output” button with a blue traffic sign in frame
Expected Output	Display the names and confidence percentage of the classification result.
Input	
Results	
Status (Pass/Fail)	Pass

Table 8.6.11 Test Case: Yellow Traffic Sign Input (Video-Based)

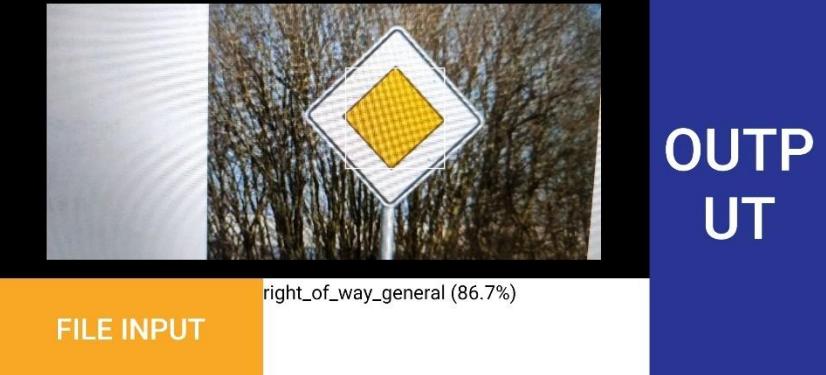
Test Case Name	Yellow Traffic Sign Input (Video-Based)
Test Case Description	Users click on the “Output” button with a yellow traffic sign in frame
Expected Output	Display the names and confidence percentage of the classification result.
Input	
Results	 <p>right_of_way_general (86.7%)</p> <p>FILE INPUT</p> <p>OUTPUT</p>
Status (Pass/Fail)	Pass

Table 8.6.12 Test Case: Traffic Sign Tracking with Bounding Box (Video-Based)

Test Case Name	Traffic Sign Tracking with Bounding Box (Video-Based)
Test Case Description	Current camera frame consists of a traffic sign
Expected Output	Track the detected traffic sign with white bounding box
Input	Current camera frame consists of a traffic sign
Results	
Status (Pass/Fail)	Pass

Table 8.6.13 Test Case: No Valid Traffic Sign (Image-Based)

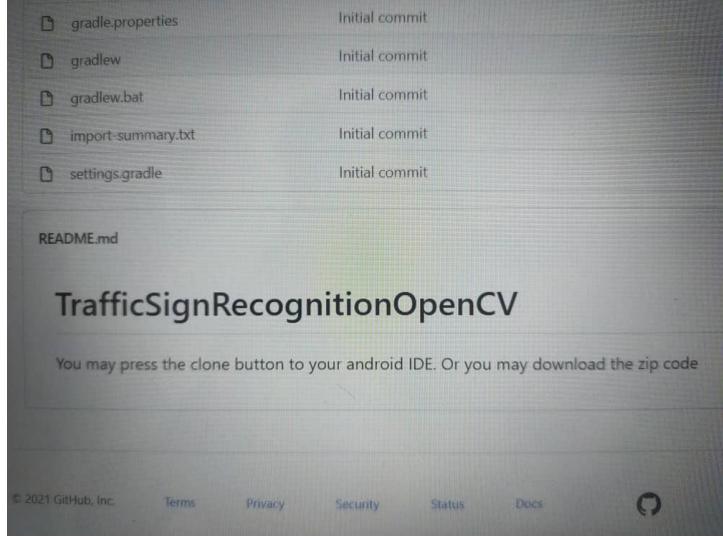
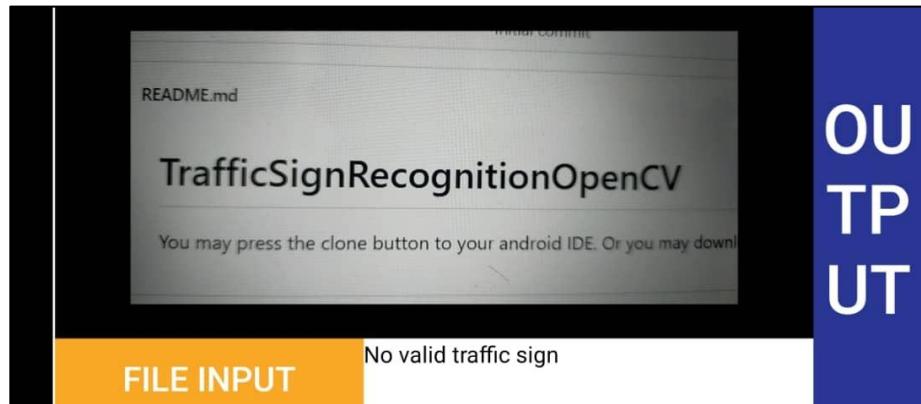
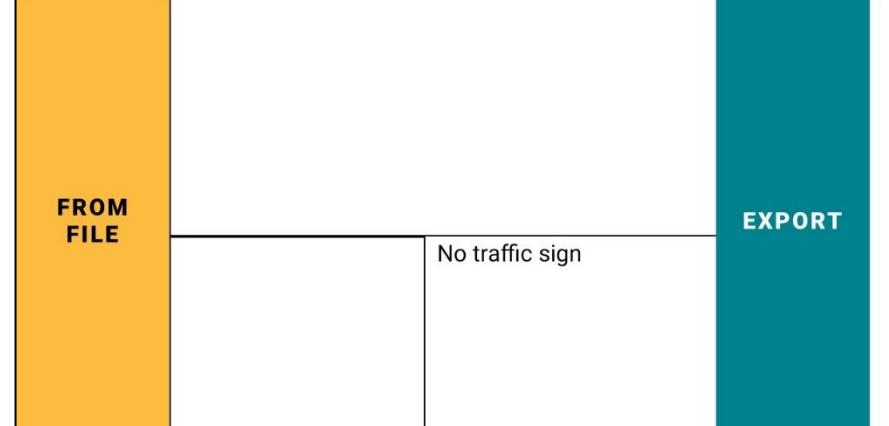
Test Case Name	No Valid Traffic Sign (Image-Based)
Test Case Description	User selects image with no traffic signs
Expected Output	Display “No valid traffic sign”
Input	
Results	
Status (Pass/Fail)	Pass

Table 8.6.14 Test Case: Button Redirection to File Input Activity (Video-Based)

Test Case Name	Button Redirection to File Input Activity (Video-Based)
Test Case Description	Users click on the “File Input” button
Expected Output	Redirect users to the image-based input activity
Input	 <p>turn_left (57.9%) turn_right (17.2%)</p>
Results	 <p>No traffic sign</p> <p>Redirected.</p>
Status (Pass/Fail)	Pass

8.7 Summary

[Done By: Jacynth Tham Ming Quan & Tan Jing Jie]

All in all, the developed mobile application was able to recognize traffic signs of various colours and various sizes through still-image inputs as well as real-time video stream inputs. Moreover, users were able to export the segmented traffic sign to their gallery from the image-based input activity and view the detected traffic sign bounded and tracked by a white bounding box in the video-based input activity. Furthermore, both the image-based and video-based input activities had voice outputs, which could help alert the user about upcoming traffic signs on the road. Having tested and validated the developed application, it was affirmed that the developed application is ready to be integrated into smart cars.

Chapter 9

Experimental Result

9.0 Overview

[Done by: Ng Jan Hui]

In this chapter, performance analysis of the trained models was done using a few classification metrics. The metrics that will be computed are the accuracy score, confusion matrix, micro averaged F1 score, micro averaged precision, and micro averaged recall. The reason behind the usage of micro averaged scores is due to the nature of the classification task being a multiclass problem. After obtaining the metrics, opinions and judgements on the model performance will be made. Figure 9.0.1 depicts the flow of this chapter.

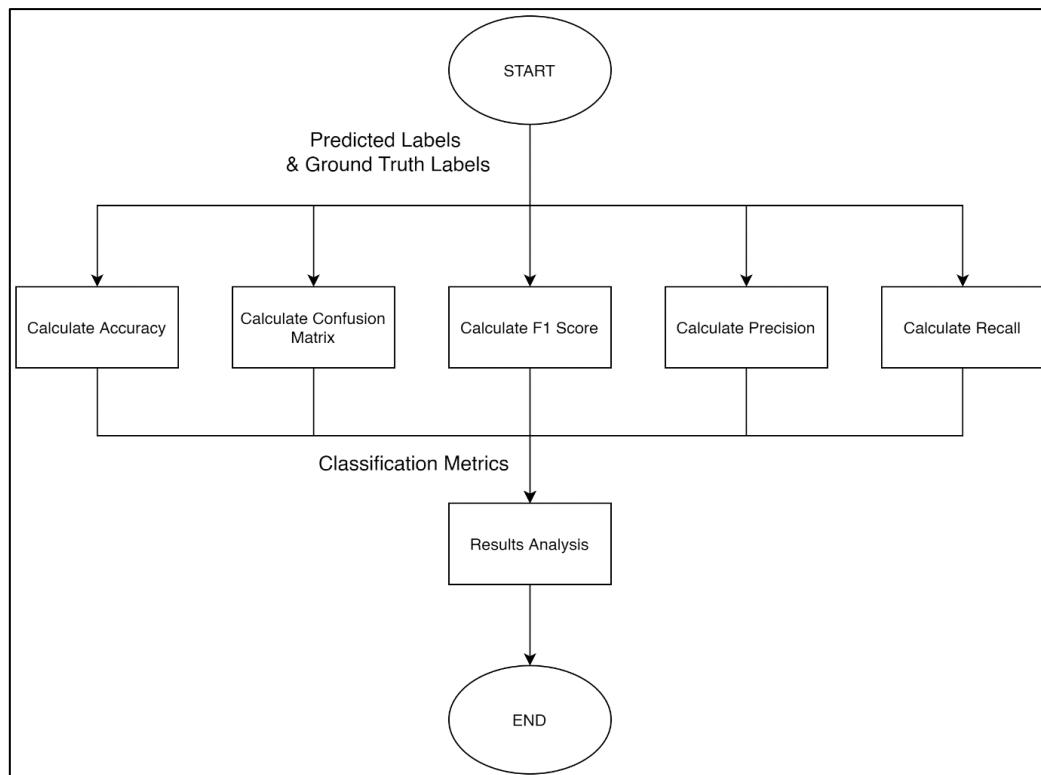


Figure 9.0.1 Block Diagram illustrating experimental results analysis flow

9.1 System Performance

[Done by: Ng Jan Hui]

9.1.1 Exporting predicted labels and ground truth labels

[Done by: Ng Jan Hui]

At the end of the model training phase, the testing set was used on the trained model to evaluate its performance on the left-out testing set. The labels predicted by both the SVM classifier and Random Forest classifier on the testing set was saved to two separate one-dimensional Mats respectively. Additionally, the ground truth labels of the traffic signs were also stored in a one-dimensional Mat.

Using the OpenCV provided formatter function, the three one-dimensional Mats were each written to a CSV file using fstream library. The reason behind exporting the labels was because C++ does not provide powerful data analysis and visualization libraries compared to libraries such as Scikit-learn and Matplotlib that are available in Python. Hence, all visualizations of the model predictions will be carried in a Python environment.

9.1.2 Computing classification metrics

[Done by: Ng Jan Hui]

$$\text{Microaverage Precision} = \frac{\sum TP}{\sum TP + \sum FP}$$

$$\text{Microaverage Recall} = \frac{\sum TP}{\sum TP + \sum FN}$$

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

In a Python notebook file, the three CSV files were first read into the code as NumPy arrays using the NumPy function `genfromtxt()`. Afterwards, the using the `accuracy_score` function provided by `sklearn.metrics` library, the accuracy scores for both the Random Forest and SVM classifiers were calculated. To verify that the accuracy score is genuine and is not affected by dataset imbalance or overfitting, the F1 score, precision and recall are then computed. Note that, based on the formula above, computing the micro average precision and recall is similarly to computing the accuracy as all class weights are considered. The computed metrics are tabulated in table 9.1.2.1 below.

Table 9.1.2.1 Classification Metrics for both classifiers

Metric	Random Forest	SVM	MobileNet
Accuracy	0.9712	0.9744	0.8125
F1 Score (Micro-Averaged)	0.9712	0.9744	0.8125
Precision (Micro-Averaged)	0.9712	0.9744	0.8125
Recall (Micro-Averaged)	0.9712	0.9744	0.8125

Looking at the metrics, it is notable that the SVM performs slightly better than the Random Forest, and the MobileNet classifier. The SVM's accuracy, F1 score, precision and recall are all higher than the Random Forest and the MobileNet classifier.

However, the performances of the SVM and Random Forest do not differ by a lot, as both model's scores are quite close. Both models had achieved scores of over 0.90 for the four metrics. This implies that very few traffic signs had been misclassified. Even though the dataset is skewed towards Class 2, 5 and 6 samples, the models are still able to predict the traffic signs correctly as indicated by the high F1 score.

As for the MobileNet classifier, it was trained on a different dataset compared to the other two classifiers. By comparison, the MobileNet was trained on a dataset consisting of 43 classes, each of which had blur, dark and weather-affected images while the SVM and Random Forest models were trained on a dataset consisting of 6 classes of traffic sign. As a result, the accuracy of the MobileNet classifier was slightly lower than the SVM and Random Forest classifiers due to the former's ability to classify a larger number of traffic sign classes.

9.1.3 Prediction performance visualization using Confusion Matrix

[Done by: Ng Jan Hui & Tan Jing Jie & Jacynth Tham Ming Quan]

To visualize and summarize the predictions made by the classifiers, the confusion matrix is then plotted. The confusion matrix is plotted using scikit-learn and matplotlib. Figure 9.1.3.1 shows the confusion matrix for the SVM classifier while Figure 9.1.3.2 shows the confusion matrix for the Random Forest classifier. Figure 9.1.3.3 shows the confusion matrix for MobileNet CNN classifier.

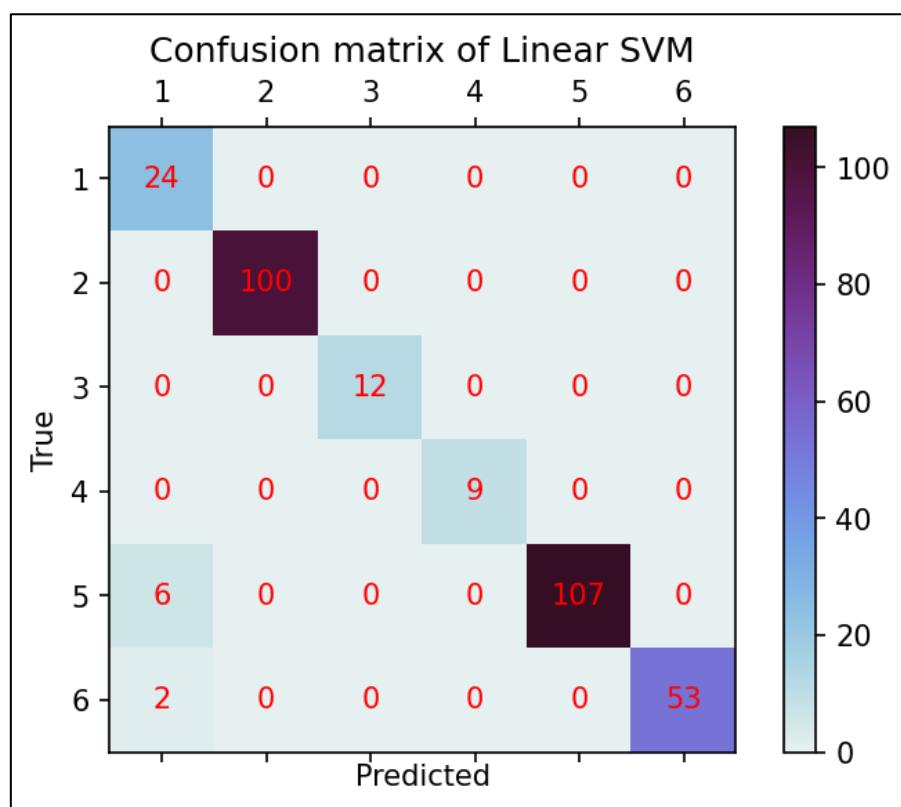


Figure 9.1.3.1 SVM Confusion Matrix

Looking at the predictions made by the SVM classifier, there were eight misclassifications occurring. The SVM had falsely predicted six No entry (Class 5) signs and two No stopping (Class 6) signs as No horn (Class 1) signs. This is probably explained as No entry and No stopping signs are circular and have red as the primary color, which are like that of the No horn sign.

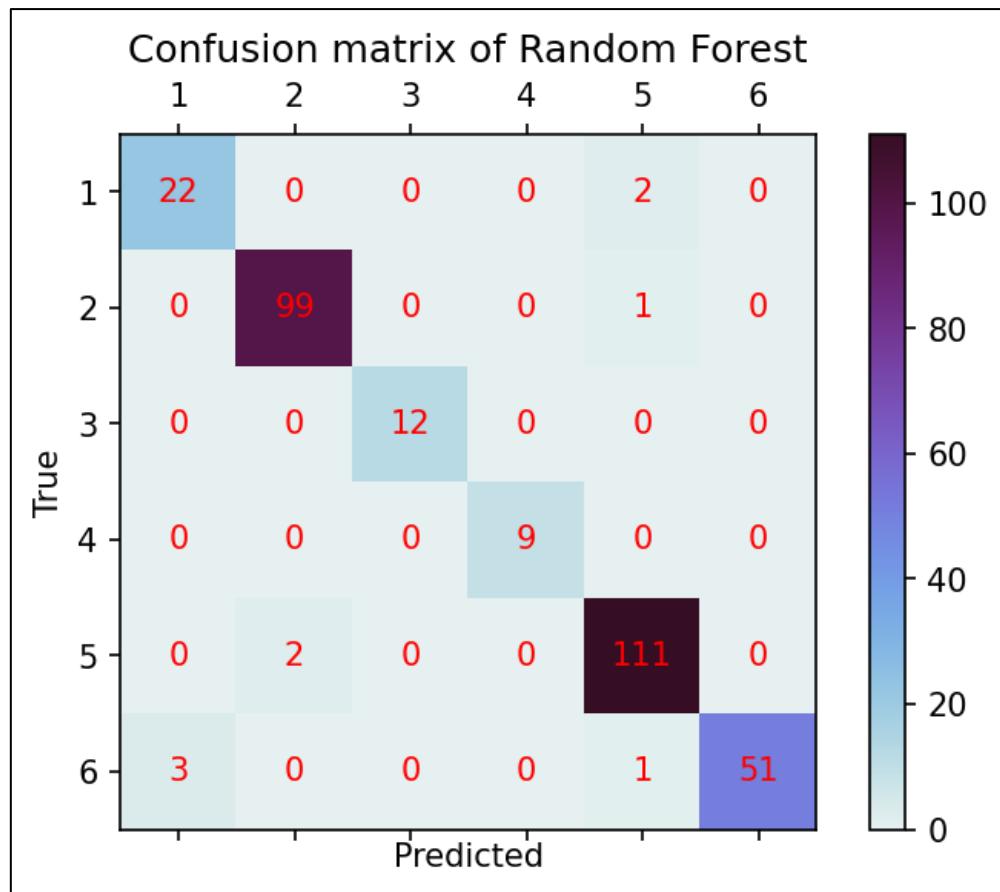


Figure 9.1.3.2 Random Forest Confusion Matrix

From the Random Forest Confusion Matrix, there were a total of 9 misclassifications that were distributed across classes 1, 2, 5 and 6. More particularly, three No entry (Class 6) signs were falsely predicted as No horn (Class 1) signs; two No stopping (Class 5) signs were falsely predicted as Car (Class 2) signs; two No horn (Class 1) signs were falsely predicted as No stopping signs (Class 5); one Car (Class 2) signs was falsely predicted as No stopping (Class 2) sign; and one No entry (Class 6) sign was predicted as No stopping signs (Class 5). Most misclassifications occurred on the No stopping (Class 5) sign.

The misclassification can be explained due to the overfitting nature of the Random Forest classifier. This causes the Random Forest to not be able to generalize well to unseen data, thus causing slightly more misclassifications.

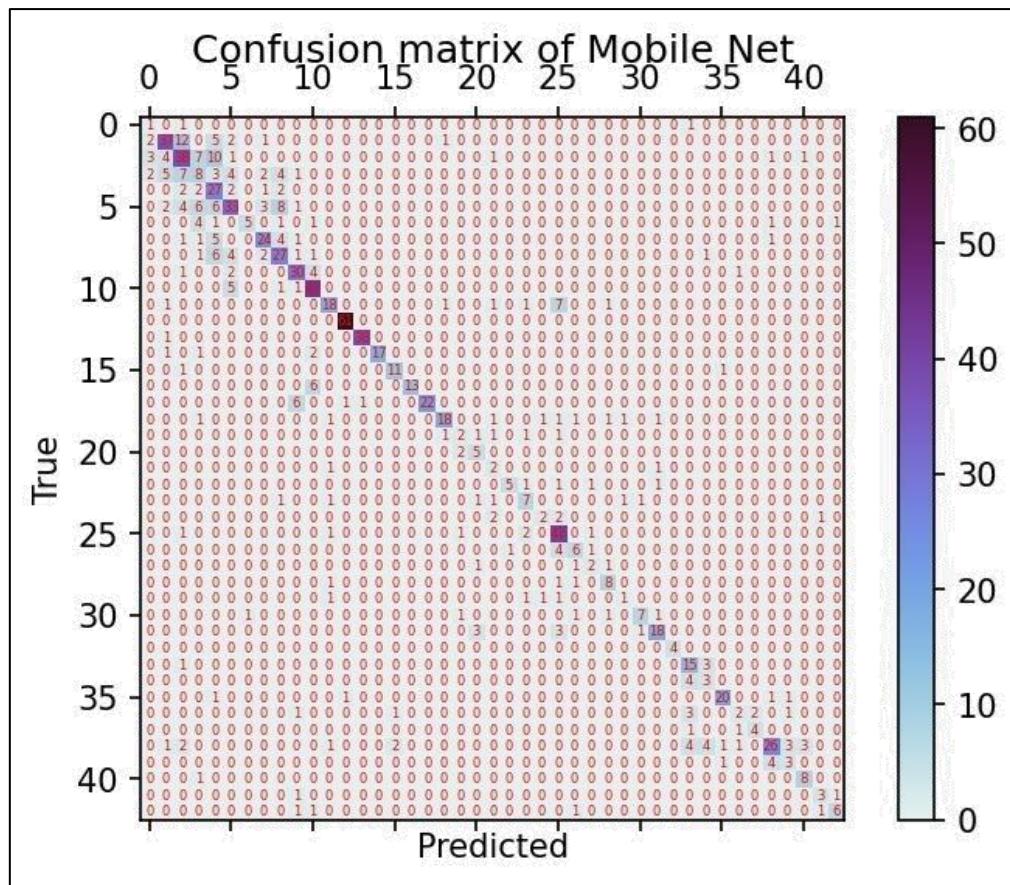


Figure 9.1.3.3 Mobile Net CNN Confusion Matrix

There are 1000 testing dataset used to build the Mobile Net Confusion Matrix. There are a total of 43 classes (0 to 42) of traffic signs to be analyzed. The label of each id is listed in [Appendix B.5](#). Overall, the prediction result is satisfied. The accuracy achieved is 81.25%. As shown in the color in the diagonal line, most of the class classified accurately.

The misclassification for several classes can be explained because there are insufficient samples in the testing set, some of the class only consist of very little sample of testing image (it can be shown for the traffic sign label 19-24). However, overall, the result from the confusion matrix can prove that the model reached satisfied accuracy.

9.2 Model Comparison

[Done by: Tan Wei Mun]

In the subsection, the three classification models that trained in the project will be compared with each other to determine which classifier achieved the highest performance. Besides, three different model will also be benchmarked with the most remarkable works in the literature review according to the classifier model.

9.2.1 Comparison between SVM, Random Forest and CNN

[Done by: Tan Wei Mun]

The calculated classification metrics are visualization into line graph as shown in figure 9.2.1.1.

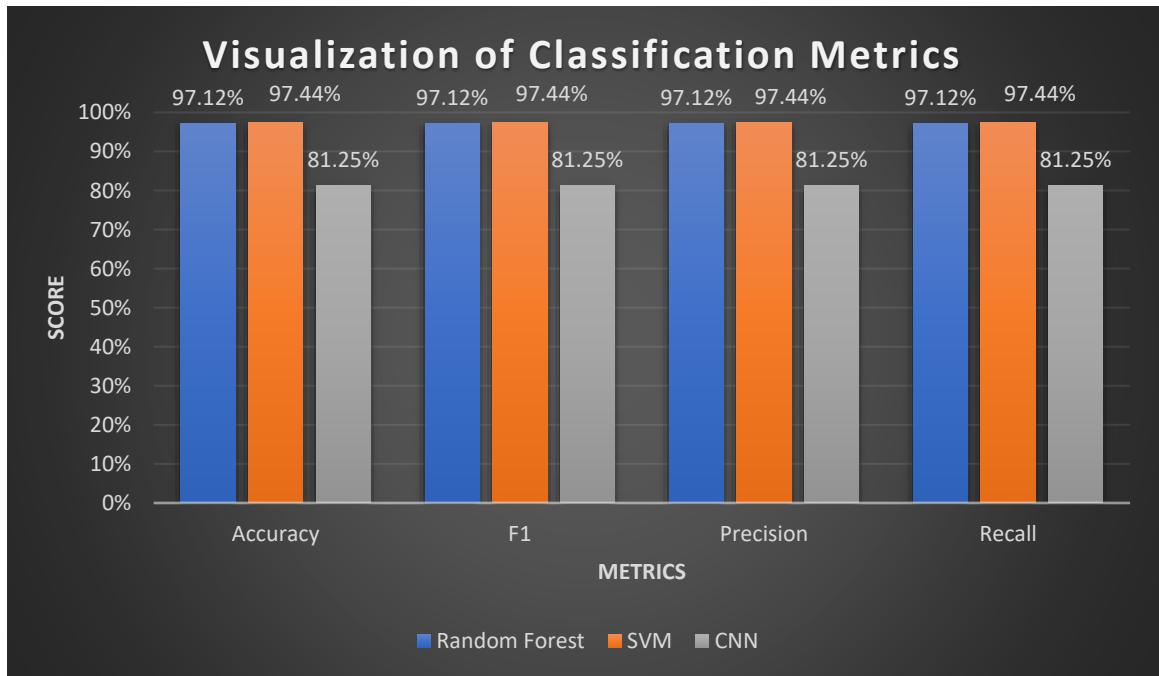


Figure 9.2.1.1 Visualization of Classification Metrics

The most accurate classifier in our project is the SVM model, which achieved 97.44% accuracy, followed by 97.12% accuracy of the Random Forest model, and lastly, the 81.25% of CNN model. Even though both the datasets used in training the models are recorded that significantly skewed and imbalanced, the situation for the GTSRB dataset used in CNN training is more serious because it contained more classes, which might be the reason for the extremely lower accuracy CNN model. However,

removing excessive data for skewing classes will make the dataset to be too small. Therefore, there is no other way but only trained the models with the skewed datasets.

As analyzing the classification metrics, the precision, recall and F1 scores for all classifiers are all found to be identical. To be explaining the situation, the formulae listed in the chapter 9.1.2. According to the formulae, the precision and recall could be the same when the total number of False Positive (FP) and False Negative (FN) are the same. While the precision and recall are the same, the calculated F1 score will also be the same as precision and recall as well.

Furthermore, the high precision and recall for both SVM and Random Forest classifiers reflects that they classified the target class quite correct in which there are very less number of other signs being classified as target class. This analysis was proven by high F1 score and the confusion matrix drawn in Figure 9.1.3.1 and Figure 9.1.3.2. On the other hand, the CNN mobileNet model has only achieved 81.25% of precision, recall and F1 score. It means that the CNN model is more likely misclassified a wrong sign into the target class compared another two models here. From the confusion matrix for CNN classifier, it is very evident that there are only few classes out from 43 classes having True Positive (TP) predictions which proven the interpretation above.

In short, the SVM classifier is the most accurate model that classified the most signs correctly while Random Forest classifier also an excellent model in which its accuracy is slightly lower than SVM only. In contrast, the CNN model might require retraining with more well processed dataset before putting into real life implementations.

9.2.2 Benchmarking with Previous Works

[Done by: Tan Wei Mun]

The three models will be compared against the three prior models from the literature review that obtained the best accuracy in this part. Our CNN (mobileNet) will be compared against Jin et al.'s CNN model, which was trained using the Hinge Loss SGD method and obtained an accuracy of 99.65%. (Jin, et al., 2014) NEXT, the SVM model will be compared to the SVM model developed by Yuan, et al., which used a fusion method and obtained 97.50% accuracy.(Yuan, et al., 2017) Finally, the Random

Forest classifier will be compared to Ellahyani, et al.'s work, which combined temporal data with Invariant Geometric moments and achieved an accuracy of 97.4%. (Ellahyani, et al., 2016)

The accuracy between these six classifiers was visualized as a bar chart, as shown in Figure 9.2.2.1.

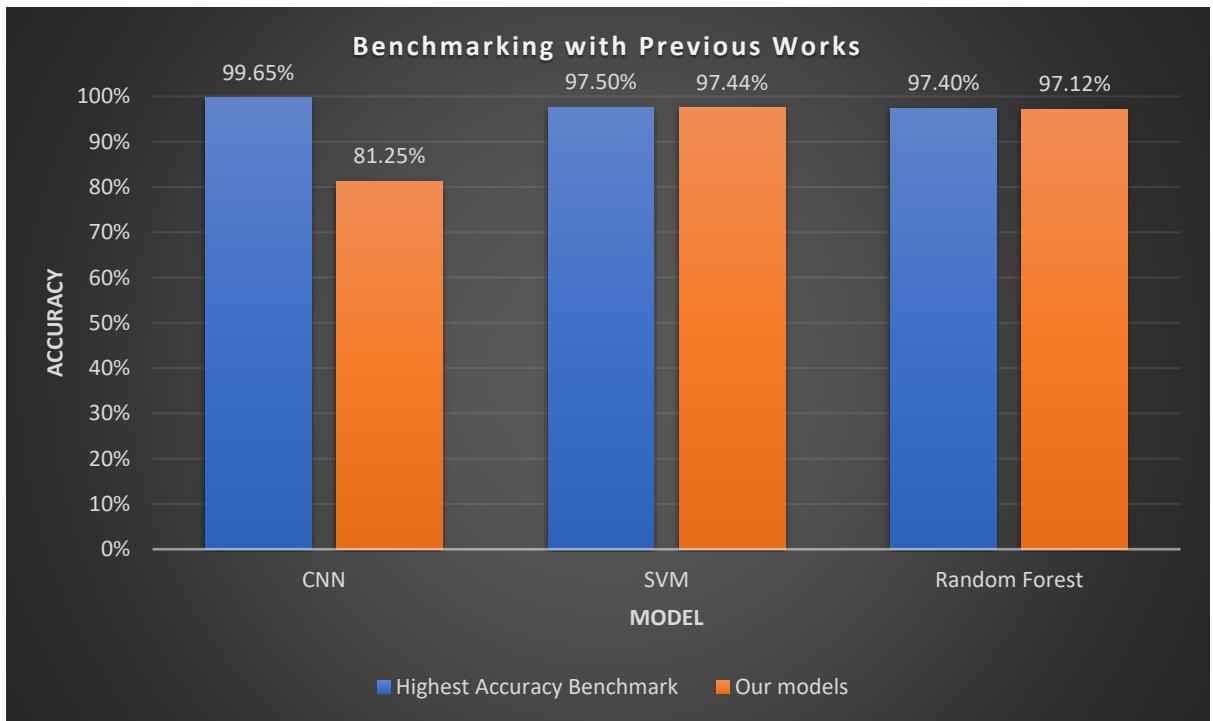


Figure 9.2.2.1 Benchmarking with Previous Works

Taking everything into account, no classifiers developed in this research outperform previous models when compared to previous studies. When comparing the models, it is clear that the prior models have all been improved, either through a different training technique or additional processed features. To exemplify, Jin, et al.'s CNN which is trained using the Hinge Loss SGD, which may take less time to train and provide a better model. Even though the training technique used for the models was sufficient to yield an acceptable accuracy, it is inspired by previous works that there are many more training strategies that may be investigated in the future to improve the models.

