

TRAFFIC SIGN SEGMENTATION

By

LEE JING YONG

LAI WEI BIN

HAR SZE HAO

TOH YUN SHUANG

A PROPOSAL

SUBMITTED TO

Universiti Tunku Abdul Rahman

in partial fulfillment of the requirements

for the degree of

BACHELOR OF COMPUTER SCIENCE (HONOURS)

Faculty of Information and Communication Technology

(Kampar Campus)

MAY 2024

ABSTRACT

Done by : Har Sze Hao

Traffic sign segmentation is crucial for computer vision and autonomous driving. Accurate detection and localization of traffic signs enable safe and efficient driving. This paper proposes a new method for segmenting traffic signs using color spaces and image processing techniques. Our method begins by collecting user-provided images containing traffic signs. These images undergo preprocessing, including resizing and noise reduction. The core step involves converting the preprocessed images to the HSV color space, which better emphasizes color differences between traffic signs and their backgrounds.

We define specific HSV threshold ranges for yellow, blue, and red traffic signs. By applying these color thresholds in the HSV space, we effectively segment traffic signs from the background. For red signs, we use two threshold ranges to accommodate the circular nature of red in the HSV space.

After segmentation, we use contour detection to identify the longest contour, presumed to be the traffic sign boundary. We then generate a complete sign mask using a flood fill algorithm. Finally, we apply this mask to the original image to extract the segmented color traffic sign.

Our method not only accurately segments traffic signs but also preserves their original color information. Results are presented as side-by-side displays of original and segmented images for easy visual comparison. Additionally, the segmented images are saved for further analysis.

Experimental results show that the method performs particularly well on yellow signs, demonstrating the effectiveness of color space analysis and threshold techniques for accurate traffic sign detection. This approach adapts to the unique color characteristics of traffic signs under different conditions, providing a robust solution for intelligent and safe transportation systems.

Future work could focus on improving segmentation for other color signs and integrating machine learning techniques to enhance robustness in complex scenarios.

TABLE OF CONTENT

ABSTRACT	2
LIST OF FIGURES	5
LIST OF TABLES	9
LIST OF ABBREVIATIONS	10
CHAPTER 1 INTRODUCTION.....	12
1.1 Motivation and Problem Statement	12
1.2 Project Scope	13
1.3 Project Objective.....	13
1.4 Impact, significance and contribution	15
1.5 Background information	15
CHAPTER 2 LITERATURE REVIEW	17
2.1 A Lightweight Traffic Sign Recognition Model Based on Improved YOLOv5	17
2.2 A robust, coarse-to-fine traffic sign detection method.....	21
2.3 Object detection based on RGC mask R-CNN. IET Image Processing	25
2.4 Simultaneous Traffic Sign Detection and Boundary Estimation Using Convolutional Neural Network	29
2.5 Comparison of the 4 techniques	33
CHAPTER 3 SYSTEM METHODOLOGY	35
3.1 Design specification.....	35
3.2 System Block Diagram and Description	36
CHAPTER 4 SYSTEM DESIGN.....	40
4.1 Data Set.....	40
4.2 Segmentation	40
4.3 Feature Extraction	43
4.4 Classifier.....	44
4.5 Testing and Evaluation	45
CHAPTER 5 SYSTEM IMPLEMENTATION	47
5.1 Image Acquisition	47
5.2 Segmentation	47
5.3 Feature Extraction	48
5.4 Feature Storage and Use.....	49

5.6	Testing and Evaluation	50
5.7	Setting and Configuration	51
5.8	System Operation (with Screenshot).....	52
5.9	Implementation Issues and Challenges	54
5.10	Concluding Remark	55
CHAPTER 6 SYSTEM EVALUATION AND DISCUSSION.....		57
6.1	System Testing and Performance Metrics.....	57
6.2	Testing Setup and Result.....	59
6.3	Performance Comparison	63
6.4	Project Challenges.....	64
6.5	Objectives Evaluation	65
6.6	Error Analysis	66
6.7	Novelties of Work.....	66
6.8	Concluding Remark	67
CHAPTER 7 CONCLUSION		69
7.1	Project Review.....	69
7.2	Discussion and Conclusion	69
7.3	Future Work.....	70
REFERENCES.....		71
Appendices.....		73

PLAGIARISM CHECK RESULT

Place your Turnitin plagiarism checking result here.

LIST OF FIGURES

Figure Number	Title	Page
Figure 2.1.1	Original YOLOv5 model	17

Figure 2.1.2	Improved YOLOv5 model	18
Figure 2.1.3	Benchmark results of YOLOv5 different versions	19
Figure 2.1.4	Performance comparison with different YOLO model	20
Figure 2.2.1	Block diagram of the proposed method	22
Figure 2.2.2	Block diagram of danger sign detection	22
Figure 2.2.3	Examples of perspective adjustment	23
Figure 2.2.4	Block diagram of mandatory sign detection	24
Figure 2.3.1	RGC Mask R-CNN.	25
Figure 2.3.2	Results of the RGC Mask R-CNN on the COCO test set	26
Figure 2.3.3	Comparison for AP score of object detection for different backbone	27
Figure 2.3.4	Comparison for AP score of instance segmentation for different backbone	27
Figure 2.3.5	Comparison for AR score of object detection for different backbone.	28
Figure 2.3.6	Comparison for AR score of instance segmentation for different backbone.	28
Figure 2.4.1	Boundary estimation using a predicted 2D pose and a shape label	29
Figure 2.4.2	The overall procedure of the proposed method	30

Figure 2.4.3	Box and pose prediction using regression vectors	31
Figure 2.4.4	Examples of traffic sign template images and their boundary corners	32
Figure 3.2.1	Block diagram of system	36
Figure 4.2.1	Result of deduction	41
Figure 4.2.2	Final segmented result	42
Figure 4.2.3.1	Segmentation process	42
Figure 4.2.3.2	Segmentation process	42
Figure 4.2.3.3	Segmentation process	42
Figure 4.5.1	Accuracy	46
Figure 5.8.1	Main menu options	52
Figure 5.8.2	Model training process	52
Figure 5.8.3	Saving features and model trained	52
Figure 5.8.4	Confusion matrix for training data	53
Figure 5.8.5	Example of correct identification	53
Figure 5.8.6	Example of incorrect identification	54
Figure 5.8.7	Error handling on invalid input	54
Figure 6.1.1	Formula for the accuracy	57
Figure 6.1.2	Example of confusion matrix	58
Figure 6.1.3	Formula of precision and recall	58

Figure 6.1.4	Formula of F1 Score	58
Figure 6.2.1.1	Accuracy result of HOG with SVM classification	60
Figure 6.2.1.2	Accuracy result of Color HOG with Random Forest classification	61
Figure 6.2.2.1	Confusion matrix for SVM classification	62
Figure 6.6.1	Misclassification of traffic sign	66

LIST OF TABLES

Table Number	Title	Page
Table 2.2	Shapes and colours of the three categories	21
Table 3.1.2.1	Specification of hardware for developing and testing	35
Table 3.1.3.1	Specification of software for developing and testing	36
Table 5.7.1	Software and libraries used	51
Table 5.7.2	Dataset settings	51
Table 5.7.3	Preprocessing steps and conditions	51
Table 5.7.4	Hyperparameters for models	51

LIST OF ABBREVIATIONS

ADAS **Advanced Driver-Assistance Systems**

<i>AEs</i>	Auto-Encoders	
<i>AI</i>	Artificial Intelligence	
<i>AP</i>	Average Precision	
<i>AR</i>	Average Recall	
<i>CBAM</i>	Convolutional Block Attention Module	

CGI **Computer-Generated Imagery**

CNNs **Convolutional Neural Networks**

<i>COCO</i>	Common Objects in Context	
--------------------	----------------------------------	--

CSV **Comma-Separated Values**

FN **False Negative**

FP **False Positive**

GANs **Generative Adversarial Networks**

<i>GN</i>	Group Normalization	
<i>GTSDb</i>	German Traffic Sign Detection Benchmark Dataset	
<i>HOG</i>	Histogram of oriented gradient	
<i>HSV</i>	Hue, Saturation, and Value	
<i>IDE</i>	Integrated Development Environment	

<i>IoU</i>	Intersection-Over-Union	
<i>LDA</i>	Linear Discriminant Analysis	
<i>LIDAR</i>	Light Detection and Ranging	
<i>mAP</i>	Mean Average Precision	
<i>NMS</i>	N-Maximal Suppression	
<i>OCR</i>	Optical Character Recognition	
<i>PNG</i>	Portable Network Graphic	
<i>RBF</i>	Radial Basis Function	
<i>R-CCN</i>	Region-based Convolutional Neural Network	
<i>RGB</i>	Red, Green, Blue	
<i>RGC</i>	ResNet Group Cascade	
<i>ROIs</i>	Regions of Interest	
<i>RPN</i>	Region Proposal Network	
<i>SSD</i>	Single Shot MultiBox Detector	
<i>STSB</i>	Simultaneous Traffic Sign Detection and Boundary Estimation	
<i>SVM</i>	Support Vector Machine	
<i>TN</i>	True Negative	
<i>TP</i>	True Positive	

TSR **Traffic Sign Recognition**

VJ

Viola-Jones

YOLO

You Look Only Once

CHAPTER 1 INTRODUCTION

1.1 Motivation and Problem Statement

Done by: Har Sze Hao

The development of self-driving vehicles is seen to bring about a future that is safer, more efficient, and more accessible with means of transportation. Perhaps one thing that deems very important is the reliable identification and detection of traffic signs that are used in controlling the guidance behavior of the vehicle on roadways. Precise traffic sign detection is thus a crucial problem for autonomous driving to assure protection and law compliance. We aim to contribute significantly to the field by building a strong system for the automatic detection of traffic signs. Our design will utilize the most current techniques in color and shape detection. This project will not only support the development of autonomous vehicles but also spur innovation and foster the growth of research capabilities within our team and enhance our career prospects.

Legible traffic signs are necessary for autonomous vehicles to operate safely and legally. Despite variations in sign colors, shapes, lighting conditions, and environmental factors, the detection of traffic signs continues to be a challenging area. Current solutions often do not achieve high accuracy or robustness in diverse conditions. Our team proposes a superior system design for traffic sign detection by integrating state-of-the-art techniques in color and shape recognition, along with focused in-depth research into the most promising and recent approaches. We also plan to explore additional methods to enhance and strengthen traffic sign recognition beyond color and shape detection. Our preliminary work provides proof of the feasibility and effectiveness of the proposed design, laying the groundwork for further development and potential financing.

1.2 Project Scope

Done by: Lee Jing Yong

The scope of the project is to conduct research and development to detect traffic sign with high consistency and efficiency for autonomous vehicles. The proposed system shall be able to detect traffic signs in real time while the vehicle is moving regardless of any weather conditions or partial blockage. A suitable dataset of images of the traffic signs shall be selected and prepared for testing the model. After all the necessary analysis and setup, we can start to test the model and examine the accuracy of the model. Tweaks and fine tuning are then done on the model to ensure the model is made to the requirements of the proposal. Once the requirements are met, the proposed solution is ready for deployment.

1.3 Project Objective

Done by: Lee Jing Yong

The first objective of the project is to segment traffic signs from images based on colors and shapes. To use colors to segment the traffic signs, we will need to identify the color range for the traffic signs and for shapes, we will have to identify the methods to remove background noises so that the edges in the images are highlighted correctly. The two methods includes careful considerations of the preprocessing steps required. The second objective of the project is to develop a machine learning model which can accurately identify traffic signs in real time for autonomous vehicles. The project however doesn't cover the identification of obstacles, humans and road structure. An analysis shall be done to choose a suitable model that can consistently and efficiently detect objects fast and accurately as is model is to be used on autonomous vehicles and needs to detect traffic signs in real time. The dataset used to train the model shall include normal images and high difficulty traffic sign images such as low resolution, raining, foggy, partially blocked and signs with wear and tear. The difficult part of the detection is to properly identify the traffic signs in real time when the image is unclear which the project shall tackle. Once the training of the model is completed, the model will be tested with a test set to ensure that it is able to accurately detect all

the 84 test images in under 2 seconds even in low specification computers. If the requirements are not met, the model will be tweaked and fine-tuned till it achieves the requirements.

1.4 Impact, significance and contribution

Done by: Lai Wei Bin

Nowadays, traffic sign detection, segmentation, and classification methods have been studied for many years and for several purposes. For example, traffic sign recognition (TSR) is an important part of some advanced driver-assistance systems (ADASs) and auto-driving systems (ADSs).[1] Traffic sign detection plays a critical role in enhancing road safety, improving traffic management, and advancing autonomous driving technologies. Providing real-time information such as hazards and traffic rules to drivers or autonomous systems significantly reduces accidents caused by human mistakes or missed signals. The integration of traffic sign detection into ADAS and ADSs ensures that these vehicles can navigate roads safely, adhere to traffic laws, and respond appropriately to various road conditions. TSR is important to both systems because both are critical systems which means that even minor errors or incorrect instructions in the system, may lead to serious safety issues. Furthermore, the adoption of traffic sign detection systems contributes to the development of smart city infrastructure, leading to more efficient traffic management, reduced congestion, and valuable data collection for traffic analysis and urban planning.

Traffic sign detection plays a crucial role beyond ensuring immediate safety on the roads. It drives research and development in machine learning, computer vision, and artificial intelligence, leading to technological innovations that have applications in various fields such as healthcare, security, and robotics. These systems also offer significant economic benefits, reducing the need for human traffic monitoring and potentially lowering insurance premiums through enhanced road safety. As these systems become more advanced and widespread, their contributions to legal compliance, technological progress, and cross-disciplinary applications will continue to grow, emphasizing their importance in modern transportation and urban development.

1.5 Background information

Done by: Toh Yun Shuang

The detection and recognition of traffic sign is using the techniques of computer vision and machine learning techniques. Computer vision is a field of artificial intelligence (AI) that focuses on enabling computers to detect, understand, and derive information from images and videos, such as objects and people. By utilizing images from camera and videos, machines can accurately identify and classify objects, and then react to what they “see” just like a human vision system.

Deep Learning methods are currently the state of-the-art in many Computer Vision and Image Processing problems, in particular image classification. After years of intensive investigation, several models have matured and become essential tools. This includes Convolutional Neural Networks (CNNs), Siamese and Triplet Networks, Auto-Encoders (AEs) and Generative Adversarial Networks (GANs). [3] The key techniques and algorithms in computer vision include image processing, feature extraction, object detection and recognition, image segmentation, image classification, generative models.

Computer vision is not only can used on traffic signs detection, but in real world, it also widely used on Optical Character Recognition (OCR), Machine inspection, Retail, 3D models building, medical imaging, Automotive safety, Match move, motion capture, surveillance, and fingerprint recognition and biometrics. [4] Retail represents the automated checkout machines that used in the shopping mall, where computer vision will perform object recognition to scan the objects we want to pay for. Match move represents a technique that widely used in movies such as Jurassic Park, where the computer vision tracks feature points in a source video to estimate the 3D camera motion and shape of the environment and fuse computer-generated imagery (CGI) with live action footage. [4]

However, there will also have some challenges to computer vision. Computer vision system has limited performance and function, therefore, cannot be expected to replicate just like the human eye. For examples, the challenges of the sensitivity of the parameters, the strength of the algorithms and the accuracy of the result. [2] Besides that, computer vision relies on analyzing the collected visual data, therefore, the poor quality of input images data such as poor lighting, backgrounds clutter, and occlusions can affect the performance of the system.[5]

CHAPTER 2 LITERATURE REVIEW

2.1 A Lightweight Traffic Sign Recognition Model Based on Improved YOLOv5

Done by: Lee Jing Yong

The model You Look Only Once which is known as the YOLO model [7], have been popular in the past few years due to its unique object detection technique. There are several versions of the YOLO model which have been tweaked for a better performance. The most recent version of YOLO is version 9 which have been released recently back in February 2024[8]. The YOLO model consists of 4 different parts which are input, backbone, neck and head or the output. The article “A Lightweight Traffic Sign Recognition Model Based on Improved YOLOv5” [6] suggests YOLOv5 model to recognize traffic signs. The early version of the YOLO model requires high computing power and takes in a lot of parameters to detect the objects. In the improved version, Ghost and C3Ghost model were added into the neck network of the YOLOv5 model. Figure 2.1.1.1 below shows the original YOLOv5 model and Figure 2.1.1.2 below shows the modified YOLOv5 model.

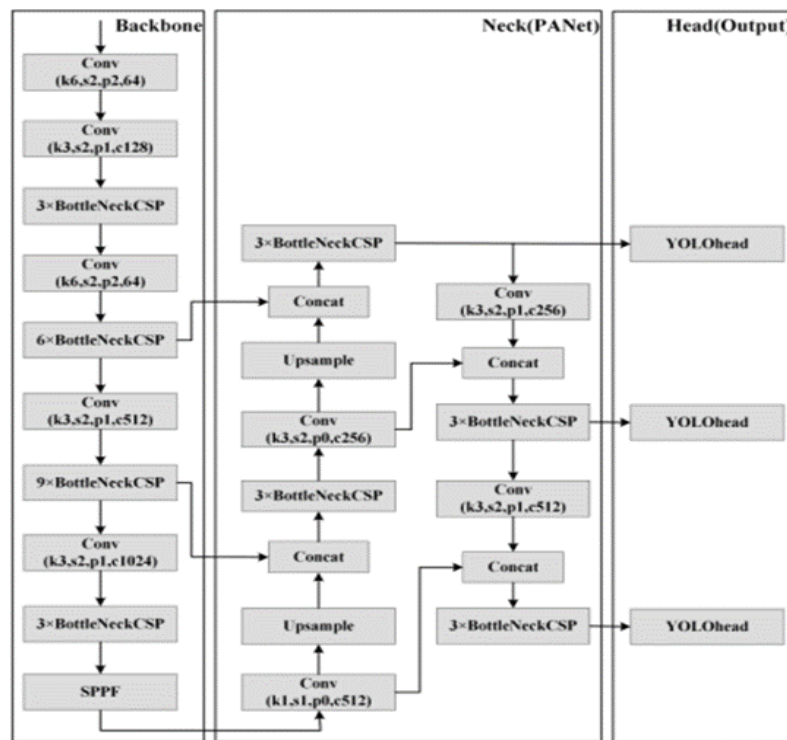


Figure 2.1.1 Original YOLOv5 model

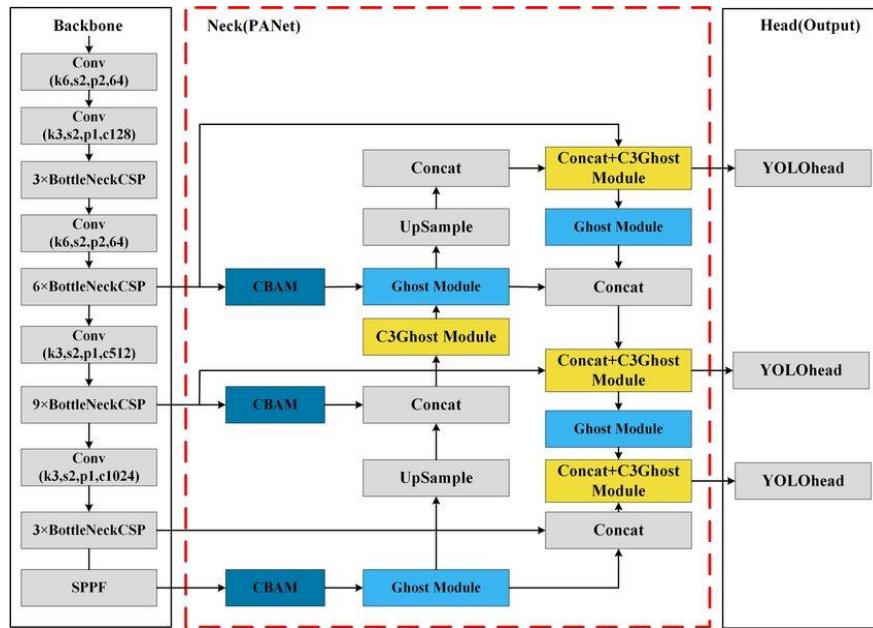


Figure 2.1.2 Improved YOLOv5 model

The aim of the addition was to reduce the cost of computing and the number of parameters, however this addition had some flaws such that the extracted data from the image is incomplete which ultimately reduces the accuracy of the detection. To overcome this flaw, CBAM attention module has been added to the backbone of the YOLO model. The attention module is used to extract information from the input to make up of the data lost and retain the accuracy of the detection with only little computation power [6]. The research clearly explained CBAM (Convolutional Block Attention Module) attention mechanism including how it works. A proper benchmark between multiple modification to the YOLOv5 model was made in the research. Figure 2.1.1.3 below shows the benchmark results.

