**R&D PROPOSAL TO DETECT TRAFFIC SIGNS**

By

ANG CHIN SIANG

CHUA JING XUAN

CORNELIUS WONG QING JUN

KOK TZE KANG

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: Ang Chin Siang

Many previous studies showed the outstanding effort paid to traffic sign detection and recognition using different methods such as YOLO, CNN, and so on. In this paper, we focus on using OpenCV as a platform to achieve computer vision for traffic sign detection and recognition. One of the literature studies shows the remarkable accuracy in Chinese character recognition on traffic signs, however, there are still gaps found in the experiment that caused the misbehavior and misjudgment of the targets. The reason is found to be less proper preprocessing of the images and a less robust model was used to recognize Chinese characters. To cope with the issues, we propose some improvements to the previous model used by adding a versatile preprocessing model before the start of the experiment and a Chinese character AI model to provide better assistance to the classifier in character recognition after the BoVM or HOG features extraction, of which is pre-trained to understand 5000+ Chinese characters. The proposed system will start with dataset segmentation for training and testing purposes, and tailored preprocessing is given to images with different levels of noise and unclarity via the versatile preprocessing model. Next, cascaded color segmentation, refined candidate regions, and BoVM or HOG features extraction are a series of actions that are performed to detect and segment the region of interest (traffic sign). The model will first be trained by training the dataset by SVM learning algorithm. Finally, the trained model is tested to distinguish the existence of traffic signs in an image for final Chinese character recognition purposes.

**TABLE OF CONTENTS**

|  |  |
| --- | --- |
| **ABSTRACT**  **LIST OF FIGURES**  **LIST OF TABLES** | **ii**  **v**  **vii** |

|  |  |
| --- | --- |
| **CHAPTER 1 Introduction** | **1** |
| 1.1 Motivation and Problem Statement  1.2 Project Scope  1.3 Project Objectives  1.4 Impact, Significance and Contribution  1.5 Background Information | 1  1  2  3  3 |
| **CHAPTER 2 LITERATURE REVIEW** | **5** |
| 2.1 Detection and Recognition of Horizontal and Vertical Chinese Text-based Characters on Traffic Sign  2.2 Simultaneous Traffic Sign Detection and Boundary Estimation Using Convolutional Neural Network  2.3 Traffic Sign Detection based on Convolutional Neural Networks  2.4 Traffic sign detection and recognition (TSDR)  2.5 Comparing 4 methods | 5  8  11  14  18 |
|  |  |
| **CHAPTER 3 System Methodology/Approach OR System Model** | **19** |
| 3.1 System Design Diagram/Equation  3.2 Color Segmentation of a Traffic Sign  3.3 Feature extraction process  3.3.1 Overall feature extraction flow   * + 1. Colour feature extraction process   3.3.3 Shape feature extraction process  3.4 Classification  3.4.1 Support Vector Machines (SVM)  3.4.2 K-Nearest Neighbour (KNN)  3.4.3 Random Forest | 19  20  23  23  24  26  26  26  27  28 |
|  |  |
| **CHAPTER 4 SYSTEM IMPLEMENTATION** | **30** |
| 4.1 Hardware Setup | 30 |
| 4.2 Software Setup | 30 |
| 4.3 Colour Segmentation Operation | 30 |
| 4.4 Demo Feature Extraction | 32 |
| 4.5 Demo Training classifier and Evaluation of classifier | 35 |
| 4.6 Implementation Issues and Challenges | 38 |
|  |  |
| **CHAPTER 5 SYSTEM EVALUATION AND DISCUSSION** | **41** |
| 5.1 Comparison of classifier results | 41 |
|  |  |
| **CHAPTER 6 CONCLUSION AND RECOMMENDATION** | **43** |
| 6.1 Conclusion | 43 |
| 6.2 Recommendation | 44 |
|  |  |
| **REFERENCES’**  **APPENDIX** | **45**  **48** |

**LIST OF FIGURES**

Figure 2.1.1.1 - Flowchart of the Chinese Text Traffic Sign in the Horizontal and Vertical Directions.