9.3 Error Analysis

[Done by: Por Teong Dean & Jacynth Tham Ming Quan]

During the error analysis, several issues were discovered with the developed desktop-based and mobile-based traffic sign recognizer. The three main issues discovered were the colour segmentation inaccuracies, vulnerabilities of the traffic sign recognizer on weather- and lighting-affected traffic sign images and lastly, the computational complexities.

Firstly, the HSV colour space segmentation process of the developed traffic sign recognizer faced issues when trying to segment traffic signs from similarly-coloured backgrounds, for example when there is a stop sign positioned in front of a red brick wall. This is due to the fact that the colour of traffic signs may deteriorate over time, therefore, it is impossible for the chosen HSV range to cover the colour variations of all traffic signs. Moreover, the colour segmentation was also vulnerable to external noise such as dark or blurred input images. As the segmentation process is crucial to determining the traffic sign class later on, errors in this stage resulted in the developed traffic sign recognizer being unable to classify traffic signs which blended into the background.

Secondly, the developer traffic sign recognizer was vulnerable to weather and lighting-affected input images. Some of the test images were taken during heavy storms or had reflections from the sun, which greatly affected the recognition accuracy of the developed traffic sign recognizer. Moreover, the lack of weather-affected traffic sign images in the training dataset also caused the developed traffic sign recognizer to be inaccurate when classify such images.

Last but not least, the computational complexities of the overall traffic sign recognizing system also led to video-based inputs on both desktops and mobile applications susceptible to lags. Sometimes, when dealing with a number of traffic signs at one go, the developed desktop-based and mobile-based applications may take up to 1 seconds per frame, which downgrades its capabilities at being implemented into a smart car's real-time safety system. This is due to the lengthy pre-processing and feature extraction process, which has to be repeated for each camera frame. Therefore, a trade-off is made here between recognition accuracy and processing time.

In short, the errors in the developed traffic sign recognizer mainly revolved around the recognition accuracies and real-time performance. Therefore, in the future,

Chapter 9 Experimental Result

several modern technologies and techniques might have to be implemented in order to conquer these system vulnerabilities.

9.4 Future Work

[Done by: Tan Jing Jie]

From the earlier error analysis subsection, the limitations of this project are concluded. Hence, 3 future works are proposed in order to improve this project from its insufficiencies. The future works are to implement neural network segmentation, to implement image augmentation to the database and to implement both tracking and fusion strategy for video stream input.

First and foremost, the reason to implement neural network segmentation instead of using the current color segmentation approach is because neural network segmentation is not easily affected by lighting issues. As shown in the figure below, the “forward” traffic sign located at the top-left corner can be segmented easily by using the neural network segmentation. Unlike using color segmentation, the traffic sign hardly segments out completely without noise due to the environment having a lot of similar colors such as trees and shadow on the road. As shown in the figure below, the neural network segmentation can directly segment out a lot of different objects at once. This can be trained by giving examples in training the model. This feature is very significant to real time as the model is trained to “know” where the objects should be located. For instance, if an area is segmented for roads, the same region will not be segmented for traffic signs again.

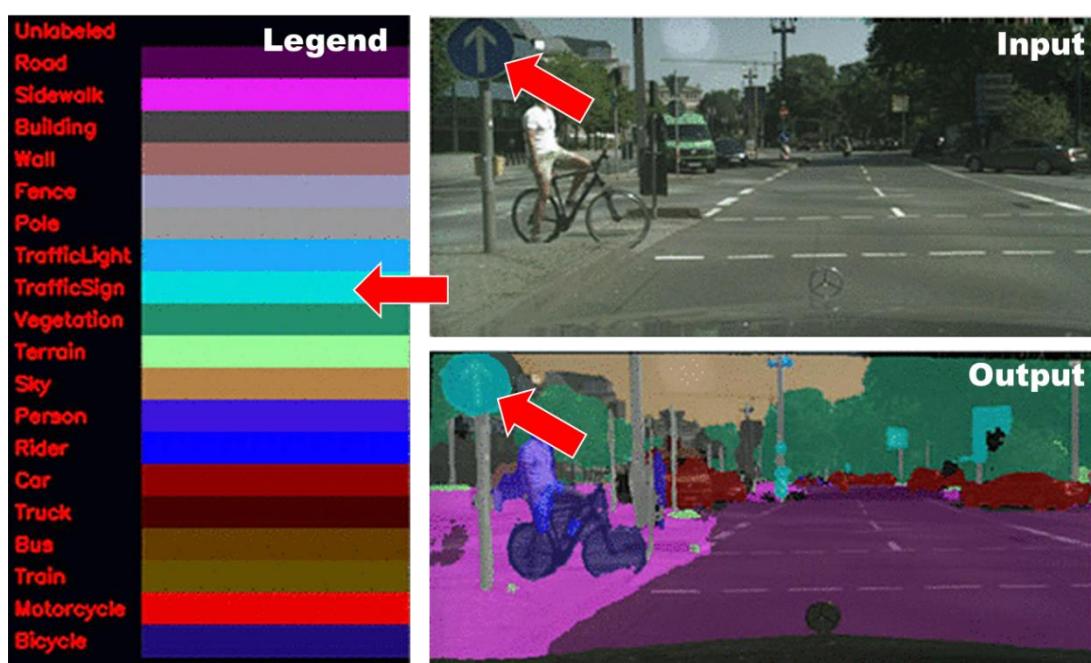


Figure 9.4.1 The neural network segmentation algorithm segment the objects

Second, the reason to implement image augmentation for the dataset is to enlarge the dataset size to resolve the insufficient training sample. In real time use, there are a lot of incidents that may occur especially the weather and the movement of cameras. Hence, image augmentation can be used to increase the training dataset in order to provide more samples for a model to learn. The image below shows how an original speed limit of 50 traffic signs underwent the image augmentation and generated 16 augmented traffic sign images. The augmentation method is able to generate the condition of rain, lighting issue and etcetera. Besides, a blur due to motion image is significant to train a model. This is because, in real time usage, this traffic sign recognition system is often used in automation vehicles. Hence, the motion blur will affect the real time camera input. Therefore, generating a motion blur image for training purposes is crucial. In short, image augmentation technique is able to overcome the insufficient training dataset as well as increase the classification accuracy.

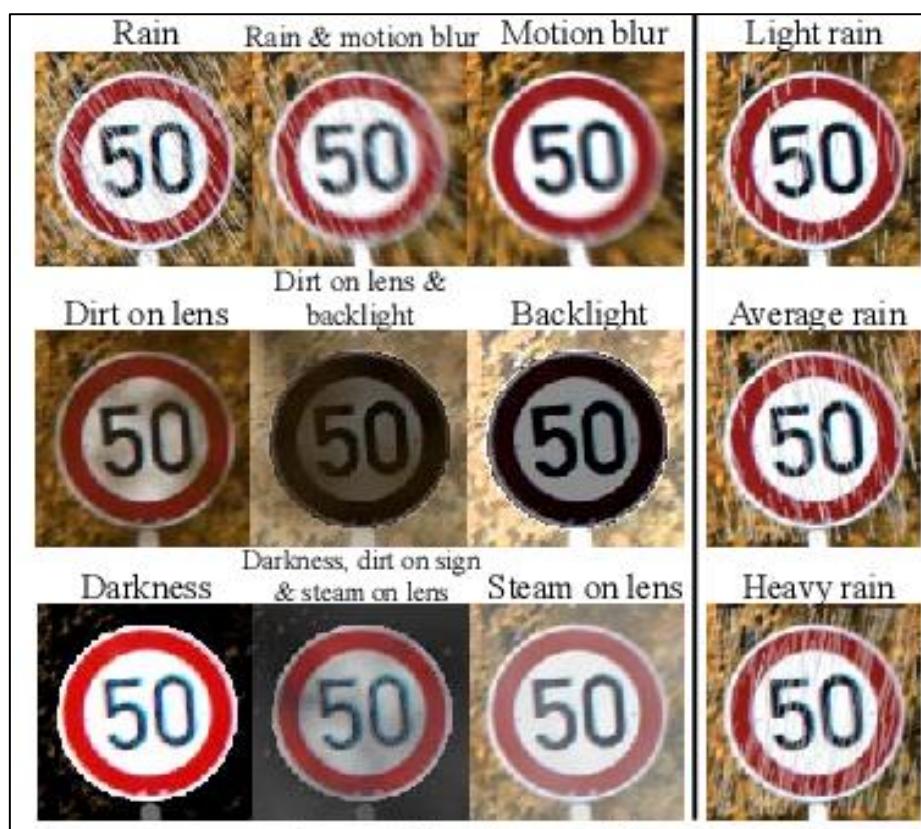


Figure 9.4.1 Image augmentation technique applied to a speed limit of 50 traffic sign.

Furthermore, the strategy of tracking and fusion approach needs to be implemented to enhance this system. The major reason is to resolve the limitation due to the algorithm complexity. As shown in figure below, the tracking process starts from

a normal first frame image. When the first frame image is obtained, the processes of image pre-processing, feature extraction and classification is run by normal. Moving on, after obtaining the coordinate in first frame segmentation, the coordinate will be enlarged with fixed scale or using prior knowledge – the x-axis will enlarge more due to the car always move to front. After the next frame is obtained, the coordinate will crop down the area, hence the dimension that to-be-analyzed is reduced, so it can reduce the system overhead by reducing the computational complexity. However, after several frames, the frame needs to fully analyze again to check for new incoming traffic signs.

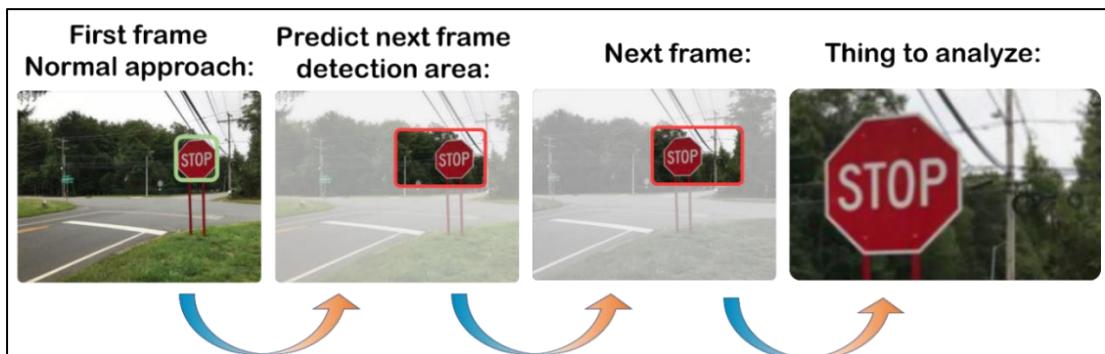


Figure 9.4.2 Flow of tracking approach

Furthermore, fusion strategy is very significant for video-based recognition, this approach can increase the reliability of the result especially for real-time recognition usage. For example, when the situation requires highly reliable results such as providing service to the automotive vehicle, it needs zero tolerance to outlier results. Hence, by fusion strategy, multiple frame results that belong to the same traffic sign can provide more reliable results. Furthermore, when the camera source is moving forward, the traffic sign image will be increased frame by frame. As larger scale traffic signs usually contain richer information and lead to more high accuracy, hence the future system can adopt a Gaussian-based weighting to each classified frame. By using the example shown in the figure below, the first frame traffic sign is misclassified as a “Stop” sign due to the distance from camera to traffic sign, but the other 3 frames are correct because they are closer to the camera. In this case, different weightages are applied to different frames according to the traffic sign size. Hence, the final result obtained is still a stop sign.

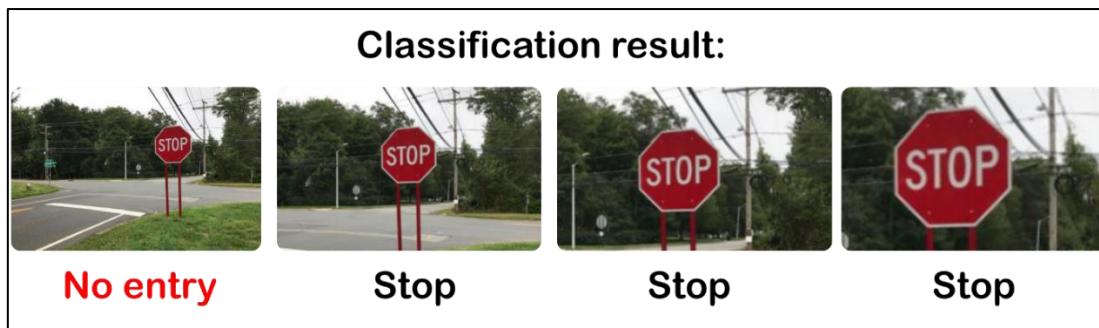


Figure 9.4.3 Fusion strategy approach

Thus, by using the tracking approach, the mathematical complexity can be reduced. Then by implementing the fusion strategy, the reliability of classification results can be improved. In other words, the accuracy of classification results can be improved without using extra complex algorithms or feature extraction. Hence, the processing time can be controlled as it is significant in real time. However, the tracking and fusion strategy need to cooperate to avoid carry-forward error. The fusion strategy can only fuse the result when the coordinate of the traffic sign is intercept. This is to ensure there is no mis-combining of the classified result when there are multiple traffic signs segmented in one frame.

All in all, after implementing the future work proposed above, this limitation of this system can be resolved. These future works can resolve the limitation of lighting issue, the insufficient dataset issue, and the time-consuming processing issue. Therefore, the scope of application of this system can be broader. After upgrading this system capability, it can provide a better result when this system is implemented into a self-driving system.

9.5 Contribution

[Done By: Jacynth Tham Ming Quan]

The contribution table below tabulates the contributions of each team member to the full project.

Table 9.5.1 Project Contribution Distribution

Contribution	Done By
Input Still Image (Desktop) Further Segmentation Hu Moments Highest Model Accuracy → 0.9744 (SVM)	Tan Jing Jie, Jacynth Tham Ming Quan, Ng Jan Hui, Tan Wei Mun, Por Teong Dean
Input Video (Desktop) File Input + Webcam Input	Tan Jing Jie, Jacynth Tham Ming Quan, Ng Jan Hui, Tan Wei Mun, Por Teong Dean
Android Application (Mobile) Input: Still Image + Real-Time Video Final Model Accuracy → 0.8125 (MobileNet)	Tan Jing Jie, Jacynth Tham Ming Quan, Ng Jan Hui, Tan Wei Mun, Por Teong Dean

The table above evidenced that each team member contributed for the development and report writing for all parts from the image pre-processing, segmentation all the way up to the integration of the classifiers into desktop-based and mobile-based applications.

Furthermore, this project contributes to society by developing a fast and efficient way of recognizing traffic signs accurately, something that existing traffic sign recognizers were not able to achieve. Moreover, many existing traffic sign recognizers only possessed fast recognizing speeds or high accuracies, but never both at once. However, with this project, a balance was able to be achieved between processing speed

and accuracy for both image-based and video-based classifications. Moving on, the integration of the developed traffic sign recognizer into an Android application contributes tremendously towards the research of smart cars. Future smart cars often have the need for integrated smart safety systems, with the ability to recognize traffic signs as one of the compulsory modules. In this project, the Android application developed along with the real-time voice outputs and high-accuracy MobileNet classifier will definitely be able to contribute to future constructions of smart cars.

9.6 Summary

[Done By: Jacynth Tham Ming Quan]

All in all, this chapter described the performance analysis of the three trained models, which were SVM, Random Forest and MobileNet respectively. Having compared the accuracy scores, confusion matrices, F1 scores and other classification metrics, it was concluded that the best performing model was the SVM classifier, which resulted in an accuracy on 0.9744. The Random Forest classifier came in second, with an accuracy of 0.9712. On the other hand, the mobile-based classifier, MobileNet, had an accuracy of 0.8125.

Furthermore, the error analysis of the developed traffic sign recognizer discovered that the classifiers were vulnerable to external noise, weather-affected images and had computational complexities. Therefore, in the future works, techniques such as neural network segmentation, image augmentation and fusion strategy should be used to conquer these vulnerabilities.

In short, the developed traffic sign recognizer was able to contribute to society by developing a fast and efficient way of recognizing traffic signs accurately through the use of desktop-based and mobile-based applications.

Chapter 10

Conclusion

10.0 Project Overview

[Done By: Tan Jing Jie, Jacynth Tham Ming Quan, Ng Jan Hui, Tan Wei Mun, Por Teong Dean]

In a nutshell, this project aims to develop a novel traffic sign recognizer that is able to keep drivers safe on the road as well as conquer the problems faced by existing traffic sign recognizers. The objective of this project is to enhance the accuracy and performance of existing traffic sign recognizers with the use of various classifiers such as SVM, Random Forest and MobileNet (CNN). This project aims to implement the future work of the existing systems reviewed, as well as integrate modified computer vision techniques in order to deal with blurred or weather-impacted traffic signs as well as processing time issues. The contribution of this project is a novel traffic sign recognizer that is able to work in real-time and in various lighting and weather conditions.

10.1 Summary of Methodology and Implementation

[Done By: Tan Jing Jie, Jacynth Tham Ming Quan, Ng Jan Hui, Tan Wei Mun, Por Teong Dean]

The novelties of this proposal algorithm are reducing the mathematical complexity as decrease of the dimension of input images. This is done by tracking the previous traffic sign of the image. Besides, this proposal proposes a high speed of recognition with high accuracy. Moreover, with the nature of this system – low computational complexity and high accuracy result, this proposal explores a mobile application-integrable traffic sign recognition system. Having trained, tested and compared the three classification models – SVM, Random Forest and MobileNet (CNN) – the models were then implemented into desktop-based and mobile-based applications, each being able to recognize traffic signs in still-images and real-time video-streams.

10.2 Future Work and Concluding Note

[Done By: Tan Jing Jie, Jacynth Tham Ming Quan, Ng Jan Hui, Tan Wei Mun, Por Teong Dean]

As there were still noticeable limitations of the developed desktop-based and mobile-based traffic sign recognizer, the future works of this system is to implement neural network segmentation, image augmentation and both tracking and fusion strategy in order to achieve the best possible balance between prediction accuracy and processing speed. All in all, the developed traffic sign recognizer was able to achieve all the objectives listed, including being able to work on both desktop-based and mobile-based applications. In the near future, the developed traffic sign recognizer can be integrated into smart cars as part of the built-in safety assistance system.

Bibliography

- Berkaya, S. K. et al., 2015. On circular traffic sign detection and recognition. *Expert Systems With Applications*, 48(1), pp. 67-75.
- Cires, a. D., Meier, U., Masci, J. & Schmidhuber, J., 2011. A Committee of Neural Networks for Traffic Sign Classification. *Proceedings of International Joint Conference on Neural Networks, San Jose, California, USA*.
- Ellahyani, A., Ansari, M. E. & Jaafari, I. E., 2016. Traffic sign detection and recognition based on random forests. *Applied Soft Computing*, Volume 46, pp. 805-815.
- Greenhalgh, J. & Mirmehdi, M., 2012. Real-Time Detection and Recognition of Road Traffic Signs. *IEEE Transactions on Intelligent Transportation Systems*, 13(4), pp. 1498-1507.
- Huang, Z., Yu, Y., Gu, J. & Liu, H., 2016. An Efficient Method for Traffic Sign Recognition Based on Extreme Learning Machine. *IEEE Transactions on Cybernetics*, 47(4), pp. 2168-2267.
- Jin, J., Fu, K. & Zhang, C., 2014. Traffic Sign Recognition With Hinge Loss Trained Convolutional Neural Networks. *IEEE Transactions on Intelligent Transportation Systems*, Volume 15, pp. 1991-2000.
- Larsson, F. & Feslberg, M., 2011. Using Fourier Descriptor and Spatial Models for Traffic Sign Recognition. *Scandinavian Conference on Image Analysis*, pp. 238-249.
- Li, J. et al., 2017. *Perceptual Generative Adversarial Networks for Small Object Detection*. s.l., 2017 IEEE Conference on Computer Vision and Pattern Recognition, pp. 1951-1959.
- Yang, Y., Luo, H., Huarong, X. & Wu, F., 2015. Towards Real-Time Traffic Sign Detection. *IEEE TRANSACTIONS ON INTELLIGENT TRANSPORTATION SYSTEMS*.
- Yuan, Y., Xiong, I. Z. & Wang, Q., 2017. An Incremental Framework for Video-Based Traffic Sign Detection, Tracking, and Recognition. *IEEE Transactions on Intelligent Transportation Systems*, 18(7), pp. 1918-1929.

- Zaklouta, F. & Stanciulescu, B., 2012. Real-time traffic sign recognition in three stages. *Robotics and Autonomous Systems Journal*.
- Zaklouta, F. & Stanciulescu, B., 2012. *Real-Time Traffic-Sign Recognition Using Tree Classifiers*. s.l., IEEE TRANSACTIONS ON INTELLIGENT TRANSPORTATION SYSTEMS, pp. 1507-1514.
- Zhang, J., Huang, M., Jin, X. & Li, X., 2017. A Real-Time Chinese Traffic Sign Detection Algorithm Based on Modified YOLOv2. *MDPI Algorithms*, 10(4), p. 127.
- Zhang, Y., Liu, Y., Zhang, Z. & Zhang, Z., 2018. Classification of Incomplete Data Based on Evidence Theory and an Extreme Learning Machine in Wireless Sensor Networks. *Sensors*, 18(4), p. 16.
- Zhu, Y. et al., 2016. Traffic sign detection and recognition using fully convolutional network guided proposals. *Neurocomputing*, Volume Volume 214, pp. Pages 758-766.
- Zhu, Z. et al., 2016. Traffic-Sign Detection and Classification in the Wild. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Appendices

A Poster

[Done by Tan Jing Jie]



The poster is titled "Traffic Sign Recognition System" and features a green Android robot icon and a road scene with speed limit signs in the background.

By Tan Jing Jie, Jacynth Tham Ming Quan, Tan Wei Mun, Ng Jan Hui, Por Teong Dean

Introduction

- Nowadays, there are many automotive cars which require the accurate traffic sign recognition technology as it is important for them to drive safely on the road.
- Besides, there are visually impaired people have difficulty to see the road sign while walking on road, this system deliverable is assist them by recognizing the traffic sign and output with simple voice recognition,

Objectives

- To develop a traffic sign recognizer that is able to recognize at least 43 traffic signs in different kinds of environments and weather conditions.
- To implement a traffic sign recognizing algorithm with more than 80% accuracy and a recognition time of fewer than 1 second per traffic sign.
- To develop a desktop-based traffic sign recognizer that is able to recognize traffic signs from still images.
- To develop a desktop-based traffic sign recognizer that is able to recognize traffic signs from file-based video input and webcam input.
- To develop a mobile-based traffic sign recognizing application that is able to recognize traffic signs from gallery-based image input and real-time video streams with voice out.

Methodology

```
graph TD; A[Image input] --> B[Image preprocessing]; B --> C[Feature Extraction]; C --> D[Classification]; D --> E[Result output];
```

This system work in both desktop and mobile application platform!!!

Results

Output: The highest accuracy obtained is 97.44% from SVM classifier.

A-1

Bachelor of Computer Science (Hons)

Faculty of Information and Communication Technology (Kampar Campus), UTAR

B Work Code

B.1 Image preprocessing

[Done by: Tan Jing Jie & Jacynth Tham Ming Quan]

This code is specially designed for showing all output of the flow of image preprocessing results. The core code of this image preprocessing (excluding the code that for displaying purpose) will be used in the training and testing process.

```
#include      <opencv2/opencv.hpp>
#include      <opencv2/highgui/highgui.hpp>
#include      <opencv2/imgproc.hpp>
#include      <opencv2/core.hpp>
#include      <iostream>
#include      <direct.h>
#include      <fstream>
#include      <math.h>
#include      "Supp.h"
#include <filesystem>
namespace fs = std::filesystem;

#define M_PI           3.14159265358979323846
using namespace cv;
using namespace std;

// Get which is larger
bool compareContourAreas(std::vector<cv::Point> contour1,
std::vector<cv::Point> contour2) {
    double i = fabs(contourArea(cv::Mat(contour1)));
    double j = fabs(contourArea(cv::Mat(contour2)));
    return (i < j);
}

// Customised function
void fillHole(const Mat srcBw, Mat& dstBw)
{
    //imshow("Original", srcBw);
    Size m_Size = srcBw.size();
    Mat Temp = Mat::zeros(m_Size.height + 2, m_Size.width + 2,
srcBw.type());
    srcBw.copyTo(Temp(Range(1, m_Size.height + 1), Range(1, m_Size.width + 1)));
    //imshow("Temp", Temp);
    cv::floodFill(Temp, Point(0, 0), Scalar(255, 255, 255));
    //imshow("After floodFill() Temp", Temp);
    Mat cutImg;
    Temp(Range(1, m_Size.height + 1), Range(1, m_Size.width + 1)).copyTo(cutImg);
    //imshow("CutImg", cutImg);
    //imshow("~CutImg", ~cutImg);
    dstBw = srcBw | (~cutImg);
    //imshow("Final Image", dstBw);
}

array<Mat, 2> furtherSegment(Mat srcI2, Mat singleMask, String name, int i) {
    Mat          largeWin, win[1 * 3], // create the new window
                legend[1 * 3]; // and the means to each sub-window
```

```

createWindowPartition(srcI2, largeWin, win, legend, 1, 3);
srcI2.copyTo(win[0]);
putText(legend[0], "First Segmented", Point(5, 11), 1, 1, Scalar(250,
250, 250), 1);

    const int          noOfImagePerCol = 2, noOfImagePerRow = 5, ///
create a 3X4 window partition
        totalRow = srcI2.rows,
        totalCol = srcI2.cols,
        ratio = 3,
        kernelSize = 3,
        size = 21;

    Mat    blurring,
           HSVImage,
           white, black, blue, mask,
           eroded,
           dilated,
           canvas,
           filledCanvasMask,
           finalMask;

    array<Mat, 2> segmented;

    Mat temp;

    RNG             rng(12345);

    canvas.create(totalRow, totalCol, CV_8UC3);
    canvas = Scalar(0, 0, 0);
    canvas.copyTo(filledCanvasMask);

    vector<vector<Point>> contours, contours_arrg;

    /*
--- */
// Prepare display enviroment
bilateralFilter(srcI2, blurring, size, size * 3, size / 3);

// Convert the image to HSV Colour Space & Grayscale
cvtColor(blurring, HSVImage, COLOR_BGR2HSV);

// Declaring the Hue Ranges for the colours
Scalar whiteLow = Scalar(0, 0, 150);
Scalar whiteHigh = Scalar(180, 50, 255);
Scalar blackLow = Scalar(0, 0, 0);
Scalar blackHigh = Scalar(180, 100, 60);
Scalar blueLow = Scalar(104, 236, 0);
Scalar blueHigh = Scalar(110, 255, 255);

// Match for Red
inRange(HSVImage, whiteLow, whiteHigh, white);
inRange(HSVImage, blackLow, blackHigh, black);
inRange(HSVImage, blueLow, blueHigh, blue);

// Remove noise
Mat element(2, 2, CV_8U, cv::Scalar(1));
morphologyEx(white, white, cv::MORPH_CLOSE, element);
morphologyEx(black, black, cv::MORPH_CLOSE, element);
morphologyEx(blue, blue, cv::MORPH_CLOSE, element);

// Merge the mask

```

```

cvtColor(singleMask, singleMask, COLOR_BGR2GRAY);
mask = (white | black | blue) & singleMask;

erode(mask, eroded, element);
dilate(eroded, dilated, element);

//Eliminate unneccesary contours
//findContours(dilated, contours, RETR_EXTERNAL, CHAIN_APPROX_NONE);
//drawContours(canvas, contours, contours.size()-1, Scalar(255, 255,
255)); // draw boundariess
//sort(contours.begin(), contours.end(), compareContourAreas);
//fillHole(canvas, filledCanvasMask);

cvtColor(dilated, finalMask, COLOR_GRAY2BGR);
//Segmented
segmented[0] = finalMask & srcI2;
segmented[0].copyTo(win[1]);
putText(legend[1], "Further Segmented", Point(5, 11), 1, 1,
Scalar(250, 250, 250), 1);

segmented[0].copyTo(segmented[1]);
segmented[1].setTo(Scalar(255, 255, 255), finalMask);
segmented[1].copyTo(win[2]);
resize(segmented[1], segmented[1], Size(80, 80), INTER_LINEAR);
putText(legend[2], "Binary output", Point(5, 11), 1, 1, Scalar(250,
250, 250), 1);

//Show content
imshow("Further Segmentation", largeWin);

// Record
imwrite("Output/" + name + "/Result/FurtherSegmentation" +
to_string(i) + ".png", largeWin);

return segmented;
}

void segmentMain(Mat in, String name) {

Mat srcI = in;

//resize(srcI,srcI, Size(320,480),0,0,1);

static int t1, t2, t3, t4;
const int noOfImagePerCol = 2, noOfImagePerRow = 5, // create a 3X4 window partition
totalRow = srcI.rows,
totalCol = srcI.cols,
ratio = 3,
kernelSize = 3,
size = 21;

Mat largeWin, summaryLargeWin, summaryWin[1 * 2],
win[noOfImagePerRow * noOfImagePerCol], // create the new window
summaryLegend[1 * 2], legend[noOfImagePerRow *
noOfImagePerCol]; // and the means to each sub-window

Mat blurring,
HSVImage,
redMask1, redMask2, blueMask, yellowMask, mask,
eroded,
dilated,

```

```

        canvas,
        finalCanvas,
        finalCanvasMask,
        segmented;

    Mat temp;
    RNG rng(12345);

    canvas.create(totalRow, totalCol, CV_8UC3);
    canvas = Scalar(0, 0, 0);
    canvas.copyTo(finalCanvasMask);

    vector<vector<Point>> contours, contours_arrg;

    /*
    --- */
    // Prepare display enviroment
    createWindowPartition(srcI, largeWin, win, legend, noOfImagePerCol,
noOfImagePerRow);
    createWindowPartition(srcI, summaryLargeWin, summaryWin,
summaryLegend, 1, 2);

    putText(legend[0], "Original", Point(5, 11), 1, 1, Scalar(250, 250,
250), 1);
    srcI.copyTo(win[0]);
    putText(summaryLegend[0], "Original", Point(5, 11), 1, 1, Scalar(250,
250, 250), 1);
    srcI.copyTo(summaryWin[0]);

    bilateralFilter(srcI, blurring, size, size * 3, size / 3);
    putText(legend[1], "Bilateral Filter", Point(5, 11), 1, 1, Scalar(250,
250, 250), 1);
    blurring.copyTo(win[1]);

    // Convert the image to HSV Colour Space & Grayscale
    cvtColor(blurring, HSVImage, COLOR_BGR2HSV);
    cvtColor(HSVImage, temp, COLOR_HSV2BGR);
    putText(legend[2], "HSV converted", Point(5, 11), 1, 1, Scalar(250,
250, 250), 1);
    temp.copyTo(win[2]);

    // Declaring the Hue Ranges for the colours
    // Hue Ranges for Red
    // According to color space, red is at the two left and right edge
    Scalar redLow1 = Scalar(150, 140, 160);
    Scalar redHigh1 = Scalar(180, 255, 255);

    Scalar redLow2 = Scalar(0, 50, 50);
    Scalar redHigh2 = Scalar(3, 255, 255);

    // Hue Ranges for Blue
    Scalar blueLow = Scalar(100, 150, 100);
    Scalar blueHigh = Scalar(128, 255, 255);

    // Hue Ranges for Yellow
    Scalar yellowLow = Scalar(14, 100, 140);
    Scalar yellowHigh = Scalar(30, 255, 255);

    // Match for Red
    inRange(HSVImage, redLow1, redHigh1, redMask1);