TABLE 3. Performance of different models.

Model	mAP(0.5)	mAP(0.5:0.95)	Speed-GPU(ms)	Params	FLOPS(G)	Weights(M)
YOLOv5s	96.8%	73.5%	3.6	7027720	16	13.7
YOLOv5s_all	96.5%	72.4%	3.3	3681120	8.0	7.4
YOLOv5s_backbone	96.4%	72.9%	3.3	5084368	10.6	10.0
YOLOv5s_neck	96.6%	73.4%	3.4	5614968	13.3	11.0
Ours	98.3%	76.7%	3.5	6035102	14.1	11.8

TABLE 4. Performance of different attention modules.

Model	mAP(0.5)	mAP(0.5:0.95)	Speed-GPU(ms)	Params	FLOPS(G)	Weights(M)
YOLOv5s_ECA	98.1%	76.6%	3.6	6001025	13.7	11.8
YOLOv5s_CA	98.1%	76.8%	3.6	6003929	14.2	11.8
YOLOv5s_SE	98.0%	76.6%	3.5	6044024	14.2	11.8
YOLOv5s_CBAM	98.3%	76.7%	3.5	6035102	14.1	11.8

TABLE 5. Performance of different loss functions.

Model	mAP(0.5)	mAP(0.5:0.95)	Speed-GPU(ms)	Params	FLOPS(G)	Weights(M)
YOLOv5s_GIoU	96.8%	73.5%	3.6	7027720	16	13.7
YOLOv5s_DIoU	97.7%	76.1%	2.8	6043318	14.1	11.8
YOLOv5s_CIoU	97.7%	76.1%	2.8	6044318	14.2	11.8
YOLOv5s_EIoU	98.3%	76.7%	3.5	6035102	14.1	11.8

Figure 2.1.3 Benchmark results of YOLOv5 different versions

From the results, the paper was able to prove that their solution works and have better performance compared to the original YOLOv5. YOLOv5 is the original model, all is the modified model where all parts are replaced with Ghost model, backbone and neck indicates that only the parts are replaced with Ghost model and finally the proposed improved model. mAP represents the mean average precision; a higher value indicates that the model have a higher accuracy. Of all the different implementations done in the benchmarks, the researchers' method appears to have the best performance. Other than comparing with modifications to the YOLOv5 model, comparison to other similar models were also made. The results are shown in Figure 2.1.1.4 below.

TABLE 7. Performance comparison with other related models.

Model	mAP(0.5)	mAP(0.5:0.95)	Speed-GPU(ms)	Params	FLOPS(G)	Weights(M)
YOLOv5s	96.8%	73.5%	3.6	7027720	16	13.7
YOLOv3-tiny	83.4%	53.5%	2.9	8674496	13	16.6
YOLOv4-tiny	85.5%	59.2	3.1	5864632	16.2	11.3
YOLOv7-tiny	84.5%	53.8%	3.4	6019432	13.4	11.6
Ours	98.3%	76.7%	3.5	6035102	14.1	11.8

TABLE 8. Performance comparison of different lightweight models.

Model	mAP(0.5)	mAP(0.5:0.95)	Speed-GPU(ms)	Params	FLOPS(G)	Weights(M)
YOLOv5s	96.8%	73.5%	3.6	7027720	16	13.7
YOLOv5s_MoblieNetv3-S	90.1%	61.7%	2.6	1382846	2.3	3.9
YOLOv5s_MoblieNetv3-L	93.5%	66.3%	3.9	2913366	5.2	5.9
YOLOv5s_ShuffleNetv2	90.3%	60.1%	3.0	1362847	3.0	3.2
Ours	98.3%	76.7%	3.5	6035102	14.1	11.8

Figure 2.1.4 Performance comparison with different YOLO model

We can see that the proposed model still stands out even when compared to other similar models. The research had proven their solution is the best with the benchmark results. This indicates that YOLO model can work well on detecting traffic signs.

2.2 A robust, coarse-to-fine traffic sign detection method

Done by: Lai Wei Bin

In this paper, it stated several traffic sign detection methods such as Histogram of oriented gradient (HOG) and a coarse-to-fine sliding window scheme. It begins by detecting candidate regions of interest (ROIs) with a small-sized window and then verifies these ROIs with a larger window for increased accuracy.

In [9], the sliding window scheme has gained popularity in object detection. Notable examples include the histogram of oriented gradient (HOG) and the Viola-Jones (VJ) methods. Both approaches have been successfully applied to detect traffic sign and yielding good results. It proposed a coarse-to-fine method based on HOG.

TABLE I
SHAPES AND COLORS OF THE THREE CATEGORIES

Category	Color	Shape
Prohibitory	Red, white	Circular
Danger	Red, white	Triangular
Mandatory	Blue, white	Circular

Table 2.2 shapes and colours of the three categories.

In [9], The coarse-to-fine traffic sign detection method is used to detect the traffic signs listed in table 2.1 and which included in the German traffic sign detection benchmark dataset (GTSDB).

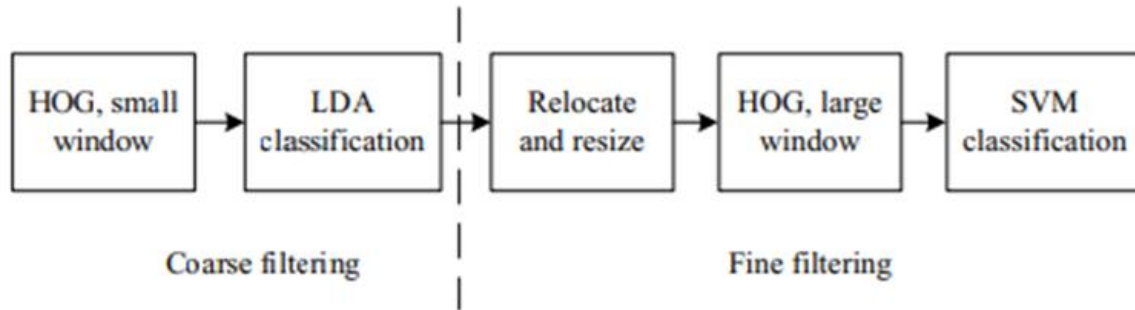


Figure 2.2.1 Block diagram of the proposed method

The original HOG algorithm first extract HOG feature from a fix-sized window and classifies the feature with a linear support vector machine (SVM). It is sliding the window across various scales to detect differently sized objects. However, it struggles with very small traffic signs (as small as 16x16 pixels) due to the small window size's classification difficulty. As the Figure 2.1.2.1, coarse-to-fine method is proposed, using two window sizes. Initially, a small-sized window performs coarse filtering to identify candidate ROIs, which are then verified with a large-sized window in fine filtering. n-maximal suppression (NMS) is applied to reduce overlapping ROIs. This method first relocates the coarse-filtered ROIs to the original image, preserving more information and improving classification accuracy. Both filtering use HOG features, with coarse filtering using LDA for efficiency and fine filtering using SVM for better accuracy.

For the colour HOG feature, the proposed method normalizes histograms of all colour channels together. This ensures that different colour channels share the same normalization factor, enhancing detection accuracy.

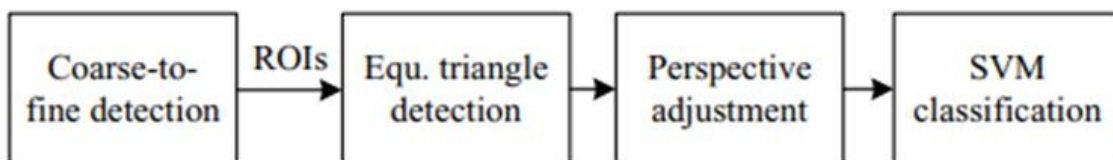


Figure 2.2.2. Block diagram of danger sign detection.

For detecting danger and mandatory signs, extra steps are added to improve precision. Red pixels are extracted from detected ROIs to create a red bitmap. A Hough transform identifies the best equilateral triangle in the bitmap, presumed to be the red rim of the danger sign. The ROI is then transformed to make the triangle equilateral, upright, and centered. Lastly, HOG and SVM classification is performed on the new ROI, filtering out most false positives without an equilateral triangular red rim and increasing precision. Hence, it can detect to various adverse situations including bad lighting condition, partial occlusion, low quality and small projective deformation.



Figure 2.2.3. Examples of perspective adjustment.

From Figure 2.1.2.3, The three images in each column are the input ROI, second and third column is extracted red bitmap and adjusted ROI, respectively. The last column is a false

positive which can be filtered out after adjustment and reclassification.

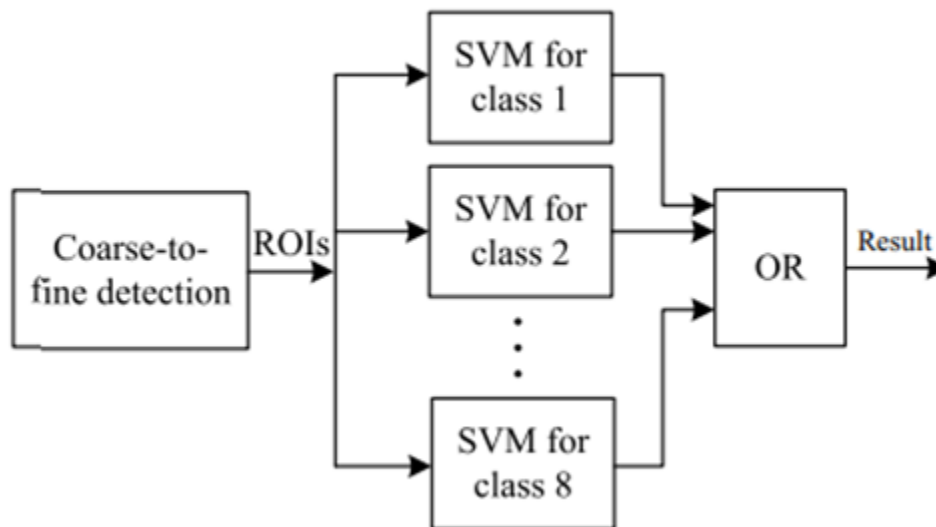


Figure 2.2.4. Block diagram of mandatory sign detection

In figure 2.1.2.4, this is for the mandatory signs, eight trained class-specific SVMs are used after fine filtering. Each SVM corresponds to a specific class of mandatory sign. Candidate ROIs have their HOG features extracted and evaluated by all SVMs. A positive response from any SVM confirms a true positive, effectively filtering out false positives, including similar-looking blue and circular signs.

In this paper, it applied training strategy to train the LDA classifier and SVM. To train the LDA classifier Positive samples are generated by selecting and deforming traffic signs from the GTSDb training images. Negative samples are randomly selected windows that do not overlap with traffic signs. It has high recall but low precision. To improve precision ratio, the following fine filtering step will do. Then, SVM training involves multiple iterations. Initially, it mirrors LDA classifier training, then detects false positives, which are used to train new SVMs in subsequent iterations until satisfactory recall and precision are achieved. For mandatory signs, eight class-specific SVMs to be trained with corresponding mandatory sign classes as positive samples and other classes and hard false positives as negative samples. After training, SVM can easily achieve 100% classification accuracy.

2.3 Object detection based on RGC mask R-CNN. IET Image Processing

Done by: Har Sze Hao

The research article discusses the development of an object detection algorithm named ResNet Group Cascade (RGC) Mask R-CNN, utilizes ResNet as the backbone network, incorporates group normalization for accuracy enhancement, and employs cascade training with various IoU thresholds to prevent overfitting. [10]

The ResNet Group Cascade (RGC) Mask R-CNN algorithm improves object detection performance over traditional Mask R-CNN through several key modifications. It employs ResNeXt-101-64×4d as the backbone network [10], which offers superior capabilities. Group normalization (GN) is integrated into the backbone, feature pyramid network neck, and bounding box head to enhance efficiency. The algorithm also adopts a cascade training approach with three different intersection-over-union (IoU) thresholds to prevent overfitting and better adapt to various proposals for detection and positioning. These enhancements collectively enable the RGC Mask R-CNN to demonstrate superior detection abilities under the same conditions compared to the traditional Mask R-CNN.

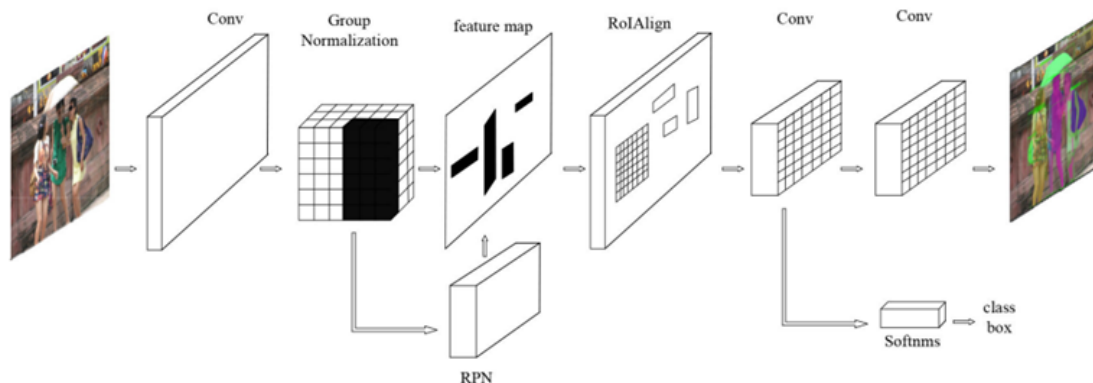


Fig 2.3.1: RGC Mask R-CNN [1]

The block diagram of RGC Mask R-CNN begins with the input image being processed by a convolutional layer to extract image features, capturing low-level elements like edges and textures. The resulting feature map undergoes group normalization, a technique ensuring stable training even with small mini-batch sizes. This group normalized data, represented as a feature map, highlights abstract features of objects and structures in the image. The feature map is then

passed to the Region Proposal Network (RPN), which suggests candidate regions where objects may be present. These candidate regions are sent to RoI Align, where the features are sampled and transformed into a fixed-size feature map. The output from RoI Align undergoes further convolution to extract higher-order features, a process repeated twice. The final feature map is processed through the Softnms function to predict the object class and bounding box. The sequence concludes with the production of an output image in which objects are identified, and a segmented mask is superimposed on the input image, allowing for the detection and segmentation of objects.

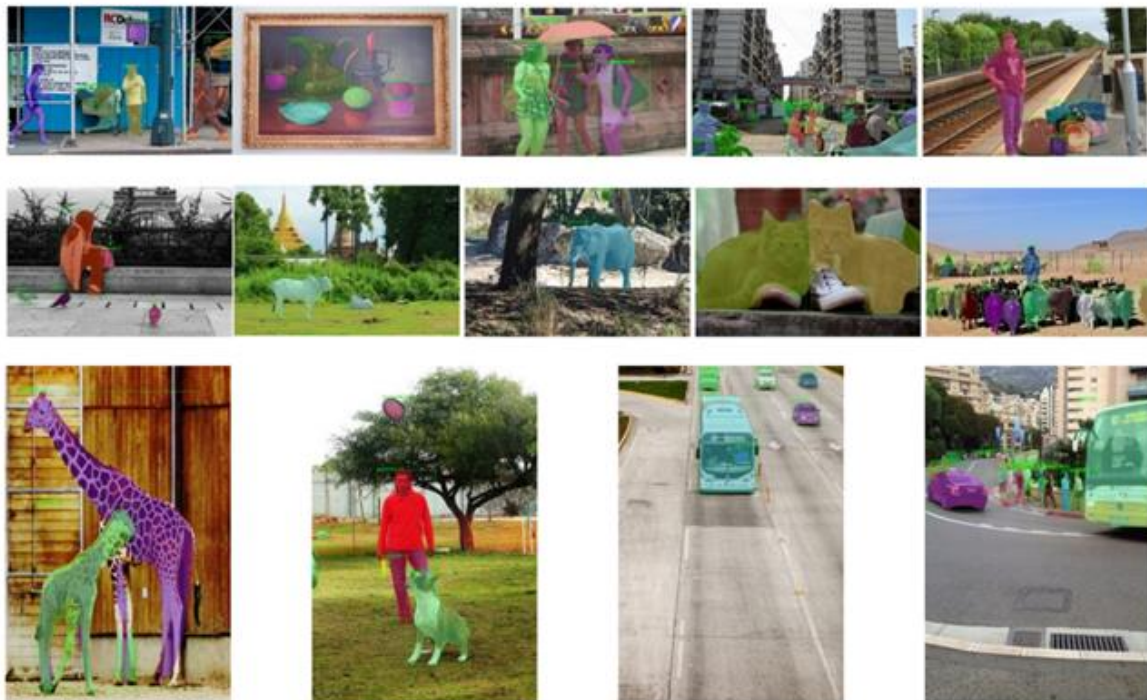


Fig 2.3.2: Results of the RGC Mask R-CNN on the COCO test set [10]

When evaluating object detection models, the most important metric is Average Precision (AP). AP is crucial because it comprehensively considers both the precision and recall of the model, providing a thorough reflection of its performance. It measures the model's average performance across different confidence thresholds, offering a detailed assessment of the model's accuracy and completeness in object detection.