```

```

inRange(HSVImage, redLow2, redHigh2, redMask2);

// Match for Blue
inRange(HSVImage, blueLow, blueHigh, blueMask);

// Match for Yellow
inRange(HSVImage, yellowLow, yellowHigh, yellowMask);

// Remove noise
Mat element(2, 2, CV_8U, cv::Scalar(1));
morphologyEx(yellowMask, yellowMask, cv::MORPH_CLOSE, element);
morphologyEx(redMask1, redMask1, cv::MORPH_CLOSE, element);
morphologyEx(redMask2, redMask2, cv::MORPH_CLOSE, element);
morphologyEx(blueMask, blueMask, cv::MORPH_CLOSE, element);

//imshow("Red mask1", redMask1);
//imshow("Red mask2", redMask2);
//imshow("Red concate", redMask1 | redMask2);

// Merge the mask
mask = redMask1 | redMask2 | blueMask | yellowMask;
cvtColor(mask, temp, COLOR_GRAY2BGR);
temp.copyTo(win[3]);
putText(legend[3], "Mask", Point(5, 11), 1, 1, Scalar(250, 250, 250),
1);

//erode
erode(mask, eroded, element);
cvtColor(eroded, temp, COLOR_GRAY2BGR);
temp.copyTo(win[4]);
putText(legend[4], "Eroded", Point(5, 11), 1, 1, Scalar(250, 250,
250), 1);

//dilate
dilate(eroded, dilated, element);
cvtColor(dilated, temp, COLOR_GRAY2BGR);
temp.copyTo(win[5]);
putText(legend[5], "Dilate", Point(5, 11), 1, 1, Scalar(250, 250,
250), 1);

// Find the contour
findContours(dilated, contours, RETR_EXTERNAL, CHAIN_APPROX_NONE);

//Display purpose(called canvas)
for (int i = 0; i < contours.size(); i++) { // Just in case there is
more than one object in image
    for (;;) { // get random colors that are not too dim
        t1 = rng.uniform(0, 255); // blue
        t2 = rng.uniform(0, 255); // green
        t3 = rng.uniform(0, 255); // red
        t4 = t1 + t2 + t3;
        if (t4 > 255) break;
    }
    drawContours(canvas, contours, i, Scalar(t1, t2, t3)); // draw
boundaries
}
putText(legend[6], "Canvas", Point(5, 11), 1, 1, Scalar(250, 250,
250), 1);

```

```

        canvas.copyTo(win[6]);

        // Sort according to size
        sort(contours.begin(), contours.end(), compareContourAreas);

        // Eliminate smaller traffic sign
        for (int i = 0; i < contours.size(); i++) {
            if (contourArea(contours[i]) * 6 >
                contourArea(contours[contours.size() - 1]))
                contours_arrg.push_back(contours[i]);
        }

        // Do the final canvas (flood)
        for (int i = 0; i < contours_arrg.size(); i++) { // Just in case there
        is more than one object in image

            drawContours(finalCanvasMask, contours_arrg, i, Scalar(255,
        255, 255)); // draw boundaries
        }

        finalCanvasMask.copyTo(win[7]);
        putText(legend[7], "Final canvas boundary", Point(5, 11), 1, 1,
        Scalar(250, 250, 250), 1);

        fillHole(finalCanvasMask, finalCanvasMask);
        putText(legend[8], "Final mask", Point(5, 11), 1, 1, Scalar(250, 250,
        250), 1);
        finalCanvasMask.copyTo(win[8]);

        //Segmented
        segmented = finalCanvasMask & srcI;
        putText(legend[9], "Final segmented", Point(5, 11), 1, 1, Scalar(250,
        250, 250), 1);
        segmented.copyTo(win[9]);
        putText(summaryLegend[1], "Final segmented", Point(5, 11), 1, 1,
        Scalar(250, 250, 250), 1);
        segmented.copyTo(summaryWin[1]);

        //Bounding box
        vector<vector<Point>> contours_poly(contours_arrg.size());
        vector<Rect> boundRect(contours_arrg.size());
        for (int i = 0; i < contours_arrg.size(); i++)
        {
            approxPolyDP(contours_arrg[i], contours_poly[i], 3, true);
            boundRect[i] = boundingRect(contours_poly[i]);
        }

        // Crop and save
        for (int i = 0; i < boundRect.size(); i++) {

            //prepare only single mask
            Mat singleMask, resized;
            singleMask.create(totalRow, totalCol, CV_8UC3);
            singleMask = Scalar(0, 0, 0);
            drawContours(singleMask, contours_arrg, i, Scalar(255, 255,
        0));
            fillHole(singleMask, singleMask);

            //get the single traffic sign with rectangle
            Mat single = segmented & singleMask;

```

```

        single(Rect(boundRect[i].tl().x, boundRect[i].tl().y,
boundRect[i].br().x - boundRect[i].tl().x, boundRect[i].br().y -
boundRect[i].tl().y)).copyTo(single);
        resize(single, resized, Size(80, 80), INTER_LINEAR);
        imwrite("Output/" + name + "/Segmented/Img_" + to_string(i) +
".png", resized);

        //Do double segment
        singleMask(Rect(boundRect[i].tl().x, boundRect[i].tl().y,
boundRect[i].br().x - boundRect[i].tl().x, boundRect[i].br().y -
boundRect[i].tl().y)).copyTo(singleMask);
        array<Mat, 2> seg = furtherSegment(single, singleMask, name,i);
        imwrite("Output/" + name + "/FurtherSegmented/Img_" +
to_string(i) + ".png", seg[0]);
        imwrite("Output/" + name + "/BinarySegment/Img_" + to_string(i) +
".png", seg[1]);
    }

    //Show content
    imshow("Image processing", largeWin);
    imshow("Summary Image processing", summaryLargeWin);

    // Record
    imwrite("Output/" + name + "/Result/Flow.png", largeWin);
}

int main(int argc, char** argv) {
    fs::remove_all("Output");
    ifstream inputPathsFile("Inputs/inputSignNames.txt");
    int count = 1;
    if (inputPathsFile.is_open()) {
        string name_temp;
        while (getline(inputPathsFile, name_temp)) {

            fs::create_directories("Output/Img_" + to_string(count) +
"/Result");

            fs::create_directories("Output/Img_" + to_string(count) +
"/Segmented");

            fs::create_directories("Output/Img_" + to_string(count) +
"/FurtherSegmented");

            fs::create_directories("Output/Img_" + to_string(count) +
"/BinarySegment");

            Mat in = imread(name_temp);
            segmentMain(in, "Img_" + to_string(count));
            count++;
        }
    }

    waitKey();
    return 0;
}

```


B.2 Feature Extraction

[Done by: Ng Jan Hui & Tan Jing Jie]

extractFeaturesFromTrafficSign Function

```
// Feature Extraction Function
void extractFeaturesFromTrafficSign(std::vector<Mat>& furtherSegments, std::vector<Mat>& initialSegments, std::vector<std::vector<double>>& retHuMomentfeature, std::vector<std::vector<float>>& retHogFeature) {

    // Calculate Hu Moments from Further Segments
    for (int i = 0; i < furtherSegments.size(); i++) {
        Mat temp;

        cvtColor(furtherSegments[i], temp, COLOR_BGR2GRAY);

        Moments moments = cv::moments(temp, true);
        std::vector<double> huMoments;
        std::vector<double> logged_huMoments;
        bool foundInf = false;
        HuMoments(moments, huMoments);

        // Push the log transform vectors
        for (int i = 0; i < 7; i++) {

            /*if (isinf(huMoments[i]) || huMoments[i] == 0) {
                foundInf = true;
            }*/
            // logged_huMoments.push_back(huMoments[i]);
            logged_huMoments.push_back(-
1 * copysign(1.0, huMoments[i]) * log10(abs(huMoments[i])));
        }

        /*if (!foundInf)*/
        retHuMomentfeature.push_back(logged_huMoments);
    }

    HOGDescriptor hog;
    hog.winSize = Size(80, 80);
    hog.cellSize = Size(32, 32);
    hog.blockSize = Size(32, 32);
    hog.blockStride = Size(8, 8);
    vector<float> descriptors;

    for (size_t i = 0; i < initialSegments.size(); i++) {
        Mat temp;

        resize(initialSegments[i], temp, Size(80, 80), INTER_LINEAR);
        hog.compute(temp, descriptors);
        retHogFeature.push_back(descriptors);
    }
}
```

Generate Feature Matrix and Write to CSV Function

```
// Write to CSV
void generateFeatureMatrix(std::vector<std::vector<double>>& HuMomentfeature, std::vector<std::vector<float>>& HogFeature, std::vector<std::string>& labelVector) {

    // Label Encoding
    /*
        017 : Horn -> 1
        028 : Car -> 2
        031 : Uturn -> 3
        037 : School -> 4
        054 : NoEntry -> 5
        055 : Stop -> 6
    */

    // DEFINING A MAP TO STORE ALL LABEL CLASSES
    std::map<string, int> label_name = {
        {"17", 1},
        {"28", 2},
        {"31", 3},
        {"37", 4},
        {"54", 5},
        {"55", 6},
    };

    // Ofstream file to save csv
    std::ofstream featureCSV;
    featureCSV.open("traffic_sign_concat.csv");

    // Generate feature rows
    for (int i = 0; i < HuMomentfeature.size(); i++) {
        for (int j = 0; j < HuMomentfeature[i].size(); j++) {
            featureCSV << HuMomentfeature[i][j] << ",";
        }

        for (int j = 0; j < HogFeature[i].size(); j++) {
            featureCSV << HogFeature[i][j] << ",";
        }

        int val = 0;
        auto it = label_name.find(labelVector[i]);
        if (it != label_name.end())
            val = it->second;

        featureCSV << val;
        featureCSV << endl;
    }

    featureCSV.close();
}
```

B.3 Feature Dataset Imputation (Python)

[Done by: Ng Jan Hui]

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:


import numpy as np
import cv2
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split


# In[2]:


df = pd.read_csv('FeatureCSV/traffic_sign_concat.csv', header=None)


# In[3]:


df.head(10)


# In[4]:


df_clean = df.replace([np.inf, -np.inf, 0], np.nan)

df_clean.isnull().sum()


# In[5]:


from sklearn.impute import SimpleImputer

imputer = SimpleImputer(strategy='median')
imputer.fit(df_clean)
df_imputed = imputer.transform(df_clean)


# In[6]:


df_imputed = pd.DataFrame(df_imputed)
```

```
df_imputed.isnull().sum()

# In[7]:


df_imputed.to_csv('FeatureCSV/traffic_sign_concat_imputed.csv', header=False, index=False)
```

B.4 End-to-end Model Training (SVM + RF)

[Done by: Ng Jan Hui & Tan Wei Mun]

```
#include "Supp.h"
#include <iostream>
#include <map>
#include <math.h>
#include <opencv2/core.hpp>
#include <opencv2/dnn.hpp>
#include <opencv2/features2d.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc.hpp>
#include <opencv2/ml.hpp>
#include "opencv2/opencv_modules.hpp"
#include "opencv2/core/core.hpp"
#include "opencv2/ml/ml.hpp"
#include "opencv2/highgui/highgui.hpp"
#include <opencv2/opencv.hpp>
#include <opencv2/objdetect.hpp>
#include <random>
#include <vector>
#include     <fstream>

# define M_PI           3.14159265358979323846

using namespace std;
using namespace cv;
using namespace cv::ml;
using namespace dnn;

// Double Segmentation Function
std::array<Mat, 2> doubleSegment(Mat srcI2, Mat singleMask, int i) {

    const int          totalRow = srcI2.rows,
                      totalCol = srcI2.cols,
                      ratio = 3,
                      kernelSize = 3,
                      size = 21;

    Mat    blurring,
           HSVImage,
           white, black, blue, mask,
           eroded,
           dilated,
           canvas,
           filledCanvasMask,
           finalMask;

    std::array<Mat, 2> segmented;

    Mat temp;

    RNG             rng(12345);

    canvas.create(totalRow, totalCol, CV_8UC3);
    canvas = Scalar(0, 0, 0);
    canvas.copyTo(filledCanvasMask);
```

```

        std::vector<std::vector<Point>> contours, contours_arrg;

        /*
-----*/
        // Prepare display enviroment

        bilateralFilter(srcI2, blurring, size, size * 3, size / 3);

        // Convert the image to HSV Colour Space & Grayscale
        cvtColor(blurring, HSVImage, COLOR_BGR2HSV);

        // Declaring the Hue Ranges for the colours
        Scalar whiteLow = Scalar(0, 0, 0);
        Scalar whiteHigh = Scalar(180, 50, 255);
        Scalar blackLow = Scalar(0, 0, 0);
        Scalar blackHigh = Scalar(180, 50, 100);
        Scalar blueLow = Scalar(104, 236, 0);
        Scalar blueHigh = Scalar(110, 255, 255);

        // Match for Red
        inRange(HSVImage, whiteLow, whiteHigh, white);
        inRange(HSVImage, blackLow, blackHigh, black);
        inRange(HSVImage, blueLow, blueHigh, blue);

        // Remove noise
        Mat element(2, 2, CV_8U, cv::Scalar(1));
        morphologyEx(white, white, cv::MORPH_CLOSE, element);
        morphologyEx(black, black, cv::MORPH_CLOSE, element);
        morphologyEx(blue, blue, cv::MORPH_CLOSE, element);

        // Merge the mask
        cvtColor(singleMask, singleMask, COLOR_BGR2GRAY);
        mask = (white | black | blue) & singleMask;

        erode(mask, eroded, element);
        dilate(eroded, dilated, element);

        cvtColor(dilated, finalMask, COLOR_GRAY2BGR);
        //Segmented
        segmented[0] = finalMask & srcI2;

        segmented[0].copyTo(segmented[1]);
        segmented[0].setTo(Scalar(255, 255, 255), finalMask);

        return segmented;
    }

    void printImage300(std::vector<Mat> imageVector) {
        for (int i = 0; i < imageVector.size(); i++) {
            Mat temp;

            resize(imageVector[i], temp, Size(290, 290), INTER_LINEAR);
            imshow("Further Segment " + to_string(i + 1), temp);
            waitKey(0);
        }
    }

    // Get which is larger
}

```

```

bool compareContourAreas(std::vector<cv::Point> contour1, std::vector<cv::Point> contour2) {
    double i = fabs(contourArea(cv::Mat(contour1)));
    double j = fabs(contourArea(cv::Mat(contour2)));
    return (i < j);
}

// Customised function
void fillHole(const Mat srcBw, Mat& dstBw)
{
    Size m_Size = srcBw.size();
    Mat Temp = Mat::zeros(m_Size.height + 2, m_Size.width + 2, srcBw.type());
    srcBw.copyTo(Temp(Range(1, m_Size.height + 1), Range(1, m_Size.width + 1)));
    cv::floodFill(Temp, Point(0, 0), Scalar(255, 255, 255));
    Mat cutImg;
    Temp(Range(1, m_Size.height + 1), Range(1, m_Size.width + 1)).copyTo(cutImg);
    dstBw = srcBw | (~cutImg);
}

// Segmentation Function (Will Trigger Double Segmentation)
void segmentMain(String path, std::vector<Mat>& returnFurtherSegments, std::vector<std::string>& returnLabelNames, std::vector<Mat>& returnFirstSegments) {

    Mat srcI = imread(path);
    /*imshow(path, srcI);
    waitKey(0);*/

    //resize(srcI, srcI, Size(320,480),0,0,1);

    static int t1, t2, t3, t4;
    const int totalRow = srcI.rows,
              totalCol = srcI.cols,
              ratio = 3,
              kernelSize = 3,
              size = 21;

    Mat blurring,
        HSVImage,
        redMask1, redMask2, blueMask, yellowMask, mask,
        eroded,
        dilated,
        canvas,
        filledCanvasMask,
        segmented;

    Mat temp;

    RNG rng(12345);

    canvas.create(totalRow, totalCol, CV_8UC3);
    canvas = Scalar(0, 0, 0);
    canvas.copyTo(filledCanvasMask);

    std::vector<std::vector<Point> > contours, contours_arrg;

```

```

bilateralFilter(srcI, blurring, size, size * 3, size / 3);

// Convert the image to HSV Colour Space & Grayscale
cvtColor(blurring, HSVImage, COLOR_BGR2HSV);
cvtColor(HSVImage, temp, COLOR_HSV2BGR);

// Declaring the Hue Ranges for the colours
// Hue Ranges for Red
// According to colour space, red is at the two left and right edges
Scalar redLow1 = Scalar(159, 140, 160);
Scalar redHigh1 = Scalar(180, 255, 255);

Scalar redLow2 = Scalar(0, 50, 50);
Scalar redHigh2 = Scalar(7, 255, 255);

// Hue Ranges for Blue
Scalar blueLow = Scalar(101, 150, 100);
Scalar blueHigh = Scalar(130, 255, 255);

// Hue Ranges for Yellow
Scalar yellowLow = Scalar(16, 100, 140);
Scalar yellowHigh = Scalar(36, 255, 255);

// Match for Red
inRange(HSVImage, redLow1, redHigh1, redMask1);
inRange(HSVImage, redLow2, redHigh2, redMask2);

// Match for Blue
inRange(HSVImage, blueLow, blueHigh, blueMask);

// Match for Yellow
inRange(HSVImage, yellowLow, yellowHigh, yellowMask);

// Remove noise
Mat element(2, 2, CV_8U, cv::Scalar(1));
morphologyEx(yellowMask, yellowMask, cv::MORPH_CLOSE, element);
morphologyEx(redMask1, redMask1, cv::MORPH_CLOSE, element);
morphologyEx(redMask2, redMask2, cv::MORPH_CLOSE, element);
morphologyEx(blueMask, blueMask, cv::MORPH_CLOSE, element);

// Merge the mask
mask = redMask1 | redMask2 | blueMask | yellowMask;
cvtColor(mask, temp, COLOR_GRAY2BGR);

// erode
erode(mask, eroded, element);
cvtColor(eroded, temp, COLOR_GRAY2BGR);

// dilate
dilate(eroded, dilated, element);
cvtColor(dilated, temp, COLOR_GRAY2BGR);

// Find the contour
findContours(dilated, contours, RETR_EXTERNAL, CHAIN_APPROX_NONE)
;

for (int i = 0; i < contours.size(); i++) { // Just in case there
    is more than one object in image
}

```

```

        for (;;) { // get random colors that are not too dim
            t1 = rng.uniform(0, 255); // blue
            t2 = rng.uniform(0, 255); // green
            t3 = rng.uniform(0, 255); // red
            t4 = t1 + t2 + t3;
            if (t4 > 255) break;
        }

        drawContours(canvas, contours, i, Scalar(t1, t2, t3)); // draw boundaries
    }

    // Sort according to size
    sort(contours.begin(), contours.end(), compareContourAreas);

    // draw the largest contour if exist
    if (contours.size() > 0) {
        contours_arrg.push_back(contours[contours.size() - 1]);

        drawContours(filledCanvasMask, contours_arrg, 0, Scalar(255, 255, 255));

        // Get the label names from the file
        std::string label_class = path.substr(15, 2);
        returnLabelNames.push_back(label_class);
    }

    fillHole(filledCanvasMask, filledCanvasMask);

    //Segmented
    segmented = filledCanvasMask & srcI;

    //Bounding box
    std::vector<std::vector<Point>> contours_poly(contours_arrg.size());
    std::vector<Rect> boundRect(contours_arrg.size());
    for (int i = 0; i < contours_arrg.size(); i++)
    {

        approxPolyDP(contours_arrg[i], contours_poly[i], 3, true);
        boundRect[i] = boundingRect(contours_poly[i]);
    }

    Mat drawing = Mat::zeros(canvas.size(), CV_8UC3);
    for (int i = 0; i < contours_arrg.size(); i++)
    {
        Rect rect = boundRect[i];

        cv::rectangle(srcI, boundRect[i].tl(), boundRect[i].br(), Scalar(255, 255, 0), 2, 0, 0);
    }

    // Crop and save
    for (int i = 0; i < boundRect.size(); i++) {
        // prepare only single mask

```

```

        Mat singleMask;
        singleMask.create(totalRow, totalCol, CV_8UC3);
        singleMask = Scalar(0, 0, 0);

        drawContours(singleMask, contours_arrg, i, Scalar(255, 255, 0));
        fillHole(singleMask, singleMask);

        // get the single traffic sign with rectangle
        Mat single = segmented & singleMask;
        returnFirstSegments.push_back(single);

        single(Rect(boundRect[i].tl().x, boundRect[i].tl().y, boundRect[i]
            ].br().x - boundRect[i].tl().x, boundRect[i].br().y - boundRect[i].tl()
            .y)).copyTo(single);

        // Do double segment

        singleMask(Rect(boundRect[i].tl().x, boundRect[i].tl().y, boundRe
            ct[i].br().x - boundRect[i].tl().x, boundRect[i].br().y - boundRect[i].tl()
            .y)).copyTo(singleMask);

        std::array<Mat, 2> seg = doubleSegment(single, singleMask, i);

        returnFurtherSegments.push_back(seg[0]);
    }

// Feature Extraction Function
void extractFeaturesFromTrafficSign(std::vector<Mat>& furtherSegments, st
d::vector<Mat>& initialSegments, std::vector<std::vector<double>>& retHuM
omentfeature, std::vector<std::vector<float>>& retHogFeature) {

    // Calculate Hu Moments from Further Segments
    for (int i = 0; i < furtherSegments.size(); i++) {
        Mat temp;

        cvtColor(furtherSegments[i], temp, COLOR_BGR2GRAY);

        Moments moments = cv::moments(temp, true);
        std::vector<double> huMoments;
        std::vector<double> logged_huMoments;
        bool foundInf = false;
        HuMoments(moments, huMoments);

        // Push the log transform vectors
        for (int i = 0; i < 7; i++) {

            /*if (isinf(huMoments[i]) || huMoments[i] == 0) {
                foundInf = true;
            }*/
            // logged_huMoments.push_back(huMoments[i]);
            logged_huMoments.push_back(-
1 * copysign(1.0, huMoments[i]) * log10(abs(huMoments[i])));
        }

        /*if (!foundInf)*/
        retHuMomentfeature.push_back(logged_huMoments);
    }
}

```

```

HOGDescriptor hog;
hog.winSize = Size(80, 80);
hog.cellSize = Size(32, 32);
hog.blockSize = Size(32, 32);
hog.blockStride = Size(8, 8);
vector<float> descriptors;

for (size_t i = 0; i < initialSegments.size(); i++) {
    Mat temp;

    resize(initialSegments[i], temp, Size(80, 80), INTER_LINEAR);
    hog.compute(temp, descriptors);
    retHogFeature.push_back(descriptors);
}

// Write to CSV
void generateFeatureMatrix(std::vector<std::vector<double>>& HuMomentfeature,
                           std::vector<std::vector<float>>& HogFeature, std::vector<std::string>& labelVector) {

    // Label Encoding
    /*
    017 : Horn -> 1
    028 : Car -> 2
    031 : Uturn -> 3
    037 : School -> 4
    054 : NoEntry -> 5
    055 : Stop -> 6
    */

    // DEFINING A MAP TO STORE ALL LABEL CLASSES
    std::map<string, int> label_name = {
        {"17", 1},
        {"28", 2},
        {"31", 3},
        {"37", 4},
        {"54", 5},
        {"55", 6},
    };

    // Ofstream file to save csv
    std::ofstream featureCSV;
    featureCSV.open("traffic_sign_concat.csv");

    // Generate feature rows
    for (int i = 0; i < HuMomentfeature.size(); i++) {
        for (int j = 0; j < HuMomentfeature[i].size(); j++) {
            featureCSV << HuMomentfeature[i][j] << ",";
        }

        for (int j = 0; j < HogFeature[i].size(); j++) {
            featureCSV << HogFeature[i][j] << ",";
        }

        int val = 0;
        auto it = label_name.find(labelVector[i]);

```

```

        if (it != label_name.end())
            val = it->second;

        featureCSV << val;
        featureCSV << endl;
    }

    featureCSV.close();
}

int main(int argc, char** argv) {
    int choice;

    cout << "Skip Segmentation (1 - Yes, 2 - No): ";
    cin >> choice;

    if (choice == 2) {
        /*STEP 1 -> SEGMENTATION - WILL SAVE TO CSV*/

        /*PREPROCESSING - Variable Declaration*/
        std::vector<Mat> furtherSegments;
        std::vector<Mat> firstSegments;
        std::vector<std::string> allLabels;

        // Reading the input paths file

        ifstream inputPathsFile("Input Dataset/inputSignNames.txt");

        // Start Segmentation Process
        std::cout << "Begin Segmentation";

        int ct = 0;

        if (inputPathsFile.is_open()) {
            std::string name_temp;
            while (getline(inputPathsFile, name_temp)) {

                std::cout << ct++ << ". " << name_temp << " Done " << endl;

                segmentMain(name_temp, furtherSegments, allLabels, firstSegments);
            // returns -> 1) Internal Segment 2) Encoded label 3) Initial Segment
            }
        }

        std::cout << "\t....Done" << endl << endl;

        /*STEP 2 - Feature Extraction*/

        // Start Feature Extraction
        std::cout << "Begin Feature Extraction";

        // Feature Vector
        std::vector<std::vector<double>> huMomentsFeatures;
        std::vector<std::vector<float>> HoGFeatures;

        extractFeaturesFromTrafficSign(furtherSegments, firstSegments, hu
        MomentsFeatures, HoGFeatures);
    }
}

```

```

        std::cout << "\t....Done" << endl << endl;

        /*STEP 3 - Writing Feature to CSV*/

        // Start Write To CSV
        std::cout << "Begin writing to CSV" ;

        generateFeatureMatrix(huMomentsFeatures, HoGFeatures, allLabels);

        std::cout << "\t....Done" << endl << endl;
    }

    /*STEP 4 - DATASET PREPROCESSING - Reading the csv dataset file*/

    Ptr<cv::ml::TrainData> dataset = cv::ml::TrainData::loadFromCSV("traffic_sign_concat_imputed.csv", 0);
    if (dataset.empty()) {
        std::cout << "CSV can't be read" << endl;
        return 0;
    }

    // Splitting the dataset into training and testing (70% Training
    and 30% Testing)
    dataset->setTrainTestSplitRatio(0.70);
    std::cout << "No. of Test / No. of Train: " << dataset-
    >getNTestSamples() << " / " << dataset->getNTrainSamples() << endl;

    Mat trainSet = dataset->getTrainSamples();
    Mat trainLabels = dataset->getTrainResponses();

    Mat testSet = dataset->getTestSamples();
    Mat testLabels = dataset->getTestResponses();

    // Converting label to use as integer for SVM
    Mat trainLabels_int;
    trainLabels.convertTo(trainLabels_int, CV_32S);

    Mat testLabels_int;
    testLabels.convertTo(testLabels_int, CV_32S);
    std::system("Pause");
    std::system("cls");

    // Print out the training data
    std::cout << "======" << endl;
    std::cout << "Training Data Matrix" << endl;
    std::cout << "======" << endl << e
ndl;
    std::cout << "\t\t\t" << "Hu Moment 1\t" << "Hu Moment 2\t" << "H
u Moment 3\t" << "Hu Moment 4\t" << "Hu Moment 5\t" << "Hu Moment 6\t" <<
    "Hu Moment 7\t" << endl << endl;
    for (int i = 0; i < trainSet.rows; i++) {
        std::cout << "Training Sample " << i + 1 << "\t";
        for (int j = 0; j < 7; j++) {

            std::cout << setw(10) << trainSet.at<float>(i, j) << "\t";
        }
    }
}

```

```

        std::cout << endl;
    }
    std::cout << endl << endl;

    // Print out the test data
    std::cout << "======" << endl;
    std::cout << "Test Data Matrix" << endl;
    std::cout << "======" << endl << endl;
    std::cout << "\t\t\t" << "Hu Moment 1\t" << "Hu Moment 2\t" << "Hu
    u Moment 3\t" << "Hu Moment 4\t" << "Hu Moment 5\t" << "Hu Moment 6\t" <<
    "Hu Moment 7\t" << endl << endl;
    for (int i = 0; i < testSet.rows; i++) {
        std::cout << "Testing Sample " << i + 1 << "\t";
        for (int j = 0; j < 7; j++) {

            std::cout << setw(10) << testSet.at<float>(i, j) << "\t";
        }
        std::cout << endl;
    }
    std::cout << endl << endl;

    std::cout << "Begin Model Training - SVM + RF" << endl << endl;

    /*STEP 6- MODEL TRAINING*/

    /*Random Forest*/
    Ptr<RTrees> rtrees = RTrees::create();
    rtrees->setMaxDepth(10);
    rtrees->setMinSampleCount(2);
    rtrees->setRegressionAccuracy(0);
    rtrees->setUseSurrogates(false);
    rtrees->setMaxCategories(16);
    rtrees->setPriors(Mat());
    rtrees->setCalculateVarImportance(true);
    rtrees->setActiveVarCount(0);
    rtrees-
    >setTermCriteria(TermCriteria(TermCriteria::MAX_ITER, 100, 0));

    // Training
    std::cout << "Begin Training Random Forest";
    rtrees->train(trainSet, ROW_SAMPLE, trainLabels_int);
    rtrees->save("T6G1_rtree.xml");
    std::cout << "\t....Done" << endl << endl;
    // Testing
    std::cout << "Begin prediction of testing set";
    Mat rtrees_testResults;
    rtrees->predict(testSet, rtrees_testResults);
    std::cout << "\t....Done" << endl << endl;

    /*SVM*/
    Ptr<SVM> svm = SVM::create();
    svm->setType(SVM::C_SVC);
    svm->setKernel(SVM::LINEAR);
    svm-
    >setTermCriteria(TermCriteria(TermCriteria::MAX_ITER, 1000, 1e-6));

    // Training

```

```

        std::cout << "Begin Training SVM";
        svm->train(trainSet, ROW_SAMPLE, trainLabels_int);
        svm->save("T6G1_SVM.xml");
        std::cout << "\t....Done" << endl << endl;
        // Testing
        std::cout << "Begin prediction of testing set";
        Mat svm_testResults;
        svm->predict(testSet, svm_testResults);
        std::cout << "\t....Done" << endl << endl;
        std::cout << "Done Model Training - SVM + RF" << endl << endl;

        std::cout << "Experimental Results - Accuracy of Classifier" << endl << endl;

        /*STEP 7 - EXPERIMENTAL RESULTS*/
        ofstream outRTree("Random_Forest_res.csv");
        ofstream outSVM("SVM_res.csv");
        ofstream outTestLabels("testLabels.csv");

        // Random Forest Results
        float rtree_accuracy = float(countNonZero(rtrees_testResults == testLabels)) / testLabels.rows;
        std::cout << "\n\nRTree Classification accuracy: " << rtree_accuracy;
        std::cout << endl;

        // SVM Results
        float svm_accuracy = float(countNonZero(svm_testResults == testLabels)) / testLabels.rows;
        std::cout << "\n\nSVM Classification accuracy: " << svm_accuracy;
        std::cout << endl;

        // Write the Predicted Labels to CSV (Use Python for Analysis)
        outRTree << format(rtrees_testResults, cv::Formatter::FMT_CSV) << endl;
        outSVM << format(svm_testResults, cv::Formatter::FMT_CSV) << endl;
        outTestLabels << format(testLabels, cv::Formatter::FMT_CSV) << endl;

        outRTree.close();
        outSVM.close();
        outTestLabels.close();

        std::cout << endl << endl;
        std::system("pause");
        return 0;
    }
}

```

B.5 Mobile Net Convolutional Neural Network Classification Training, Testing and Experimental result

[Done by: Tan Jing Jie & Jacynth Tham Ming Quan]

The CNN classifier code is documented in ipynb notebook. The link of the notebook is: <https://colab.research.google.com/drive/1VtMPejj-EviEMj1-kKhvQjFWPaxydak?usp=sharing>.

The label for traffic sign is start from 0 to 42 which sorted as following.