On the other hand, Average Recall (AR) evaluates the model's average recall at varying numbers of predicted boxes. AR emphasizes the model's ability to detect all objects in an image, focusing on how well the model identifies every target present.

Figure 2.1.3.3, 2.1.3.4, 2.1.3.5 and 2.1.3.6 below represents the comparison between Mask R-CNN and RGC Mask R-CNN on object detection bounding box and instance segmentation using the COCO test-dev.

Table 1 Object detection bounding box AP on COCO test-dev

	Backbone	AP	AP _{S0}	AP ₇₅	AP _S	AP _M	AP _L
Mask R-CNN	ResNet-50	37.3	59.1	40.3	22.0	40.9	48.2
	ResNet-101	39.4	61.0	43.3	23.1	43.7	51.3
	ResNeXt-101-32 × 4d	41.2	62.9	45.1	24.1	45.5	52.6
	ResNeXt-101-64 × 4d	42.2	64.0	46.3	24.6	46.6	55.3
RGC Mask R-CNN	ResNet-50	41.3	59.1	45.1	23.3	44.6	54.5
	ResNet-101	42.7	60.8	46.8	23.8	46.5	56.9
	ResNeXt-101-32 × 4d	44.4	62.8	48.7	25.5	48.2	58.7
	ResNeXt-101-64 × 4d	45.5	63.9	49.7	26.0	49.2	60.6

Note: AP: at IoU = 0.50:0.05:0.95, AP_{S0}: AP at IoU = 0.50, AP₇₅: AP at IoU = 0.75, AP_S: AP for small objects: area < 32², AP_M: AP for medium objects: 32² < area < 96², AP_L: AP for large objects: area > 96².

Bold values are the best values in the comparison

Fig 2.3.3: Comparison for AP score of object detection for different backbone [10]

Table 2 Instance segmentation Mask AP on COCO test-dev. The same evaluation criteria are taken as the bounding box

	Backbone	AP	AP _{S0}	AP ₇₅	AP _S	AP _M	AP _L
Mask R-CNN	ResNet-50	34.2	55.9	36.2	18.2	37.5	46.5
	ResNet-101	35.9	57.7	38.4	19.2	39.7	49.7
	ResNeXt-101-32 × 4d	37.2	59.5	39.8	19.7	41.1	50.2
	ResNeXt-101-64 × 4d	38.1	60.7	40.9	20.3	42.1	52.4
RGC Mask R-CNN	ResNet-50	35.7	56.4	38.7	18.6	38.7	49.1
	ResNet-101	37.1	58.1	39.9	19.0	40.6	51.4
	ResNeXt-101-32 × 4d	38.3	59.8	41.2	20.3	41.9	52.4
	ResNeXt-101-64 × 4d	39.2	61.2	42.1	20.6	42.7	54.1

Bold values are the best values in the comparison

Fig 2.3.4: Comparison for AP score of instance segmentation for different backbone [10]

Table 3 Object detection bounding box AR on COCO test-dev.

	Backbone	AR	AR_{50}	AR_{75}	AR_S	AR_M	AR_L
Mask R-CNN	ResNet-50	31.0	49.7	52.4	32.8	56.1	66.5
	ResNet-101	32.3	51.5	54.3	34.9	58.7	68.5
	ResNeXt-101-32 × 4d	33.1	52.4	55.1	35.3	59.7	68.2
	ResNeXt-101-64 × 4d	33.9	53.3	55.9	35.9	59.9	70.5
RGC Mask R-CNN	ResNet-50	33.6	53.0	55.6	34.2	59.7	71.3
	ResNet-101	34.5	54.2	56.7	35.3	60.9	73.0
	ResNeXt-101-32 × 4d	35.3	55.1	57.7	35.8	61.8	73.7
	ResNeXt-101-64 × 4d	35.8	55.8	58.5	37.2	62.3	74.3

Bold values are the best values in the comparison

Note: AR: at IoU = 0.50:0.05:0.95, AR_{50} : AR at IoU = 0.50, AR_{75} : AR at IoU = 0.75, AR_S : AR for small objects: area < 32², AR_M : AR for medium objects: 32² < area < 96², AR_L : AR for large objects: area > 96².

*Fig 2.3.5: Comparison for AR score of object detection for different backbone [10]***Table 4** Instance segmentation Mask AR on COCO test-dev.

	Backbone	AR	AR_{50}	AR_{75}	AR_S	AR_M	AR_L
Mask R-CNN	ResNet-50	29.4	45.8	48.0	27.6	52.0	63.9
	ResNet-101	30.5	47.3	49.6	29.7	53.8	65.8
	ResNeXt-101-32 × 4d	30.9	47.8	50.0	29.5	54.5	65.4
	ResNeXt-101-64 × 4d	31.6	48.7	50.8	30.5	54.8	67.0
RGC Mask R-CNN	ResNet-50	30.0	46.1	48.1	27.4	52.0	64.1
	ResNet-101	30.8	47.1	49.2	28.7	53.2	65.7
	ResNeXt-101-32 × 4d	31.2	47.9	50.0	28.8	54.2	65.9
	ResNeXt-101-64 × 4d	31.8	48.4	50.5	29.8	54.4	66.9

Bold values are the best values in the comparison

Fig 2.3.6: Comparison for AR score of instance segmentation for different backbone [10]

2.4 Simultaneous Traffic Sign Detection and Boundary Estimation Using Convolutional Neural Network

Done by: Toh Yun Shuang

The paper integrated a traffic sign detection system that simultaneously estimates the location and precise boundary of traffic signs using a convolutional neural network (CNN). In this paper, the boundary estimation of traffic sign is formulated as 2D pose and shape class prediction problem, effectively solved by a single CNN. Figure 1 shows an example of boundary estimation using Convolutional Neural Network.

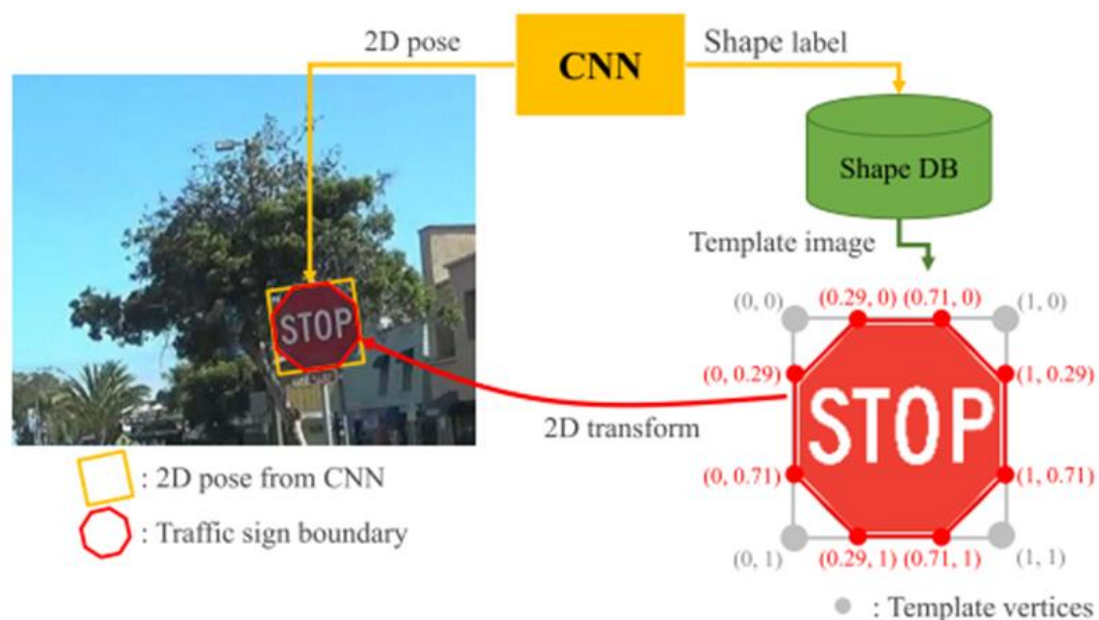


Figure 2.4.1 Boundary estimation using a predicted 2D pose and a shape label[6]

The 2D pose of planar targets can be represented as 8-dimensional vectors, such as the coordinates of four vertices. A convolutional neural network (CNN) can accurately predict these poses while also predicting the scores for each shape label. Using the predicted 2D poses and shape labels, the boundary corners of a traffic sign are calculated by projecting the boundary corners of the corresponding template image into the image coordinate system. As illustrated in the Figure 2.1.4.1, the CNN retrieves the template image of the predicted shape label from the shape database and projects the boundary corners of the template image onto the observed image using the predicted pose.

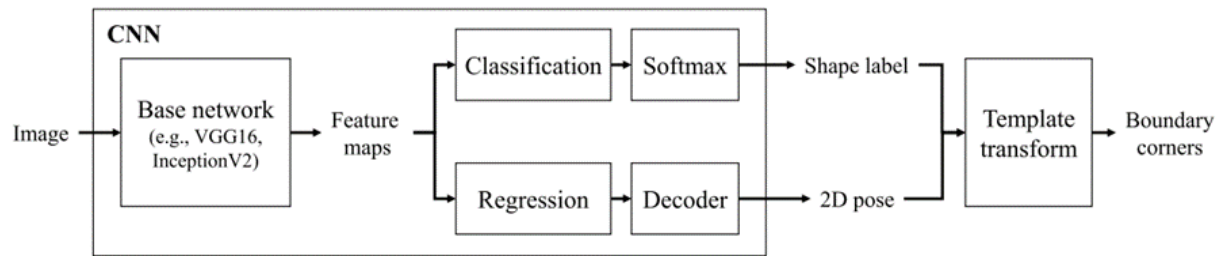


Figure 2.4.2 The overall procedure of the proposed method.[11]

The traffic signs detection process start with the input image, using the base network such as VGG16 and Inception V2 to extract feature maps. These features serve as the input for pose regression and shape classification, predicted the 2D poses and shape labels. The corners of template images that correspond to the shape label are then transformed using the 2D pose to get the boundary corners for the input image.

The proposed method begins with Convolutional Feature Extraction. In this step, features are extracted from images using the chosen base networks. The most important criterion for choosing the base network is the accuracy and speed trade-off. We choose to present the detection using VGG16 and InceptionV2 base networks. These features serve as the input for subsequent tasks, including pose regression and shape classification. The implementation employs Single Shot MultiBox Detector (SSD) architecture to predict the 2D poses and shape class probabilities of traffic signs. This involves leveraging multiple feature maps at different spatial resolutions to improve the robustness and accuracy of the predictions.

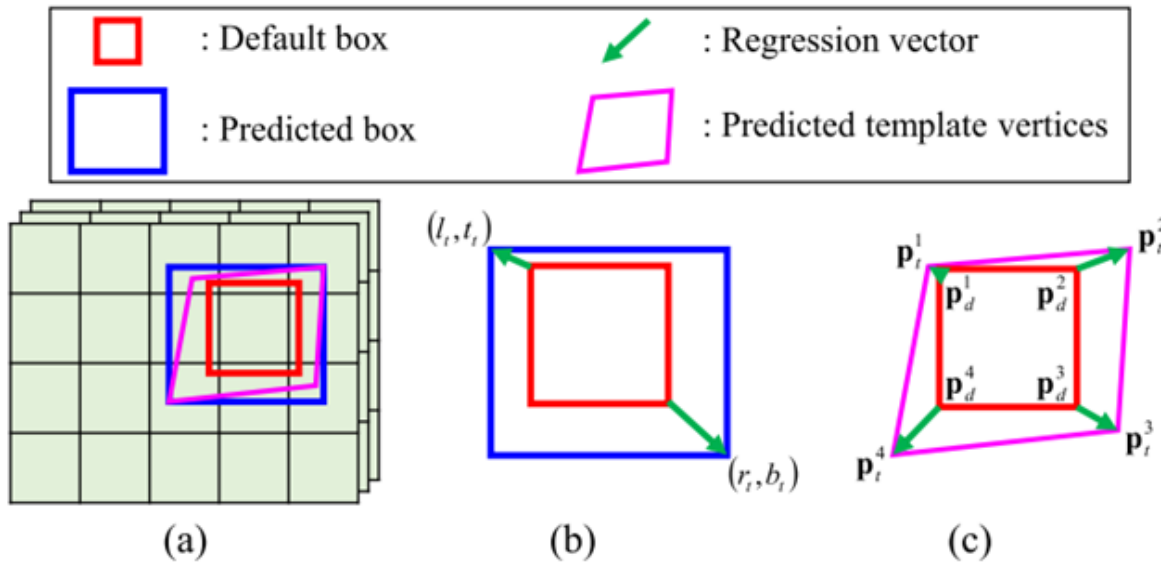


Figure 2.4.3 Box and pose prediction using regression vectors.[11]

Next the system comes to 2D Pose Prediction. Figure 2.1.4.3 shows the bounding box regression used in recent CNN-based object detectors. For the first figure, the system assigns default boxes and obtains regression vectors for every unit of feature maps in the regression layer. For the second figure, in conventional object detection networks, two regression vectors are predicted for bounding box estimation. In the last figure, four regression vectors are predicted from the network for 2D pose estimation, which is the system use in this paper. The system generalizes the bounding box regression approach to vertex regression, allowing the network to predict the 2D poses of traffic signs. Each traffic sign is represented by an 8-dimensional vector corresponding to the four vertices of the sign's boundary in the image.

C. Boundary Corner Estimation

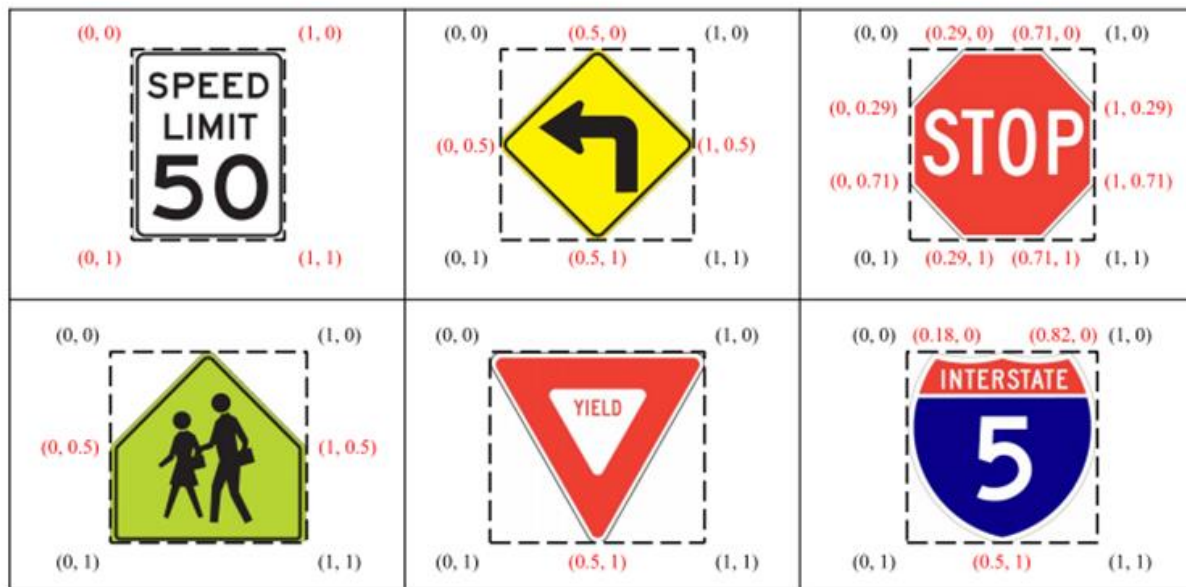


Figure 2.4.4 Examples of traffic sign template images and their boundary corners.[11]

After that, the system straightforward to Boundary Corner Estimation. Instead of directly predicting bounding box coordinates, the method focuses on pose estimation. The predicted 2D poses are then used to project the boundary corners of predefined traffic sign templates into the image coordinate space, thus estimating the boundaries of the detected traffic signs accurately.

2.5 Comparison of the 4 techniques

Done by: everyone

In the paper [9], the techniques are designed to enhance the accuracy designed to enhance the accuracy and efficiency of traffic sign detection, particularly in challenging conditions such as poor lighting, partial occlusion, low quality, and small projective deformation. It combines Histogram of Oriented Gradient (HOG) features with a coarse-to-fine sliding window scheme and utilizes color information to improve detection precision. Therefore, it has high accuracy of traffic sign detection because the coarse-to-fine approach ensures that even the smallest traffic signs are detected, and the subsequent verification step improves precision, leading to high overall accuracy. It has the advantage of robustness The method is effective in various challenging conditions, including poor lighting, partial occlusion, low quality, and small projective deformation. the techniques which are efficient to use color information. It normalizes histograms of all color channels together to enhance detection accuracy and leverages color information more effectively than previous approaches.

However, the method used is expensive due to the computational complexity. The coarse-to-fine approach and the use of multiple SVMs can be computationally intensive, particularly when processing large datasets or high-resolution images. It led to resource intensive as well because it requires significant computational resources for both training and inference, which may not be applicable for real-time applications or devices with limited processing power. Compared to the method used in “A Lightweight Traffic Sign Recognition Model Based on Improved YOLOv5” [6], which is more suitable for real-time applications. The lightweight nature of the model ensures that it can run efficiently on devices with limited computational resources. Furthermore, it has potential overfitting. The high precision achieved through multiple iterations and fine-tuning may lead to overfitting, making the method less generalizable to new, unseen data.

When comparing RGC Mask-RCNN, Simultaneous Traffic Sign Detection and Boundary Estimation (STSB) using Convolutional Neural Networks (CNNs) with Histogram of Oriented Gradients (HOG), and the YOLO model, several factors come into play.

RGC Mask-RCNN combines region-based convolutional neural networks (R-CNN) with mask prediction for pixel-wise object segmentation. Its advantage lies in its ability to provide precise instance segmentation, which is crucial for applications needing detailed object outlines. This model excels in distinguishing overlapping objects and achieving high accuracy in complex scenes. However, it is computationally intensive and slower than some other models due to its multi-stage process, which might not be suitable for real-time applications.

STSB using CNN and HOG integrates the robustness of CNNs in feature extraction with the edge detection strength of HOG. This hybrid approach enhances traffic sign detection by leveraging the spatial and contextual information provided by CNNs and the gradient-based features from HOG. While this method improves detection accuracy and boundary estimation, it may still suffer from slower processing speeds and reduced performance in highly cluttered environments compared to more modern approaches.

YOLO (You Only Look Once) is renowned for its real-time object detection capabilities, processing images in a single pass to deliver fast and efficient results. Its advantage lies in its speed and ability to handle multiple objects in a scene, making it ideal for real-time applications. However, YOLO may trade off some accuracy for speed, particularly in cases requiring fine-grained object segmentation or when objects are very small or densely packed.

In summary, RGC Mask-RCNN is suited for applications needing detailed object segmentation but may be too slow for real-time use. STSB with CNN and HOG offers a balanced approach for traffic sign detection with decent accuracy but can be slower. YOLO excels in speed and real-time performance but may sacrifice some accuracy, particularly in more complex scenarios.