```
0: '20_speed',
1: '30_speed',
2: '50_speed',
3: '60_speed',
4: '70_speed',
5: '80_speed',
6: '80_lifted',
7: '100_speed',
8: '120_speed',
9: 'no_overtaking_general',
10: 'no_overtaking_trucks',
11: 'right_of_way_crossing',
12: 'right_of_way_general',
13: 'give_way',
14: 'stop',
15: 'no_way_general',
16: 'no_way_trucks',
17: 'no_way_one_way',
18: 'attention_general',
19: 'attention_left_turn',
20: 'attention_right_turn',
21: 'attention_curvy',
22: 'attention_bumpers',
23: 'attention_slippery',
24: 'attention_bottleneck',
25: 'attention_construction',
26: 'attention_traffic_light',
27: 'attention_pedestrian',
28: 'attention_children',
29: 'attention_bikes',
30: 'attention_snowflake',
31: 'attention_deer',
32: 'lifted_general',
33: 'turn_right',
34: 'turn_left',
35: 'turn_straight',
```

```
36: 'turn_straight_right',
37: 'turn_straight_left',
38: 'turn_right_down',
39: 'turn_left_down',
40: 'turn_circle',
41: 'lifted_no_overtaking_general',
42: 'lifted_no_overtaking_trucks'
```

```
# -*- coding: utf-8 -*-
"""MobileNet Using Traffic Signs.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1VtMPejj-EviEMj1-kKhvQjF-WPaxydak

# MobileNet CNN Model (UCCC2513 Mini Project)

Created By: Team 1

[Tan Jing Jie & Jacynth Tham Ming Quan]

This python notebook demonstrates the retraining and testing of Tensorflow's MobileNet Convolutional Neural Network. The output of this notebook is a TensorFlowLite file that is able to work efficiently on mobile applications.

## Notebook Setup

First, the **TensorFlow Hub** dependency has to be added in order to obtain the MobileNet CNN model.

"""

# install dependency if not already installed
!pip install tensorflow_hub
!pip install cmocean

from PIL import Image
import os
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import zipfile
import tensorflow_hub as hub
from tensorflow.keras import layers
import pandas as pd
```

```

from sklearn.metrics import accuracy_score
import sklearn.metrics

"""### Configure Project Space

Determine directories where the output model file is stored
"""

OUTPUT_ROOT_DIR = "output/"
OUTPUT_TFLITE_MODEL = os.path.join(OUTPUT_ROOT_DIR, "mobilenet.tflite")
OUTPUT_LABELS = os.path.join(OUTPUT_ROOT_DIR, "mobilenetlabel.txt")
OUTPUT_READABLE_LABELS = os.path.join(OUTPUT_ROOT_DIR, "mobilenetlabel_readable.txt")

"""### Configure MobileNet Model"""

mobilenet_model = "https://tfhub.dev/google/imagenet/mobilenet_v1_100_224/feature_vector/4"

image_size = (224, 224)

model_input_height = 224
model_input_width = 224
model_input_mean = 0
MODEL_INPUT_STD = 255

#Placeholder is the name of the input layer of the MobileNet model
model_input_layer = "Placeholder"

#final_result is the name of the output layer of the MobileNet model
model_output_layer = "final_result"

"""## Preparing Dataset"""

tmp_dir = "dataset/tmp"
tmp_label_dir = os.path.join(tmp_dir, "GTSRB/Final_Test")

train_data_dir = "dataset/training"
validation_data_dir = "dataset/validation"

#Download images from GTSRB website

#Images for training
!curl -L https://sid.erda.dk/public/archives/daaeac0d7ce1152aea9b61d9f1e19370/GTSRB_Final_Training_Images.zip

```

```

#Images for validation
!curl -
LOC - https://sid.elda.dk/public/archives/daaeac0d7ce1152aea9b61d9f1e
19370/GTSRB_Final_Test_Images.zip

#Labels for validation
!curl -
LOC - https://sid.elda.dk/public/archives/daaeac0d7ce1152aea9b61d9f1e
19370/GTSRB_Final_Test_GT.zip

# Commented out IPython magic to ensure Python compatibility.
# %time
# unload = [
#     ("GTSRB_Final_Training_Images.zip", tmp_dir),
#     ("GTSRB_Final_Test_Images.zip", tmp_dir),
#     ("GTSRB_Final_Test_GT.zip", tmp_label_dir)
# ]
#
# for file, directory in unload:
#     print("Unzipping {} zip file to {} directory.".format(file, directory))
#
#     with zipfile.ZipFile(file,"r") as zip_reference:
#         zip_reference.extractall(directory)

"""### Prepare Labels, Training Data and Testing Data

#### Labels
"""

#All the labels of the traffic signs in the dataset
label_map = {
    0: '20_speed',
    1: '30_speed',
    2: '50_speed',
    3: '60_speed',
    4: '70_speed',
    5: '80_speed',
    6: '80_lifted',
    7: '100_speed',
    8: '120_speed',
    9: 'no_overtaking_general',
    10: 'no_overtaking_trucks',
    11: 'right_of_way_crossing',
    12: 'right_of_way_general',
    13: 'give_way',
    14: 'stop',
    15: 'no_way_general',
}

```

```

16: 'no_way_trucks',
17: 'no_way_one_way',
18: 'attention_general',
19: 'attention_left_turn',
20: 'attention_right_turn',
21: 'attention_curvy',
22: 'attention_bumpers',
23: 'attention_slippery',
24: 'attention_bottleneck',
25: 'attention_construction',
26: 'attention_traffic_light',
27: 'attention_pedestrian',
28: 'attention_children',
29: 'attention_bikes',
30: 'attention_snowflake',
31: 'attention_deer',
32: 'lifted_general',
33: 'turn_right',
34: 'turn_left',
35: 'turn_straight',
36: 'turn_straight_right',
37: 'turn_straight_left',
38: 'turn_right_down',
39: 'turn_left_down',
40: 'turn_circle',
41: 'lifted_no_overtaking_general',
42: 'lifted_no_overtaking_trucks'
}

#if the output directory does not exist, make one
if not os.path.exists(OUTPUT_ROOT_DIR):
    os.makedirs(OUTPUT_ROOT_DIR)

file = open(OUTPUT_READABLE_LABELS, 'w')

#write all the labels into the output label file
for key, val in sorted(label_map.items()):
    file.write("{}\n".format(val))

file.close()

"""

#### Training Set"""

# Input all PPM files and labels

```

```

tmp_train_data_dir = os.path.join(tmp_dir, "GTSRB/Final_Training/Images")

# Get all subdirectories of data_dir. One subdirectory = one label
subdirectories = [d for d in os.listdir(tmp_train_data_dir)
                  if os.path.isdir(os.path.join(tmp_train_data_dir, d))]

training_data = []
training_labels = []

#for all subdirectories, collect the list of images and labels (as a pair)
for class_directory in subdirectories:
    label_dir = os.path.join(tmp_train_data_dir, class_directory)
    file_names = [os.path.join(label_dir, f)
                  for f in os.listdir(label_dir) if f.endswith(".ppm")]
]

    for image_file in file_names:
        training_data.append(image_file)
        training_labels.append(class_directory)

# Sort the image and label lists
training_data.sort()
training_labels.sort()

# Commented out IPython magic to ensure Python compatibility.
# %time
# # Change file path for all images + convert ppm to jpg
#
# for ppm_input_file, label in zip(training_data, training_labels):
#     image = Image.open(ppm_input_file)
#     directory = os.path.join(train_data_dir, label)
#     image_filename = "{}.jpg".format(os.path.splitext(os.path.basename(ppm_input_file))[0])
#
#     if not os.path.exists(directory):
#         os.makedirs(directory)
#
#     image.save(os.path.join(directory, image_filename))

#Show all stop image
print(train_data_dir)
training_images =[]
a=os.listdir(os.path.join(train_data_dir, "00018"))

for b in a:
    if "00018." in str(b):

```

```

        training_images.append(os.path.join(train_data_dir, "00018", str(b)))

print(training_images)

i=0
#plot out all the categories with one exp image
plt.figure(figsize=(17, 30))
for image in training_images:
    plt.subplot(10,7, i + 1)
    plt.axis('off')
    plt.title("{}".format(label_map[12])) #label
    plt.imshow(Image.open(image))
    i = i+1
    if(i==30):
        break
plt.show()

print(len(training_images))

# Show all categories of signs with example images
training_dir = [d for d in os.listdir(train_data_dir)
                if os.path.isdir(os.path.join(train_data_dir, d))]

training_dir.sort()

training_images = []
for training_dir in training_dir:
    training_images.append(os.path.join(train_data_dir, training_dir,
                                        "00000_00000.jpg"))

#set counter to 0
i = 0

#plot out all the categories with one exp image
plt.figure(figsize=(17, 30))
for image in training_images:
    plt.subplot(10,7, i + 1)
    plt.axis('off')
    plt.title("{}".format(label_map[i])) #label
    i += 1
    plt.imshow(Image.open(image))
plt.show()

print(i)

```

```

"""All of TensorFlow Hub's image modules expect float inputs in the `rescale` parameter to achieve this.

The image size will be handled later.

"""

#Resale the images to float inputs of between [0, 1] so that it can be used by TensorFlow Hub

image_float = tf.keras.preprocessing.image.ImageDataGenerator(rescale=1/255)
image_float_data = image_float.flow_from_directory(str(train_data_dir), target_size=image_size)

# results from the previous block of code are pairs of image and label
for image_batch, label_batch in image_float_data:
    print("Image shape: ", image_batch.shape)
    print("Label shape: ", label_batch.shape)
    break

"""#### Testing Dataset"""

#Load testing set
testing_data_dir = os.path.join(tmp_dir, "GTSRB/Final_Test/Images")

# Commented out IPython magic to ensure Python compatibility.
# %time
#
# #Same process as observed in the Training Dataset just now
#
# #find all ppm files
# testing_data = [f for f in os.listdir(testing_data_dir) if f.endswith(".ppm")]
#
# testing_images = []
#
# #convert ppm to jpg
# for ppm_file in testing_data:
#     image_dir = os.path.join(testing_data_dir, ppm_file)
#     image = Image.open(image_dir)
#     directory = validation_data_dir

```

```

#     image_filename = "{}.jpg".format(os.path.splitext(os.path.basename(ppm_file))[0])
#
#     if not os.path.exists(directory):
#         os.makedirs(directory)
#
#     final_image = os.path.join(directory, image_filename)
#     image.save(final_image)
#
#     testing_images.append(final_image)
#     testing_images.sort()
#
# print("Testing dataset images:", len(testing_images))

"""#### Exploration"""

number_files = len(subdirectories)
print("The number of traffic sign class: " + str(number_files))

number_training_data = len(training_data)
print("The number of training data: " + str(number_training_data))

number_testing_data = len(testing_data)
print("The number of testing data: " + str(number_testing_data))

number_total_data = number_training_data + number_testing_data
print("The number of total data (training +testing): " + str(number_total_data))

"""## Training The Model"""

#Download the mobilenet model

mobilenet_url = "https://tfhub.dev/google/imagenet/mobilenet_v2_100_224/feature_vector/4" #@param {type:"string"}
mobilenet_layer = hub.KerasLayer(mobilenet_url,
                                  input_shape=(224,224,3))

mobilenet_batch = mobilenet_layer(image_batch)

print(mobilenet_batch.shape)

#Setting this parameter to false means the variables in the feature extraction layer will be frozen
#Thus, the training process only affects the classifying layer

mobilenet_layer.trainable = False

```

```

#Add a head to the downloaded mobilenet model

model = tf.keras.Sequential([
    mobilenet_layer,
    layers.Dense(image_float_data.num_classes)
])

model.summary()

initial_predictions = model(image_batch)

initial_predictions.shape

#Compile the model and prepare for training

model.compile(
    optimizer=tf.keras.optimizers.Adam(),
    loss=tf.keras.losses.CategoricalCrossentropy(from_logits=True),
    metrics=['acc'])

#Define custom class and method to handle the batch training process

class CollectBatchStats(tf.keras.callbacks.Callback):
    def __init__(self):
        self.batch_losses = []
        self.batch_acc = []

    def on_train_batch_end(self, batch, logs=None):
        self.batch_losses.append(logs['loss'])
        self.batch_acc.append(logs['acc'])
        self.model.reset_metrics()

#Start training process

steps_per_epoch = np.ceil(image_float_data.samples/image_float_data.batch_size)

batch_stats_callback = CollectBatchStats()

history = model.fit(image_float_data, epochs=20,
                     steps_per_epoch=steps_per_epoch,
                     callbacks=[batch_stats_callback])

"""Now after, even just a few training iterations, we can already see
that the model is making progress on the task."""

#Plot the Loss Against the Training Steps
plt.figure()

```

```

plt.ylabel("Loss")
plt.xlabel("Training Steps")
plt.ylim([0,2])
plt.plot(batch_stats_callback.batch_losses)

#Plot the Accuracy Against Training Steps

plt.figure()
plt.ylabel("Accuracy")
plt.xlabel("Training Steps")
plt.ylim([0,1])
plt.plot(batch_stats_callback.batch_acc)

"""## Evaluate the Model

"""

#Load the test data from the CSV file into a dataframe. Then change all the ppm to jpg in the Filename column.
training_labels_csv = os.path.join(tmp_label_dir, "GT-final_test.csv")
test_df = pd.read_csv(training_labels_csv, header=0, sep=';')
test_df['Filename'] = test_df['Filename'].str.replace('.ppm','.jpg')
test_df['ClassId'] = test_df['ClassId'].astype(str).str.zfill(5)

image_val_data = image_float.flow_from_dataframe(test_df, x_col="Filename", batch_size=12630 ,directory=validation_data_dir, y_col="ClassId", target_size=image_size)

for image_test, label_test in image_val_data:
    print("Image shape: ", image_test.shape)
    print("Label shape: ", label_test.shape)
    break

"""### Test Images

"""

predicted_image = model.predict(image_test)
predicted_id = np.argmax(predicted_image, axis=-1)
label_id = np.argmax(label_test, axis=-1)

"""##### Please skip this, as the count of images is too big. It may led to memory leak, or you may just edit the batch size to 1000"""

batch_size = image_test.shape[0]
num_plot_column = 5

```

```

num_plot_row = batch_size // num_plot_column + (batch_size % num_plot
_column > 0)

plt.figure(figsize=(10,9))
plt.subplots_adjust(hspace=0.5)
for n in range(batch_size):
    plt.subplot(num_plot_row,num_plot_column,n+1)
    plt.imshow(image_test[n])
    color = "green" if predicted_id[n] == label_id[n] else "red"
    plt.title(label_map[predicted_id[n]].title(), color=color)
    plt.axis('off')
_ = plt.suptitle("Model Predictions (green: correct, red: wrong)")

print("Accuracy of the tested batch of images:")
accuracy_score(label_id, predicted_id)

#Calculate evaluated loss and accuracy of model

score = model.evaluate(x=image_val_data, batch_size=image_val_data.ba
tch_size, steps=image_val_data.samples/image_val_data.batch_size)
print("Loss: ", score[0], "Accuracy: ", score[1])

from sklearn.metrics import precision_score
precision=precision_score(label_id, predicted_id, average='micro')
print(precision)

from sklearn.metrics import recall_score
recall = recall_score(label_id, predicted_id, average='micro')
print(recall)

from sklearn.metrics import f1_score
f1 = f1_score(label_id, predicted_id, zero_division=1,average='micro'
)
print(f1)

from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import cmocean

labels = label_map.items()

cm = confusion_matrix(label_id, predicted_id)
print(cm)

fig = plt.figure(dpi=150)
ax = fig.add_subplot(111)
cax = ax.matshow(cm, cmap = cmocean.cm.dense)

```

```

for (i, j), z in np.ndenumerate(cm):
    ax.text(j, i, '{}'.format(z), ha='center', va='center', color='red')

plt.title('Confusion matrix of Mobile Net')
fig.colorbar(cax)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()

"""## Save Trained Model"""

import time
t = time.time()

export_path = "OUTPUT_ROOT_DIR/model{}".format(int(t))
model.save(export_path, save_format='tf')

export_path

# Convert the model to TensorFlowLite
converter = tf.lite.TFLiteConverter.from_keras_model(model)
tflite_model = converter.convert()

# Save the TensorFlowLite model
with tf.io.gfile.GFile(OUTPUT_TFLITE_MODEL, 'wb') as f:
    f.write(tflite_model)

"""### Test Converted TensorFlowLite Model"""

# Load the TensorFlowLite model into interpreter
interpreter = tf.lite.Interpreter(model_path=OUTPUT_TFLITE_MODEL)
interpreter.allocate_tensors()

input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()

batch_size = image_test.shape[0]
predicted_id = np.zeros(batch_size)

#Predict a batch of images and get output ID (recognized labels)
for i, image in enumerate(np.split(image_test, batch_size)):
    interpreter.set_tensor(input_details[0]['index'], image)
    interpreter.invoke()
    output_data = interpreter.get_tensor(output_details[0]['index'])
    predicted_id[i] = np.argmax(output_data)

label_id = np.argmax(label_test, axis=-1)

```

```

#Plot the recognized images
num_plot_column = 5
num_plot_row = batch_size // num_plot_column + (batch_size % num_plot
_column > 0)

plt.figure(figsize=(10,9))
plt.subplots_adjust(hspace=0.5)
for n in range(batch_size):
    plt.subplot(num_plot_row,num_plot_column,n+1)
    plt.imshow(image_test[n])
    color = "green" if predicted_id[n] == label_id[n] else "red"
    plt.title(label_map[predicted_id[n]].title(), color=color)
    plt.axis('off')
_ = plt.suptitle("Model final predictions (green: correct, red: wrong
)")

print("Accuracy of the TensorFlowLite MobileNet Model:")
accuracy_score(label_id, predicted_id)

```

B.6 Desktop application

User specified traffic sign image input (Still Image File)

[Done By: Ng Jan Hui, Tan Jing Jie & Jacynth Tham Ming Quan]

```
#include <iostream>
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/features2d.hpp>
#include <opencv2/imgproc.hpp>
#include <opencv2/objdetect.hpp>
#include <opencv2/ml.hpp>
#include <random>
#include <vector>
#include <fstream>
#include <map>
#include <iomanip>

using namespace std;
using namespace cv;

// Double Segmentation Function
std::array<Mat, 2> doubleSegment(Mat srcI2, Mat singleMask, int i) {

    const int totalRow = srcI2.rows,
              totalCol = srcI2.cols,
              ratio = 3,
              kernelSize = 3,
              size = 21;

    Mat blurring,
        HSVImage,
        white, black, blue, mask,
        eroded,
        dilated,
        canvas,
        filledCanvasMask,
        finalMask;

    std::array<Mat, 2> segmented;

    Mat temp;

    RNG rng(12345);

    canvas.create(totalRow, totalCol, CV_8UC3);
    canvas = Scalar(0, 0, 0);
    canvas.copyTo(filledCanvasMask);

    std::vector<std::vector<Point>> contours, contours_arrg;

    /*-----*/
    // Prepare display enviroment

    bilateralFilter(srcI2, blurring, size, size * 3, size / 3);
```

```

// Convert the image to HSV Colour Space & Grayscale
cvtColor(blurring, HSVImage, COLOR_BGR2HSV);

// Declaring the Hue Ranges for the colours
Scalar whiteLow = Scalar(0, 0, 150);
Scalar whiteHigh = Scalar(180, 50, 255);
Scalar blackLow = Scalar(0, 0, 0);
Scalar blackHigh = Scalar(180, 50, 60);
Scalar blueLow = Scalar(104, 236, 0);
Scalar blueHigh = Scalar(110, 255, 255);

// Match for Red
inRange(HSVImage, whiteLow, whiteHigh, white);
inRange(HSVImage, blackLow, blackHigh, black);
inRange(HSVImage, blueLow, blueHigh, blue);

// Remove noise
Mat element(2, 2, CV_8U, cv::Scalar(1));
morphologyEx(white, white, cv::MORPH_CLOSE, element);
morphologyEx(black, black, cv::MORPH_CLOSE, element);
morphologyEx(blue, blue, cv::MORPH_CLOSE, element);

// Merge the mask
cvtColor(singleMask, singleMask, COLOR_BGR2GRAY);
mask = (white | black | blue) & singleMask;

erode(mask, eroded, element);
dilate(eroded, dilated, element);

cvtColor(dilated, finalMask, COLOR_GRAY2BGR);
//Segmented
segmented[0] = finalMask & srcI2;

segmented[0].copyTo(segmented[1]);
segmented[0].setTo(Scalar(255, 255, 255), finalMask);

return segmented;
}

// Print Image with size 300 by 300
void printImage300(std::vector<Mat> imageVector) {
    for (int i = 0; i < imageVector.size(); i++) {
        Mat temp;

        resize(imageVector[i], temp, Size(290, 290), INTER_LINEAR);
        imshow("Further Segment " + to_string(i + 1), temp);
        waitKey(0);
    }
}

// Get which is larger
bool compareContourAreas(std::vector<cv::Point> contour1, std::vector<cv::Point> contour2) {
    double i = fabs(contourArea(cv::Mat(contour1)));
    double j = fabs(contourArea(cv::Mat(contour2)));
    return (i < j);
}

// Customised function

```

```

void fillHole(const Mat srcBw, Mat& dstBw)
{
    Size m_Size = srcBw.size();
    Mat Temp = Mat::zeros(m_Size.height + 2, m_Size.width + 2, srcBw.type());
    srcBw.copyTo(Temp(Range(1, m_Size.height + 1), Range(1, m_Size.width + 1)));
    cv::floodFill(Temp, Point(0, 0), Scalar(255, 255, 255));
    Mat cutImg;
    Temp(Range(1, m_Size.height + 1), Range(1, m_Size.width + 1)).copyTo(cutImg);
    dstBw = srcBw | (~cutImg);
}

// Segmentation Function (Will Trigger Double Segmentation)
void segmentMain(String path, std::vector<Mat>& returnFurtherSegments, std::vector<std::string>& returnLabelNames, std::vector<Mat>& returnFirstSegments, std::vector<std::string>& returnPaths) {

    Mat srcI = imread(path);
    /*imshow(path, srcI);
    waitKey(0);*/

    //resize(srcI,srcI, Size(320,480),0,0,1);

    static int t1, t2, t3, t4;
    const int totalRow = srcI.rows,
              totalCol = srcI.cols,
              ratio = 3,
              kernelSize = 3,
              size = 21;

    Mat blurring,
        HSVImage,
        redMask1, redMask2, blueMask, yellowMask, mask,
        eroded,
        dilated,
        canvas,
        filledCanvasMask,
        segmented;

    Mat temp;

    RNG rng(12345);

    canvas.create(totalRow, totalCol, CV_8UC3);
    canvas = Scalar(0, 0, 0);
    canvas.copyTo(filledCanvasMask);

    std::vector<std::vector<Point> > contours, contours_arrg;

    bilateralFilter(srcI, blurring, size, size * 3, size / 3);

    // Convert the image to HSV Colour Space & Grayscale
    cvtColor(blurring, HSVImage, COLOR_BGR2HSV);
    cvtColor(HSVImage, temp, COLOR_HSV2BGR);

    // Declaring the Hue Ranges for the colours
    // Hue Ranges for Red
}

```

```

        // According to colour space, red is at the two left and right edge
        Scalar redLow1 = Scalar(159, 140, 160);
        Scalar redHigh1 = Scalar(180, 255, 255);

        Scalar redLow2 = Scalar(0, 50, 50);
        Scalar redHigh2 = Scalar(7, 255, 255);

        // Hue Ranges for Blue
        Scalar blueLow = Scalar(101, 150, 100);
        Scalar blueHigh = Scalar(130, 255, 255);

        // Hue Ranges for Yellow
        Scalar yellowLow = Scalar(16, 100, 140);
        Scalar yellowHigh = Scalar(36, 255, 255);

        // Match for Red
        inRange(HSVImage, redLow1, redHigh1, redMask1);
        inRange(HSVImage, redLow2, redHigh2, redMask2);

        // Match for Blue
        inRange(HSVImage, blueLow, blueHigh, blueMask);

        // Match for Yellow
        inRange(HSVImage, yellowLow, yellowHigh, yellowMask);

        // Remove noise
        Mat element(2, 2, CV_8U, cv::Scalar(1));
        morphologyEx(yellowMask, yellowMask, cv::MORPH_CLOSE, element);
        morphologyEx(redMask1, redMask1, cv::MORPH_CLOSE, element);
        morphologyEx(redMask2, redMask2, cv::MORPH_CLOSE, element);
        morphologyEx(blueMask, blueMask, cv::MORPH_CLOSE, element);

        // Merge the mask
        mask = redMask1 | redMask2 | blueMask | yellowMask;
        cvtColor(mask, temp, COLOR_GRAY2BGR);

        //erode
        erode(mask, eroded, element);
        cvtColor(eroded, temp, COLOR_GRAY2BGR);

        //dilate
        dilate(eroded, dilated, element);
        cvtColor(dilated, temp, COLOR_GRAY2BGR);

        // Find the contour
        findContours(dilated, contours, RETR_EXTERNAL, CHAIN_APPROX_NONE)
;

        for (int i = 0; i < contours.size(); i++) { // Just in case there
is more than one object in image
            for (;;) { // get random colors that are not too dim
                t1 = rng.uniform(0, 255); // blue
                t2 = rng.uniform(0, 255); // green
                t3 = rng.uniform(0, 255); // red
                t4 = t1 + t2 + t3;
                if (t4 > 255) break;
            }
        }
    }
}

```

```

        drawContours(canvas, contours, i, Scalar(t1, t2, t3)); // draw boundaryless

    }

    // Sort according to size
    sort(contours.begin(), contours.end(), compareContourAreas);

    // draw the largest contour if exist
    if (contours.size() > 0) {
        contours_arrg.push_back(contours[contours.size() - 1]);

        drawContours(filledCanvasMask, contours_arrg, 0, Scalar(255, 255, 255));

        // Get the label names from the file
        std::string label_class = path.substr(15, 2);
        returnLabelNames.push_back(label_class);
        returnPaths.push_back(path);
    }

    fillHole(filledCanvasMask, filledCanvasMask);

    //Segmented
    segmented = filledCanvasMask & srcI;

    //Bounding box
    std::vector<std::vector<Point>> contours_poly(contours_arrg.size());
    std::vector<Rect> boundRect(contours_arrg.size());
    for (int i = 0; i < contours_arrg.size(); i++)
    {

        approxPolyDP(contours_arrg[i], contours_poly[i], 3, true);
        boundRect[i] = boundingRect(contours_poly[i]);
    }

    Mat drawing = Mat::zeros(canvas.size(), CV_8UC3);
    for (int i = 0; i < contours_arrg.size(); i++)
    {
        Rect rect = boundRect[i];

        cv::rectangle(srcI, boundRect[i].tl(), boundRect[i].br(), Scalar(255, 255, 0), 2, 0, 0);
    }

    // Crop and save
    for (int i = 0; i < boundRect.size(); i++) {
        // prepare only single mask
        Mat singleMask;
        singleMask.create(totalRow, totalCol, CV_8UC3);
        singleMask = Scalar(0, 0, 0);

        drawContours(singleMask, contours_arrg, i, Scalar(255, 255, 0));
        fillHole(singleMask, singleMask);
    }
}

```

```

        // get the single traffic sign with rectangle
        Mat single = segmented & singleMask;
        returnFirstSegments.push_back(single);

        single(Rect(boundRect[i].tl().x, boundRect[i].tl().y, boundRect[i]
        ].br().x - boundRect[i].tl().x, boundRect[i].br().y - boundRect[i].tl().y
        )).copyTo(single);

        // Do double segment

        singleMask(Rect(boundRect[i].tl().x, boundRect[i].tl().y, boundRe
        ct[i].br().x - boundRect[i].tl().x, boundRect[i].br().y - boundRect[i].tl()
        (.y)).copyTo(singleMask);

        std::array<Mat, 2> seg = doubleSegment(single, singleMask, i);

        returnFurtherSegments.push_back(seg[0]);
    }
}

// Feature Extraction Function
void extractFeaturesFromTrafficSign(std::vector<Mat>& furtherSegments, st
d::vector<Mat>& initialSegments, std::vector<std::vector<double>>& retHuM
omentfeature, std::vector<std::vector<float>>& retHogFeature) {

    // Calculate Hu Moments from Further Segments
    for (int i = 0; i < furtherSegments.size(); i++) {
        Mat temp;

        cvtColor(furtherSegments[i], temp, COLOR_BGR2GRAY);

        Moments moments = cv::moments(temp, true);
        std::vector<double> huMoments;
        std::vector<double> logged_huMoments;
        bool foundInf = false;
        HuMoments(moments, huMoments);

        // Push the log transform vectors
        for (int i = 0; i < 7; i++) {

            /*if (isinf(huMoments[i]) || huMoments[i] == 0) {
                foundInf = true;
            }*/
            // logged_huMoments.push_back(huMoments[i]);
            logged_huMoments.push_back(-
1 * copysign(1.0, huMoments[i]) * log10(abs(huMoments[i])));
        }

        /*if (!foundInf)*/
        retHuMomentfeature.push_back(logged_huMoments);
    }

    HOGDescriptor hog;
    hog.winSize = Size(80, 80);
    hog.cellSize = Size(32, 32);
    hog.blockSize = Size(32, 32);
    hog.blockStride = Size(8, 8);
    vector<float> descriptors;
}

```

```

        for (size_t i = 0; i < initialSegments.size(); i++) {
            Mat temp;

            resize(initialSegments[i], temp, Size(80, 80), INTER_LINEAR);
            hog.compute(temp, descriptors);
            retHogFeature.push_back(descriptors);
        }

    }

// Write to CSV
void generateFeatureMatrix(std::vector<std::vector<double>>& HuMomentfeature,
                           std::vector<std::vector<float>>& HogFeature, std::vector<std::string>& labelVector, Mat& retMat, Mat& retLabels) {

    // Label Encoding
    /*
     * 017 : Horn -> 1
     * 028 : Car -> 2
     * 031 : Uturn -> 3
     * 037 : School -> 4
     * 054 : NoEntry -> 5
     * 055 : Stop -> 6
     */
    // DEFINING A MAP TO STORE ALL LABEL CLASSES
    std::map<string, int> label_name = {
        {"17", 1},
        {"28", 2},
        {"31", 3},
        {"37", 4},
        {"54", 5},
        {"55", 6},
    };

    // Create a Mat
    Mat featureMatrix(Size(HuMomentfeature[0].size() + HogFeature[0].size(),
                           labelVector.size()), CV_32F);
    retMat = featureMatrix;
    Mat labels(Size(1, labelVector.size()), CV_32F);
    retLabels = labels;
    int i, j, k;

    for (i = 0; i < featureMatrix.rows; i++) {

        for (j = 0; j < HuMomentfeature[i].size(); j++) {

            featureMatrix.at<float>(i, j) = HuMomentfeature[i][j];
        }

        for (k = 0; k < HogFeature[i].size(); k++, j++) {

            featureMatrix.at<float>(i, j) = HogFeature[i][k];
        }

        int val = 7;
        auto it = label_name.find(labelVector[i]);
        if (it != label_name.end())
    }
}

```

```

        val = it->second;

        labels.at<float>(i) = val;
    }

int main() {

    cv::setNumThreads(0);
    system("cls");
    /*STEP 1 -> SEGMENTATION*/

    /*PREPROCESSING - Variable Declaration*/
    std::vector<Mat> furtherSegments;
    std::vector<Mat> firstSegments;
    std::vector<std::string> allLabels;
    std::vector<std::string> inputNames;
    std::vector<std::string> userInputs;

    // Reading the input paths file (user specified)
    ifstream inputPathsFile("Input Dataset/testInputs.txt");

    // Start Segmentation Process
    std::cout << "Begin Segmentation" << endl << endl;

    int ct = 0;

    if (inputPathsFile.is_open()) {
        std::string name_temp;
        while (getline(inputPathsFile, name_temp)) {
            userInputs.push_back(name_temp);

            segmentMain(name_temp, furtherSegments, allLabels, firstSegments,
            inputNames); // returns -
            > 1) Internal Segment 2) Encoded label 3) Initial Segment 4) Names of successfully segmented
        }
    }

    for (int i = 0; i < userInputs.size(); i++) {
        string statusMsg;

        if (std::count(inputNames.begin(), inputNames.end(), userInputs[i])) {
            statusMsg = "Segmented Successfully";
        }
        else {

            statusMsg = "Segmentation Failed (Bad Lighting Condition)";
        }

        cout << left << setw(3) << ++ct;

        cout << left << setw(35) << setfill(' ') << userInputs[i];

        cout << left << setw(35) << setfill(' ') << statusMsg << endl;
    }
}

```

```

/*STEP 2 - Feature Extraction*/

// Start Feature Extraction
std::cout << "\nBegin Feature Extraction";