CHAPTER 3 SYSTEM METHODOLOGY

3.1 Design specification

Done by: Lee Jing Yong

3.1.1 System overview

The traffic sign segmentation system that we develop are made to take in images and segment out the traffic signs in the image to be use in later for traffic sign recognition. The segmented traffic sign shall include only the clear image of the traffic sign without background and other objects. The below figure shows the flow of the system that we have developed.

3.1.2 Hardware used

The following table shows the hardware specification that are used to develop and test the project.

Operating System	Windows 10 64 bit
Processor	AMD Ryzen 5 3600 6-Core Processor 3.59 GHz
Ram	16.0 GB
GPU	NVIDIA GeForce RTX 3060

Table 3.1.2.1 Specification of hardware for developing and testing

3.1.3 Software used

The following table below summarizes the software used in the project

Operating System	Windows
------------------	---------

Programming language	C++
IDE	Visual Studio 2022
Libraries	OpenCV

Table 3.1.3.1 Specification of software for developing and testing

3.2 System Block Diagram and Description

Done by: Toh Yun Shuang

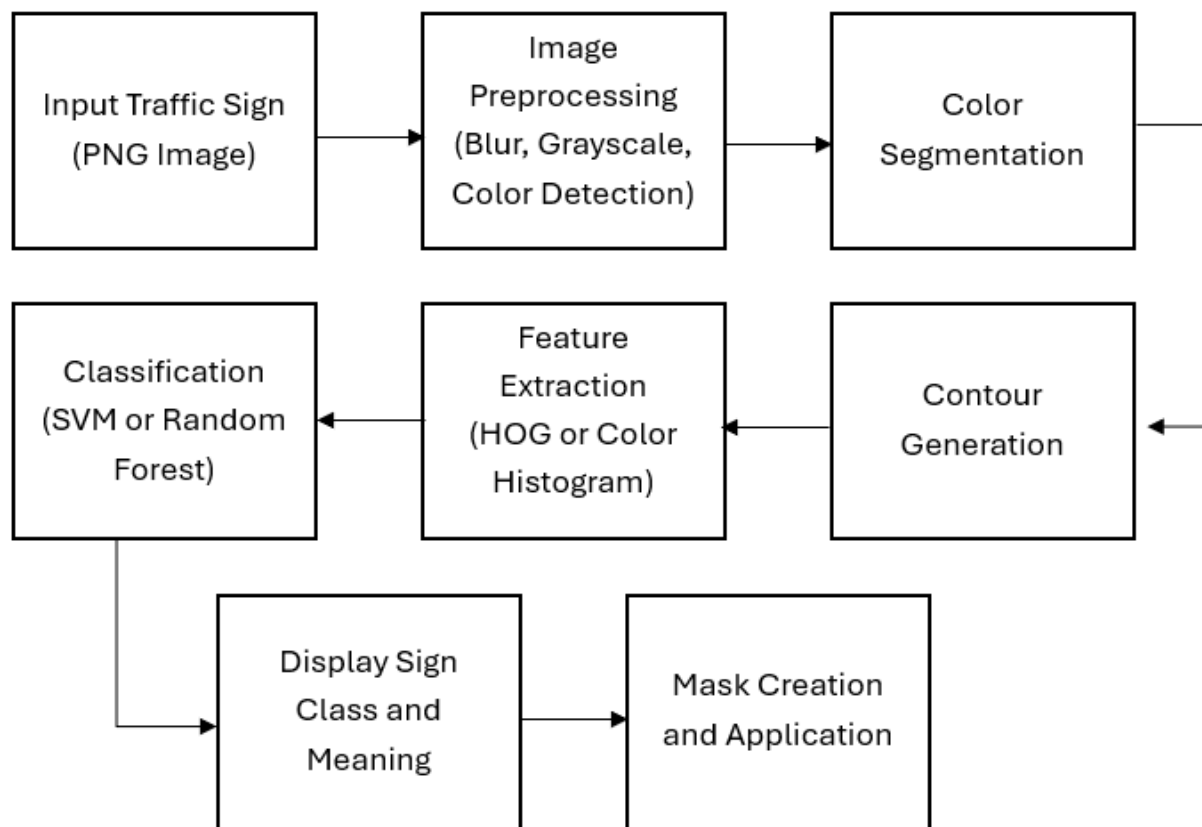


Figure 3.2.1 Block diagram of system

3.2.1 Input Traffic Sign

The system will first read the image files that contain traffic signs. The system will only input the images that are in PNG format and loaded into the system memory for further processing. This step involves converting the image from its file format into a matrix format so that the system can easily manipulate the image.

3.2.2 Image Preprocessing

The image preprocessing step is aimed to clean up the image that contains noise, lighting conditions, or occlusions so that the system can identify the traffic sign in the image easily and prepare for further analysis. The process contains blurring, grayscale conversion, and color detection. Blurring that used in the project is median blurring. Median blurring is proceeded by replacing each pixel's value of the image with the median value of the surrounding pixel neighborhood. Median blurring is useful in removing salt-and-pepper noise, which means the random white and black pixels, without blurring the edges of the image and preserving the shape of the traffic signs. The Grayscale Conversion process will convert the image into a grayscale format that reduces the multiple color channels into a single intensity channel. This can help in increasing the accuracy of edge detection and contour generation by highlighting the contrasts between the traffic sign and the background. For the color detection, specific colors such as red, blue, or yellow are used to identify regions of interest. Color detection will help in localize the traffic sign in the image for further analysis.

3.2.3 Color Segmentation

After detecting the main color of traffic signs, the system will proceed to color segmentation process where the system will isolate the regions of the image that match the color detected. The system will use the color thresholds to create a binary mask where the regions within the traffic signs color are marked as foreground and the rest are marked as background. This step is aimed to separate the parts of the image that are likely to contain the traffic sign based on color so that the system can only focus on the relevant regions and reduce the area that needs to be processed further.

3.2.4 Contour Segmentation

After color segmentation, the system next identifies the shape of the traffic sign using contour detection. In this process, contours are generated around the detected regions in color segmentation by joining all the continuous points along the boundary of an object which is having the same color or intensity in the image. After generating all the contours on the image, the system will filter them based on area, circularity, or aspect ratio and choose only one contour that is most likely to be the traffic sign to eliminate false positives. After contour segmentation process, the system can identify the shape of the contour whether it is circular, triangular, or rectangular to define the types of traffic signs.

3.2.5 Feature Extraction

The Feature Extraction process will identify and select the relevant characteristics from images for distinguishing traffic signs from the background and other objects. It simplifies the input data to improve the efficiency and accuracy of the subsequent classification stage. Feature Extraction that used in this project is Histogram of Oriented Gradients (HOG) and Color Histogram. HOG is a technique focusing on capturing the edge and the gradient information to detect the traffic sign's shape and structure. For example, the circle shape for the traffic sign is a regulatory sign, rectangle is an information sign, and the triangles is a warning sign. Color Histogram is a technique that captures the color distribution of the traffic signs and builds a histogram for each color channel so that the traffic signs can be distinguished based on that color. For example, the different colors of traffic sign have distinct meanings such as red for stop sign, yellow for warning sign, and blue for informational sign. After the feature extraction, the result is then used as an input feature for the classification part.

3.2.6 Classification

Classification involves assigning detected traffic signs to predefined categories or classes based on the features extracted. It provides help in accurately identifying the type of each traffic sign. The classifiers used in this project are Support Vector Machine (SVM) and Random Forest. SVM is a powerful classification algorithm that attempts to find the optimal boundary that separates

different classes of traffic signs to classify the detected traffic signs into predefined categories such as stop signs, speed limit signs, and others. It works by finding the hyperplane that separates traffic signs of different classes in a multidimensional feature space. While Random Forest is an algorithm that builds multiple decision trees during training and outputs the mode of their predictions. Each tree is trained on a random subset of the data features, and the final classification is based on the majority vote from all trees.

3.2.7 Display Sign Class and Meaning

After the classification, the system will display the identified class of the traffic sign and its meaning such as “No Entry”, “Speed Limit”, “Go Straight and Turn Right”, and others. It is important as it could provide feedback that indicates what the detected traffic sign is and what it signifies to the user or system. In real-world applications, this could be used to warn the driver while using the application or helps autonomous vehicle in decision making.

3.2.8 Mask Creation and Application

The last step of the traffic sign detection project is the creation and application of the mask. The system will create a mask to isolate the detected traffic sign from the rest of the image to ensure that only the region of the image that contains the traffic sign is highlighted. The mask can be further used in an autonomous vehicle and helps in influence the vehicle’s behavior such as slowing down for a limited speed or stopping for a stop sign.

CHAPTER 4 SYSTEM DESIGN

4.1 Data Set

Done by: Har Sze Hao

The Chinese Traffic Sign Database is a database containing 6,164 clear images of various road signs in the Chinese environment, which could be a very excellent training and testing foundation for those traffic sign identification models developed through machine learning and computer vision under OpenCV projects.

The database is divided thoughtfully into training and testing sets, comprising 4,170 images allocated for training and 1,994 images reserved for testing. This split will enable effective model development and its validation. It covers a big area of traffic signs, ranging up to 58 different types; hence, it ensures very varied representation of signs usually met on Chinese roads.

Sample images are taken from the database to give an idea of the various types of signs the database contains: speed limit indicators, direction signs, prohibitive signs like "No stopping," and behavioral instructions like "No Horn." This diversity will enable the development of models that can identify and interpret a wide spectrum of traffic signage.

4.2 Segmentation

Done by: Lee Jing Yong

Using the imread function, we load the image at the start of the image processing pipeline. After making a copy of the original image, a Gaussian filter with a kernel size of (9,9) is used to blur it. We convert this blurry image to grayscale. Figure 4.2.1 illustrates the result of subtracting the grayscale blurred image from the original to decrease background noise and provide a clearer image with better focus on the regions of concern. This is a critical step in removing unwanted details that could negatively impact accurate detection. We further enhance the image by calculating the average intensity of the red channel because certain images are taken in low lighting, which might make traffic signs harder to see. The color intensity of the red channel is doubled to

increase its visibility in the processed image if the average red intensity is found to be less than 10. The picture is then transformed into the HSV color space, which works better for color-based segmentation.

Next, the red color is separated by the given range of `redLower(0, 0, 35)` and `redUpper(0, 255, 165)` to build a binary mask using the `inRange` function. The `split` function is then used to divide this binary mask into its separate color channels, enabling more accurate manipulation of the red channel. In order to further isolate the red areas, a red mask with a threshold value of 15 is created using a thresholding technique. The `erode` function is used with a kernel size of (4,3) in order to reduce noise. Next, a custom function named `generateContour` is used to generate the contours. After the contour is created, we check the area of the mask that the contour covers. Over 80% coverage of the image by the contour mask suggests a possible inaccuracy in the contour detection process. In order to overcome this, we apply a smaller erosion kernel size (3,3) and fix minor gaps that may reduce the detection accuracy using the `dilate` function with a kernel size (6,4). To further smooth out the final image, a `medianBlur` with a size of 1 is applied. Finally, the outcomes of these processing stages are shown in Figure 4.2.2 below, showing greater accuracy in identifying and segmenting traffic signs. The whole process of segmentation is shown in Figure 4.2.3, 4.2.4, 4.2.5 below.

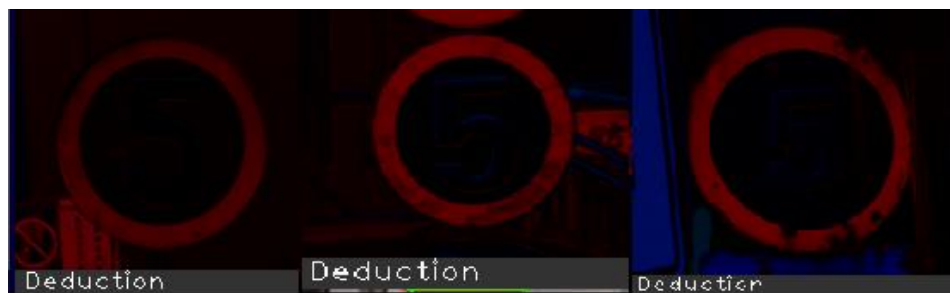


Figure 4.2.1 Result of deduction



Figure 4.2.2 Final segmented result

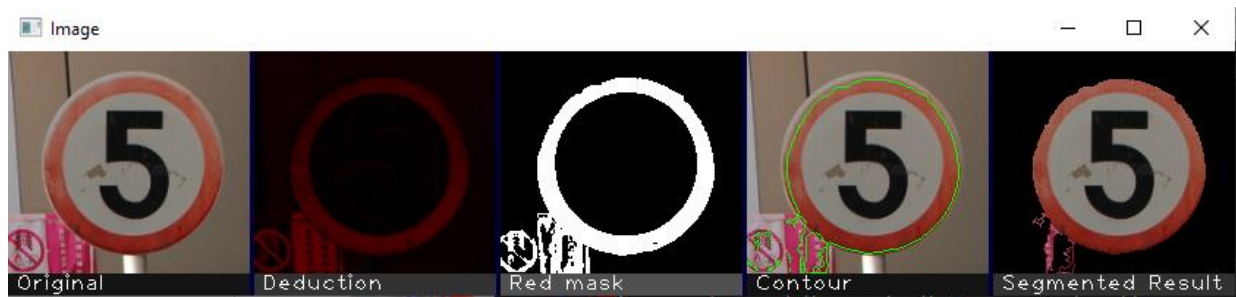


Figure 4.2.3.1 Segmentation process



Figure 4.2.3.2 Segmentation process



Figure 4.2.3.3 Segmentation process

4.3 Feature Extraction

Done by: Lai Wei Bin

In this project, we have implemented the Histogram of Oriented Gradients (HOG) and Color Histogram feature extraction approaches for the image feature extraction. Both methods are critical for capturing different information and aspects of an image. The HOG technique focuses on the structural characteristics of the image such as edges and textures, while the color histogram provides information on the color distribution. This combination allows your model to make more accurate predictions, especially in scenarios where both shape and color are important such as traffic sign recognition. This is because the traffic signal is fixed colors and shapes. Therefore, both feature extraction approaches are suitable in extracting features in shape and color from the traffic sign.

The HOG feature extraction process in this project is designed to capture gradient-based information from the images. By resizing each image to a uniform 64x64 pixels, it is to ensure consistency size across the dataset. The resized image size matches the HOG window size so that the gradient computation can be applied to the entire image. Furthermore, the block size of 8x8 pixels divides the window into smaller regions and each block is split into 8x8 cells. This is to allow a fine-grained capture of local gradient information. Then, we used 9 orientation bins to compute gradients of the image in different directions and to ensure that the classification model can effectively recognize edges and shapes of traffic sign. This approach has the benefit that relatively robust to variations in lighting condition of the image and minor changes in the orientation of the objects.

On the other hand, the Color Histogram feature extraction focuses on the color distribution within the images. This approach is equally important as the HOG because color is a critical factor in traffic sign recognition. By converting the images to the HSV color space, we are able to separate the color intensity and the brightness from the actual color information and then represent it with the hue and saturation. This makes the model less sensitive to variation of lighting conditions while still capturing the essential color characteristics from the image. The use of 5 bins for each of the H, S, and V channels results in a 125-dimensional feature vector for each image. The channels comprehensively represent color distribution. This is particularly important for tasks where color

is a key factor. For instance, the color distribution is useful to distinguish different types of traffic signs which are similar shapes but different colors.

Lastly, the use of both HOG and Color Histogram features extraction ensures that your feature set is varied and has multiple dimensions. By conducting both feature extraction, it can capture both the structural and color-based characteristics of the images. This is to improve the model's performance in classification task. At the end of feature extraction, we save all these features in CSV files which are "HOGFeatures.csv" and "ColorHistogramFeatures.csv" for the process for model training. Each row in the CSV file represents the extracted features for one image to allow machine learning models like Support Vector Machines (SVM) and Random Forests to have an easy input.

4.4 Classifier

Done by: Har Sze Hao

In our traffic sign classification project, we have implemented a comprehensive approach that leverages the strengths of two powerful machine learning algorithms: **Support Vector Machines (SVM) and Random Forests**. SVMs, known for their effectiveness in high-dimensional spaces, excel at finding optimal hyperplanes that maximize the margin between different classes [13]. This characteristic makes them particularly well-suited for classifying the Histogram of Oriented Gradients (HOG) features we extract from traffic sign images. Random Forests, on the other hand, offer the advantage of ensemble learning, capturing complex, non-linear relationships in data through multiple decision trees [14]. Our implementation carefully balances these algorithms with strong image processing techniques.

The SVM implementation utilizes a linear kernel (SVM::LINEAR), chosen for its efficiency with high-dimensional HOG features [13]. The regularization parameter C was set to 1, balancing margin maximization and classification error minimization. Although not used in the linear kernel, the gamma parameter was set to 0.5 for potential RBF kernel experimentation. These parameters were selected to optimize the trade-off between model complexity and generalization capability.

The Random Forest configuration employs a maximum depth of 10 and a minimum sample count of 2 for node splitting. These settings are designed to mitigate overfitting while maintaining the model's capacity to capture complex data patterns [14]. The maximum number of categories was set to 10, accommodating the multi-class nature of the traffic sign problem.

The segmentation process employs the HSV color space with optimized thresholds for red, blue, and yellow signs. This color-based methodology, inspired and employed by Zaklouta and Stanciulescu [13], facilitates robust sign detection across varied lighting conditions.

The integration of these techniques and carefully calibrated parameters has resulted in a system capable of addressing the complexities inherent in real-world traffic sign recognition. This approach illustrates the potential for enhancing driver assistance systems through the application of advanced computer vision and machine learning techniques.

Lastly, after training the models with their respective parameters, both the SVM and Random Forest classifiers were serialized and saved to disk in YAML (.yaml) file format. The SVM model was saved as 'svm_modelHOG.yaml', while the Random Forest model was saved as 'randomForest_modelHOG.yaml'. This approach of saving trained models to external files is a standard practice in machine learning workflows. It allows for efficient model persistence, enabling the trained classifiers to be easily loaded and reused for future predictions without the need for retraining.

4.5 Testing and Evaluation

Done by: Toh Yun Shuang

After we done the feature extraction and classification, we need to know that whether the model is performing as expected. Therefore, we need to do testing and evaluation on the results to assess the performance, reliability, and generalizability of the model. In this step, we use accuracy to evaluate the performance of the model.

Accuracy is one of the most basic and widely used metrics that used to evaluate the classification models and indicates the proportion of correctly classified instances to the total instances.

Accuracy is easy to calculate and understand as it works by divides the total number of predictions by number of correct predictions. Accuracy can also count by True Positive (TP) + True Negative (TN) divides TP + TN + False Positive (FP) + False Negative (FN).

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$

Figure 4.5.1 Accuracy

If the model has a high accuracy score, it means that the model is correctly predicted most of the data. This indicates that both positive and negative classes are being well classified, and the model effectively captures the underlying patterns in the data. However, it is possible that the high accuracy will cause an overfitting where the model performs well on training data but poorly on unseen data. While the low accuracy scores suggest that the model is less classify correctly and may need a better features or different model. A low accuracy model shows that the model is difficult to classify the data accurately.

CHAPTER 5 SYSTEM IMPLEMENTATION

5.1 Image Acquisition

Done by: Toh Yun Shuang

Firstly, the user needs to find and download the Chinese Traffic Sign Database from an online resource. The Chinese Traffic Sign Database includes 58 different classes or categories of traffic signs and each class corresponding to a specific types of traffic sign such as speed limit, roundabout, no entry, U-turn sign, and many more. The dataset also includes a total of 4170 images covering all the classes. Each image represents a unique traffic sign that will be used for further analysis. After downloading the dataset, the user should extract it for unzipping the compressed file so that the individual images can be accessed. The dataset should be extracted into the designated program folder where the user's code and application are located for the system to read the images and proceed with further steps.

5.2 Segmentation

Done by: Lee Jing Yong

The proposed approach employs a multi-stage preprocessing for noise reduction and image enhancement. The implementation approach to reduce the noise of the images by applying Gaussian blur, followed by color space conversions and image subtraction to enhance the color channel, which is important to detect traffic signs with color. The use of deduction between the original and grayscale images to isolate background and traffic sign, and enhance color contrast before applying color-based masking.

Beside that, the approach includes adaptive mask generation based on average color intensity. Based on the average blue intensity. The algorithm dynamically adjusts the brightness of the mask. If the average color intensity is too low, the image is brightened to ensure that low-intensity color regions are adequately captured. The approach helps to improve the reliability of the color-based mask for more accurate color-based segmentation in subsequent steps.

Furthermore, the approach includes iterative refinement and contour selection. The iterative refinement process addressed the issue of over segmentation. Firstly, the initial mask and contours

are generated. Then, the approached algorithm is used to evaluate the coverage of the segmented area. If the coverage exceeds a certain threshold, it is the potential inclusion of non-sign regions. The initial mask undergoes further refinement through erosion, dilation, and blurring, to ensure the mask is not including the background. This process reduces the likelihood of including background areas in the final segmentation. Moreover, the method prioritizes selecting the longest contour, which typically corresponds to the traffic sign, and refines it to ensure that only the relevant traffic sign region is segmented.

5.3 Feature Extraction

Done by: Toh Yun Shuang

Feature extraction of the project is handled through two primary methods which is Histogram of Oriented Gradients (HOG) and Color Histogram. For the HOG method, the system first resizes the input image to a standard size which is 64 x 64 pixel to ensure the consistency of all the images. After that, the resized image will be converted to grayscale as it does not require color information for HOG. Next, HOG will compute gradients and orientations and create histogram of gradient orientation. The parameters for HOG Descriptor were implemented with a 64x64 pixel window size, 16x16 block size, 8x8 block stride, 8x8 cell size, and 9 histogram bins. These parameters were selected based on their established effectiveness in comparable object detection tasks [13]. Lastly, the system uses `hog.compute()` function to calculate the HOG features for the grayscale image and the extracted features will be written to a CSV file along with the image filename and label.

For the Color histogram, the system will also first be resizing the image to 64 x 64 pixels. After that, the system converts the image from RGB color space to the HSV color space because HSV separates color information from intensity information. After converting, the system will build histograms separately for the Hue, Saturation, and Value channels. By using 5 bins per channel, these bins were incorporated to provide supplementary color information to the shape-based HOG features. Lastly, the system normalizes the histogram by ensuring that the feature values are in range [0,1].

5.4 Feature Storage and Use

Done by: Har Sze Hao

Features extracted from the image dataset, including Histogram of Oriented Gradients (HOG) and color histograms, are systematically stored in CSV files named "HOGFeatures.csv" and "ColorHistogramFeatures.csv" respectively. These files follow a consistent format, with each row containing an image filename, class label, and a series of feature values. This storage method allows for a clear separation between the computationally intensive feature extraction process and subsequent model training phases.

When utilizing these features, the project employs a loading mechanism through the `loadCSV()` function, which efficiently parses the stored data into memory as Mat objects. This approach facilitates flexible data manipulation, including random shuffling and splitting into training and testing sets via the `shuffleAndSplit()` function. The stored features are then directly fed into the training processes for both Support Vector Machine (SVM) and Random Forest classifiers, streamlining the model development pipeline. During the prediction phase, the `predictAndShowMeaning()` function seamlessly integrates feature extraction from new images with the loaded pre-trained models, ensuring consistent feature representation between training and inference stages. This comprehensive approach to feature management not only optimizes computational resources but also enhances the project's modularity, allowing for easy experimentation with different classification algorithms on a consistent feature set.

5.5 Train & Test Dataset

Done by: Har Sze Hao

This traffic sign classification project's dataset management technique shows an effective approach to model development and assessment. Random shuffling is applied to the entire dataset at first to guarantee that all classes are fairly represented. After that, this randomized dataset is carefully divided, with 20% set aside for testing and 80% designated for training. An important subset is

retained for performance evaluation, and a major chunk is provided for model training in this 80/20 split, which is a generally recognized standard in machine learning.

The training set, comprising 80% of the data, serves as the foundation for building both the Support Vector Machine (SVM) and Random Forest models. This significant part enables the models to learn from a wide variety of examples, improving their capacity to generalize across different kinds and circumstances of traffic signs. On the other hand, the 20% testing set is essential for impartially assessing how well the models function with unknown data. This division is essential for evaluating the trained models' actual prediction power and providing information about their practicality. In addition, this thorough testing process helps identify possible overfitting, a common problem in machine learning when models work incredibly well on training data but don't transfer to new, unobserved samples. The research guarantees an objective assessment of model correctness by keeping a distinct testing set, which offers a trustworthy gauge of the classifiers' performance in traffic sign recognition tasks.

5.6 Testing and Evaluation

Done by: Lee Jing Yong

The traffic sign classification system is tested using the chinese traffic sign dataset. The dataset includes a total of 58 different classes of traffic signs. The image in the dataset includes images of traffic signs from various angles, lightings, and backgrounds. The varying conditions of the images greatly affected the accuracy of the system.

For our HOG and SVM model, the model achieved a high training accuracy of 99.04%. However, when it is tested the accuracy of the model drops drastically to 78.57%. For the color hog and random forest model, the training accuracy is 100% and the testing accuracy is 82.14%.

Upon testing the models, it indicates that the model we trained is overfitted. However, it can also be untrue as some of the segmentations are incorrect. This leads to the model being unable to correctly classify the segmented traffic sign image. By comparing the accuracy, the color HOG and random forest model is better compared to the SVM model because both the models are classifying the same segmented image.

5.7 Setting and Configuration

Done by: Lee Jing Yong

Software and libraries

IDE	Visual Studio
Image processing	OpenCV
Programming language	C++

Table 5.7.1 Software and libraries used

Dataset

Source	Chinese traffic sign database
Training and testing	80% training 20% testing
Classes	58
Total images	4170

Table 5.7.2 Dataset settings

Preprocessing steps

Image size	All resized to 64x64
Color segmentation	Apply red, blue, yellow segmentation to all images
Contour selection	Select the largest contour from the 3 colors

Table 5.7.3 Preprocessing steps and conditions

Model Hyperparameters

SVM	
Max iterations	100
Gamma	0.5
Regularization	1
Random forest	
Max iterations	100
Max depth	10
Minimum sample	2
Max categorical data	10

Table 5.7.4 Hyperparameters for models

5.8 System Operation (with Screenshot)

Done by: Lee Jing Yong

Main menu

Firstly, the program will display the menu when executed. The user shall then select the option that they want.

```
Please choose number for the following task.
1. Segmentation of traffic sign (HOG + SVM)
2. Segmentation of traffic sign (colorHOG + Random Forest)
3. HOG Extraction and SVM classification model training
4. Color HOG Extraction and Random Forest model training
0. Exit
Enter your choice: _
```

Figure 5.8.1 Main menu options

Model training

If the user did not train any model before, they shall proceed to train the model first either SVM or random forest model. The model is required in the segmentation process for classification process. The model training process will take some time and users will be waiting in the screen below.

```
Enter your choice: 3
Extracting features..... Please wait for a while
```

Figure 5.8.2 Model training process

Once the features are extracted and the model is trained, the features will be saved into a csv format file while the trained model will be saved into a yaml format file.

```
Feature extraction complete and saved to HOGFeatures.csv
Loaded features: 4170x1764
Loaded labels: 4170x1
SVM model trained and saved to svm_modelHOG.yaml
```

Figure 5.8.3 Saving features and model trained

The trained model will go through testing. The confusion matrix of the training set will then be calculated and displayed to the user.

```
True label: 17, Predicted label: 17
True label: 35, Predicted label: 35
True label: 28, Predicted label: 28
True label: 28, Predicted label: 28
True label: 17, Predicted label: 17
True label: 42, Predicted label: 42
True label: 16, Predicted label: 16
True label: 10, Predicted label: 10
Confusion Matrix:
19 0 0
0 9 0
0 0 17
```

Figure 5.8.4 Confusion matrix for training data

Traffic sign segmentation and classification

Depending on which model the user has trained, they can now proceed to segment and classify the traffic signs. After selecting the model to use, a window with both comparisons of the original image and the segmented image will be displayed. The figure below shows both correct and wrong classification of the traffic signs.



Figure 5.8.5 Example of correct identification



Figure 5.8.6 Example of incorrect identification

Error Handling

In the menu page, the program only takes in integers within the given options. However, if the user input something unexpected the program might crash. In order to prevent this, proper handling of the user input is made so that the program can perform normally.

```
Please choose number for the following task.
1. Segmentation of traffic sign (HOG + SVM)
2. Segmentation of traffic sign (colorHOG + Random Forest)
3. HOG Extraction and SVM classification model training
4. Color HOG Extraction and Random Forest model training
0. Exit
Enter your choice: one
Invalid input. Please input integer only.

Please choose number for the following task.
1. Segmentation of traffic sign (HOG + SVM)
2. Segmentation of traffic sign (colorHOG + Random Forest)
3. HOG Extraction and SVM classification model training
4. Color HOG Extraction and Random Forest model training
0. Exit
Enter your choice: 6
Invalid input. Please choose a valid option.

Please choose number for the following task.
1. Segmentation of traffic sign (HOG + SVM)
2. Segmentation of traffic sign (colorHOG + Random Forest)
3. HOG Extraction and SVM classification model training
4. Color HOG Extraction and Random Forest model training
0. Exit
Enter your choice: _
```

Figure 5.8.7 Error handling on invalid input

5.9 Implementation Issues and Challenges

Done by: Lai Wei Bin

One of the primary challenges during the implementation phase in our proposed project is dealing with noise in the images. Noise of the image is often caused by random variations in brightness and lighting during the image capture. This noise can significantly affect the accuracy of color-based segmentation methods and make it difficult to isolate the traffic sign from the background. Additionally, when images are taken from different angles, not directly from the front, the perceived color and brightness of the sign can vary. This variation complicates the preprocessing stage, as the color thresholds used to generate the mask may not work consistently across all images, leading to incomplete or inaccurate segmentation although the image pre-processing method and thresholding has been used to reduce the noise of the image.

Another critical issue arises when the background colors are similar with the color of the traffic sign. In such cases, distinguishing the sign from its surroundings becomes challenging, especially when using color-based techniques. This can lead to the generation of contours for both the background and the traffic sign, and if the longest contour includes both, then it may result in inaccurate segmentation. Despite applying algorithms designed to ignore background contours, some images may still generate contours for both the background and the sign. Moreover, Variation of lighting conditions and the presence of shadows may aggravate the problem, which can alter the apparent color of both the sign and the background, making the sign and background look similar. These factors necessitate more robust preprocessing techniques, such as machine learning models, to reduce the impact of noise, varying angles, and similar background colors, and accurately segment the sign.

5.10 Concluding Remark

Done by: Lee Jing Yong

In our implementation, we have succeeded in properly segmenting traffic signs from their images and also correctly identify the traffic signs. Through comparing both SVM model and random forest models, it has been shown that both of them have their own strengths and weaknesses. The SVM model is able to process faster to provide a prediction. However, the accuracy of the model is not that pleasant. For the random forest model, it has a higher accuracy when compared to the

SVM model. Although the random forest model has a higher accuracy, the processing time is remarkably slow. The slow processing time can be an issue if it were to be used in the real world autonomous vehicle. Despite that, the processing time is still under 2 seconds for each image which fits into our requirements

Further fine-tuning shall be made to overcome the issues that we have faced in the project. One of the issues is background noises affecting the segmented results. This causes the image to include irrelevant parts of the image when the color is similar to the traffic sign and affect the accuracy of the prediction model.

CHAPTER 6 SYSTEM EVALUATION AND DISCUSSION

6.1 System Testing and Performance Metrics

Done by: Lai Wei Bin

System testing is an important part in the project, where the entire system is tested for functionality, performance, and reliability of the classification tasks. In this project, integration testing was applied to ensure that the system meets the classification requirements. Key performance metrics such as accuracy, precision, recall, F1-score, and the confusion matrix were employed to evaluate the effectiveness and performance of the classification models, particularly for the SVM and Random Forest classifiers.

The system's accuracy measures the proportion of correct predictions to total predictions, while the confusion matrix provides detailed insights into true positives, false positives, true negatives, and false negatives. From the confusion matrix, we calculate precision, recall, and F1-score, which are critical in assessing the system's efficiency under different workloads. By analyzing these metrics, potential bottlenecks can be identified, allowing us to optimize the system's overall performance for real-world usage.

In this project, we integrated testing directly into each feature extraction and classification process. As a result, we could immediately evaluate the accuracy and confusion matrix after running either the HOG feature extraction with SVM classification or the Color HOG feature extraction with Random Forest classification. By analyzing these metrics, we can easily assess performance and adjust parameters to achieve better results.

$$\text{Accuracy} = \frac{\text{Number of correctly predicted samples}}{\text{Number of samples}}$$

Figure 6.1.1 formula for the accuracy

From figure 6.1.1, the accuracy of a classification model is calculated by dividing the number of correctly predicted samples by the total number of samples. It reflects how well the model performs overall by showing the proportion of correct predictions out of all predictions made.

		Predicted	
		Positive	Negative
Actual	Positive	True positive	False negative
	Negative	False positive	True negative

Figure 6.1.2 example of confusion matrix

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

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

Figure 6.1.3 formula of precision and recall

$$\text{F1 Score} = \frac{2}{\frac{1}{\text{Precision}} + \frac{1}{\text{Recall}}}$$

$$= \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

Figure 6.1.4 formula of F1 Score

From figure 6.1.2, it shows the confusion matrix, and it is a useful tool to visualize the performance of a classification model. It breaks down the predictions into four categories which is True Positive (TP) number of positive samples are correctly classified as positive, False Negative (FN) number of positive samples are wrongly classified as negative, False Positive (FP) number of negative samples are wrongly classified as positive and True Negative (TN) The number of negative samples are correctly classified as negative.

From the confusion matrix, we can derive several important performance metrics, including precision, recall, and F1-score, which provide more detailed insights into the classifier's behavior. The formular are shown in figure 6.1.3 and figure 6.1.4.

6.2 Testing Setup and Result

Done by: Lai Wei Bin and Lee Jing Yong

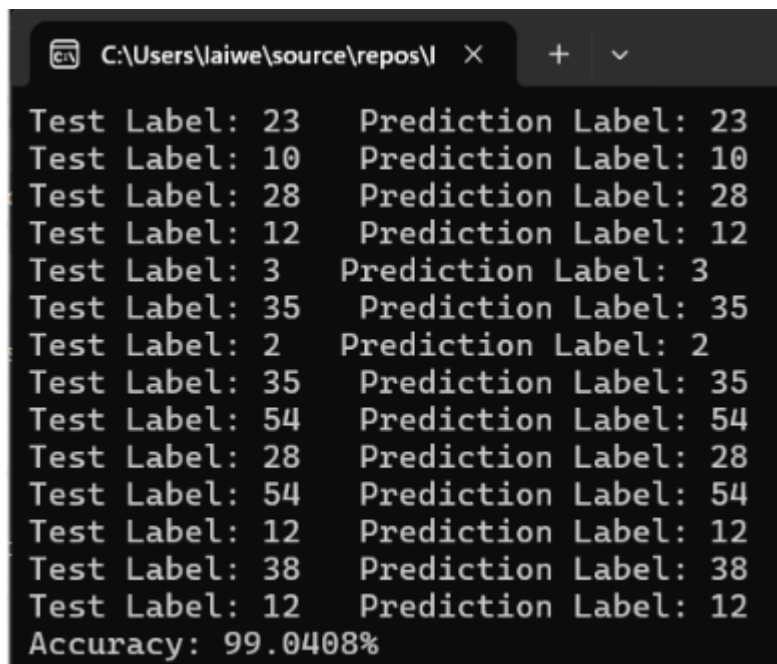
In this project, system testing was conducted using the trsd-train dataset, which contains labeled Chinese traffic sign images. The dataset was provided from the internet and used to train and evaluate two approaches which are HOG feature extraction combined with SVM classification and Color HOG feature extraction combined with Random Forest classification. HOG extracts gradient-based features, while Color HOG incorporates color information to enhance recognition in traffic signs with distinct color features. For classification, the SVM model was applied in the approach to separate traffic sign classes based on edge and orientation features. while the Random Forest model was used in the second approach that leverages an ensemble of decision trees for classification tasks. After the feature extraction and classification, it will directly calculate accuracy and confusion matrix as it is integrating testing. This integration of testing ensures that performance metrics are promptly available, allowing for real-time assessment and adjustment of parameters to optimize the system's effectiveness for traffic sign recognition.

The system's performance was measured using key metrics such as accuracy, precision, recall, F1-score, and the confusion matrix. Accuracy reflected the proportion of correctly classified traffic signs, while the confusion matrix provided insights into true positives (TP), false positives (FP), true negatives (TN), and false negatives (FN). From the confusion matrix, precision and

recall were calculated, and the F1-score was derived to balance the two metrics. Both approaches demonstrated strong performance. The SVM model performs excellent in detecting traffic signs based on edge features and the Random Forest model shows perfect performance when color information was critical. This testing process allowed for further parameter tuning and system optimization to ensure the model's effectiveness in real-world applications. The next part will show the result of the accuracy and confusion matrix for both approaches.