// Feature Vector
std::vector<std::vector<double>> huMomentsFeatures;
std::vector<std::vector<float>> HoGFeatures;

// Call feature extraction function
extractFeaturesFromTrafficSign(furtherSegments, firstSegments, hu
MomentsFeatures, HoGFeatures);

std::cout << "\t....Done" << endl << endl;

/*STEP 3 - Writing Feature to Matrix*/

std::cout << "Generating feature matrix";

Mat features;
Mat labels;

generateFeatureMatrix(huMomentsFeatures, HoGFeatures, allLabels,
features, labels);

std::cout << "\t....Done" << endl << endl;

/*STEP 5 - Predicting using on pretrained model*/

std::cout << "Begin prediction";

// Load Models
Ptr<ml::SVM> pre_svm = Algorithm::load<ml::SVM>("Models/NoStd/T6G
1_SVM.xml");

Ptr<ml::RTrees> pre_rtrees = Algorithm::load<ml::RTrees>("Models/
NoStd/T6G1_rtree.xml");

// Predict (With Confidence)
// SVM
Mat svm_res;
pre_svm->predict(features, svm_res);

// Random Forest
Mat rtree_res;
Mat rtree_votes;
pre_rtrees->predict(features, rtree_res);
pre_rtrees->getVotes(features, rtree_votes, 0);

std::cout << "\t....Done" << endl << endl;

/*STEP 6 - Analysis*/

std::string signNames[] = { "No horn", "Car", "Uturn", "People cr
ossing", "No stopping", "No entry", "Invalid Type" };

```

```

        cout << "\n======" << endl;
===== " << endl;
        cout << "Predicted signs based on user input" << endl;
        cout << "======" << endl << endl;
        cout << left << setw(34) << setfill(' ') << "Sign Input Name";
        cout << left << setw(18) << setfill(' ') << "Actual Label";
        cout << left << setw(18) << setfill(' ') << "SVM Prediction";
        cout << left << setw(20) << setfill(' ') << "Random Forest Prediction (Confidence)" << endl << endl;

        for (int i = 0; i < inputNames.size(); i++) {
            string curLabel;
            string svmPredLabel;
            string rtreePredLabel;
            double max = 0;

            Mat temp = imread(inputNames[i]);
            resize(temp, temp, Size(150, 150), cv::INTER_LINEAR);
            imshow("Current Image", temp);

            // Process Confidence
            Mat signProbasRow = rtree_votes.row(i + 1);
            minMaxLoc(signProbasRow, NULL, &max, NULL, NULL);

            // Rounding
            std::stringstream stream;
            stream << std::fixed << std::setprecision(2) << max;
            std::string probaVal = stream.str();

            // Prepping strings to be read
            curLabel = signNames[int(labels.at<float>(i)) - 1];
            svmPredLabel = signNames[int(svm_res.at<float>(i)) - 1];

            rtreePredLabel = signNames[int(rtree_res.at<float>(i)) - 1] + " (" +
" + probaVal + "%)";

            // Print results

            cout << left << setw(34) << setfill(' ') << inputNames[i] + ":";
            cout << left << setw(18) << setfill(' ') << curLabel;

            cout << left << setw(18) << setfill(' ') << svmPredLabel;

            cout << setprecision(2) << left << setw(20) << setfill(' ') << rtreePredLabel;
            cout << endl << endl;

            waitKey(0);
        }

        return 0;
    }
}

```

Webcam-based Implementation (Webcam Input)

[Done By: Por Teong Dean]

```
#include <iostream>
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/features2d.hpp>
#include <opencv2/imgproc.hpp>
#include <opencv2/objdetect.hpp>
#include <opencv2/ml.hpp>
#include <random>
#include <vector>
#include <fstream>
#include <map>
#include <iomanip>

using namespace std;
using namespace cv;

// Double Segmentation Function
std::array<Mat, 2> doubleSegment(Mat srcI2, Mat singleMask, int i) {

    const int totalRow = srcI2.rows,
              totalCol = srcI2.cols,
              ratio = 3,
              kernelSize = 3,
              size = 21;

    Mat blurring,
        HSVImage,
        white, black, blue, mask,
        eroded,
        dilated,
        canvas,
        filledCanvasMask,
        finalMask;

    std::array<Mat, 2> segmented;

    Mat temp;

    RNG rng(12345);

    canvas.create(totalRow, totalCol, CV_8UC3);
    canvas = Scalar(0, 0, 0);
    canvas.copyTo(filledCanvasMask);
```

```

std::vector<std::vector<Point>> contours, contours_arrg;

/*-----
-----*/
// Prepare display enviroment

bilateralFilter(srcI2, blurring, size, size * 3, size / 3);

// Convert the image to HSV Colour Space & Grayscale
cvtColor(blurring, HSVImage, COLOR_BGR2HSV);

// Declaring the Hue Ranges for the colours
Scalar whiteLow = Scalar(0, 0, 150);
Scalar whiteHigh = Scalar(180, 50, 255);
Scalar blackLow = Scalar(0, 0, 0);
Scalar blackHigh = Scalar(180, 50, 60);
Scalar blueLow = Scalar(104, 236, 0);
Scalar blueHigh = Scalar(110, 255, 255);

// Match for Red
inRange(HSVImage, whiteLow, whiteHigh, white);
inRange(HSVImage, blackLow, blackHigh, black);
inRange(HSVImage, blueLow, blueHigh, blue);

// Remove noise
Mat element(2, 2, CV_8U, cv::Scalar(1));
morphologyEx(white, white, cv::MORPH_CLOSE, element);
morphologyEx(black, black, cv::MORPH_CLOSE, element);
morphologyEx(blue, blue, cv::MORPH_CLOSE, element);

// Merge the mask
cvtColor(singleMask, singleMask, COLOR_BGR2GRAY);
mask = (white | black | blue) & singleMask;

erode(mask, eroded, element);
dilate(eroded, dilated, element);

cvtColor(dilated, finalMask, COLOR_GRAY2BGR);
//Segmented
segmented[0] = finalMask & srcI2;

segmented[0].copyTo(segmented[1]);
segmented[0].setTo(Scalar(255, 255, 255), finalMask);

return segmented;
}

// Print Image with size 300 by 300

```

```

void printImage300(std::vector<Mat> imageVector) {
    for (int i = 0; i < imageVector.size(); i++) {
        Mat temp;
        resize(imageVector[i], temp, Size(290, 290), INTER_LINEAR);
        imshow("Further Segment " + to_string(i + 1), temp);
        waitKey(0);
    }
}

// Get which is larger
bool compareContourAreas(std::vector<cv::Point> contour1, std::vector<cv::Point> contour2) {
    double i = fabs(contourArea(cv::Mat(contour1)));
    double j = fabs(contourArea(cv::Mat(contour2)));
    return (i < j);
}

// Customised function
void fillHole(const Mat srcBw, Mat& dstBw)
{
    Size m_Size = srcBw.size();
    Mat Temp = Mat::zeros(m_Size.height + 2, m_Size.width + 2, srcBw.type());
    srcBw.copyTo(Temp(Range(1, m_Size.height + 1), Range(1, m_Size.width + 1)));
    cv::floodFill(Temp, Point(0, 0), Scalar(255, 255, 255));
    Mat cutImg;
    Temp(Range(1, m_Size.height + 1), Range(1, m_Size.width + 1)).copyTo(cutImg);
    dstBw = srcBw | (~cutImg);
}

// Segmentation Function (Will Trigger Double Segmentation)
void segmentMain(Mat srcI, std::vector<Mat>& returnFurtherSegments, std::vector<Mat>& returnFirstSegments) {

    /*imshow(path, srcI);
    waitKey(0);*/

    //resize(srcI,srcI, Size(320,480),0,0,1);

    static int      t1, t2, t3, t4;
    const int       totalRow = srcI.rows,
                    totalCol = srcI.cols,
                    ratio = 3,
                    kernelSize = 3,
                    size = 21;
}

```

```

    Mat      blurring,
    HSVImage,
    redMask1, redMask2, blueMask, yellowMask, mask,
    eroded,
    dilated,
    canvas,
    filledCanvasMask,
    segmented;

    Mat temp;

    RNG          rng(12345);

    canvas.create(totalRow, totalCol, CV_8UC3);
    canvas = Scalar(0, 0, 0);
    canvas.copyTo(filledCanvasMask);

    std::vector<std::vector<Point> >    contours, contours_arrg;

    bilateralFilter(srcI, blurring, size, size * 3, size / 3);

    // Convert the image to HSV Colour Space & Grayscale
    cvtColor(blurring, HSVImage, COLOR_BGR2HSV);
    cvtColor(HSVImage, temp, COLOR_HSV2BGR);

    // Declaring the Hue Ranges for the colours
    // Hue Ranges for Red
    // According to colour space, red is at the two left and right edges
    Scalar redLow1 = Scalar(159, 140, 160);
    Scalar redHigh1 = Scalar(180, 255, 255);

    Scalar redLow2 = Scalar(0, 50, 50);
    Scalar redHigh2 = Scalar(7, 255, 255);

    // Hue Ranges for Blue
    Scalar blueLow = Scalar(101, 150, 100);
    Scalar blueHigh = Scalar(130, 255, 255);

    // Hue Ranges for Yellow
    Scalar yellowLow = Scalar(16, 100, 140);
    Scalar yellowHigh = Scalar(36, 255, 255);

    // Match for Red
    inRange(HSVImage, redLow1, redHigh1, redMask1);
    inRange(HSVImage, redLow2, redHigh2, redMask2);

    // Match for Blue

```

```

inRange(HSVImage, blueLow, blueHigh, blueMask);

// Match for Yellow
inRange(HSVImage, yellowLow, yellowHigh, yellowMask);

// Remove noise
Mat element(2, 2, CV_8U, cv::Scalar(1));
morphologyEx(yellowMask, yellowMask, cv::MORPH_CLOSE, element);
morphologyEx(redMask1, redMask1, cv::MORPH_CLOSE, element);
morphologyEx(redMask2, redMask2, cv::MORPH_CLOSE, element);
morphologyEx(blueMask, blueMask, cv::MORPH_CLOSE, element);

// Merge the mask
mask = redMask1 | redMask2 | blueMask | yellowMask;
cvtColor(mask, temp, COLOR_GRAY2BGR);

//erode
erode(mask, eroded, element);
cvtColor(eroded, temp, COLOR_GRAY2BGR);

//dilate
dilate(eroded, dilated, element);
cvtColor(dilated, temp, COLOR_GRAY2BGR);

// Find the contour
findContours(dilated, contours, RETR_EXTERNAL, CHAIN_APPROX_NONE)
;

for (int i = 0; i < contours.size(); i++) { // Just in case there
is more than one object in image
    for (;;) { // get random colors that are not too dim
        t1 = rng.uniform(0, 255); // blue
        t2 = rng.uniform(0, 255); // green
        t3 = rng.uniform(0, 255); // red
        t4 = t1 + t2 + t3;
        if (t4 > 255) break;
    }

    drawContours(canvas, contours, i, Scalar(t1, t2, t3)); // dra
w boundaries
}

// Sort according to size
sort(contours.begin(), contours.end(), compareContourAreas);

// draw the largest contour if exist
if (contours.size() > 0) {

```

```

        contours_arrg.push_back(contours[contours.size() - 1]);
        drawContours(filledCanvasMask, contours_arrg, 0, Scalar(255,
255, 255));
    }

    fillHole(filledCanvasMask, filledCanvasMask);

    //Segmented
    segmented = filledCanvasMask & srcI;

    //Bounding box
    std::vector<std::vector<Point>> contours_poly(contours_arrg.size());
    std::vector<Rect> boundRect(contours_arrg.size());
    for (int i = 0; i < contours_arrg.size(); i++)
    {
        approxPolyDP(contours_arrg[i], contours_poly[i], 3, true);
        boundRect[i] = boundingRect(contours_poly[i]);
    }

    Mat drawing = Mat::zeros(canvas.size(), CV_8UC3);
    for (int i = 0; i < contours_arrg.size(); i++)
    {
        Rect rect = boundRect[i];
        cv::rectangle(srcI, boundRect[i].tl(), boundRect[i].br(), Scalar(255, 255, 0), 2, 0, 0);
    }

    // Crop and save
    for (int i = 0; i < boundRect.size(); i++) {
        // prepare only single mask
        Mat singleMask;
        singleMask.create(totalRow, totalCol, CV_8UC3);
        singleMask = Scalar(0, 0, 0);
        drawContours(singleMask, contours_arrg, i, Scalar(255, 255, 0));
        fillHole(singleMask, singleMask);

        // get the single traffic sign with rectangle
        Mat single = segmented & singleMask;
        returnFirstSegments.push_back(single);
        single(Rect(boundRect[i].tl().x, boundRect[i].tl().y, boundRect[i].br().x - boundRect[i].tl().x, boundRect[i].br().y - boundRect[i].tl().y)).copyTo(single);

        // Do double segment
    }
}

```

```

        singleMask(Rect(boundRect[i].tl().x, boundRect[i].tl().y, boundRect[i].br().x - boundRect[i].tl().x, boundRect[i].br().y - boundRect[i].tl().y)).copyTo(singleMask);
        std::array<Mat, 2> seg = doubleSegment(single, singleMask, i)
    ;

        returnFurtherSegments.push_back(seg[0]);
    }
}

// Feature Extraction Function
void extractFeaturesFromTrafficSign(std::vector<Mat>& furtherSegments,
, std::vector<Mat>& initialSegments, std::vector<std::vector<double>>& retHuMomentfeature, std::vector<std::vector<float>>& retHogFeature)
{

    // Calculate Hu Moments from Further Segments
    for (int i = 0; i < furtherSegments.size(); i++) {
        Mat temp;

        cvtColor(furtherSegments[i], temp, COLOR_BGR2GRAY);

        Moments moments = cv::moments(temp, true);
        std::vector<double> huMoments;
        std::vector<double> logged_huMoments;
        bool foundInf = false;
        HuMoments(moments, huMoments);

        // Push the log transform vectors
        for (int i = 0; i < 7; i++) {
            /*if (isinf(huMoments[i]) || huMoments[i] == 0) {
                foundInf = true;
            }*/
            // logged_huMoments.push_back(huMoments[i]);
            logged_huMoments.push_back(-
1 * copysign(1.0, huMoments[i]) * log10(abs(huMoments[i])));
        }

        /*if (!foundInf)*/
        retHuMomentfeature.push_back(logged_huMoments);
    }

    HOGDescriptor hog;
    hog.winSize = Size(80, 80);
    hog.cellSize = Size(32, 32);
    hog.blockSize = Size(32, 32);
    hog.blockStride = Size(8, 8);
    vector<float> descriptors;
}

```

```

        for (size_t i = 0; i < initialSegments.size(); i++) {
            Mat temp;
            resize(initialSegments[i], temp, Size(80, 80), INTER_LINEAR);
            hog.compute(temp, descriptors);
            retHogFeature.push_back(descriptors);
        }

    }

// Write to CSV
void generateFeatureMatrix(std::vector<std::vector<double>>& HuMoment
feature, std::vector<std::vector<float>>& HogFeature, Mat& retMat, st
d::vector<Mat> segment) {

    // Label Encoding
    /*
    017 : Horn -> 1
    028 : Car -> 2
    031 : Uturn -> 3
    037 : School -> 4
    054 : NoEntry -> 5
    055 : Stop -> 6
    */
}

// DEFINING A MAP TO STORE ALL LABEL CLASSES
std::map<string, int> label_name = {
    {"17", 1},
    {"28", 2},
    {"31", 3},
    {"37", 4},
    {"54", 5},
    {"55", 6},
};

// Create a Mat
Mat featureMatrix(Size(HuMomentfeature[0].size() + HogFeature[0].
size(), segment.size()), CV_32F);
retMat = featureMatrix;
int i, j, k;

for (i = 0; i < featureMatrix.rows; i++) {

    for (j = 0; j < HuMomentfeature[i].size(); j++) {
        featureMatrix.at<float>(i, j) = HuMomentfeature[i][j];
    }

    for (k = 0; k < HogFeature[i].size(); k++, j++) {
}

```

```

        featureMatrix.at<float>(i, j) = HogFeature[i][k];
    }
}

int main(int argc, char** argv)
{
    VideoCapture cap;
    if (!cap.open(0))
        return 0;

    int frame = 1;
    system("cls");
    cout << "Classifiers" << "\t" << "SVM" << "\t" << "RF" << endl;
    for (;;)
    {
        Mat srcI;
        cap >> srcI;
        if (srcI.empty()) break; // end of video stream

        std::vector<Mat> furtherSegments;
        std::vector<Mat> firstSegments;
        segmentMain(srcI, furtherSegments, firstSegments);

        std::vector<std::vector<double>> huMomentsFeatures;
        std::vector<std::vector<float>> HoGFeatures;

        // Call feature extraction function
        extractFeaturesFromTrafficSign(furtherSegments, firstSegments,
            huMomentsFeatures, HoGFeatures);

        Mat features;
        Mat labels;

        generateFeatureMatrix(huMomentsFeatures, HoGFeatures, feature
s, furtherSegments);

        Ptr<ml::SVM> classifier = Algorithm::load<ml::SVM>("Models/T6
G1_SVM.xml");
        Mat testResults;
        classifier->predict(features, testResults);
        Ptr<ml::RTrees> pre_rtrees = Algorithm::load<ml::RTrees>("Mod
els/T6G1_rtree.xml");
        Mat testResults2;
        pre_rtrees->predict(features, testResults2);
        if (frame == 1) {
            system("cls");
        }
    }
}

```

```

        cout << "Classifiers" << "\t" << "SVM" << "\t" << "RF" <<
endl;
    }
    std::string signNames[] = { "No horn", "Car", "Uturn", "People crossing", "No stopping", "No entry" };
    if (furtherSegments.size() > 1) {
        for (int i = 0; i <= furtherSegments.size(); i++) {
            cout << "Prediction" << ":" " \t" << signNames[int(testResults.at<float>(i)) - 1] << "\t"
                << signNames[int(testResults2.at<float>(i)) - 1]
<< endl;
        }
    }
    else {
        cout << "Prediction" << ":" " \t" << signNames[int(testResults.at<float>(0)) - 1] << "\t"
            << signNames[int(testResults2.at<float>(0)) - 1] << endl << endl;
    }
    imshow("Traffic Sign Detection Using Web Camera", srcI);

    if (waitKey(10) == 27) break; // stop capturing by pressing ESC
    frame++;
}
return 0;
}

```

Video-based Implementation (Video file)

[Done By: Tan Wei Mun]

```

#include <iostream>
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/features2d.hpp>
#include <opencv2/imgproc.hpp>
#include <opencv2/objdetect.hpp>
#include <opencv2/ml.hpp>
#include <random>
#include <vector>
#include <fstream>
#include <map>
#include <iomanip>

using namespace std;

```

```

using namespace cv;

// Double Segmentation Function
std::array<Mat, 2> doubleSegment(Mat srcI2, Mat singleMask, int i) {

    const int      totalRow = srcI2.rows,
                   totalCol = srcI2.cols,
                   ratio = 3,
                   kernelSize = 3,
                   size = 21;

    Mat blurring,
        HSVImage,
        white, black, blue, mask,
        eroded,
        dilated,
        canvas,
        filledCanvasMask,
        finalMask;

    std::array<Mat, 2> segmented;

    Mat temp;

    RNG          rng(12345);

    canvas.create(totalRow, totalCol, CV_8UC3);
    canvas = Scalar(0, 0, 0);
    canvas.copyTo(filledCanvasMask);

    std::vector<std::vector<Point>> contours, contours_arrg;

    /*-----*/
    // Prepare display enviroment

    bilateralFilter(srcI2, blurring, size, size * 3, size / 3);

    // Convert the image to HSV Colour Space & Grayscale
    cvtColor(blurring, HSVImage, COLOR_BGR2HSV);

    // Declaring the Hue Ranges for the colours
    Scalar whiteLow = Scalar(0, 0, 150);
    Scalar whiteHigh = Scalar(180, 50, 255);
    Scalar blackLow = Scalar(0, 0, 0);
    Scalar blackHigh = Scalar(180, 50, 60);
    Scalar blueLow = Scalar(104, 236, 0);
    Scalar blueHigh = Scalar(110, 255, 255);

```

```

// Match for Red
inRange(HSVImage, whiteLow, whiteHigh, white);
inRange(HSVImage, blackLow, blackHigh, black);
inRange(HSVImage, blueLow, blueHigh, blue);

// Remove noise
Mat element(2, 2, CV_8U, cv::Scalar(1));
morphologyEx(white, white, cv::MORPH_CLOSE, element);
morphologyEx(black, black, cv::MORPH_CLOSE, element);
morphologyEx(blue, blue, cv::MORPH_CLOSE, element);

// Merge the mask
cvtColor(singleMask, singleMask, COLOR_BGR2GRAY);
mask = (white | black | blue) & singleMask;

erode(mask, eroded, element);
dilate(eroded, dilated, element);

cvtColor(dilated, finalMask, COLOR_GRAY2BGR);
//Segmented
segmented[0] = finalMask & srcI2;

segmented[0].copyTo(segmented[1]);
segmented[0].setTo(Scalar(255, 255, 255), finalMask);

return segmented;
}

// Print Image with size 300 by 300
void printImage300(std::vector<Mat> imageVector) {
    for (int i = 0; i < imageVector.size(); i++) {
        Mat temp;
        resize(imageVector[i], temp, Size(290, 290), INTER_LINEAR);
        imshow("Further Segment " + to_string(i + 1), temp);
        waitKey(0);
    }
}

// Get which is larger
bool compareContourAreas(std::vector<cv::Point> contour1, std::vector<cv::Point> contour2) {
    double i = fabs(contourArea(cv::Mat(contour1)));
    double j = fabs(contourArea(cv::Mat(contour2)));
    return (i < j);
}

// Customised function

```

```

void fillHole(const Mat srcBw, Mat& dstBw)
{
    Size m_Size = srcBw.size();
    Mat Temp = Mat::zeros(m_Size.height + 2, m_Size.width + 2, srcBw.type());
    srcBw.copyTo(Temp(Range(1, m_Size.height + 1), Range(1, m_Size.width + 1)));
    cv::floodFill(Temp, Point(0, 0), Scalar(255, 255, 255));
    Mat cutImg;
    Temp(Range(1, m_Size.height + 1), Range(1, m_Size.width + 1)).copyTo(cutImg);
    dstBw = srcBw | (~cutImg);
}

// Segmentation Function (Will Trigger Double Segmentation)
void segmentMain(Mat srcI, std::vector<Mat>& returnFurtherSegments, std::vector<Mat>& returnFirstSegments) {

    /*imshow(path, srcI);
    waitKey(0);*/

    //resize(srcI,srcI, Size(320,480),0,0,1);

    static int      t1, t2, t3, t4;
    const int       totalRow = srcI.rows,
                    totalCol = srcI.cols,
                    ratio = 3,
                    kernelSize = 3,
                    size = 21;

    Mat      blurring,
            HSVImage,
            redMask1, redMask2, blueMask, yellowMask, mask,
            eroded,
            dilated,
            canvas,
            filledCanvasMask,
            segmented;

    Mat temp;

    RNG          rng(12345);

    canvas.create(totalRow, totalCol, CV_8UC3);
    canvas = Scalar(0, 0, 0);
    canvas.copyTo(filledCanvasMask);

    std::vector<std::vector<Point> > contours, contours_arrg;
}

```

```

bilateralFilter(srcI, blurring, size, size * 3, size / 3);

// Convert the image to HSV Colour Space & Grayscale
cvtColor(blurring, HSVImage, COLOR_BGR2HSV);
cvtColor(HSVImage, temp, COLOR_HSV2BGR);

// Declaring the Hue Ranges for the colours
// Hue Ranges for Red
// According to colour space, red is at the two left and right edges
Scalar redLow1 = Scalar(159, 140, 160);
Scalar redHigh1 = Scalar(180, 255, 255);

Scalar redLow2 = Scalar(0, 50, 50);
Scalar redHigh2 = Scalar(7, 255, 255);

// Hue Ranges for Blue
Scalar blueLow = Scalar(101, 150, 100);
Scalar blueHigh = Scalar(130, 255, 255);

// Hue Ranges for Yellow
Scalar yellowLow = Scalar(16, 100, 140);
Scalar yellowHigh = Scalar(36, 255, 255);

// Match for Red
inRange(HSVImage, redLow1, redHigh1, redMask1);
inRange(HSVImage, redLow2, redHigh2, redMask2);

// Match for Blue
inRange(HSVImage, blueLow, blueHigh, blueMask);

// Match for Yellow
inRange(HSVImage, yellowLow, yellowHigh, yellowMask);

// Remove noise
Mat element(2, 2, CV_8U, cv::Scalar(1));
morphologyEx(yellowMask, yellowMask, cv::MORPH_CLOSE, element);
morphologyEx(redMask1, redMask1, cv::MORPH_CLOSE, element);
morphologyEx(redMask2, redMask2, cv::MORPH_CLOSE, element);
morphologyEx(blueMask, blueMask, cv::MORPH_CLOSE, element);

// Merge the mask
mask = redMask1 | redMask2 | blueMask | yellowMask;
cvtColor(mask, temp, COLOR_GRAY2BGR);

// erode
erode(mask, eroded, element);

```

```

cvtColor(eroded, temp, COLOR_GRAY2BGR);

//dilate
dilate(eroded, dilated, element);
cvtColor(dilated, temp, COLOR_GRAY2BGR);

// Find the contour
findContours(dilated, contours, RETR_EXTERNAL, CHAIN_APPROX_NONE)
;

for (int i = 0; i < contours.size(); i++) { // Just in case there
is more than one object in image
    for (;;) { // get random colors that are not too dim
        t1 = rng.uniform(0, 255); // blue
        t2 = rng.uniform(0, 255); // green
        t3 = rng.uniform(0, 255); // red
        t4 = t1 + t2 + t3;
        if (t4 > 255) break;
    }

    drawContours(canvas, contours, i, Scalar(t1, t2, t3)); // draw boundaries
}

// Sort according to size
sort(contours.begin(), contours.end(), compareContourAreas);

// draw the largest contour if exist
if (contours.size() > 0) {
    contours_arrg.push_back(contours[contours.size() - 1]);
    drawContours(filledCanvasMask, contours_arrg, 0, Scalar(255,
255, 255));
}

fillHole(filledCanvasMask, filledCanvasMask);

//Segmented
segmented = filledCanvasMask & srcI;

//Bounding box
std::vector<std::vector<Point>> contours_poly(contours_arrg.size());
std::vector<Rect> boundRect(contours_arrg.size());
for (int i = 0; i < contours_arrg.size(); i++)
{
    approxPolyDP(contours_arrg[i], contours_poly[i], 3, true);
}

```

```

        boundRect[i] = boundingRect(contours_poly[i]);
    }

    Mat drawing = Mat::zeros(canvas.size(), CV_8UC3);
    for (int i = 0; i < contours_arrg.size(); i++)
    {
        Rect rect = boundRect[i];
        cv::rectangle(srcI, boundRect[i].tl(), boundRect[i].br(), Scalar(255, 255, 0), 2, 0, 0);
    }

    // Crop and save
    for (int i = 0; i < boundRect.size(); i++) {
        // prepare only single mask
        Mat singleMask;
        singleMask.create(totalRow, totalCol, CV_8UC3);
        singleMask = Scalar(0, 0, 0);
        drawContours(singleMask, contours_arrg, i, Scalar(255, 255, 0));
        fillHole(singleMask, singleMask);

        // get the single traffic sign with rectangle
        Mat single = segmented & singleMask;
        returnFirstSegments.push_back(single);
        single(Rect(boundRect[i].tl().x, boundRect[i].tl().y, boundRect[i].br().x - boundRect[i].tl().x, boundRect[i].br().y - boundRect[i].tl().y)).copyTo(single);

        // Do double segment
        singleMask(Rect(boundRect[i].tl().x, boundRect[i].tl().y, boundRect[i].br().x - boundRect[i].tl().x, boundRect[i].br().y - boundRect[i].tl().y)).copyTo(singleMask);
        std::array<Mat, 2> seg = doubleSegment(single, singleMask, i);
    }

    returnFurtherSegments.push_back(seg[0]);
}
}

// Feature Extraction Function
void extractFeaturesFromTrafficSign(std::vector<Mat>& furtherSegments, std::vector<Mat>& initialSegments, std::vector<std::vector<double>>& retHuMomentfeature, std::vector<std::vector<float>>& retHogFeature)
{
    // Calculate Hu Moments from Further Segments
    for (int i = 0; i < furtherSegments.size(); i++) {

```

```

    Mat temp;

    cvtColor(furtherSegments[i], temp, COLOR_BGR2GRAY);

    Moments moments = cv::moments(temp, true);
    std::vector<double> huMoments;
    std::vector<double> logged_huMoments;
    bool foundInf = false;
    HuMoments(moments, huMoments);

    // Push the log transform vectors
    for (int i = 0; i < 7; i++) {
        /*if (isinf(huMoments[i]) || huMoments[i] == 0) {
            foundInf = true;
        }*/
        // logged_huMoments.push_back(huMoments[i]);
        logged_huMoments.push_back(-
1 * copysign(1.0, huMoments[i]) * log10(abs(huMoments[i])));
    }

    /*if (!foundInf)*/
    retHuMomentfeature.push_back(logged_huMoments);
}

HOGDescriptor hog;
hog.winSize = Size(80, 80);
hog.cellSize = Size(32, 32);
hog.blockSize = Size(32, 32);
hog.blockStride = Size(8, 8);
vector<float> descriptors;

for (size_t i = 0; i < initialSegments.size(); i++) {
    Mat temp;
    resize(initialSegments[i], temp, Size(80, 80), INTER_LINEAR);
    hog.compute(temp, descriptors);
    retHogFeature.push_back(descriptors);
}

}

// Write to CSV
void generateFeatureMatrix(std::vector<std::vector<double>>& HuMoment
feature, std::vector<std::vector<float>>& HogFeature, Mat& retMat, st
d::vector<Mat> segment) {

    // Label Encoding
    /*
        017 : Horn -> 1

```

```

028 : Car -> 2
031 : Uturn -> 3
037 : School -> 4
054 : NoEntry -> 5
055 : Stop -> 6
*/
// DEFINING A MAP TO STORE ALL LABEL CLASSES
std::map<string, int> label_name = {
    {"17", 1},
    {"28", 2},
    {"31", 3},
    {"37", 4},
    {"54", 5},
    {"55", 6},
};

// Create a Mat
Mat featureMatrix(Size(HuMomentfeature[0].size() + HogFeature[0].
size(), segment.size()), CV_32F);
retMat = featureMatrix;
int i, j, k;

for (i = 0; i < featureMatrix.rows; i++) {

    for (j = 0; j < HuMomentfeature[i].size(); j++) {
        featureMatrix.at<float>(i, j) = HuMomentfeature[i][j];
    }

    for (k = 0; k < HogFeature[i].size(); k++, j++) {
        featureMatrix.at<float>(i, j) = HogFeature[i][k];
    }
}
int main(int argc, char** argv)
{
    // Reading the input paths file
    ifstream inputPathsFile("Input Dataset/inputSignNamesVideo.txt");

    if (inputPathsFile.is_open()) {
        std::string name_temp;
        while (getline(inputPathsFile, name_temp)) {
            cout << "Reading " << name_temp << endl << endl;
            system("Pause");
            VideoCapture cap(name_temp);
            int width = (int)cap.get(CAP_PROP_FRAME_WIDTH),
            // get video width & height
            height = (int)cap.get(CAP_PROP_FRAME_HEIGHT);
    }
}

```

```

        double          fps = cap.get(CAP_PROP_FPS);
        Size    S = Size(width / 3, height / 3);
        int frame = 1;
        system("cls");
        cout << "Classifiers" << "\t" << "SVM" << "\t\t" << "RF"
<< endl;
        while (1)
        {
            Mat srcI;
            cap >> srcI;
            if (srcI.empty()) {
                cout << "End of " << name_temp << endl << endl;
                break;
            } // end of video stream
            cv::resize(srcI, srcI, S);

            std::vector<Mat> furtherSegments;
            std::vector<Mat> firstSegments;
            segmentMain(srcI, furtherSegments, firstSegments);
            imshow("Traffic Sign Detection Using Video From File"
, srcI);
            if (furtherSegments.size() < 1) {
                cout << "No signs detected" << endl << endl;
                continue;
            }
            std::vector<std::vector<double>> huMomentsFeatures;
            std::vector<std::vector<float>> HoGFeatures;

            // Call feature extraction function
            extractFeaturesFromTrafficSign(furtherSegments, first
Segments, huMomentsFeatures, HoGFeatures);
            Mat features;
            Mat labels;

            generateFeatureMatrix(huMomentsFeatures, HoGFeatures,
features, furtherSegments);

            Ptr<ml::SVM> classifier = Algorithm::load<ml::SVM>("M
odels/T6G1_SVM.xml");
            Mat testResults;
            classifier->predict(features, testResults);
            Ptr<ml::RTrees> pre_rtrees = Algorithm::load<ml::RTre
es>("Models/T6G1_rtree.xml");

            // Random Forest
            Mat rtree_res;
            Mat rtree_votes;
            pre_rtrees->predict(features, rtree_res);

```

```

        pre_rtrees->getVotes(features, rtree_votes, 0);
        cout << "Classifiers" << "\t" << "SVM" << "\t\t" << "
RF (Confidence)" << endl;
        std::string signNames[] = { "No horn", "Car", "Uturn"
, "People crossing", "No stopping", "No entry" };
        for (int i = 0; i < furtherSegments.size(); i++) {
            double max = 0;
            Mat signProbasRow = rtree_votes.row(i + 1);
            minMaxLoc(signProbasRow, NULL, &max, NULL, NULL);

            // Rounding
            std::stringstream stream;
            stream << std::fixed << std::setprecision(2) << m
ax;
            std::string probaVal = stream.str();
            cout << "Prediction" << ":" " " "\t" << signNames[int(testResults.at<float>(i)) - 1] << "\t\t" << signNames[int(rtree_re
s.at<float>(i)) - 1] + " (" + probaVal + "%)" << endl << endl;
        }

        imshow("Traffic Sign Detection Using Video From File"
, srcI);
        //outputVideo << canvas;
        if (waitKey(10) == 27) break; // stop capturing by pr
essing ESC
    }
}
cout << "End of txt file" << endl;
}
else {
    cout << "The is failed to open" << endl;
}
system("Pause");

return 0;
}

```

B.7 Mobile application

[Done by: Tan Jing Jie & Jacynth Tham Ming Quan, Tan Wei Mun]

The version control of this project is managed by Git with the Github interface. The code and application installer can found in the Github repository: <https://github.com/JJTAN0611/TrafficSignRecognitionOpenCV.git>.

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.opencvproject">

    <uses-permission android:name="android.permission.CAMERA" />

    <uses-feature
        android:name="android.hardware.camera"
        android:required="false" />
    <uses-feature
        android:name="android.hardware.camera.autofocus"
        android:required="false" />
    <uses-feature
        android:name="android.hardware.camera.front"
        android:required="false" />
    <uses-feature
        android:name="android.hardware.camera.front.autofocus"
        android:required="false" />

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:screenOrientation="landscape"
        android:supportsRtl="true"
        android:theme="@style/Theme.OpenCVProject">
        <activity
            android:name=".FromFileRecognitionActivity"
            android:screenOrientation="landscape" />
        <activity
            android:name=".FromCameraRecognitionActivity"
            android:screenOrientation="landscape" />
        <activity android:name=".SplashScreen">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name=".MainActivity"></activity>
    </application>

</manifest>
```

Classification.java

```

package com.example.opencvproject;

public class Classification {
    public final String title;
    public final float confidence;

    public Classification(String title, float confidence) {
        this.title = title;
        this.confidence = confidence;
    }

    @Override
    public String toString() {
        return title + " " + String.format("(%.1f%%)", confidence * 100.0f);
    }
}

```

FromCameraRecognitionActivity.java

```

package com.example.opencvproject;

import java.io.FileInputStream;
import java.io.IOException;
import java.nio.MappedByteBuffer;
import java.nio.channels.FileChannel;
import java.util.ArrayList;
import java.util.List;
import java.util.Locale;

import org.opencv.android.BaseLoaderCallback;
import org.opencv.android.CameraBridgeViewBase.CvCameraViewFrame;
import org.opencv.android.LoaderCallbackInterface;
import org.opencv.android.OpenCVLoader;
import org.opencv.android.Utils;
import org.opencv.core.CvException;
import org.opencv.core.CvType;
import org.opencv.core.Mat;
import org.opencv.core.Scalar;
import org.opencv.android.CameraBridgeViewBase;
import org.opencv.android.CameraBridgeViewBase.CvCameraViewListener2;
import org.opencv.imgproc.Imgproc;
import org.tensorflow.lite.Interpreter;

import android.app.Activity;
import android.app.ProgressDialog;
import android.content.Intent;
import android.content.pm.ActivityInfo;
import android.content.res.AssetFileDescriptor;
import android.graphics.Bitmap;
import android.media.ThumbnailUtils;
import android.os.Bundle;
import android.speech.tts.TextToSpeech;
import android.util.Log;
import android.view.View;
import android.view.Window;
import android.view.WindowManager;
import android.view.SurfaceView;
import android.widget.Button;
import android.widget.ImageView;
import android.widget.TextView;

```

```

import android.widget.Toast;

import static org.opencv.imgproc.Imgproc.contourArea;
import static org.opencv.imgproc.Imgproc.dilate;
import static org.opencv.imgproc.Imgproc.drawContours;
import static org.opencv.imgproc.Imgproc.erode;
import static org.opencv.imgproc.Imgproc.findContours;
import static org.opencv.imgproc.Imgproc.floodFill;

import
com.google.android.material.floatingactionbutton.FloatingActionButton;

public class FromCameraRecognitionActivity extends Activity implements
CvCameraViewListener2 {
    private static final String TAG = "FromCamera";
    int count = 0;

    TextView textView;
    Button capture;
    private Mat mRgba;
    private GtsrbClassifier gtsrbClassifier;
    private CameraBridgeViewBase mOpenCvCameraView;
    Button navigate_from_file;
    List<Mat> result= new ArrayList<>();
    TextToSpeech tts;

    private BaseLoaderCallback mLoaderCallback = new
BaseLoaderCallback(this) {
        @Override
        public void onManagerConnected(int status) {
            switch (status) {
                case LoaderCallbackInterface.SUCCESS: {
                    Log.i(TAG, "OpenCV loaded successfully");
                    mOpenCvCameraView.enableView();
                }
                break;
                default: {
                    super.onManagerConnected(status);
                }
                break;
            }
        }
    };

    public FromCameraRecognitionActivity() {
        Log.i(TAG, "Instantiated new " + this.getClass());
    }

    /**
     * Called when the activity is first created.
     */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        Log.i(TAG, "called onCreate");
        super.onCreate(savedInstanceState);
        requestWindowFeature(Window.FEATURE_NO_TITLE);
        setContentView(R.layout.activity_from_camera_recognition);
        getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
                            WindowManager.LayoutParams.FLAG_FULLSCREEN);

        setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE);

        loadGtsrbClassifier();

        mOpenCvCameraView = (CameraBridgeViewBase)
findViewById(R.id.cameralview);
        mOpenCvCameraView.setVisibility(SurfaceView.VISIBLE);
    }
}

```

```

mOpenCvCameraView.setCvCameraViewListener(this);

capture = findViewById(R.id.take_photo_btn);
textView = findViewById(R.id.textView);

textView.setText("Hello");
capture.bringToFront();

tts=new TextToSpeech(FromCameraRecognitionActivity.this, new
TextToSpeech.OnInitListener() {

    @Override
    public void OnInit(int status) {
        // TODO Auto-generated method stub
        if(status == TextToSpeech.SUCCESS){
            int result=tts.setLanguage(Locale.US);
            if(result==TextToSpeech.LANG_MISSING_DATA ||
               result==TextToSpeech.LANG_NOT_SUPPORTED){
                Log.e("error", "This Language is not supported");
            }
            else{
                tts.setLanguage(Locale.US);
            }
        }
        else{
            Log.e("error", "Initialization Failed!");
        }
    }
});

//voice output
//export
navigate_from_file = findViewById(R.id.navigate_from_file);
navigate_from_file.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Toast.makeText(getApplicationContext(), "Add face
activity", Toast.LENGTH_SHORT).show();
        Intent intent = new
Intent(FromCameraRecognitionActivity.this,
FromFileRecognitionActivity.class);
        startActivity(intent);
        finish();
    }
});

capture.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        if (result.size() == 0)
            return;
        Mat currentCapture = result.get(0);
        Log.d("tmp", currentCapture.height() + " | " +
currentCapture.width());
        try {
            Bitmap bmp = convertMatToBitmap(currentCapture);
            //imageView.setImageBitmap(
convertMatToBitmap(currentCapture));
            //imageView.invalidate();
            Log.d("Exception", "here");
            Bitmap squareBitmap =

```

```

ThumbnailUtils.extractThumbnail(bmp, bmp.getWidth(), bmp.getHeight());
        Bitmap preprocessedImage =
ImageUtils.prepareImageForClassification(squareBitmap);
        List<Classification> recognitions =
gtsrbClassifier.recognizeImage(preprocessedImage);
        String a="";
        for (Classification b:recognitions)
            a=a+b;
        textView.setText(a);
        textView.invalidate();

tts.speak(textView.getText(),TextToSpeech.QUEUE_ADD,null,"0");
    } catch (Exception e) {
        Log.d("Exception", ""+e.getMessage());
    }
}
});

private static Bitmap convertMatToBitMap(Mat input){
    Bitmap bmp = null;
    Mat rgb = new Mat();
    Imgproc.cvtColor(input, rgb, Imgproc.COLOR_BGR2RGB);
    Imgproc.cvtColor(rgb, rgb, Imgproc.COLOR_RGB2BGR);
    try {
        bmp = Bitmap.createBitmap(rgb.cols(), rgb.rows(),
Bitmap.Config.ARGB_8888);
        Utils.matToBitmap(rgb, bmp);
    }
    catch (CvException e){
        Log.d("Exception",e.getMessage());
    }
    return bmp;
}

private void loadGtsrbClassifier() {
    try {
        gtsrbClassifier = new GtsrbClassifier(new
Interpreter(loadModelFile(this, "gtsrb_model.tflite")));
    } catch (IOException e) {
        Toast.makeText(this, "GTSRB model couldn't be loaded. Check
logs for details.", Toast.LENGTH_SHORT).show();
        e.printStackTrace();
    }
}

public MappedByteBuffer loadModelFile(Activity activity, String
MODEL_FILE) throws IOException {
    AssetFileDescriptor fileDescriptor =
activity.getAssets().openFd(MODEL_FILE);
    FileInputStream inputStream = new
FileInputStream(fileDescriptor.getFileDescriptor());
    FileChannel fileChannel = inputStream.getChannel();
    long startOffset = fileDescriptor.getStartOffset();
    long declaredLength = fileDescriptor.getDeclaredLength();
    return fileChannel.map(FileChannel.MapMode.READ_ONLY, startOffset,
declaredLength);
}

@Override
public void onPause() {
    super.onPause();
    if (mOpenCvCameraView != null)
        mOpenCvCameraView.disableView();
}

@Override

```

```

    public void onResume() {
        super.onResume();
        if (!OpenCVLoader.initDebug()) {
            Log.d(TAG, "Internal OpenCV library not found. Using OpenCV
Manager for initialization");
            OpenCVLoader.initAsync(OpenCVLoader.OPENCV_VERSION_3_0_0,
this, mLoaderCallback);
        } else {
            Log.d(TAG, "OpenCV library found inside package. Using it!");
        }
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
        if (mOpenCvCameraView != null)
            mOpenCvCameraView.disableView();
    }

    public void onCameraViewStarted(int width, int height) {
        mRgba = new Mat(height, width, CvType.CV_8UC4);
    }

    public void onCameraViewStopped() {
        mRgba.release();
    }

    public Mat onCameraFrame(CvCameraViewFrame inputFrame) {

        Mat input = inputFrame.rgba();
        ImagePreprocess ip = new ImagePreprocess();
        Thread thread = new Thread() {
            public void run() {
                result = ip.processCamera(input);
                mRgba = input;
            }
        };
        thread.run();
        return mRgba;
    }
}

```

FromFileRecognitionActivity.java

```

package com.example.opencvproject;

import androidx.annotation.NonNull;
import androidx.annotation.Nullable;
import androidx.annotation.RequiresApi;
import androidx.appcompat.app.AppCompatActivity;
import androidx.viewpager2.widget.ViewPager2;

import android.annotation.SuppressLint;
import android.app.Activity;
import android.app.AlertDialog;
import android.content.ContentResolver;
import android.content.ContentValues;

```

```

import android.content.Context;
import android.content.Intent;
import android.content.pm.ActivityInfo;
import android.content.res.AssetFileDescriptor;
import android.database.Cursor;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.media.ThumbnailUtils;
import android.net.Uri;
import android.os.Build;
import android.os.Bundle;
import android.os.Environment;
import android.provider.MediaStore;
import android.speech.tts.TextToSpeech;
import android.util.Log;
import android.view.DragEvent;
import android.view.View;
import android.view.Window;
import android.view.WindowManager;
import android.widget.Button;
import android.widget.ImageView;
import android.widget.TextView;
import android.widget.Toast;

import org.opencv.android.BaseLoaderCallback;
import org.opencv.android.LoaderCallbackInterface;
import org.opencv.android.OpenCVLoader;
import org.opencv.android.Utils;
import org.opencv.core.CvException;
import org.opencv.core.Mat;
import org.opencv.imgproc.Imgproc;
import org.tensorflow.lite.Interpreter;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.nio.MappedByteBuffer;
import java.nio.channels.FileChannel;
import java.util.ArrayList;
import java.util.Date;
import java.util.List;
import java.util.Locale;

public class FromFileRecognitionActivity extends AppCompatActivity {
    private static final String TAG = "FromFiles";
    private static final int RESULT_LOAD_IMAGE = 1;

    Button fromfile;
    Button export;
    TextView textView;
    ImageView input;
    ViewPager2 viewPager2;
    private GtsrbClassifier gtsrbClassifier;
    TextToSpeech tts;
    ProgressDialog nDialog;
    List<String> recognitions;
    List<String> voices;
    List<Bitmap> bsegments;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        requestWindowFeature(Window.FEATURE_NO_TITLE);
        setContentView(R.layout.activity_from_file_recognition);
    }
}

```

```

        getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
                WindowManager.LayoutParams.FLAG_FULLSCREEN);

        setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE);
        getSupportActionBar().hide();

        loadGtsrbClassifier();

        fromfile = findViewById(R.id.from_file);
        export = findViewById(R.id.export);
        textView = findViewById(R.id.outputText);
        input = findViewById(R.id.input);
        viewPager2 = findViewById(R.id.result);
        textView.setText("No traffic sign");
        textView.invalidate();

        nDialog = new ProgressDialog(this);
        nDialog.setMessage("Loading..");
        nDialog.setTitle("Get Traffic Sign");
        nDialog.setIndeterminate(false);
        nDialog.setCancelable(true);

        recognitions = new ArrayList<String>();
        voices=new ArrayList<String>();
        bsegments=new ArrayList<Bitmap>();

        fromfile.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent photoPickerIntent = new Intent(Intent.ACTION_PICK);
                photoPickerIntent.setType("image/*");
                startActivityForResult(photoPickerIntent,
                        RESULT_LOAD_IMAGE);
            }
        });

        tts=new TextToSpeech(FromFileRecognitionActivity.this, new
        TextToSpeech.OnInitListener() {

            @Override
            public void onInit(int status) {
                // TODO Auto-generated method stub
                if(status == TextToSpeech.SUCCESS) {
                    int result=tts.setLanguage(Locale.US);
                    if(result==TextToSpeech.LANG_MISSING_DATA ||
                            result==TextToSpeech.LANG_NOT_SUPPORTED) {
                        Log.e("error", "This Language is not supported");
                    }
                    else{
                        tts.setLanguage(Locale.US);
                    }
                }
                else
                    Log.e("error", "Initialization Failed!");
            }
        });

        textView.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                int currentItem = viewPager2.getCurrentItem();
                if(voices.size()==0)
                    tts.speak("No traffic
sign.",TextToSpeech.QUEUE_ADD,null,"0");
                else

```

```

        tts.speak("("+currentItem+1)+"/"
"+voices.get(currentItem),TextToSpeech.QUEUE_ADD,null,"0");
    }
});

@RequiresApi(api = Build.VERSION_CODES.M)
@Override
protected void onActivityResult(int requestCode, int resultCode,
@Nullable Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
    Bitmap selectedImage;

    if (resultCode == RESULT_OK) {

        try {
            final Uri imageUri = data.getData();
            final InputStream imageStream =
getContentResolver().openInputStream(imageUri);
            selectedImage = BitmapFactory.decodeStream(imageStream);
            input.setImageBitmap(selectedImage);
        } catch (FileNotFoundException e) {
            e.printStackTrace();
            Toast.makeText(this, "Something went wrong",
Toast.LENGTH_LONG).show();
            return;
        }
    } else {
        Toast.makeText(this, "You haven't picked Image",
Toast.LENGTH_LONG).show();
        return;
    }

    nDialog.show();

    Mat mat = new Mat();
    Bitmap bmp32 = selectedImage.copy(Bitmap.Config.ARGB_8888, true);
    Utils.bitmapToMat(bmp32, mat);

    ImagePreprocess ip = new ImagePreprocess();
    List<Mat> segment = ip.processImage(mat);

    if(segment.size()==0) {
        Toast.makeText(getApplicationContext(),"No traffic sign found",
Toast.LENGTH_LONG);
        textView.setText("No traffic sign.");
        return;
    }

    viewPager2.setAdapter(new SliderAdapter(segment,
getApplicationContext()));

    recognitions = new ArrayList<String>();
    voices=new ArrayList<String>();
    bsegments=new ArrayList<Bitmap>();
    for (Mat s : segment) {
        Bitmap bmp = convertMatToBitmap(s);
        bsegments.add(bmp);
        Bitmap squareBitmap = ThumbnailUtils.extractThumbnail(bmp,
bmp.getWidth(), bmp.getHeight());
        Bitmap preprocessedImage =
ImageUtils.prepareImageForClassification(squareBitmap);
        List<Classification> r =

```

```

gtsrbClassifier.recognizeImage(preprocessedImage);
    try {
        recognitions.add(r.toString());
        String temp = r.toString();
        String arr[] = temp.split(" ", 2);
        temp=arr[0].replace("_","");
        temp=temp.replace("[","");
        voices.add(temp);
    } catch (Exception e) {
        recognitions.add("No found");
        voices.add("No found");
    }
}

viewPager2.registerOnPageChangeCallback(new
ViewPager2.OnPageChangeCallback() {
    @Override
    public void onPageScrolled(int position, float positionOffset,
int positionOffsetPixels) {
        super.onPageScrolled(position, positionOffset,
positionOffsetPixels);
        if (recognitions == null)
            return;
        int currentItem = viewPager2.getCurrentItem();
        textView.setText((currentItem+1) + "/" +
recognitions.size() + " : " + recognitions.get(currentItem));
        textView.invalidate();
    }
});

export.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        String time = (new Date()).getTime()+"";
        if(bsegments.size()==0) {
            Toast.makeText(getApplicationContext(), "No traffic
sign", Toast.LENGTH_SHORT);
            textView.setText("No traffic sign!!!!");
        }
        for(int i=0;i<bsegments.size();i++)
        {
            try {
                saveImage(bsegments.get(i),voices.get(i),time);
            } catch (IOException e) {
                e.printStackTrace();
            }
            Toast.makeText(getApplicationContext(), "Successfully
saved", Toast.LENGTH_SHORT);
        }
    }
});
nDialog.dismiss();
}

private void saveImage(Bitmap bitmap, @NonNull String name, String
time) throws IOException {
    boolean saved;
    OutputStream fos;

    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.Q) {
        ContentResolver resolver = getContentResolver();
        ContentValues contentValues = new ContentValues();
        contentValues.put(MediaStore.MediaColumns.DISPLAY_NAME, name);
        contentValues.put(MediaStore.MediaColumns.MIME_TYPE,
"image/png");
        contentValues.put(MediaStore.MediaColumns.RELATIVE_PATH,
"DCIM/openCVExport_" + time);
        Uri imageUri =

```

```

resolver.insert(MediaStore.Images.Media.EXTERNAL_CONTENT_URI,
contentValues);
    fos = resolver.openOutputStream(imageUri);
} else {
    String imagesDir =
Environment.getExternalStoragePublicDirectory(
    Environment.DIRECTORY_DCIM).toString() +
File.separator + "openCVExport_" + time;

    File file = new File(imagesDir);

    if (!file.exists()) {
        file.mkdir();
    }

    File image = new File(imagesDir, name + ".png");
    fos = new FileOutputStream(image);

}

saved = bitmap.compress(Bitmap.CompressFormat.PNG, 100, fos);
fos.flush();
fos.close();
}

public void loadGtsrbClassifier() {
    try {
        gtsrbClassifier = new GtsrbClassifier(new
Interpreter(loadModelFile(this, "gtsrb_model.tflite")));
    } catch (IOException e) {
        Toast.makeText(this, "GTSRB model couldn't be loaded. Check
logs for details.", Toast.LENGTH_SHORT).show();
        e.printStackTrace();
    }
}

public MappedByteBuffer loadModelFile(Activity activity, String
MODEL_FILE) throws IOException {
    AssetFileDescriptor fileDescriptor =
activity.getAssets().openFd(MODEL_FILE);
    FileInputStream inputStream = new
FileInputStream(fileDescriptor.getFileDescriptor());
    FileChannel fileChannel = inputStream.getChannel();
    long startOffset = fileDescriptor.getStartOffset();
    long declaredLength = fileDescriptor.getDeclaredLength();
    return fileChannel.map(FileChannel.MapMode.READ_ONLY, startOffset,
declaredLength);
}

static Bitmap convertMatToBitMap(Mat input) {
    Bitmap bmp = null;
    Mat rgb = new Mat();
    Imgproc.cvtColor(input, rgb, Imgproc.COLOR_BGR2RGB);
    Imgproc.cvtColor(rgb, rgb, Imgproc.COLOR_RGB2BGR);
    try {
        bmp = Bitmap.createBitmap(rgb.cols(), rgb.rows(),
Bitmap.Config.ARGB_8888);
        Utils.matToBitmap(rgb, bmp);
    } catch (CvException e) {
        Log.d("Exception", e.getMessage());
    }
    return bmp;
}

private BaseLoaderCallback mLoaderCallback = new
BaseLoaderCallback(this) {
    @Override

```

```

        public void onManagerConnected(int status) {
            switch (status) {
                case LoaderCallbackInterface.SUCCESS:
                    break;
                default:
                    super.onManagerConnected(status);
                    break;
            }
        }

    @Override
    public void onResume() {
        super.onResume();
        if (!OpenCVLoader.initDebug()) {
            Log.d(TAG, "Internal OpenCV library not found. Using OpenCV Manager for initialization");
            OpenCVLoader.initAsync(OpenCVLoader.OPENCV_VERSION_3_0_0,
this, mLoaderCallback);
        } else {
            Log.d(TAG, "OpenCV library found inside package. Using it!");
        }

        mLoaderCallback.onManagerConnected(LoaderCallbackInterface.SUCCESS);
    }

}

```

GtsrbClassifier.java

```

package com.example.opencvproject;

import android.content.res.AssetFileDescriptor;
import android.content.res.AssetManager;
import android.graphics.Bitmap;

import org.tensorflow.lite.Interpreter;

import java.io.FileInputStream;
import java.io.IOException;
import java.nio.ByteBuffer;
import java.nio.ByteOrder;
import java.nio.channels.FileChannel;
import java.util.ArrayList;
import java.util.List;
import java.util.PriorityQueue;

import static
com.example.opencvproject.GtsrbModelConfig.CLASSIFICATION_THRESHOLD;
import static com.example.opencvproject.GtsrbModelConfig.IMAGE_MEAN;
import static com.example.opencvproject.GtsrbModelConfig.IMAGE_STD;
import static
com.example.opencvproject.GtsrbModelConfig.INPUT_IMG_SIZE_HEIGHT;
import static
com.example.opencvproject.GtsrbModelConfig.INPUT_IMG_SIZE_WIDTH;
import static
com.example.opencvproject.GtsrbModelConfig.MAX_CLASSIFICATION_RESULTS;
import static com.example.opencvproject.GtsrbModelConfig.MODEL_INPUT_SIZE;

public class GtsrbClassifier {

```

```

private final Interpreter interpreter;

public GtsrbClassifier(Interpreter interpreter) {
    this.interpreter = interpreter;
}

public List<Classification> recognizeImage(Bitmap bitmap) {
    ByteBuffer byteBuffer = convertBitmapToByteBuffer(bitmap);
    float[][] result = new float[1][GtsrbModelConfig.OUTPUT_LABELS.size()];
    interpreter.run(byteBuffer, result);
    return getSortedResult(result);
}

private ByteBuffer convertBitmapToByteBuffer(Bitmap bitmap) {
    ByteBuffer byteBuffer =
    ByteBuffer.allocateDirect(MODEL_INPUT_SIZE);
    byteBuffer.order(ByteOrder.nativeOrder());
    int[] intValues = new int[INPUT_IMG_SIZE_WIDTH *
INPUT_IMG_SIZE_HEIGHT];
    bitmap.getPixels(intValues, 0, bitmap.getWidth(), 0, 0,
    bitmap.getWidth(), bitmap.getHeight());
    int pixel = 0;
    for (int i = 0; i < INPUT_IMG_SIZE_WIDTH; ++i) {
        for (int j = 0; j < INPUT_IMG_SIZE_HEIGHT; ++j) {
            final int val = intValues[pixel++];
            byteBuffer.putFloat(((val >> 16) & 0xFF) - IMAGE_MEAN) /
IMAGE_STD);
            byteBuffer.putFloat(((val >> 8) & 0xFF) - IMAGE_MEAN) /
IMAGE_STD);
            byteBuffer.putFloat(((val) & 0xFF) - IMAGE_MEAN) /
IMAGE_STD);
        }
    }
    return byteBuffer;
}

private List<Classification> getSortedResult(float[][] resultsArray) {
    PriorityQueue<Classification> sortedResults = new PriorityQueue<>(
        MAX_CLASSIFICATION_RESULTS,
        (lhs, rhs) -> Float.compare(rhs.confidence,
        lhs.confidence)
    );

    for (int i = 0; i < GtsrbModelConfig.OUTPUT_LABELS.size(); ++i) {
        float confidence = resultsArray[0][i];
        if (confidence > CLASSIFICATION_THRESHOLD) {
            GtsrbModelConfig.OUTPUT_LABELS.size());
            sortedResults.add(new
Classification(GtsrbModelConfig.OUTPUT_LABELS.get(i), confidence));
        }
    }

    return new ArrayList<>(sortedResults);
}
}

```

GtsrbModelConfig.java

```

package com.example.opencvproject;

import java.util.Arrays;

```

```

import java.util.Collections;
import java.util.List;

/**
 * The most of those information can be found in
 GTSRB_TensorFlow_MobileNet.ipynb
 */
public class GtsrbModelConfig {
    public static String MODEL_FILENAME = "gtsrb_model.tflite";

    public static final int INPUT_IMG_SIZE_WIDTH = 224;
    public static final int INPUT_IMG_SIZE_HEIGHT = 224;
    public static final int FLOAT_TYPE_SIZE = 4;
    public static final int PIXEL_SIZE = 3;
    public static final int MODEL_INPUT_SIZE = FLOAT_TYPE_SIZE *
INPUT_IMG_SIZE_WIDTH * INPUT_IMG_SIZE_HEIGHT * PIXEL_SIZE;
    public static final int IMAGE_MEAN = 0;
    public static final float IMAGE_STD = 255.0f;

    //This list can be taken from notebooks/output/labels_readable.txt
    public static final List<String> OUTPUT_LABELS =
Collections.unmodifiableList(
    Arrays.asList(
        "20_speed",
        "30_speed",
        "50_speed",
        "60_speed",
        "70_speed",
        "80_speed",
        "80_lifted",
        "100_speed",
        "120_speed",
        "no_overtaking_general",
        "no_overtaking_trucks",
        "right_of_way_crossing",
        "right_of_way_general",
        "give_way",
        "stop",
        "no_way_general",
        "no_way_trucks",
        "no_way_one_way",
        "attention_general",
        "attention_left_turn",
        "attention_right_turn",
        "attention_curvy",
        "attention_bumpers",
        "attention_slippery",
        "attention_bottleneck",
        "attention_construction",
        "attention_traffic_light",
        "attention_pedestrian",
        "attention_children",
        "attention_bikes",
        "attention_snowflake",
        "attention_deer",
        "lifted_general",
        "turn_right",
        "turn_left",
        "turn_straight",
        "turn_straight_right",
        "turn_straight_left",
        "turn_right_down",
        "turn_left_down",
        "turn_circle",
        "lifted_no_overtaking_general",
        "lifted_no_overtaking_trucks"
    )
);
}

```

```

    public static final int MAX_CLASSIFICATION_RESULTS = 3;
    public static final float CLASSIFICATION_THRESHOLD = 0.1f;
}

```

ImagePreprocess.java

```

package com.example.opencvproject;

import android.graphics.Bitmap;
import android.util.Log;
import android.widget.TextView;

import org.opencv.android.Utils;
import org.opencv.core.Core;
import org.opencv.core.CvException;
import org.opencv.core.Mat;
import org.opencv.core.MatOfPoint;
import org.opencv.core.Point;
import org.opencv.core.Rect;
import org.opencv.core.Scalar;
import org.opencv.core.Size;
import org.opencv.imgproc.Imgproc;

import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.List;

import static org.opencv.core.CvType.CV_8UC3;
import static org.opencv.core.CvType.CV_8U;

public class ImagePreprocess{
    private Mat input;

    Scalar redLow1;
    Scalar redHigh1;

    Scalar redLow2 ;
    Scalar redHigh2 ;

    // Hue Ranges for Blue
    Scalar blueLow;
    Scalar blueHigh ;

    // Hue Ranges for Yellow
    Scalar yellowLow ;
    Scalar yellowHigh ;

    ImagePreprocess() {
        redLow1 = new Scalar(150, 140, 160);
        redHigh1 = new Scalar(180, 255, 255);

        redLow2 = new Scalar(0, 50, 50);
        redHigh2 = new Scalar(3, 255, 255);

        // Hue Ranges for Blue
        blueLow = new Scalar(100, 150, 100);
        blueHigh = new Scalar(128, 255, 255);

        // Hue Ranges for Yellow
        yellowLow = new Scalar(14, 100, 140);
        yellowHigh = new Scalar(30, 255, 255);
    }
}