6.2.1 Accuracy

Done by: Lai Wei Bin



```
C:\Users\laiwe\source\repos\
Test Label: 23 Prediction Label: 23
Test Label: 10 Prediction Label: 10
Test Label: 28 Prediction Label: 28
Test Label: 12 Prediction Label: 12
Test Label: 3 Prediction Label: 3
Test Label: 35 Prediction Label: 35
Test Label: 2 Prediction Label: 2
Test Label: 35 Prediction Label: 35
Test Label: 54 Prediction Label: 54
Test Label: 28 Prediction Label: 28
Test Label: 54 Prediction Label: 54
Test Label: 12 Prediction Label: 12
Test Label: 38 Prediction Label: 38
Test Label: 12 Prediction Label: 12
Accuracy: 99.0408%
```

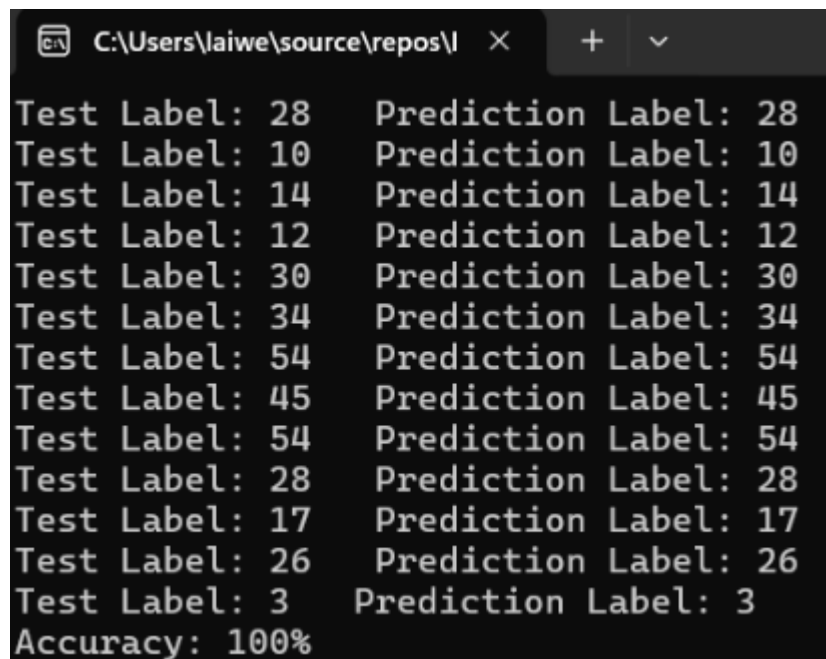
Figure 6.2.1.1 accuracy result of HOG with SVM classification

The accuracy of a model is calculated as the proportion of correct predictions out of the total number of predictions made on the test set and also training set. In this project, the Support Vector Machine SVM model achieved an impressive 99% accuracy as shown in Figure 6.2.1.1. This means that out of every 100 traffic sign images in the test set, 99 were correctly classified by the model. In the testing set, 83 traffic signs in a total of 84 images are successfully classified.

This high accuracy indicates that the SVM was highly effective in distinguishing between different traffic sign categories. The success of the SVM model can be attributed to its ability to find the

optimal hyperplane that separates different classes of traffic signs with the maximum margin. By maximizing this margin, the SVM model minimizes misclassification errors and generalizes well to unseen data.

The 99% accuracy suggests that the model has effectively learned the underlying patterns in the training data and is able to apply its learning to correctly classify most test samples. However, there remains a small error rate (1%), which may be due to a few misclassified signs. These misclassifications could result from ambiguous or noisy images, variation of lightning and other outliers. Overall, the 99% accuracy reflects the strong performance and generalization capability of the SVM model, showing that it is well-suited for traffic sign recognition tasks.



```
C:\Users\laiwe\source\repos\l  X  +  v
Test Label: 28 Prediction Label: 28
Test Label: 10 Prediction Label: 10
Test Label: 14 Prediction Label: 14
Test Label: 12 Prediction Label: 12
Test Label: 30 Prediction Label: 30
Test Label: 34 Prediction Label: 34
Test Label: 54 Prediction Label: 54
Test Label: 45 Prediction Label: 45
Test Label: 54 Prediction Label: 54
Test Label: 28 Prediction Label: 28
Test Label: 17 Prediction Label: 17
Test Label: 26 Prediction Label: 26
Test Label: 3 Prediction Label: 3
Accuracy: 100%
```

Figure 6.2.1.2 accuracy result of Color HOG with Random Forest classification

The accuracy of a model is calculated as the proportion of correct predictions out of the total number of predictions made on the test set. In this project, the Random Forest model achieved a perfect 100% accuracy as shown in Figure 6.2.1.2. This means that all traffic sign images in the test set were correctly classified by the model. In the testing set, all the traffic signs in a total of 84 images are successfully classified.

This perfect accuracy highlights Random Forest model has the strength in handling complex classification tasks in traffic sign recognition. The Random Forest model works by constructing multiple decision trees during training and combining their outputs to make the final prediction. By aggregating predictions from different trees, the model reduces the risk of overfitting and increases robustness, leading to its exceptional performance.

The 100% accuracy suggests that the model not only learned the key features of different traffic sign categories but was also able to apply its learning to correctly classify every image in the test set. This perfect classification rate shows that the Random Forest effectively handles variations in the images like different angles, lighting conditions, or even slight occlusions of the traffic signs.

However, it is crucial to ensure that this performance is maintained when it is applied to new and unseen data. This is because high accuracy might also indicate potential overfitting to the test set. Overall, the 100% accuracy achieved by the Random Forest model reflects its superior classification ability and robustness, making it an excellent tool for this traffic sign recognition task.

6.2.2 Confusion matrix

Done by: Lai Wei Bin

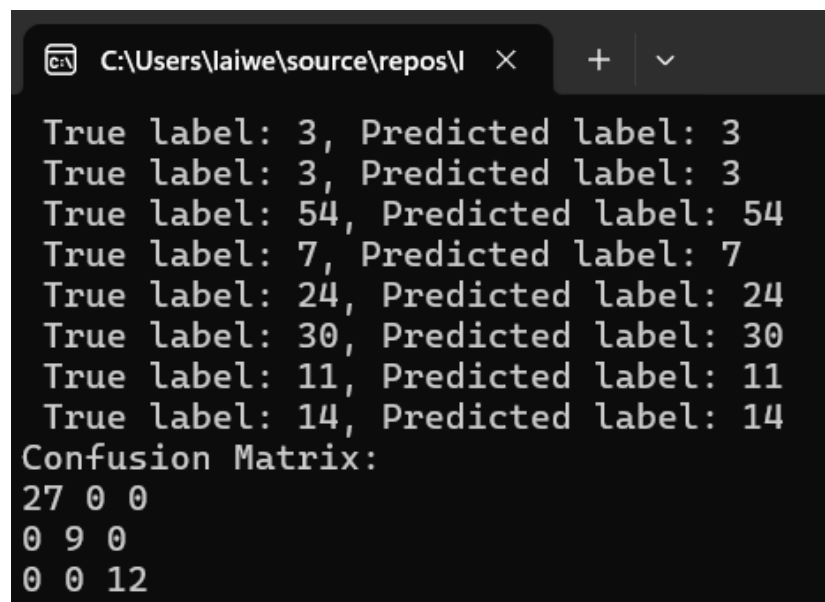


Figure 6.2.2.1 confusion matrix for SVM classification

From Figure 6.2.2.1, it shows the confusion matrix of SVM classification. From this confusion matrix, we can calculate precision, recall and F1-score to evaluate the performance of the model.

6.3 Performance Comparison

Done by: Lai Wei Bin

In this project, both SVM and Random Forest classifiers demonstrated high accuracy, with SVM achieving 99% accuracy on the test set and Random Forest reaching 100% accuracy. While Random Forest performs better than SVM with a small margin, the perfect accuracy of Random Forest can sometimes indicate potential overfitting to the training data. This happens when the model becomes too closely specific and memorizes the training set and this may lead to generalizing badly to unseen data, especially in cases where noise or irrelevant details in the training set may overly influence the model. On the other hand, SVM's slightly lower accuracy may actually indicate better generalization to new data compared to Random Forest model. Both classifiers are highly effective for this problem, but the differences in accuracy need to be balanced against their potential for overfitting and generalization.

Support Vector Machine (SVM) has better generalization properties because it works by finding the optimal hyperplane that maximizes the margin between classes. This wide margin helps ensure that the model performs well on the training data and also the unseen test data. Due to this margin maximization principle, SVM typically has less chance of overfitting and especially when the dataset is high-dimensional. This makes SVM a robust choice for classification tasks where you want to avoid overfitting and ensure that the model performs well in real-world scenarios.

In contrast, Random Forest is highly accurate, while it builds multiple decision trees during training and averages their predictions to make final decisions. This process can lead to high accuracy on the training set and test set. However, Random Forest model has more likelihood to overfitting. This is obviously shown when many trees are used or when noise is present in the data. Overfitting can result in model capturing patterns that are specific to the training set but do not generalize well to new or unseen data.

In terms of computational efficiency, there are notable differences between SVM and Random Forest. SVM can be computationally expensive when dealing with large datasets or high-

dimensional data. The process of finding the optimal hyperplane, particularly when using non-linear kernels like radial basis function (RBF). It involves solving complex optimization problems and takes some time and resources. The computational cost of SVM grows with the number of training samples, making it slower to train on large datasets.

On the other hand, Random Forest can often be more computationally efficient in model training. The ability of Random Forest to parallelize the training process allows it to scale more efficiently with larger datasets. The model can be trained faster in some cases because it leverages multi-core processors. However, the time it takes to make predictions can be slower than SVM due to the need to query multiple trees for each prediction when the forest is large.

In conclusion, the choice between SVM and Random Forest comes to consideration between generalization, overfitting risk, and computational cost for the classification task. SVM offers strong generalization with lower risk of overfitting but slower to train. While Random Forest is faster to train and more scalable. It offers very high accuracy at the cost of increased overfitting risk when dealing with complex datasets, making it critical to carefully manage the model parameters.

6.4 Project Challenges

Done by: Lai Wei Bin

The first challenge of the project is both HOG and Color HOG feature extraction makes computational complexity. HOG can be computationally intensive, especially for large images or high-resolution datasets. The time required for feature extraction may increase significantly. Due to computational complexity, the training time will be the challenges of the project. Depending on the dataset size and complexity, training both SVM with HOG features and color HOG features extraction can take a significant amount of time, especially if hyperparameter tuning is involved.

Moreover, HOG and Color HOG feature extraction have several parameters. Therefore, careful parameter selection is a challenge for us to optimize performance of the feature extraction, as HOG and Color HOG require tuning various settings, such as cell size, block size, and color spaces.

Finding the optimal tuning can be time-consuming and difficult because it may require cross-validation. Additionally, both feature extraction techniques can be sensitive to image quality, noise, and lighting conditions which may significantly impact classification accuracy. Poor-quality images can significantly affect the feature extraction and the accuracy of classification results.

Furthermore, there are some challenges faced when building a random forest classification model. The first challenge is choosing the Right Number of Trees will be difficult. Determining the optimal number of trees in the Random Forest can require experimentation, as too few trees may lead to underfitting while too many can lead to overfitting or increased computational load. Moreover, Random Forest may struggle with highly correlated features, which can lead to redundancy and may affect model performance.

6.5 Objectives Evaluation

Done by: Lee Jing Yong

The first objective of the project is to segment traffic signs using colors and shapes. In the implementation of the project, we have successfully segmented traffic signs using red, blue, and yellow. All of the three colors used to segment the traffic signs have achieved accuracy of more than 80%. However, shape segmentation in this project did not perform as expected with relatively low accuracy rate of not more than 20%. Due to the low accuracy, shape segmentation was not included in the final combined segmentation process. The final segmentation process includes only color segmentation. The second objective is to develop a machine learning model to identify traffic signs from images. The models that were developed are using the SVM and random forest model. Both models achieved an accuracy rate of more than 85%. However, the speed of each model varies. The SVM model takes a longer time to start up during the first load, but it uses very less processing power to predict the traffic sign. The random forest model, however, consumes a lot more processing power. This causes the model to take more time to output the prediction of the traffic sign. Despite this, both models have successfully met the second objective of the project and contributed to the success of this project.

6.6 Error Analysis

Done by: Lai Wei Bin



Figure 6.6.1 misclassification of traffic sign

In Figure 6.6.1, there is a classification error for the traffic sign. The originally image indicating "go straight and turn right ahead". This error appears to be due to issues during image segmentation. Failed to detect the traffic sign from the segmented image because the sign itself was missing from the image. This might be because of poor segmentation. The poor segmentation seems to have been caused by bad lighting conditions. The image is too dark for the segmentation process to correctly isolate and segment the sign. Consequently, the misclassification occurred as the system could not accurately identify the traffic sign from the bad segmented image. Lastly, the improvement of this system to handle the error is to enhance segmentation under challenging lighting conditions. It would be critical to address such errors in future classification tasks.

6.7 Novelties of Work

Done by: Lai Wei Bin

The proposed approach employs multi-stage preprocessing for noise reduction and image enhancement. The implementation approach to reduce the noise of the images by applying Gaussian blur, followed by color space conversions and image subtraction to enhance the color channel, which is important to detect traffic signs with color. The use of deduction between the original and grayscale images to isolate background and traffic sign and enhance color contrast before applying color-based masking.

Besides that, the approach includes adaptive mask generation based on average color intensity. Based on the average blue intensity. The algorithm dynamically adjusts the brightness of the mask. If the average color intensity is too low, the image is brightened to ensure that low-intensity color regions are adequately captured. The approach helps to improve the reliability of the color-based mask for more accurate color-based segmentation in subsequent steps.

Furthermore, the approach includes iterative refinement and contour selection. The iterative refinement process addressed the issue of over segmentation. Firstly, the initial mask and contours are generated. Then, the approached algorithm is used to evaluate the coverage of the segmented area. If the coverage exceeds a certain threshold, it is the potential inclusion of non-sign regions. The initial mask undergoes further refinement through erosion, dilation, and blurring. to ensure the mask does not include the background. This process reduces the likelihood of including background areas in the final segmentation. Moreover, the method prioritizes selecting the longest contour, which typically corresponds to the traffic sign, and refines it to ensure that only the relevant traffic sign region is segmented.

Moreover, the contribution to the field of traffic sign recognition is the high accuracy in identifying and classifying traffic signs. The 99% accuracy for HOG feature extraction with SVM classification model and 100% accuracy for Color HOG feature extraction with Random Forest model achieved in traffic sign recognition. The high accuracy of traffic sign classification is a critical component of autonomous driving and advanced driver-assistance systems (ADAS). Although the risk of overfitting exists, this exceptionally high accuracy highlights the effectiveness of the feature extraction and classification model approach in recognizing traffic signs with high accuracy. In traffic sign detection, even a minor improvement in accuracy can translate into huge impacts on road safety and reliability of the traffic sign recognition system. The high accuracy makes them well-suited for real-world applications such as avoiding misclassification of critical traffic signs that could lead to accidents.

6.8 Concluding Remark

Done by: Lai Wei Bin and Lee Jing Yong

In conclusion, the system's classification performance was thoroughly evaluated by using performance metrics such as accuracy, precision, recall, F1-score, and the confusion matrix.

Both the SVM and Random Forest classifiers achieved high accuracy, 99% of accuracy for HOG feature extraction with SVM and 100% of accuracy for Color HOG with Random Forest. The SVM performs excellent in generalization due to its margin maximization approach, while Random Forest showed perfect performance by leveraging color information in the classification task. However, Random Forest's perfect accuracy could lead to overfitting when the presence of noise in the dataset. This highlights the importance of balancing model performance with generalization ability. Next, the integration of testing after each feature extraction and classification is done. The subsequent analysis of accuracy and confusion matrix results further illuminating the strengths and weakness of both approaches and to decide which are perform better in real-world traffic sign recognition applications

Then, project challenges such as poor segmentation under low lighting especially with shape-based segmentation were identified. Next, by analyzing the errors in classification, the cause of poor segmentation under difficult lighting conditions will be key to further enhancing the system's performance and reliability. Moreover, both models met the project's objectives of accurately classifying traffic signs. In terms of computational efficiency, the SVM being slower during initial training but more efficient in prediction, while Random Forest consumed more processing power but provided high accuracy.

For the novelty and contribution, the results of 99% accuracy using HOG with SVM and 100% using Color HOG with Random Forest are highly valuable for real-world applications like autonomous driving and ADAS. Overall, this project highlights the potential of SVM and Random Forest models for traffic sign recognition, contributing to road safety by minimizing misclassification risks.

CHAPTER 7 CONCLUSION

7.1 Project Review

Done by: Toh Yun Shuang

In this project, we have achieved the goal which is to find a convenient way that could help drivers and autonomous vehicles in accurately recognizing traffic signs and understanding their meanings so that the misunderstanding of traffic sign which can lead to unsafe driving condition and accidents can be reduce. To achieve this goal, we utilized a structured approach that began with acquiring images of the traffic signs, followed by image preprocessing to make it easier for the further process. After that, they proceed with the color segmentation and contours segmentation to isolate relevant regions containing traffic signs using the color and the shape of the traffic signs. Feature extraction techniques were employed to identify distinguishing characteristics of traffic signs and then used to train classifiers for accurate recognition. Finally, the created mask of traffic signs and the meaning of signs have successfully displayed and can be used in applications.

7.2 Discussion and Conclusion

Done by: Toh Yun Shuang

Throughout the project, there are several insights that we can discuss about the process of traffic signs detection. Firstly, the quality and variously of the dataset is mostly affect to the result of traffic sign detection. The challenges such as the noise, environmental conditions, and image quality of the dataset are solved by applying median blurring, grayscale, and color detection. Secondly, effective feature extraction is important for effectively capture and represent the distinctive characteristics of traffic signs for accurate classification and recognition. Therefore, we employed Histogram of Oriented Gradients (HOG) and Color Histogram to extract relevant features from segmented images, enabling the models to distinguish between several types of signs effectively. Lastly for the classification, the classifiers demonstrated varying levels of accuracy depending on the features used to ensure that each detected sign is correctly identified. The project used Support Vector Machines (SVM) and got an accuracy result of 99.04% and 100% for Random Forest Classifier. The high accuracy ensures that autonomous systems can reliably interpret traffic signs and deliver a consistent performance, building trust in autonomous systems.

Overall, the project successfully demonstrated the application of a systematic approach to traffic sign detection, providing insights into the strengths and limitations of current techniques. The result reaffirmed the potential of deep learning methods while also highlighting the importance of dataset quality and the need for refined preprocessing steps.

7.3 Future Work

Done by: Har Sze Hao

The future direction of this project focuses on three key areas for improvement and expansion.

Firstly, the project aims to enhance segmentation techniques by incorporating more sophisticated methodologies, particularly leveraging deep learning-based models. These advanced approaches are expected to significantly improve the system's capability to isolate and identify traffic signs in complex and dynamic visual environments, thereby increasing detection accuracy and robustness. Second, future research will explore advanced feature extraction techniques, particularly those that utilize deep feature hierarchies. By focusing on more detailed and informative representations of traffic signs, this approach has the potential to enhance both classification accuracy and the model's ability to generalize across diverse traffic scenarios.

Finally, the project envisions expanding into multimodal detection by integrating visual data with supplementary sensor inputs such as radar and LIDAR. This integration would enable the system to maintain high detection accuracy even in challenging conditions, such as low visibility or adverse weather. By leveraging data from multiple sources, the system could achieve more reliable and consistent performance in real-world applications, advancing the practical usability of traffic sign recognition technology.