```

```

}

public List<Mat> processImage(Mat in) {

    input=in;
    Mat temp=in.clone();
    Imgproc.cvtColor(temp, temp, Imgproc.COLOR_RGBA2BGR);
    Mat blur=new Mat();
    Imgproc.bilateralFilter(temp, blur, 21, 21*3, 7);
    Mat hsv = new Mat();
    Imgproc.cvtColor(blur, hsv, Imgproc.COLOR_BGR2HSV);

    Mat Image= new Mat(),
        redMask1= new Mat(), redMask2= new Mat(), blueMask= new
    Mat(), yellowMask= new Mat(), mask= new Mat(),
        canvas= new Mat();

    List<Mat> result =new ArrayList<Mat>();

    canvas.create(input.rows(), input.cols(), CV_8UC3);
    input.copyTo(canvas);

    // Match for Red
    Core.inRange(hsv, redLow1, redHigh1, redMask1);
    Core.inRange(hsv, redLow2, redHigh2, redMask2);

    // Match for Blue
    Core.inRange(hsv, blueLow, blueHigh, blueMask);

    // Match for Yellow
    Core.inRange(hsv, yellowLow, yellowHigh, yellowMask);

    //Remove noise
    Mat element = new Mat(2,2,CV_8U,new Scalar(1));
    Imgproc.morphologyEx(yellowMask,yellowMask, Imgproc.MORPH_CLOSE,
element);
    Imgproc.morphologyEx(redMask1,redMask1, Imgproc.MORPH_CLOSE,
element);
    Imgproc.morphologyEx(redMask2,redMask2, Imgproc.MORPH_CLOSE,
element);
    Imgproc.morphologyEx(blueMask,blueMask, Imgproc.MORPH_CLOSE,
element);

    //Merged
    Core.bitwise_or(redMask1, redMask2, mask);
    Core.bitwise_or(mask, blueMask, mask);
    Core.bitwise_or(mask, yellowMask, mask);

    //Convert to do bitwise
    Imgproc.cvtColor( mask,mask, Imgproc.COLOR_GRAY2BGR);
    Imgproc.cvtColor( hsv,Image, Imgproc.COLOR_HSV2BGR);
    Core.bitwise_and(mask, Image , Image);

    // Do contour
    Imgproc.erode(Image,Image, element);
    Imgproc.dilate(Image,Image, element);
    Mat cannyOutput = new Mat();
    Imgproc.Canny(Image, cannyOutput, 100, 100 * 2);
    List<MatOfPoint> contours = new ArrayList<>();
    Mat hierarchy = new Mat();
    Imgproc.findContours(cannyOutput, contours,hierarchy,
Imgproc.RETR_EXTERNAL, Imgproc.CHAIN_APPROX_NONE);
}

```

```

Collections.sort(contours, new Comparator<MatOfPoint>() {
    @Override
    public int compare(MatOfPoint o1, MatOfPoint o2) {
        long sumMop1 = 0;
        long sumMop2 = 0;
        for( Point p: o1.toList() ) {
            sumMop1 += p.x + p.y;
        }
        for( Point p: o2.toList() ) {
            sumMop2 += p.x + p.y;
        }
        if( sumMop1 > sumMop2)
            return 1;
        else if( sumMop1 < sumMop2 )
            return -1;
        else
            return 0;
    }
});

List<MatOfPoint> filter = new ArrayList<>(contours.size());

for (int i=0; i<contours.size();i++) {

if(Imgproc.contourArea(contours.get(i))*6>Imgproc.contourArea(contours.get(contours.size()-1))){
    filter.add(contours.get(i));
}
}

Rect boundRect;
for (int i=0;i< filter.size();i++){
    Mat filledCanvasMask=new Mat(input.rows(),
input.cols(), CV_8UC3,new Scalar(0,0,0));
    Imgproc.fillConvexPoly(filledCanvasMask,filter.get(i), new
Scalar(255,255,255));
    Mat segment= new Mat();
    Imgproc.cvtColor( hsv,segment, Imgproc.COLOR_HSV2BGR);
    Core.bitwise_and(filledCanvasMask, segment , segment);
    boundRect = Imgproc.boundingRect((MatOfPoint) filter.get(i));
    Imgproc.rectangle( input, boundRect.tl(), boundRect.br(), new
Scalar(255,255,255), 2, Imgproc.LINE_AA, 0 );
    Mat crop=segment.submat(boundRect);
    Imgproc.cvtColor(crop,crop,Imgproc.COLOR_BGR2RGBA);
    result.add(crop);
}

if(result.size()<=6)
    return result;
return result.subList(result.size()-6,result.size()-1);
}

public List<Mat> processCamera(Mat in) {

    input=in;
    Mat temp=in.clone();
    Imgproc.cvtColor(temp, temp, Imgproc.COLOR_RGBA2BGR);
    Mat blur=new Mat();
    Imgproc.bilateralFilter(temp, blur, 21, 21*3, 7);
    Mat hsv = new Mat();
    Imgproc.cvtColor(blur, hsv, Imgproc.COLOR_BGR2HSV);

    Mat Image= new Mat(),
    redMask1= new Mat(), redMask2= new Mat(), blueMask= new
Mat(), yellowMask= new Mat(), mask= new Mat(),
    canvas= new Mat();

    List<Mat> result =new ArrayList<Mat>();
}

```

```

        canvas.create(input.rows(), input.cols(), CV_8UC3);
        input.copyTo(canvas);

        // Match for Red
        Core.inRange(hsv, redLow1, redHigh1, redMask1);
        Core.inRange(hsv, redLow2, redHigh2, redMask2);

        // Match for Blue
        Core.inRange(hsv, blueLow, blueHigh, blueMask);

        // Match for Yellow
        Core.inRange(hsv, yellowLow, yellowHigh, yellowMask);

        //Remove noise
        Mat element = new Mat(2,2,CV_8U,new Scalar(1));
        Imgproc.morphologyEx(yellowMask,yellowMask, Imgproc.MORPH_CLOSE,
element);
        Imgproc.morphologyEx(redMask1,redMask1, Imgproc.MORPH_CLOSE,
element);
        Imgproc.morphologyEx(redMask2,redMask2, Imgproc.MORPH_CLOSE,
element);
        Imgproc.morphologyEx(blueMask,blueMask, Imgproc.MORPH_CLOSE,
element);

        //Merged
        Core.bitwise_or(redMask1, redMask2, mask);
        Core.bitwise_or(mask, blueMask, mask);
        Core.bitwise_or(mask, yellowMask, mask);

        //Convert to do bitwise
        Imgproc.cvtColor( mask,mask, Imgproc.COLOR_GRAY2BGR);
        Imgproc.cvtColor( hsv,Image, Imgproc.COLOR_HSV2BGR);
        Core.bitwise_and(mask, Image , Image);

        // Do contour
        Imgproc.erode(Image,Image, element);
        Imgproc.dilate(Image,Image, element);
        Mat cannyOutput = new Mat();
        Imgproc.Canny(Image, cannyOutput, 100, 100 * 2);
        List<MatOfPoint> contours = new ArrayList<>();
        Mat hierarchy = new Mat();
        Imgproc.findContours(cannyOutput, contours,hierarchy,
Imgproc.RETR_EXTERNAL, Imgproc.CHAIN_APPROX_NONE);

        Collections.sort(contours, new Comparator<MatOfPoint>() {
            @Override
            public int compare(MatOfPoint o1, MatOfPoint o2) {
                long sumMop1 = 0;
                long sumMop2 = 0;
                for( Point p: o1.toList() ) {
                    sumMop1 += p.x + p.y;
                }
                for( Point p: o2.toList() ) {
                    sumMop2 += p.x + p.y;
                }
                if( sumMop1 > sumMop2)
                    return 1;
                else if( sumMop1 < sumMop2 )
                    return -1;
                else
                    return 0;
            }
        });
        List<MatOfPoint> filter = new ArrayList<>(contours.size());
    
```

```

        for (int i=0; i<contours.size();i++) {

    if(Imgproc.contourArea(contours.get(i))*6>Imgproc.contourArea(contours.get
    (contours.size()-1))){
        filter.add(contours.get(i));
    }
}

if(filter.size()==0)
    return result;

Rect boundRect;
for (int i=filter.size()-1;i< filter.size();i++) {
    Mat filledCanvasMask=new Mat(input.rows(),
input.cols(),CV_8UC3,new Scalar(0,0,0));
    Imgproc.fillConvexPoly(filledCanvasMask,filter.get(i), new
Scalar(255,255,255));
    Mat segment= new Mat();
    Imgproc.cvtColor( hsv,segment, Imgproc.COLOR_HSV2BGR);
    Core.bitwise_and(filledCanvasMask, segment , segment);
    boundRect = Imgproc.boundingRect((MatOfPoint) filter.get(i));
    Imgproc.rectangle( input, boundRect.tl(), boundRect.br(), new
Scalar(255,255,255), 2, Imgproc.LINE_AA, 0 );
    Mat crop=segment.submat(boundRect);
    Imgproc.cvtColor(crop,crop,Imgproc.COLOR_BGR2RGBA);
    result.add(crop);
}

if(result.size()<=6)
    return result;
return result.subList(result.size()-6,result.size()-1);
}

private Scalar converScalarHsv2Rgba(Scalar hsvColor) {
    Mat pointMatRgba = new Mat();
    Mat pointMatHsv = new Mat(1, 1, CV_8UC3, hsvColor);
    Imgproc.cvtColor(pointMatHsv, pointMatRgba,
    Imgproc.COLOR_HSV2RGB_FULL, 4);

    return new Scalar(pointMatRgba.get(0, 0));
}

private static Bitmap convertMatToBitMap(Mat input){
    Bitmap bmp = null;
    Mat rgb = new Mat();
    Imgproc.cvtColor(input, rgb, Imgproc.COLOR_BGR2RGB);
    Imgproc.cvtColor(rgb, rgb, Imgproc.COLOR_RGB2BGR);
    try {
        bmp = Bitmap.createBitmap(rgb.cols(), rgb.rows(),
        Bitmap.Config.ARGB_8888);
        Utils.matToBitmap(rgb, bmp);
    }
    catch (CvException e){
        Log.d("Exception",e.getMessage());
    }
    return bmp;
}
}

```

ImageUtils.java

```

package com.example.opencvproject;

import android.graphics.Bitmap;
import android.graphics.Canvas;
import android.graphics.Paint;

public class ImageUtils {
    /**
     * Make bitmap appropriate size for used model (224x224x).
     * We don't do any preprocessing, because training dataset didn't have
     * any...
     */
    public static Bitmap prepareImageForClassification(Bitmap bitmap) {
        Paint paint = new Paint();
        Bitmap finalBitmap = Bitmap.createScaledBitmap(
            bitmap,
            GtsrbModelConfig.INPUT_IMG_SIZE_WIDTH,
            GtsrbModelConfig.INPUT_IMG_SIZE_HEIGHT,
            false);
        Canvas canvas = new Canvas(finalBitmap);
        canvas.drawBitmap(finalBitmap, 0, 0, paint);
        return finalBitmap;
    }
}

```

MainActivity.java

```

package com.example.opencvproject;

import java.io.FileInputStream;
import java.io.IOException;
import java.nio.MappedByteBuffer;
import java.nio.channels.FileChannel;
import java.util.ArrayList;
import java.util.List;

import org.opencv.android.BaseLoaderCallback;
import org.opencv.android.CameraBridgeViewBase.CvCameraViewFrame;
import org.opencv.android.LoaderCallbackInterface;
import org.opencv.android.OpenCVLoader;
import org.opencv.android.Utils;
import org.opencv.core.CvException;
import org.opencv.core.CvType;
import org.opencv.core.Mat;
import org.opencv.core.Scalar;
import org.opencv.android.CameraBridgeViewBase;
import org.opencv.android.CameraBridgeViewBase.CvCameraViewListener2;
import org.opencv.imgproc.Imgproc;
import org.tensorflow.lite.Interpreter;

import android.app.Activity;
import android.content.Intent;
import android.content.pm.ActivityInfo;
import android.content.res.AssetFileDescriptor;
import android.graphics.Bitmap;
import android.media.ThumbnailUtils;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.view.Window;
import android.view.WindowManager;
import android.view.SurfaceView;
import android.widget.Button;

```

```

import android.widget.ImageView;
import android.widget.TextView;
import android.widget.Toast;

import static org.opencv.imgproc.Imgproc.contourArea;
import static org.opencv.imgproc.Imgproc.dilate;
import static org.opencv.imgproc.Imgproc.drawContours;
import static org.opencv.imgproc.Imgproc.erode;
import static org.opencv.imgproc.Imgproc.findContours;
import static org.opencv.imgproc.Imgproc.floodFill;

import
com.google.android.material.floatingactionbutton.FloatingActionButton;

public class MainActivity extends Activity{
    private static final String TAG = "MainActivity";
    int count = 0;

    Button navigate_from_file;
    Button navigate_from_camera;
    List<Mat> result= new ArrayList<>();

    public MainActivity() {
        Log.i(TAG, "Instantiated new " + this.getClass());
    }

    /**
     * Called when the activity is first created.
     */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        Log.i(TAG, "called onCreate");
        super.onCreate(savedInstanceState);
        requestWindowFeature(Window.FEATURE_NO_TITLE);
        setContentView(R.layout.activity_main);
        getWindow().setFlags WindowManager.LayoutParams.FLAG_FULLSCREEN,
            WindowManager.LayoutParams.FLAG_FULLSCREEN);

        setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE);
        setContentView(R.layout.activity_main);

        //voice output
        //export
        navigate_from_file = findViewById(R.id.from_file);
        navigate_from_file.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Toast.makeText(getApplicationContext(), "Add face
activity", Toast.LENGTH_SHORT).show();
                Intent intent = new Intent(MainActivity.this,
FromFileRecognitionActivity.class);
                startActivity(intent);
            }
        });

        navigate_from_camera = findViewById(R.id.from_camera);
        navigate_from_camera.setOnClickListener(new View.OnClickListener()
{
            @Override
            public void onClick(View v) {
                Toast.makeText(getApplicationContext(), "Add face
activity", Toast.LENGTH_SHORT).show();
                Intent intent = new Intent(MainActivity.this,
FromCameraRecognitionActivity.class);
            }
        });
    }
}

```

```

        startActivity(intent);
    }
});

}

}

```

SliderAdapter.java

```

package com.example.opencvproject;

import android.annotation.SuppressLint;
import android.content.Context;
import android.graphics.Bitmap;
import android.util.Log;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ImageView;
import android.widget.TextView;

import androidx.annotation.NonNull;
import androidx.recyclerview.widget.RecyclerView;

import org.opencv.android.Utils;
import org.opencv.core.CvException;
import org.opencv.core.Mat;
import org.opencv.imgproc.Imgproc;

import java.util.List;

public class SliderAdapter extends
RecyclerView.Adapter<SliderAdapter.SliderViewHolder>{

    private List<Mat> tsList;
    private Context context;

    public SliderAdapter(List<Mat> tsList, Context context) {
        this.tsList = tsList;
        this.context=context;
    }

    @NonNull
    @Override
    public SliderViewHolder onCreateViewHolder(@NonNull ViewGroup parent,
int viewType) {

        return new
SliderViewHolder(LayoutInflater.from(parent.getContext()).inflate(
            R.layout.slide_item_container,
            parent,
            false
        ));
    }

    @Override
    public void onBindViewHolder(@NonNull SliderViewHolder holder, int
position) {

```

```

        holder.setImage(tsList.get(position));
    }

    @Override
    public int getItemCount() {
        return tsList.size();
    }

    class SliderViewHolder extends RecyclerView.ViewHolder{

        public SliderViewHolder(View itemView) {
            super(itemView);
        }

        void setImage(Mat ts){
            ImageView imageView= itemView.findViewById(R.id.imageSlide);
            imageView.setImageBitmap(convertMatToBitMap(ts));
        }

        Bitmap convertMatToBitMap(Mat input){
            Bitmap bmp = null;
            Mat rgb = new Mat();
            Imgproc.cvtColor(input, rgb, Imgproc.COLOR_BGR2RGB);
            Imgproc.cvtColor(rgb, rgb, Imgproc.COLOR_RGB2BGR);
            try {
                bmp = Bitmap.createBitmap(rgb.cols(), rgb.rows(),
                Bitmap.Config.ARGB_8888);
                Utils.matToBitmap(rgb, bmp);
            }
            catch (CvException e){
                Log.d("Exception",e.getMessage());
            }
            return bmp;
        }
    }
}

```

SplashScreen.java

```

package com.example.opencvproject;

import androidx.appcompat.app.AppCompatActivity;

import android.Manifest;
import android.annotation.SuppressLint;
import android.content.Intent;
import android.content.pm.ActivityInfo;
import android.os.Bundle;
import android.os.Handler;
import android.widget.Toast;

import com.tbruyelle.rxpermissions2.RxPermissions;

public class SplashScreen extends AppCompatActivity {

    @SuppressLint("CheckResult")
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_splash_screen);
        getSupportActionBar().hide();
    }
}

```

```

setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE);
    Handler handler=new Handler();
    RxPermissions rxPermissions = new RxPermissions(this);
    rxPermissions
        .request(Manifest.permission.CAMERA)
        .subscribe(granted -> {
            if (granted) {
                handler.postDelayed(new Runnable() {
                    @Override
                    public void run() {
                        Intent intent =new
Intent(SplashScreen.this, MainActivity.class);
                        finish();
                        startActivity(intent);
                    }
                },2000);
                // All requested permissions are granted
            } else {
                Toast.makeText(this, "Permissions Denied. Please
Accept the Needed Permissions. ", Toast.LENGTH_LONG).show();
                // At least one permission is denied
            }
        });
    }

}
}

```

activity from camera recognition.xml

```

<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/activity_from_camera_recognition"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_alignParentStart="true"
        android:layout_alignParentTop="true"
        android:layout_alignParentEnd="true"
        android:layout_alignParentBottom="true"
        android:layout_marginStart="2dp"
        android:layout_marginTop="-9dp"
        android:layout_marginEnd="-1dp"
        android:layout_marginBottom="0dp"
        android:orientation="horizontal">

        <LinearLayout
            android:layout_width="wrap_content"
            android:layout_height="match_parent"
            android:orientation="vertical">

            <org.opencv.android.JavaCameraView
                android:id="@+id/cameraview"
                android:layout_width="666dp"

```

```

        android:layout_height="300dp"/>

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="horizontal">

        <Button
            android:id="@+id/navigate_from_file"
            android:layout_width="271dp"
            android:layout_height="match_parent"
            android:layout_marginStart="0dp"
            android:layout_marginTop="0dp"
            android:background="#F9A825"
            android:textColor="#FFFFFF"
            android:text="File input"
            android:textSize="30dp"/>
        <TextView
            android:id="@+id/textView"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:text="TextView"
            android:textColor="#000000"
            android:textSize="20dp"
            app:autoSizeTextType="uniform" />

    </LinearLayout>

</LinearLayout>

<Button
    android:id="@+id/take_photo_btn"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#283593"
    android:text="Output"
    android:textSize="60dp"
    android:textColor="#FFFFFF"/>

</LinearLayout>
</RelativeLayout>

```

activity from file recognition.xml

```

<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".FromFileRecognitionActivity">

    <ProgressBar
        android:id="@+id/progress_loader"
        style="?android:attr/progressBarStyle"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerInParent="true"

```

```

        android:visibility="visible"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="@+id/linearLayout"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <LinearLayout
        android:id="@+id/linearLayout"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="horizontal"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent">

        <Button
            android:id="@+id/from_file"
            android:layout_width="132dp"
            android:layout_height="match_parent"
            android:layout_alignParentTop="true"
            android:layout_alignParentEnd="true"
            android:backgroundTint="#FBBC3F"
            android:insetTop="0dp"
            android:insetBottom="0dp"
            android:text="From file"
            android:textColor="@color/black"
            android:textSize="20dp"
            android:textStyle="bold"
            app:layout_constraintBottom_toBottomOf="parent"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintTop_toTopOf="parent" />

        <LinearLayout
            android:layout_width="470dp"
            android:layout_height="match_parent"
            android:orientation="vertical">

            <androidx.viewpager2.widget.ViewPager2
                android:id="@+id/result"
                android:layout_width="478dp"
                android:layout_height="211dp"
                android:layout_marginStart="0dp"
                android:layout_marginTop="0dp"
                android:layout_marginEnd="0dp"
                android:background="@drawable/image_border">

        </androidx.viewpager2.widget.ViewPager2>

        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:orientation="horizontal">

            <ImageView
                android:id="@+id/input"
                android:layout_width="220dp"
                android:layout_height="match_parent"
                android:layout_marginStart="0dp"
                android:layout_marginTop="0dp"
                android:layout_marginBottom="0dp"
                android:background="@drawable/image_border"
                app:layout_constraintBottom_toBottomOf="parent"
                app:layout_constraintStart_toEndOf="@+id/from_file"
                app:layout_constraintTop_toBottomOf="@+id/result" />

```

```

        <TextView
            android:id="@+id/outputText"
            android:layout_width="250dp"
            android:layout_height="199dp"
            android:layout_alignParentTop="true"
            android:layout_alignParentEnd="true"
            android:layout_marginStart="10dp"
            android:layout_marginTop="0dp"
            android:layout_marginEnd="0dp"
            android:layout_marginBottom="0dp"
            android:gravity="left"
            android:text="The text output will show here. This is
scrollable. Touch to repeat the voice result output again!"
            android:textColor="@color/black"
            android:textSize="20dp" />
    </LinearLayout>

</LinearLayout>

<Button
    android:id="@+id/export"
    android:layout_width="132dp"
    android:layout_height="match_parent"
    android:layout_alignParentTop="true"
    android:layout_alignParentEnd="true"
    android:backgroundTint="#00838F"
    android:insetTop="0dp"
    android:insetBottom="0dp"
    android:text="Export"
    android:textColor="@color/white"
    android:textSize="20dp"
    android:textStyle="bold"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintTop_toTopOf="parent" />

</LinearLayout>

</androidx.constraintlayout.widget.ConstraintLayout>

```

activity_main.xml

```

<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:id="@+id/activity_main"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="horizontal">
        <Button
            android:id="@+id/from_file"
            android:layout_width="360dp"
            android:layout_height="match_parent"
            android:layout_alignParentTop="true"
            android:layout_alignParentEnd="true"
            android:backgroundTint="#FBBC3F"
            android:insetTop="0dp" />
    
```

```

        android:insetBottom="0dp"
        android:text="From file"
        android:textColor="@color/black"
        android:textSize="20dp"
        android:textStyle="bold"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
    <Button
        android:id="@+id/from_camera"
        android:layout_width="360dp"
        android:layout_height="match_parent"
        android:layout_alignParentTop="true"
        android:layout_alignParentEnd="true"
        android:backgroundTint="#FB3F3F"
        android:insetTop="0dp"
        android:insetBottom="0dp"
        android:text="From camera"
        android:textColor="@color/black"
        android:textSize="20dp"
        android:textStyle="bold"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
    </LinearLayout>
</RelativeLayout>

```

activity splash screen.xml

```

<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".SplashScreen">

    <pl.droidsonroids.gif.GifImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        android:src="@drawable/logo"
        android:background="#FFFFFF" />

</androidx.constraintlayout.widget.ConstraintLayout>

```

slide item container.xml

```

<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/slide_item_container"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

```

```

<ImageView
    android:id="@+id/imageSlide"
    android:layout_width="match_parent"
    android:layout_height="731dp"
    android:layout_alignParentStart="true"
    android:layout_alignParentTop="true"
    android:layout_alignParentEnd="true"
    android:layout_alignParentBottom="true"
    android:layout_marginStart="0dp"
    android:layout_marginTop="0dp"
    android:layout_marginEnd="0dp"
    android:layout_marginBottom="0dp" />

</RelativeLayout>

```

build.gradle

```

plugins {
    id 'com.android.application'
}

android {
    compileSdkVersion 30
    buildToolsVersion "30.0.3"

    defaultConfig {
        applicationId "com.example.opencvproject"
        minSdkVersion 21
        targetSdkVersion 30
        versionCode 1
        versionName "1.0"

        testInstrumentationRunner
"androidx.test.runner.AndroidJUnitRunner"
    }

    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android-
optimize.txt'), 'proguard-rules.pro'
        }
    }
    compileOptions {
        sourceCompatibility JavaVersion.VERSION_1_8
        targetCompatibility JavaVersion.VERSION_1_8
    }
    buildToolsVersion '30.0.3'
    aaptOptions {
        noCompress "tflite"
    }
    buildFeatures {
        mlModelBinding true
        viewBinding true
    }
}

dependencies {
    implementation 'androidx.appcompat:appcompat:1.3.0'
}

```

```
implementation 'com.google.android.material:material:1.3.0'
implementation project(path: ':openCVLibrary343')
implementation 'org.tensorflow:tensorflow-lite-metadata:0.1.0'
implementation 'androidx.constraintlayout:constraintlayout:2.1.0'
implementation 'androidx.navigation:navigation-fragment:2.3.5'
implementation 'androidx.navigation:navigation-ui:2.3.5'
testImplementation 'junit:junit:4.+'
androidTestImplementation 'androidx.test.ext:junit:1.1.2'
androidTestImplementation 'androidx.test.espresso:espresso-core:3.3.0'
implementation 'com.github.tbruyelle:rxpermissions:0.10.1'
implementation "io.reactivex.rxjava2:rxjava:2.1.17"
implementation 'io.reactivex.rxjava2:rxandroid:2.0.1'
implementation 'pl.droidsonroids.gif:android-gif-drawable:1.2.19'

// Import tflite dependencies
implementation 'org.tensorflow:tensorflow-lite:2.4.0'
implementation 'org.tensorflow:tensorflow-lite-gpu:0.0.0-nightly-
SNAPSHOT'
implementation 'org.tensorflow:tensorflow-lite-support:0.2.0'
}
```

B.8 Experimental Results (Python)

[Done by: Ng Jan Hui]

```
#!/usr/bin/env python
# coding: utf-8

# In[7]:


import numpy as np


# In[8]:


svm_results = np.genfromtxt('SVM_res.csv')
rf_results = np.genfromtxt('Random_Forest_res.csv')
truth = np.genfromtxt('testLabels.csv')


# In[14]:


from sklearn.metrics import accuracy_score

test_acc = accuracy_score(truth, rf_results)
test_acc_svc = accuracy_score(truth, svm_results)

print("RF Testing Accuracy: {:.4f}".format(test_acc))
print("SVM Testing Accuracy: {:.4f}".format(test_acc_svc))


# In[15]:


from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import cmocean

labels = [1,2,3,4,5,6]

cm = confusion_matrix(truth, rf_results)
print(cm)

fig = plt.figure(dpi=150)
ax = fig.add_subplot(111)
cax = ax.matshow(cm, cmap = cmocean.cm.dense)

for (i, j), z in np.ndenumerate(cm):
```

```

    ax.text(j, i, '{}'.format(z), ha='center', va='center', color='red')

plt.title('Confusion matrix of Random Forest')
fig.colorbar(cax)
ax.set_xticklabels([''] + labels)
ax.set_yticklabels([''] + labels)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()

# In[11]:


from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import cmocean

labels = [1,2,3,4,5,6]

cm = confusion_matrix(truth, svm_results)
print(cm)

fig = plt.figure(dpi=150)
ax = fig.add_subplot(111)
cax = ax.matshow(cm, cmap = cmocean.cm.dense)

for (i, j), z in np.ndenumerate(cm):
    ax.text(j, i, '{}'.format(z), ha='center', va='center', color='red')

plt.title('Confusion matrix of Linear SVM')
fig.colorbar(cax)
ax.set_xticklabels([''] + labels)
ax.set_yticklabels([''] + labels)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()

# In[12]:


from sklearn.metrics import precision_score, recall_score, f1_score

print('precision = ', precision_score(truth, rf_results, average='micro'))
print('recall = ', recall_score(truth, rf_results, average='micro'))
print('f1 score = ', f1_score(truth, rf_results, average='micro'))

```

```
# In[13]:
```

```
from sklearn.metrics import precision_score, recall_score, f1_score

print('precision = ', precision_score(truth, svm_results, average='micro'))
print('recall = ', recall_score(truth, svm_results, average='micro'))
print('f1 score = ', f1_score(truth, svm_results, average='micro'))
```

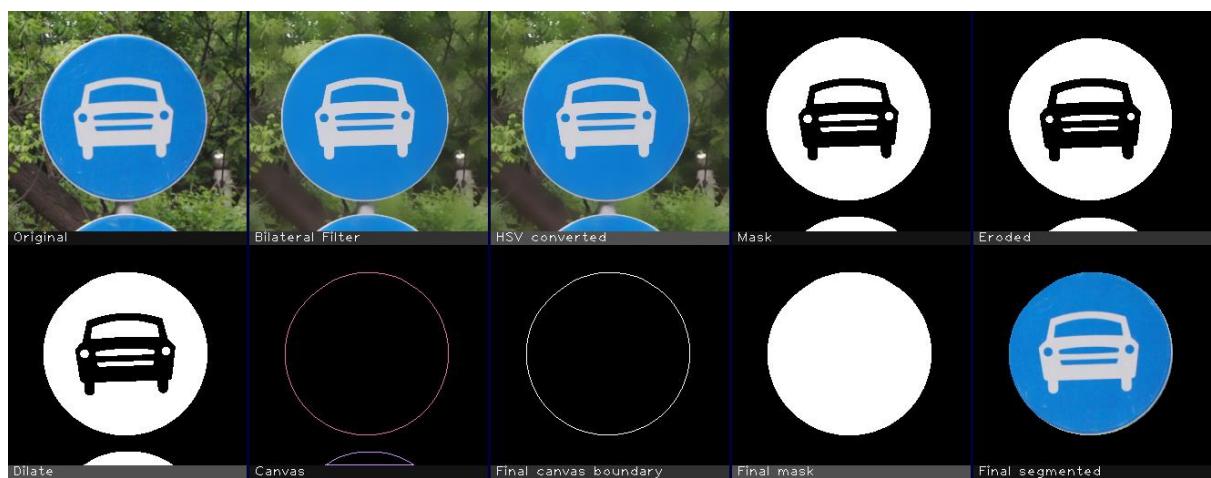
C Image pre-processing flow

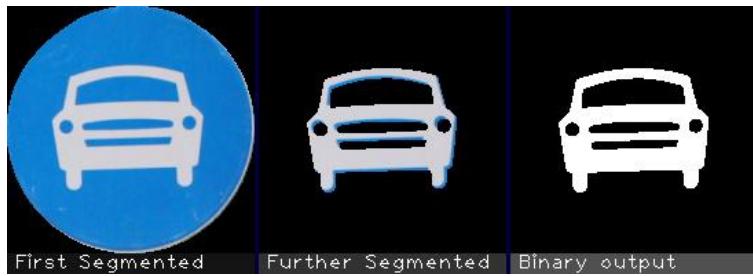
[Done by: Tan Jing Jie]

C.1 No horn sign

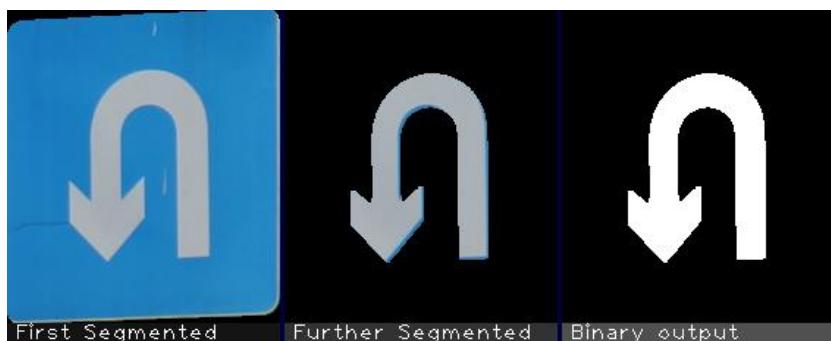
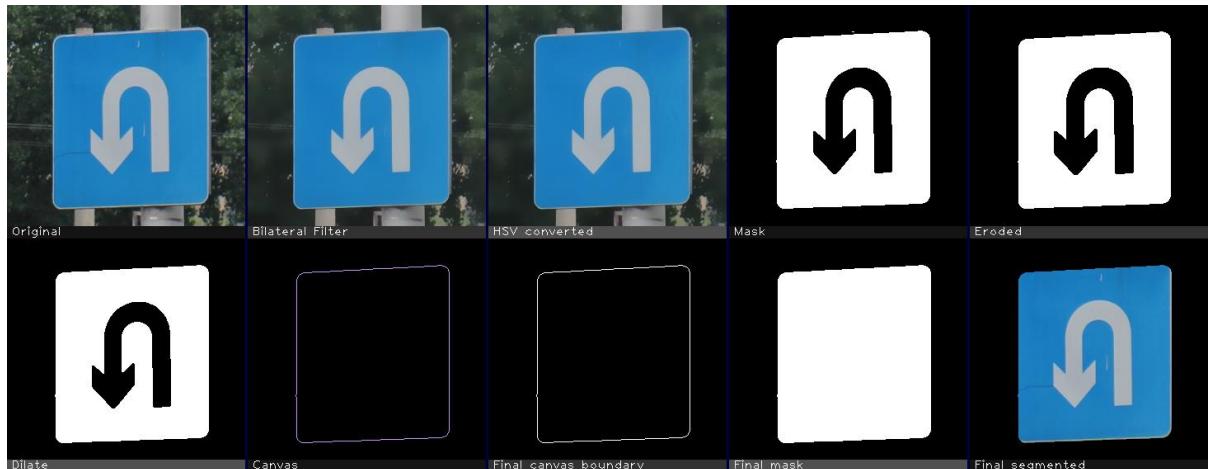


C.2 Car sign





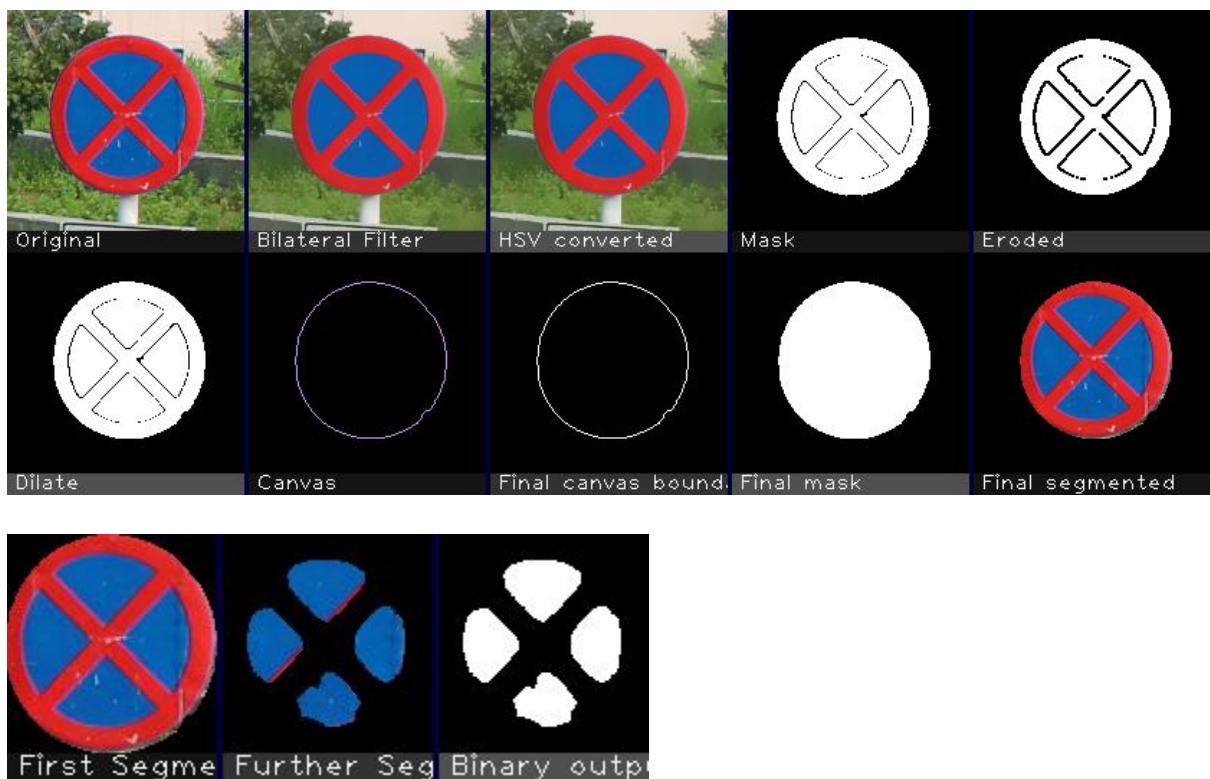
C.3 U-turn



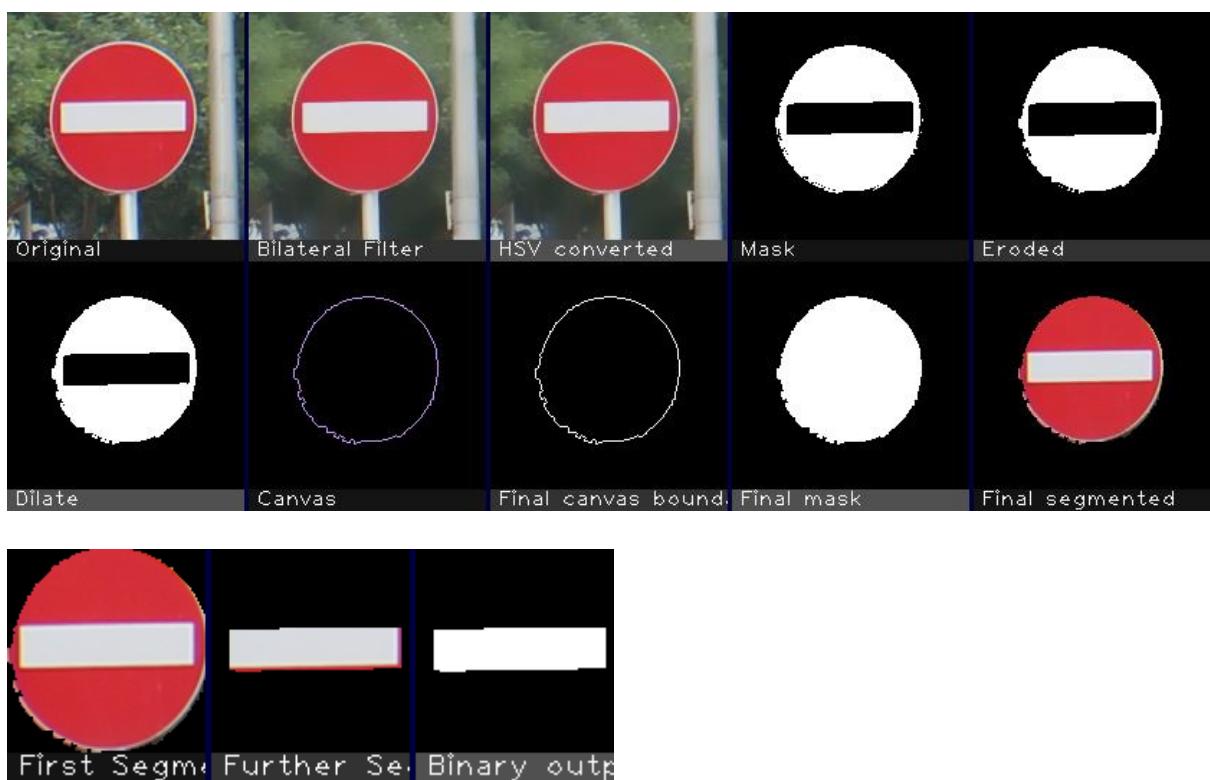
C.4 Crossing



C.5 No stop

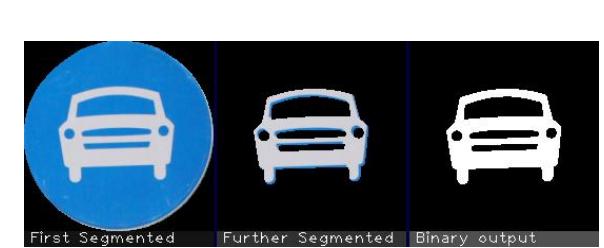
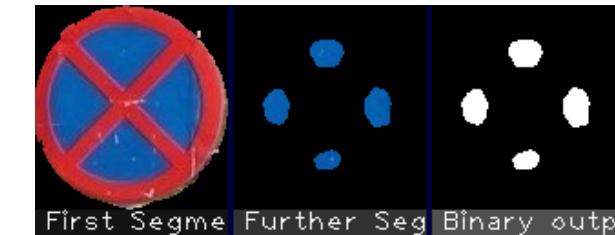
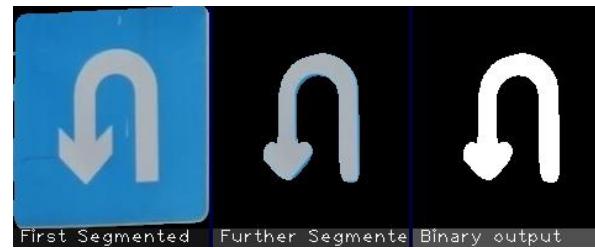
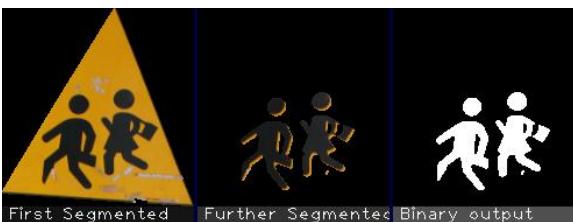
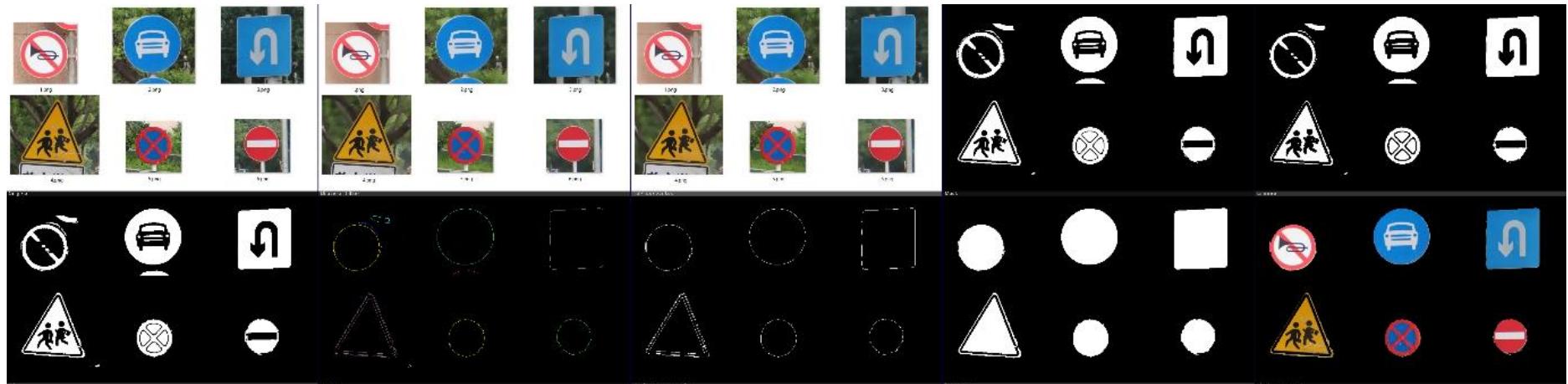


C.6 No entry

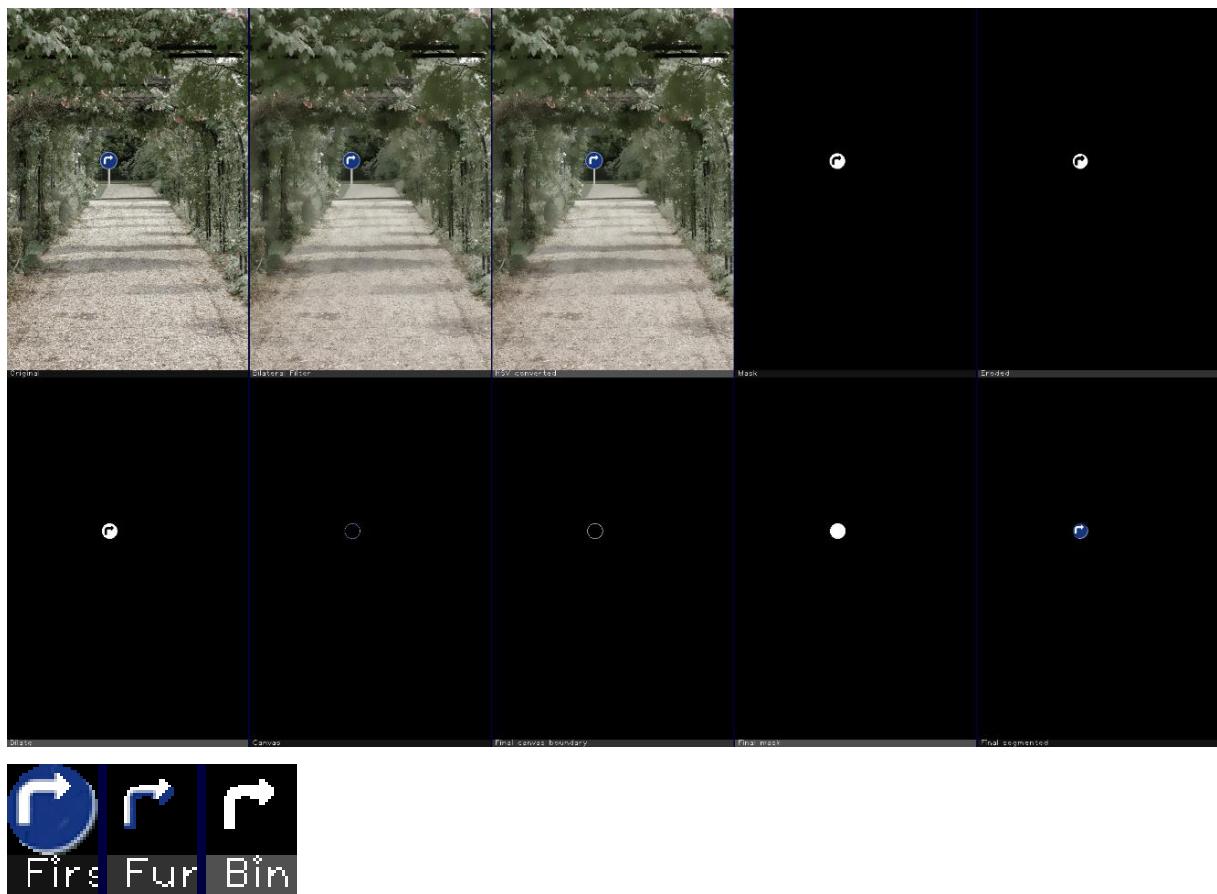


C-4

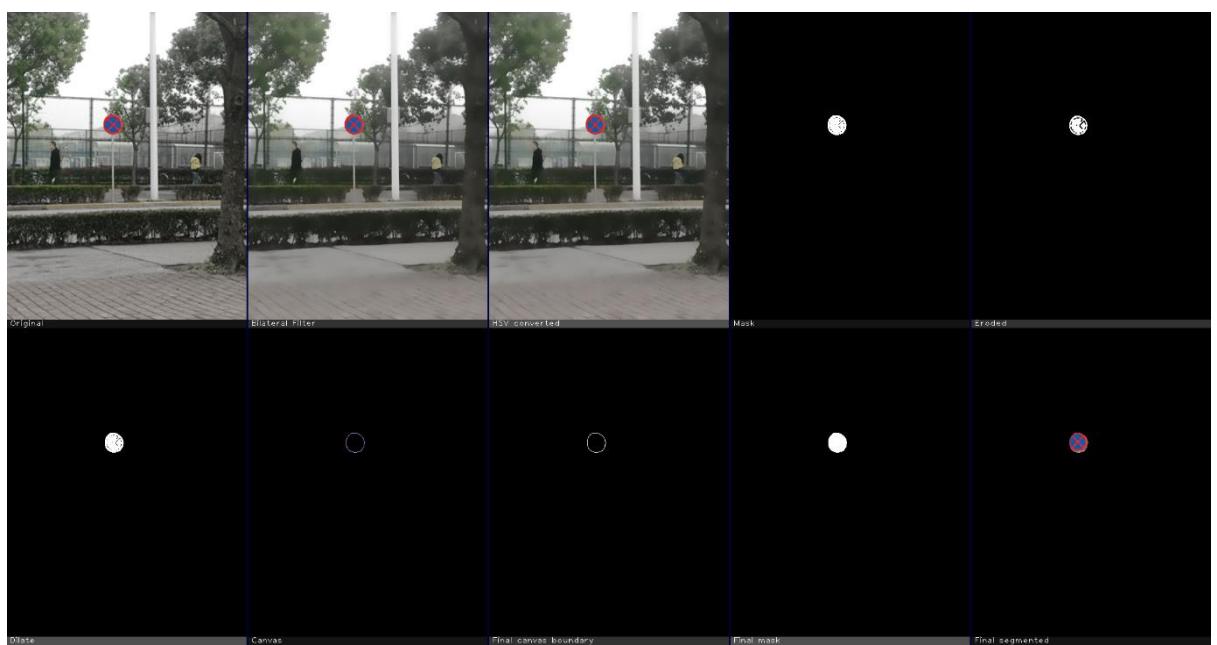
C.7 Multiple traffic sign. (Combination of six traffic sign)



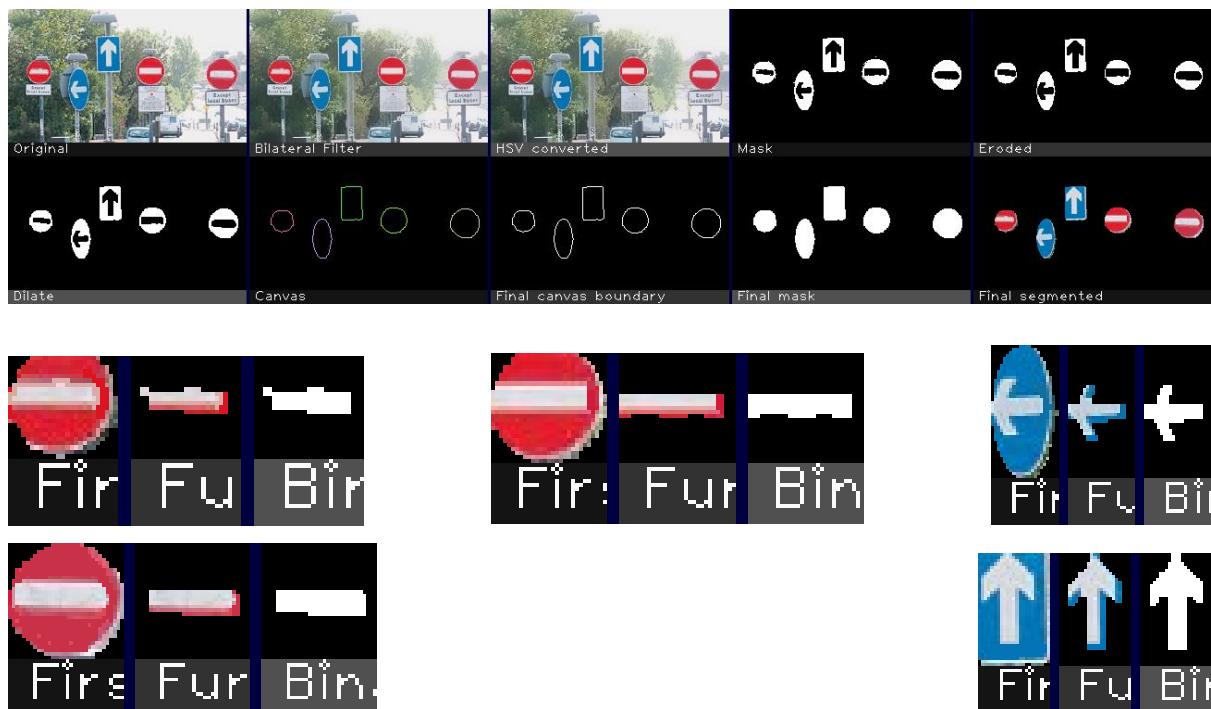
C.8 Small traffic sign (blue)



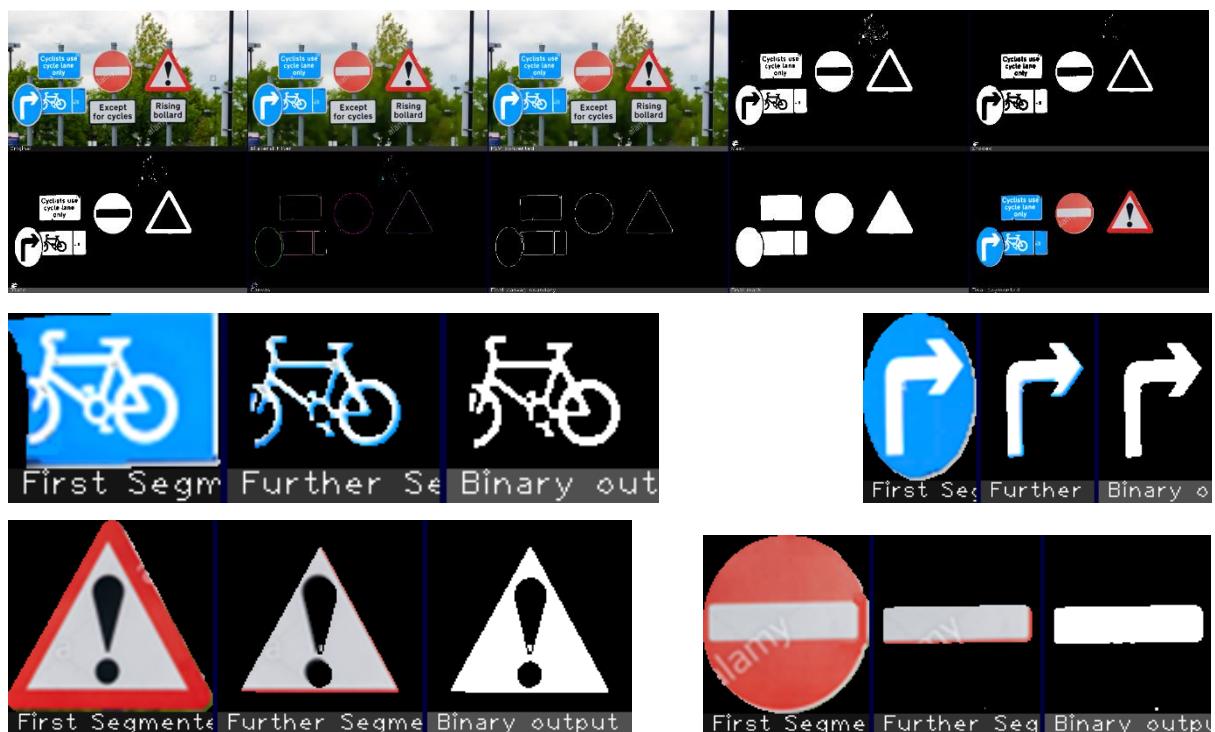
C.9 Small traffic sign (red).



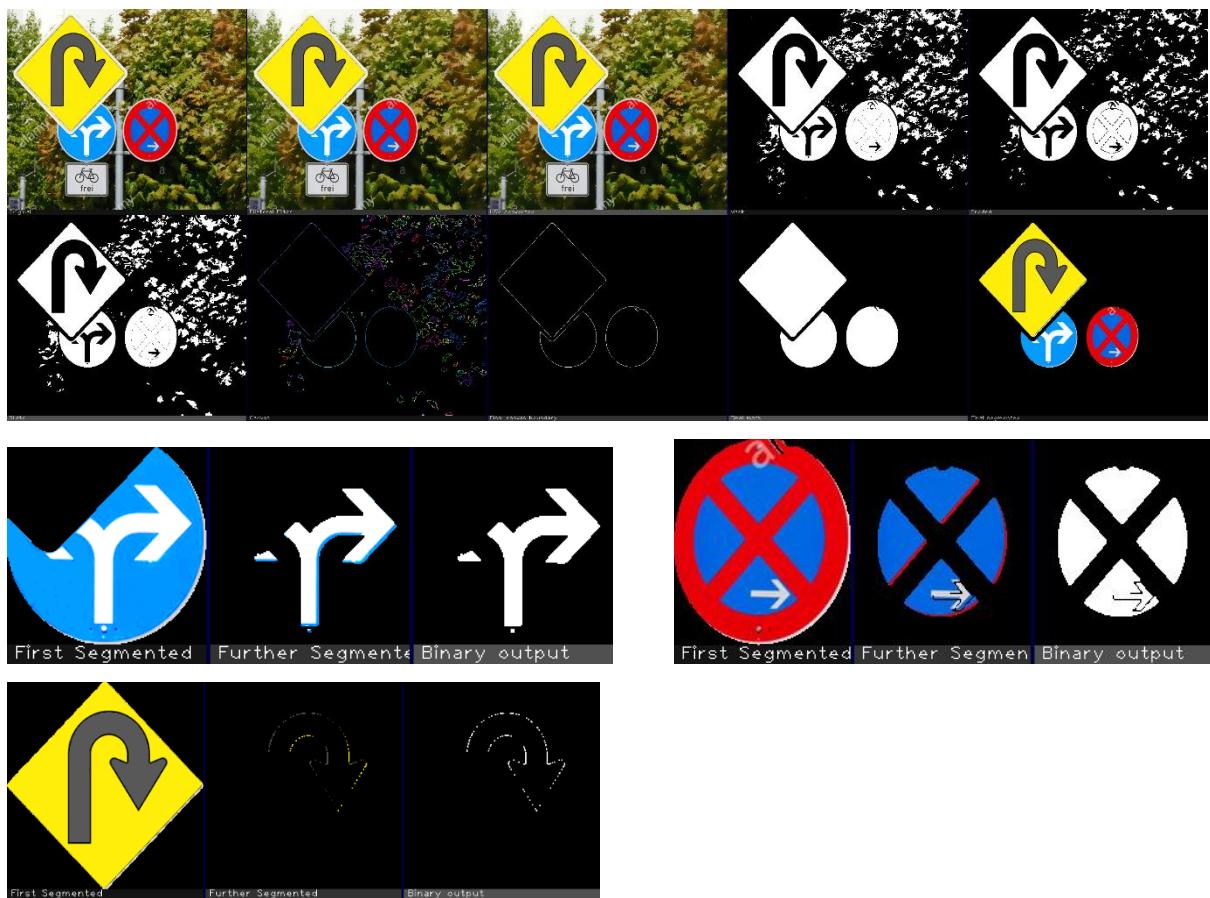
C.10 Multiple traffic sign. (Real environment and overlapped traffic sign)



C.11 Multiple traffic sign. (Real environment and overlapped traffic sign)

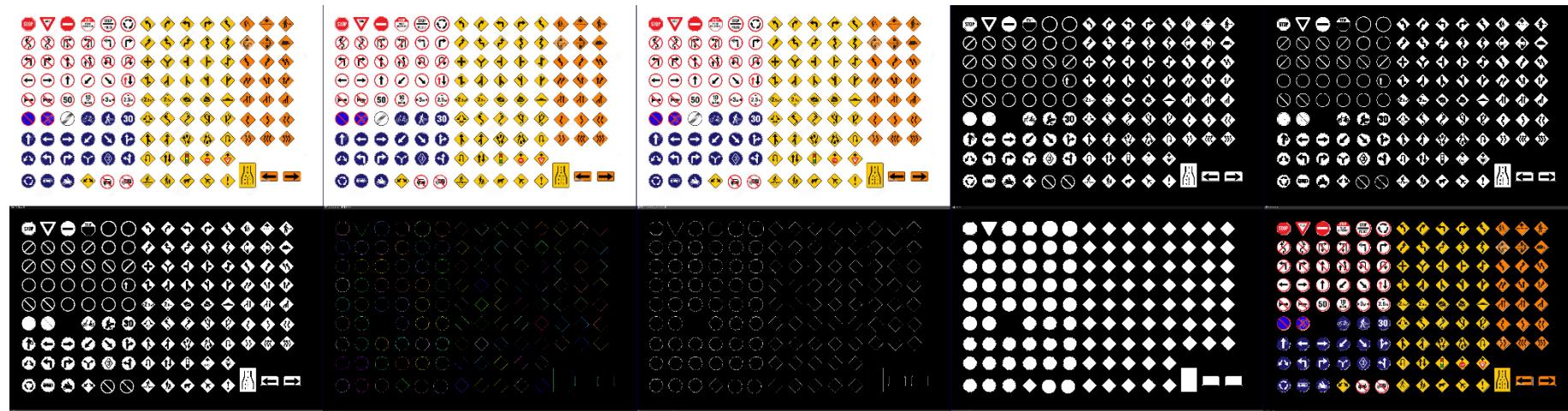


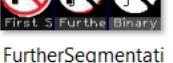
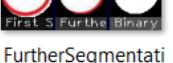
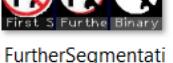
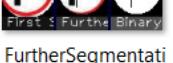
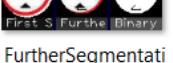
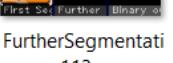
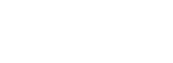
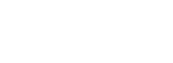
C.12 Multiple traffic sign. (Real environment and overlapped traffic sign)



C.13 Multiple traffic sign. (Check the image preprocessing coverage)

The flow from input to Segmented and Further Segmented



 First S Furthe Binary	 First S Furthe Binary	 First S Furthe Binary	 First S Furthe Binary	 First S Furthe Binary	 First S Furthe Binary	 First S Furthe Binary	 First S Furthe Binary
FurtherSegmentation64.png	FurtherSegmentation65.png	FurtherSegmentation66.png	FurtherSegmentation67.png	FurtherSegmentation68.png	FurtherSegmentation69.png	FurtherSegmentation70.png	FurtherSegmentation71.png
 First S Furthe Binary	 First S Furthe Binary	 First S Furthe Binary	 First S Furthe Binary	 First S Furthe Binary	 First S Furthe Binary	 First S Furthe Binary	 First S Furthe Binary
FurtherSegmentation72.png	FurtherSegmentation73.png	FurtherSegmentation74.png	FurtherSegmentation75.png	FurtherSegmentation76.png	FurtherSegmentation77.png	FurtherSegmentation78.png	FurtherSegmentation79.png
 First S Furthe Binary	 First S Furthe Binary	 First S Furthe Binary	 First S Furthe Binary	 First S Furthe Binary	 First S Furthe Binary	 First S Furthe Binary	 First S Furthe Binary
FurtherSegmentation80.png	FurtherSegmentation81.png	FurtherSegmentation82.png	FurtherSegmentation83.png	FurtherSegmentation84.png	FurtherSegmentation85.png	FurtherSegmentation86.png	FurtherSegmentation87.png
 First S Furthe Binary	 First S Furthe Binary	 First S Furthe Binary	 First S Furthe Binary	 First S Furthe Binary	 First S Furthe Binary	 First S Furthe Binary	 First S Furthe Binary
FurtherSegmentation88.png	FurtherSegmentation89.png	FurtherSegmentation90.png	FurtherSegmentation91.png	FurtherSegmentation92.png	FurtherSegmentation93.png	FurtherSegmentation94.png	FurtherSegmentation95.png
 First S Furthe Binary	 First S Furthe Binary	 First S Furthe Binary	 First S Furthe Binary	 First S Furthe Binary	 First S Furthe Binary	 First S Furthe Binary	 First S Furthe Binary
FurtherSegmentation96.png	FurtherSegmentation97.png	FurtherSegmentation98.png	FurtherSegmentation99.png	FurtherSegmentation100.png	FurtherSegmentation101.png	FurtherSegmentation102.png	FurtherSegmentation103.png
 First S Furthe Binary	 First S Furthe Binary	 First S Furthe Binary	 First S Furthe Binary	 First S Furthe Binary	 First S Furthe Binary	 First S Furthe Binary	 First S Furthe Binary
FurtherSegmentation104.png	FurtherSegmentation105.png	FurtherSegmentation106.png	FurtherSegmentation107.png	FurtherSegmentation108.png	FurtherSegmentation109.png	FurtherSegmentation110.png	FurtherSegmentation111.png
 First S Furthe Binary	 First S Furthe Binary	 First S Furthe Binary	 First S Furthe Binary	 First S Furthe Binary	 First S Furthe Binary	 First S Furthe Binary	 First S Furthe Binary
FurtherSegmentation112.png	FurtherSegmentation113.png	FurtherSegmentation114.png	FurtherSegmentation115.png	FurtherSegmentation116.png	FurtherSegmentation117.png	FurtherSegmentation118.png	FurtherSegmentation119.png
 First S Furthe Binary	 First S Furthe Binary	 First S Furthe Binary	 First S Furthe Binary	 First S Furthe Binary	 First S Furthe Binary	 First S Furthe Binary	 First S Furthe Binary
FurtherSegmentation120.png	FurtherSegmentation121.png						

**** This page intentionally leave blank. ****