REFERENCES

- [1] C. Liu, S. Li, F. Chang, and Y. Wang, "Machine Vision Based Traffic Sign Detection Methods: Review, Analyses and Perspectives," *IEEE Access*, vol. 7, pp. 86578–86596, 2019, doi: <https://doi.org/10.1109/access.2019.2924947>.
- [2] V. Wiley and T. Lucas, "Computer Vision and Image Processing: A Paper Review," *International Journal of Artificial Intelligence Research*, vol. 2, no. 1, pp. 29–36, Jun. 2018, doi: <https://doi.org/10.29099/ijair.v2i1.42>
- [3] M. A. Ponti, L. S. F. Ribeiro, T. S. Nazare, T. Bui and J. Collomosse, "Everything You Wanted to Know about Deep Learning for Computer Vision but Were Afraid to Ask," *2017 30th SIBGRAPI Conference on Graphics, Patterns and Images Tutorials (SIBGRAPI-T)*, Niteroi, Brazil, 2017, pp. 17-41, doi: 10.1109/SIBGRAPI-T.2017.12.
- [4] R. Szeliski, *Computer Vision: Algorithms and Applications*. Springer Nature, 2022. Accessed: Jul. 21, 2024. [Online]. Available: https://books.google.com.my/books?hl=zh-CN&lr=&id=QptXEAAAQBAJ&oi=fnd&pg=PR9&dq=computer+vision+used&ots=BNwbz3Vxul&sig=g7BKwg7u_zjAEIvpbK19ng74qvY&redir_esc=y#v=onepage&q=computer%20vision%20used&f=false
- [5] S. Paneru and I. Jeelani, "Computer vision applications in construction: Current state, opportunities & challenges," *Automation in Construction*, vol. 132, p. 103940, Dec. 2021, doi: <https://doi.org/10.1016/j.autcon.2021.103940>.
- [6] J. Yang, T. Sun, W. Zhu, and Z. Li, "A Lightweight Traffic Sign Recognition Model Based on Improved YOLOv5," *IEEE access*, vol. 11, pp. 115998–116010, Jan. 2023, doi: <https://doi.org/10.1109/access.2023.3326000>.
- [7] P. Jiang, D. Ergu, F. Liu, Y. Cai, and B. Ma, "A Review of Yolo Algorithm Developments," *Procedia Computer Science*, vol. 199, pp. 1066–1073, Jan. 2022, doi: <https://doi.org/10.1016/j.procs.2022.01.135>.
- [8] G. Boesch, "YOLOv9: Advancements in Real-time Object Detection (2024)," *viso.ai*, Feb. 27, 2024. <https://viso.ai/computer-vision/yolov9/#:~:text=YOLOv9%20is%20the%20latest%20version>
- [9] G. Wang, G. Ren, Z. Wu, Y. Zhao, and L. Jiang, "A robust, coarse-to-fine traffic sign detection method," Aug. 2013, doi: <https://doi.org/10.1109/ijcnn.2013.6706812>.

- [10] Wu, M., Yue, H., Wang, J., Huang, Y., Liu, M., Jiang, Y., Ke, C. and Zeng, C. (2020), Object detection based on RGC mask R-CNN. IET Image Processing, 14: 1502-1508. <https://doi.org/10.1049/iet-ipr.2019.0057>
- [11] H. S. Lee and K. Kim, "Simultaneous Traffic Sign Detection and Boundary Estimation Using Convolutional Neural Network," in IEEE Transactions on Intelligent Transportation Systems, vol. 19, no. 5, pp. 1652-1663, May 2018, doi: 10.1109/TITS.2018.2801560.
- [12] "Traffic Sign Recognition Database," *nlpr.ia.ac.cn*. <https://nlpr.ia.ac.cn/pal/trafficdata/recognition.html>
- [13] F. Zaklouta and B. Stanciulescu, "Real-Time Traffic-Sign Recognition Using Tree Classifiers," IEEE Transactions on Intelligent Transportation Systems, vol. 13, no. 4, pp. 1507-1514, Dec. 2012, doi: 10.1109/TITS.2012.2225618.
- [14] C. G. Kiran, L. V. Prabhu, V. Abdu Rahiman and K. Rajeev, "Traffic Sign Detection and Pattern Recognition Using Support Vector Machine," 2009 Seventh International Conference on Advances in Pattern Recognition, Kolkata, India, 2009, pp. 87-90, doi: 10.1109/ICAPR.2009.58.

Appendices

```

1  #include <opencv2/core/core.hpp>
2  #include <opencv2/highgui/highgui.hpp>
3  #include <opencv2/imgproc/imgproc.hpp>
4  #include <opencv2/opencv.hpp>
5  #include <opencv2/ml.hpp>
6  #include <iostream>
7  #include <fstream>
8  #include <vector>
9  #include <string>
10 #include <algorithm>
11 #include <random>
12 #include <numeric>
13 #include "Supp.h"
14
15 using namespace std;
16 using namespace cv;
17 using namespace cv::ml;
18
19 Ptr<ml::StatModel> SVMmodel;
20 Ptr<ml::StatModel> ForestModel;
21
22 string classifyShape(const vector<Point>& contour);
23 String predictAndShowMeaning(const string&, const string&, const Mat);
24 string getClassMeaning(int);
25
26 int calculateCoverage(Mat source);
27
28 ////////////////////////////////////////////////// Segmentation part////////////////////////////////////
29 void generateContour(Mat originalImage, Mat processedImage, Mat& outputImage, Mat& outputMask, Mat& segmentedImage) {
30
31     Point2i center;
32     vector<vector<Point>> contours;
33     vector<Vec4i> hierarchy;
34     findContours(processedImage, contours, hierarchy, RETR_EXTERNAL, CHAIN_APPROX_NONE);
35
36     // Draw contours on the original image
37     Mat contourImage = originalImage.clone();
38     drawContours(contourImage, contours, -1, Scalar(0, 255, 0), 1); // Draw contours in green
39     contourImage.copyTo(outputImage);
40
41
42     Mat canvasColor, canvasGray;
43     canvasColor.create(originalImage.rows, originalImage.cols, CV_8UC3);
44     canvasGray.create(originalImage.rows, originalImage.cols, CV_8U);
45
46     int index = 0, max = 0; // used to record down the longest contour
47
48     for (int i = 0; i < contours.size(); i++) { // We could have more than one sign in image
49         canvasGray = 0;
50         if (max < contours[i].size()) { // Find the longest contour as sign boundary
51             max = contours[i].size();
52             index = i;
53         }
54         drawContours(canvasColor, contours, i, Scalar(0, 255, 0)); // draw boundaries
55         drawContours(canvasGray, contours, i, 255);
56
57         // The code below compute the center of the region
58         Moments M = moments(canvasGray);

```

```

59     center.x = M.m10 / M.m00;
60     center.y = M.m01 / M.m00;
61
62     floodFill(canvasGray, center, 255); // fill inside sign boundary
63 }
64 canvasGray = 0;
65 drawContours(canvasGray, contours, index, 255);
66
67 Moments M = moments(canvasGray);
68 center.x = M.m10 / M.m00;
69 center.y = M.m01 / M.m00;
70
71 // generate mask of the sign
72 if (center.x > 0 && center.y > 0) { //Ensure the center is found
73     floodFill(canvasGray, center, 255); // fill inside sign boundary
74     cvtColor(canvasGray, canvasGray, COLOR_GRAY2BGR);
75     canvasGray.copyTo(outputMask);
76
77     canvasColor = canvasGray & originalImage;
78     canvasColor.copyTo(segmentedImage);
79 }
80 else {
81     canvasGray.copyTo(outputMask);
82     originalImage.copyTo(segmentedImage);
83 }
84 }
85
86
87
88 Mat RedSegmentation(Mat win[], int& winCount, Mat legends[], Mat deduct) {
89     Mat redMask, mask;
90
91     //Get the average red color
92     int avg, pixels = deduct.rows * deduct.cols;
93     Scalar deductSum = cv::sum(deduct);
94     avg = deductSum[2] / pixels;
95
96     if (avg < 10) deduct *= 2; // brighten the image if too low
97
98
99     Mat hsv, lowerRed;
100    cvtColor(deduct, hsv, COLOR_BGR2HSV);
101
102
103    Scalar redLower(0, 0, 35);
104    Scalar redUpper(0, 255, 165);
105
106    inRange(hsv, redLower, redUpper, lowerRed); //Binary mask for red color in range
107
108    cvtColor(lowerRed, lowerRed, COLOR_GRAY2BGR); //Binary mask to BGR
109
110
111    // Split the image into color channels
112    std::vector<cv::Mat> colorChannels;
113    split(lowerRed, colorChannels);
114

```

```

115
116 // Get the red channel (colorChannels[2] is the red channel)
117 // Convert red channel to BGR format (just to visualize)
118 cvtColor(colorChannels[2], redMask, COLOR_GRAY2BGR);
119 //putText(legends[winCount], "Red mask", Point(5, 11), 1, 1, Scalar(250, 250, 250), 1);
120 //redMask.copyTo(win[winCount++]);
121
122
123 // Threshold the red channel to create a binary mask
124 threshold(colorChannels[2], mask, 15, 255, THRESH_BINARY);
125
126 return mask;
127 }
128
129
130 Mat BlueSegmentation(Mat win[], int& winCount, Mat legends[], Mat deduct) {
131     Mat blueMask, mask;
132
133     //Get the average red color
134     int avg, pixels = deduct.rows * deduct.cols;
135     Scalar deductSum = cv::sum(deduct);
136     avg = deductSum[0] / pixels; //Blue channel
137
138     if (avg < 10) deduct *= 5; // brighten the image if too low
139
140
141     Mat hsv, lowerBlue;
142     cvtColor(deduct, hsv, COLOR_BGR2HSV);
143
144     // Define the blue color range in HSV
145     Scalar blueLower(100, 90, 35); // Lower bound for blue
146     Scalar blueUpper(140, 255, 255); // Upper bound for blue
147
148     inRange(hsv, blueLower, blueUpper, lowerBlue); //Binary mask for red color in range
149
150     cvtColor(lowerBlue, lowerBlue, COLOR_GRAY2BGR); //Binary mask to BGR
151
152
153     // Split the image into color channels
154     std::vector<cv::Mat> colorChannels;
155     split(lowerBlue, colorChannels);
156
157
158 // Get the red channel (colorChannels[2] is the red channel)
159 // Convert red channel to BGR format (just to visualize)
160 cvtColor(colorChannels[2], blueMask, COLOR_GRAY2BGR);
161 //putText(legends[winCount], "Blue mask", Point(5, 11), 1, 1, Scalar(250, 250, 250), 1);
162 //blueMask.copyTo(win[winCount++]);
163
164
165 // Threshold the red channel to create a binary mask
166 threshold(colorChannels[2], mask, 15, 255, THRESH_BINARY);

```

```

167     return mask;
168 }
169
170
171
172 Mat YellowSegmentation(Mat win[], int& winCount, Mat legends[], Mat deduct) {
173     Mat hsv, yellowMask;
174     cvtColor(deduct, hsv, COLOR_BGR2HSV);
175
176     // Define the blue color range in HSV
177     Scalar yellowLower(14, 120, 0);
178     Scalar yellowUpper(38, 255, 255);
179
180     inRange(hsv, yellowLower, yellowUpper, yellowMask); //Binary mask for red color in range
181
182     cvtColor(yellowMask, yellowMask, COLOR_GRAY2BGR); //Binary mask to BGR
183
184     Mat kernel = getStructuringElement(MORPH_ELLIPSE, Size(3, 3));
185     morphologyEx(yellowMask, yellowMask, MORPH_CLOSE, kernel);
186     morphologyEx(yellowMask, yellowMask, MORPH_OPEN, kernel);
187
188
189     //putText(legends[winCount], "Yellow Mask", Point(5, 11), 1, 1, Scalar(250, 250, 250), 1);
190     //yellowMask.copyTo(win[winCount++]);
191
192     cvtColor(yellowMask, yellowMask, COLOR_BGR2GRAY);
193
194     return yellowMask;
195 }
196
197 int calculateCoverage(Mat source) {
198     // Convert the image to grayscale
199     cv::Mat grayImage;
200
201     if (source.channels() == 3) {
202         cvtColor(source, grayImage, COLOR_BGR2GRAY);
203     }
204     else {
205         source.copyTo(grayImage);
206     }
207
208     // Count non-zero pixels in the grayscale image
209     int nonZeroCount = cv::countNonZero(grayImage);
210
211     // if higher than 80% its most likely wrong
212     int pixels = source.rows * source.cols;
213     int coverage = double(nonZeroCount) / double(pixels) * 100;
214     return coverage;
215 }
216
217 void segmentation1() {
218     int const noCols = 2;
219     int const noRows = 1;
220     int const heightGap = 15, widthGap = 3;
221
222     String imgPattern("Inputs/Traffic signs/*.png");
223     vector<string> imageNames;
224     Mat originalImage, processedImage, largeWin, win[noCols * noRows], legends[noCols * noRows];

```

```

225 Mat deduct, mask;
226
227 // Get all the image names
228 cv::glob(imgPattern, imageNames, true);
229
230 for (int i = 0; i < imageNames.size(); i++) {
231     originalImage = imread(imageNames[i]);
232
233     createWindowPartition(originalImage, largeWin, win, legends, noRows, noCols);
234
235     int winCount = 0;
236     putText(legends[winCount], "Original", Point(5, 11), 1, 1, Scalar(250, 250, 250), 1);
237     originalImage.copyTo(win[winCount++]);
238
239     GaussianBlur(originalImage, processedImage, Size(9, 9), 0, 0); // Apply gaussian blur to reduce noise
240
241     Mat blurredImage;
242     processedImage.copyTo(blurredImage);
243
244     cvtColor(originalImage, processedImage, COLOR_BGR2GRAY); // Change to gray scale
245     cvtColor(processedImage, processedImage, COLOR_GRAY2BGR); // Change back to rgb so that it is visible
246
247     deduct = originalImage - processedImage;
248
249     //putText(legends[winCount], "Deduction", Point(5, 11), 1, 1, Scalar(250, 250, 250), 1);
250     //deduct.copyTo(win[winCount++]);
251
252
253
254     Mat erode1 = cv::getStructuringElement(cv::MORPH_ELLIPSE, cv::Size(4, 3));
255     Mat erode2 = cv::getStructuringElement(cv::MORPH_ELLIPSE, cv::Size(3, 3));
256     Mat dilate1 = cv::getStructuringElement(cv::MORPH_ELLIPSE, Size(6, 4));
257
258     Mat masks[3];
259     masks[0] = RedSegmentation(win, winCount, legends, deduct);
260     masks[1] = BlueSegmentation(win, winCount, legends, deduct);
261     masks[2] = YellowSegmentation(win, winCount, legends, blurredImage);
262
263     Mat finalSegmentedOutput;
264     int finalCoverage = 0;
265
266     for (int j = 0; j < 3; j++) { ... }
267
268     if (finalCoverage == NULL) { // If no assignment
269         originalImage.copyTo(finalSegmentedOutput);
270     }
271
272     String name = predictAndShowMeaning("svm_modelHog.yml", "SVM", originalImage);
273
274     putText(legends[winCount], name, Point(5, 11), 1, 1, Scalar(250, 250, 250), 1);
275     finalSegmentedOutput.copyTo(win[winCount++]);
276     cout << imageNames[i];
277
278     // Display the large window with sub-windows
279     imshow("Classified Image", largeWin);
280
281     waitKey(0); // Wait for a key press
282
283     destroyAllWindows(); //Close the window

```

```

320     }
321 }
322
323 }
324 void segmentation2() {
325     int const noCols = 2;
326     int const noRows = 1;
327     int const heightGap = 15, widthGap = 3;
328
329     String imgPattern("Inputs/Traffic signs/*.png");
330     vector<string> imageNames;
331     Mat originalImage, processedImage, largeWin, win[noCols * noRows], legends[noCols * noRows];
332     Mat deduct, mask;
333
334     // Get all the image names
335     cv::glob(imgPattern, imageNames, true);
336     cout << imageNames.size() << endl;
337
338     for (int i = 0; i < imageNames.size(); i++) {
339         originalImage = imread(imageNames[i]);
340
341
342         createWindowPartition(originalImage, largeWin, win, legends, noRows, noCols);
343
344         int winCount = 0;
345         putText(legends[winCount], "Original", Point(5, 11), 1, 1, Scalar(250, 250, 250), 1);
346         originalImage.copyTo(win[winCount++]);
347
348         GaussianBlur(originalImage, processedImage, Size(9, 9), 0, 0); // Apply gaussian blur to reduce noise
349
350         Mat blurredImage;
351         processedImage.copyTo(blurredImage);
352
353         cvtColor(originalImage, processedImage, COLOR_BGR2GRAY); // Change to gray scale
354         cvtColor(processedImage, processedImage, COLOR_GRAY2BGR); // Change back to rgb so that it is visible
355
356         deduct = originalImage - processedImage;
357
358         //putText(legends[winCount], "Deduction", Point(5, 11), 1, 1, Scalar(250, 250, 250), 1);
359         //deduct.copyTo(win[winCount++]);
360
361
362         Mat erode1 = cv::getStructuringElement(cv::MORPH_ELLIPSE, cv::Size(4, 3));
363         Mat erode2 = cv::getStructuringElement(cv::MORPH_ELLIPSE, cv::Size(3, 3));
364         Mat dilate1 = cv::getStructuringElement(cv::MORPH_ELLIPSE, Size(6, 4));
365
366         Mat masks[3];
367         masks[0] = RedSegmentation(win, winCount, legends, deduct);
368         masks[1] = BlueSegmentation(win, winCount, legends, deduct);
369         masks[2] = YellowSegmentation(win, winCount, legends, blurredImage);
370
371         Mat finalSegmentedOutput;
372         int finalCoverage = 0;
373
374         for (int j = 0; j < 3; j++) { ... }
375
376         if (finalCoverage == NULL) { // If no assignment

```

```

415         originalImage.copyTo(finalSegmentedOutput);
416     }
417     String name = predictAndShowMeaning("randomForest_modelHOG.yml", "RTrees", originalImage);
418     putText(legends[winCount], name, Point(5, 11), 1, 1, Scalar(250, 250, 250), 1);
419     finalSegmentedOutput.copyTo(win[winCount++]);
420
421
422     // Display the large window with sub-windows
423     imshow("Classified Image", largeWin);
424
425     waitKey(0); // Wait for a key press
426
427     destroyAllWindows(); //Close the window
428 }
429
430
431 /////////////////////////////////////////////////////////////////// Feature extraction part/////////////////////////////////////////////////////////////////
432 // Function to extract HOG features from an image
433 void extractHOGFeatures(const Mat& image, vector<float>& features, const Size& resizeSize = Size(64, 64)) { ... }
434
435 // Function to write HOG features to CSV
436 void writeHOGFeaturesToCSV(const string& csvPath, const vector<string>& imageNames) { ... }
437
438 // Function to extract color histogram features from an image
439 void extractColorHistogram(const Mat& image, vector<float>& features) { ... }
440
441 void generateConfusionMatrix(const Mat& trueLabels, const Mat& predictedLabels, Mat& confusionMatrix) {
442     int numLabels = 3; // Replace with the actual number of classes
443     confusionMatrix = Mat::zeros(numLabels, numLabels, CV_32S);
444
445     // Convert predictions to integer format if necessary
446     Mat tempPredictions;
447     predictedLabels.convertTo(tempPredictions, CV_32S);
448
449     // Check dimensions
450     if (trueLabels.rows != tempPredictions.rows) {
451         cerr << "Error: Number of true labels does not match number of predicted labels." << endl;
452         return;
453     }
454
455     // Populate confusion matrix
456     for (int i = 0; i < trueLabels.rows; ++i) {
457         int trueLabel = trueLabels.at<int>(i, 0);
458         int predictedLabel = tempPredictions.at<int>(i, 0);
459
460         if (trueLabel >= 0 && trueLabel < numLabels && predictedLabel >= 0 && predictedLabel < numLabels) {
461             confusionMatrix.at<int>(trueLabel, predictedLabel)++;
462         }
463         else {
464             cerr << " True label: " << trueLabel << ", Predicted label: " << predictedLabel << endl;
465         }
466     }
467
468     // Print the confusion matrix
469     cout << "Confusion Matrix:" << endl;
470     for (int i = 0; i < numLabels; ++i) {
471         for (int j = 0; j < numLabels; ++j) {
472             cout << confusionMatrix.at<int>(i, j) << " ";

```



```

571     }
572     cout << endl;
573 }
574 }
575
576 void HOG() {
577     string folderPath = "Inputs/tsrd-train/"; // Replace with your folder path
578     vector<string> imageNames;
579     glob(folderPath + "*.png", imageNames, true);
580
581     if (imageNames.empty()) {
582         cerr << "Error: No images found in the directory: " << folderPath << endl;
583         return;
584     }
585
586     string csvPath = "HOGFeatures.csv";
587     writeHOGFeaturesToCSV(csvPath, imageNames);
588 }
589
590 // Function to load features and labels from CSV
591
592 void ColorHOG() {
593     // Folder path for images
594     string folderPath = "Inputs/tsrd-train/"; // Replace with your folder path
595     vector<string> imageNames;
596     glob(folderPath + "*.png", imageNames, true);
597
598     // Open CSV file for writing
599     ofstream csvFile("ColorHistogramFeatures.csv");
600     if (!csvFile.is_open()) {
601         cerr << "Error: Could not open CSV file for writing." << endl;
602         return;
603     }
604     cout << "Extracting features..... Please wait for a while" << endl;
605     // Write CSV header
606     csvFile << "filename,label,feature_0,feature_1,...,feature_N" << endl;
607
608     for (const string& imagePath : imageNames) {
609         Mat image = imread(imagePath);
610         if (image.empty()) {
611             cerr << "Error: Could not open or find the image " << imagePath << endl;
612             continue;
613         }
614
615         resize(image, image, Size(64, 64));
616
617         vector<float> features;
618         extractColorHistogram(image, features);
619
620         // Extract filename from path
621         string filename = imagePath.substr(imagePath.find_last_of("/") + 1);
622
623         // Extract label from the first 3 digits of the filename
624         string labelStr = filename.substr(0, 3); // Extract first 3 characters
625         int label = stoi(labelStr); // Convert to integer
626
627         // Write features to CSV
628         csvFile << filename << "," << label;

```

```

629     for (float feature : features) {
630         csvFile << "," << feature;
631     }
632     csvFile << endl;
633 }
634
635 csvFile.close();
636 cout << "Feature extraction complete and saved to ColorHistogramFeatures.csv" << endl;
637 }
638 }
639
640
641
642 ////////////////////////////////////////////////// Classification part ////////////////////////////////////////////
643 // Function to load features and labels from CSV
644 void loadCSV(const string& filename, Mat& features, Mat& labels) {
645     ifstream file(filename);
646     if (!file.is_open()) {
647         cerr << "Error: Could not open file " << filename << endl;
648         return;
649     }
650
651     string line;
652
653     // Skip header line
654     getline(file, line);
655
656     vector<float> rowFeatures;
657     vector<int> rowLabels;
658
659     while (getline(file, line)) {
660         stringstream ss(line);
661         string item;
662
663         // Read filename (skip it)
664         getline(ss, item, ',');
665
666         // Read label
667         int label;
668         if (!(getline(ss, item, ',') && istringstream(item) >> label)) {
669             cerr << "Error: Could not parse label from line: " << line << endl;
670             continue; // Skip this line if label parsing fails
671         }
672         rowLabels.push_back(label);
673
674         // Read features
675         rowFeatures.clear();
676         while (getline(ss, item, ',')) {
677             try {
678                 float feature = stof(item);
679                 rowFeatures.push_back(feature);
680             }
681             catch (const invalid_argument& e) {
682                 cerr << "Error: Could not parse feature from line: " << line << endl;
683                 rowFeatures.clear();
684                 break; // Skip this line if feature parsing fails
685             }
686         }

```

```

687
688 // Convert rowFeatures to a Mat if it is not empty
689 if (!rowFeatures.empty()) {
690     Mat row(rowFeatures);
691     features.push_back(row.t());
692 }
693
694
695 // Convert labels vector to Mat
696 labels = Mat(rowLabels, true).reshape(1, rowLabels.size());
697 labels.convertTo(labels, CV_32S); // Ensure labels are in correct format
698 features.convertTo(features, CV_32F); // Ensure features are in correct format
699
700 // Debug: Print the size of the features and labels
701 cout << "Loaded features: " << features.rows << "x" << features.cols << endl;
702 cout << "Loaded labels: " << labels.rows << "x" << labels.cols << endl;
703 }
704
705 // Function to shuffle and split data
706 void shuffleAndSplit(const Mat& features, const Mat& labels, Mat& trainFeatures, Mat& trainLabels, Mat& testFeatures, Mat& testLabels, float trainRatio) {
707     // Create an index vector
708     vector<int> indices(features.rows);
709     iota(indices.begin(), indices.end(), 0); // Fill with 0, 1, ..., features.rows-1
710
711     // Shuffle indices
712     random_device rd;
713     mt19937 g(rd());
714     shuffle(indices.begin(), indices.end(), g);
715
716     // Split indices
717     int trainSize = static_cast<int>(trainRatio * features.rows);
718
719     // Split data
720     trainFeatures.create(trainSize, features.cols, features.type());
721     trainLabels.create(trainSize, labels.cols, labels.type());
722     testFeatures.create(features.rows - trainSize, features.cols, features.type());
723     testLabels.create(features.rows - trainSize, labels.cols, labels.type());
724
725     for (int i = 0; i < features.rows; ++i) {
726         if (i < trainSize) {
727             features.row(indices[i]).copyTo(trainFeatures.row(i));
728             labels.row(indices[i]).copyTo(trainLabels.row(i));
729         }
730         else {
731             features.row(indices[i]).copyTo(testFeatures.row(i - trainSize));
732             labels.row(indices[i]).copyTo(testLabels.row(i - trainSize));
733         }
734     }
735 }
736
737 void SVMClassification() {
738     // Load all data
739     Mat allFeatures, allLabels;
740     loadCSV("HOGFeatures.csv", allFeatures, allLabels);
741
742     // Shuffle and split data
743     Mat trainFeatures, trainLabels, testFeatures, testLabels;
744     shuffleAndSplit(allFeatures, allLabels, trainFeatures, trainLabels, testFeatures, testLabels, 0.80f);

```

```

745 // Define SVM parameters
746 Ptr<SVM> svm = SVM::create();
747 svm->setKernel(SVM::LINEAR); // or SVM::RBF for non-linear
748 svm->setC(1); // Regularization parameter
749 svm->setGamma(0.5); // Used if kernel is RBF
750 svm->setTermCriteria(TermCriteria(TermCriteria::MAX_ITER, 100, 1e-6));
751
752 // Train SVM
753 svm->train(trainFeatures, ROW_SAMPLE, trainLabels);
754 svm->save("svm_modelHOG.yml");
755
756 cout << "SVM model trained and saved to svm_modelHOG.yml" << endl;
757
758 // Predict and evaluate
759 Mat predictions;
760 Ptr<SVM> loadedSVM = SVM::load("svm_modelHOG.yml");
761 if (loadedSVM.empty()) {
762     cerr << "Error: Could not load the SVM model." << endl;
763     return;
764 }
765
766 loadedSVM->predict(testFeatures, predictions);
767
768 // Calculate accuracy
769 int correct = 0;
770 for (int i = 0; i < testLabels.rows; ++i) {
771     cout << "Test Label: " << testLabels.at<int>(i, 0) << " Prediction Label: " << static_cast<int>(predictions.at<float>(i, 0)) << endl;
772     if (testLabels.at<int>(i, 0) == static_cast<int>(predictions.at<float>(i, 0))) {
773         ++correct;
774     }
775 }
776
777 float accuracy = static_cast<float>(correct) / testLabels.rows;
778 cout << "Accuracy: " << accuracy * 100 << "%" << endl;
779
780 // Generate and display the confusion matrix
781 Mat confusionMatrix;
782 generateConfusionMatrix(testLabels, predictions, confusionMatrix);
783
784 }
785
786 void RandomForest() {
787     // Load all data
788     Mat allFeatures, allLabels;
789     loadCSV("HOGFeatures.csv", allFeatures, allLabels);
790
791     // Shuffle and split data
792     Mat trainFeatures, trainLabels, testFeatures, testLabels;
793     shuffleAndSplit(allFeatures, allLabels, trainFeatures, trainLabels, testFeatures, testLabels, 0.99f);
794
795     // Define Random Forest parameters
796     Ptr<RTrees> randomForest = RTrees::create();
797     randomForest->setMaxDepth(10); // Maximum depth of the tree
798     randomForest->setMinSampleCount(2); // Minimum number of samples required at a leaf node
799     randomForest->setRegressionAccuracy(0); // Regression accuracy: N/A here
800     randomForest->setUseSurrogates(false); // Use surrogate splits
801
802

```

```

803 randomForest->setMaxCategories(10); // Maximum number of categories (useful for categorical data)
804 randomForest->setPriors(Mat()); // The prior probabilities of the classes
805 randomForest->setTermCriteria(TermCriteria(TermCriteria::MAX_ITER, 100, 1e-6)); // Termination criteria
806
807 // Train Random Forest
808 randomForest->train(trainFeatures, ROW_SAMPLE, trainLabels);
809 randomForest->save("randomForest_modelHOG.yml");
810
811 cout << "Random Forest model trained and saved to randomForest_modelHOG.yml" << endl;
812
813 // Predict and evaluate
814 Mat predictions;
815 Ptr<RTrees> loadedRandomForest = RTrees::load("randomForest_modelHOG.yml");
816 if (loadedRandomForest.empty()) {
817     cerr << "Error: Could not load the Random Forest model." << endl;
818     return;
819 }
820
821 loadedRandomForest->predict(testFeatures, predictions);
822
823 // Calculate accuracy
824 int correct = 0;
825 for (int i = 0; i < testLabels.rows; ++i) {
826     cout << "Test Label: " << testLabels.at<int>(i, 0) << " Prediction Label: " << static_cast<int>(predictions.at<float>(i, 0)) << endl;
827     if (testLabels.at<int>(i, 0) == static_cast<int>(predictions.at<float>(i, 0))) {
828         ++correct;
829     }
830 }
831
832 Mat confusionMatrix;
833 generateConfusionMatrix(testLabels, predictions, confusionMatrix);
834
835 float accuracy = static_cast<float>(correct) / testLabels.rows;
836 cout << "Accuracy: " << accuracy * 100 << "%" << endl;
837
838 }
839
840
841
842
843 ////////////////////////////////////////////////// Display part ////////////////////////////////////////////
844 // Function to get the name of the class
845 string getClassMeaning(int classNumber) {
846     map<int, string> classMeanings = {
847         {0, "Speed limit (5km/h)"},
848         {1, "Speed limit (15km/h)"},
849         {2, "Speed limit (30km/h)"},
850         {3, "Speed limit (40km/h)"},
851         {4, "Speed limit (50km/h)"},
852         {5, "Speed limit (60km/h)"},
853         {6, "Speed limit (70km/h)"},
854         {7, "Speed limit (80km/h)"},
855         {8, "No straight&turn left"},
856         {9, "No straight&turn right"},
857         {10, "No go straight"},
858         {11, "No turn left"},
859         {12, "No turn left&right"},
860         {13, "No turn right"},

```

```

861     {14, "No overtaking"},
862     {15, "No U-turn"},
863     {16, "No vehicle allowed"},
864     {17, "No horn allowed"},
865     {18, "Speed limit (40km/h)"},
866     {19, "Speed limit (50km/h)"},
867     {20, "Go straight or turn right"},
868     {21, "Go straight"},
869     {22, "Turn left ahead"},
870     {23, "Turn left &right ahead"},
871     {24, "Turn right ahead"},
872     {25, "Keep left"},
873     {26, "Keep right"},
874     {27, "Roundabout"},
875     {28, "Only car allowed"},
876     {29, "Sound horn sign"},
877     {30, "Bicycle lane"},
878     {31, "U-turn sign"},
879     {32, "Road divides"},
880     {33, "Traffic light sign"},
881     {34, "Warning sign"},
882     {35, "Pedestrian crossing symbol"},
883     {36, "Bicycle traffic warning"},
884     {37, "School crossing sign"},
885     {38, "Sharp bend"},
886     {39, "Sharp bend"},
887     {40, "Danger steep hill ahead warning"},
888     {41, "Danger steep hill ahead warning"},
889     {42, "Slowing sign"},
890     {43, "T-junction ahead"},
891     {44, "T-junction ahead"},
892     {45, "Village warning sign"},
893     {46, "Snake road"},
894     {47, "Railroad level crossing sign"},
895     {48, "Under construction"},
896     {49, "Snake road"},
897     {50, "Railroad level crossing sign"},
898     {51, "Accident frequent happened sign"},
899     {52, "Stop"},
900     {53, "No entry"},
901     {54, "No Stopping"},
902     {55, "No entry"},
903     {56, "Give way"},
904     {57, "Stop for checking purpose"}
905 };
906
907 if (classMeanings.find(classNumber) != classMeanings.end()) {
908     return classMeanings[classNumber];
909 }
910 else {
911     return "Unknown traffic sign";
912 }
913 }
914
915 void showWindow(Mat src, Mat seg, String class_name) {
916     int const noOfImagePerCol = 1, noOfImagePerRow = 2;
917     Mat largeWin, win[noOfImagePerRow * noOfImagePerCol], legend[noOfImagePerRow * noOfImagePerCol];
918     createWindowPartition(src, largeWin, win, legend, noOfImagePerCol, noOfImagePerRow);

```

```

919     src.copyTo(win[0]);
920     putText(legend[0], "Original", Point(5, 11), 1, 1, Scalar(250, 250, 250), 1);
921     putText(legend[1], class_name, Point(5, 11), 1, 1, Scalar(250, 250, 250), 1);
922     seg.copyTo(win[1]);
923     imshow("Window", largeWin);
924     waitKey(0);
925     destroyAllWindows();
926 }
927
928 String predictAndShowMeaning(const string& modelPath, const string& modelType, const Mat inputImage) {
929     string folderPath = "Inputs/Traffic signs/"; // Folder containing the images
930     Ptr<ml::StatModel> model;
931     Mat image;
932     inputImage.copyTo(image);
933
934     // Load the appropriate model based on modelType
935     if (modelType == "SVM") {
936         if (SVMmodel == NULL) {
937             cout << "Loading SVM model ....." << endl;
938             SVMmodel = StatModel::load<SVM>(modelPath);
939             cout << "SVM model loaded!" << endl;
940         }
941         model = SVMmodel;
942     }
943
944     else if (modelType == "RTrees") {
945         model = StatModel::load<RTrees>(modelPath);
946     }
947     else {
948         cerr << "Error: Unsupported model type." << endl;
949         return "Error model type";
950     }
951
952     // Check if model is loaded correctly
953     if (model.empty()) {
954         cerr << "Error: Failed to load the model from: " << modelPath << endl;
955         return "Error loading model";
956     }
957
958     if (image.empty()) {
959         cerr << "Error: Failed to load image: " << endl;
960         return "Error loading image";
961     }
962
963     resize(image, image, Size(64, 64));
964     // Extract features based on the chosen model type
965     vector<float> features;
966
967     if (modelType == "SVM") {
968         //extractColorHistogram(image, features);
969         extractHOGFeatures(image, features);
970     }
971     else if (modelType == "RTrees") {
972         extractHOGFeatures(image, features);
973     }
974
975     // Convert features to Mat
976

```

```

977     Mat featureMat(1, (int)features.size(), CV_32F, features.data());
978
979     // Predict using the loaded model
980     Mat predictions;
981     model->predict(featureMat, predictions);
982
983     int classNumber = static_cast<int>(predictions.at<float>(0));
984
985     // Show the image and print the predicted class meaning
986     string className = getClassMeaning(classNumber);
987     cout << "Image Class: " << classNumber << " (" << className << ")" << endl;
988     return className;
989 }
990
991 int main() {
992     int choice = -1; // Initialize choice to an invalid value
993
994     while (choice != 0) {
995         cout << "Please choose number for the following task." << endl;
996         cout << "1. Segmentation of traffic sign (HOG + SVM) " << endl;
997         cout << "2. Segmentation of traffic sign (colorHOG + Random Forest) " << endl;
998         cout << "3. HOG Extraction and SVM classification model training" << endl;
999         cout << "4. Color HOG Extraction and Random Forest model training" << endl;
1000        cout << "0. Exit" << endl;
1001        cout << "Enter your choice: ";
1002
1003        // Check if input is an integer
1004        cin >> choice;
1005        if (cin.fail()) {
1006            cout << "Invalid input. Please input integer only." << endl << endl;
1007            cin.clear(); // Clear the error state
1008            cin.ignore(numeric_limits<streamsize>::max(), '\n'); // Ignore the rest of the invalid input
1009            choice = -1;
1010            continue; // Skip to the next iteration of the loop
1011        }
1012
1013        if (choice == 1) {
1014            /// traffic sign segmentation of (SVM)
1015            segmentation1();
1016        }
1017        else if (choice == 2) {
1018            /// traffic sign segmentation of (Random Forest)
1019            segmentation2();
1020        }
1021        else if (choice == 3) {
1022            // Histogram extraction and SVM training
1023
1024            HOG();
1025            SVMClassification();
1026            main();
1027        }
1028        else if (choice == 4) {
1029            // HOG extraction and Random Forest training
1030
1031            ColorHOG();
1032            RandomForest();

```

```

1035        main();
1036    }
1037    else if (choice == 0) {
1038        cout << "Exiting the program..." << endl;
1039    }
1040    else {
1041        cout << "Invalid input. Please choose a valid option." << endl << endl;
1042    }
1043 }
1044
1045 return 0;
1046 }

```