Figure 2.1.1.2 - Algorithm for Blue Object Extraction with a Specific Threshold

Figure 2.1.1.3 - Enhance the Smoothness Object Detected

Figure 2.1.1.4 - Algorithm for the Blue Mask Section Extraction with a Specific Threshold

Figure 2.1.1.5 - Algorithm to Filter Insignificant Captured Images

Figure 2.1.1.6 - Algorithm for Shrinking the Detected Object for Higher Efficiency in Result Computation

Figure 2.1.1.7 - Algorithm for Gaps in Chinese Character Calculation

Figure 2.1.1.8 - Result of Character Pairing

Figure 2.1.1.9 - Algorithm to Calculate the Deviation for Word Pairing

Figure 2.1.1.10 - Result of Deviation Calculation Algorithm

Figure 2.2.1 - Block diagram of system detection using CNN

Figure 2.2.2 Box and pose prediction using regression vectors

Figure 2.2.3 Boundary corner to image coordinate

Figure 2.3.1 HOG feature extraction process

Figure 2.3.2 Block diagram traffic sign detection and classification

Figure 2.3.3 Network structure or architecture of CNN

Figure 2.4.1 An example of a TSDR system includes tracking process based on Kalman filter

Figure 2.4.2 Procedure of the Traffic Sign Recognition System

Figure 3.1 Design Diagram of Traffic Sign Detection and Recognition Model

Figure 3.2.1 Processes of color segmentation of a traffic sign

Figure 3.2.2 Steps of Color Segmentation of a Traffic Sign

Figure 3.2.3 Flowchart showcase of color masking processes

Figure 3.3.1.1 feature extraction process flow

Figure 3.3.2.1 BGR Color Feature extraction workflow

Figure 3.3.3.1 shape feature extraction flow

Figure 3.4.1 A simple pictorial overview of the K-Nearest Neighbor(KNN) algorithm

Figure 4.3.1 Raw Image Preprocessing Process Function

Figure 4.3.2 Preprocessed Image

Figure 4.3.3 Four Color Masking Function

Figure 4.3.4 Red Color Masked Result

Figure 4.3.5 Longest Contour Finding and Drawing Function

Figure 4.3.6 Longest Contour of Color Masked Result

Figure 4.3.7 White-Filled Longest Contour Result

Figure 4.3.8 Color Segmentation of a Traffic Sign Using AND Operator

Figure 4.3.9 White-Filled Longest Contour Result

Figure 4.4.1 The number of values obtained in BRG feature extraction of an image

Figure 4.4.2 Mask chosen by dominant color

Figure 4.4.3 result of shape feature extraction (number of vertices and Hu moments)

Figure 4.4.4 Hu moments reference value of each type of shape in labelDetermine function

Figure 4.4.5 Logic assigning label and comparing reference Hu moment of shape

Figure 4.5.1 The accuracy of the 3 trained classifiers: SVM, k-NN, and Random Forest.

Figure 4.5.2 The confusion matrix of the 3 trained classifiers: SVM, k-NN, and Random Forest for (a) training data and (b) testing data

Figures 4.5.3 The precision, recall and F1 score for 10 (training data) classes of the 3 trained

classifiers: SVM, k-NN, and Random Forest.

Figure 4.5.4 The precision, recall and F1 score for 10 testing data classes of the 3 trained classifiers: SVM, k-NN, and Random Forest.

Figure 4.6.1 Similar background color with traffic sign

Figure 4.6.2 Background color as the dominant color and captured

Figure 4.6.3 Longest contour problem in yellow

Figure 4.6.4 Longest contour detect fail in yellow

Figure 5.1 The result of precision, recall and F1 Score

**LIST OF TABLES**

Table Name

Table 3.3.1.2 Label list

Table 3.4.1 Random Forest Parameters Setting

Table 4.1 Hardware specification

Table 4.3.1 Color Segmentation Operation

Table 5.1 Classifiers performance comparison

**Chapter 1**

**Introduction**

* 1. **Problem Statement and Motivation**

Done by: Ang Chin Siang

As of December 2023, [16] Malaysia had a total vehicle registration number of 33.3 million, among them nearly 24 million vehicles are active and still operating on the road. The Transport Minister, Mr. Anthony Loke said that there are a total of 17 million cars out of all registered vehicles, which can be said, accounted for more than half of Malaysian transportation. Besides, from the data from the Malaysia Ministry of Transport’s official website [17], we can see the yearly increasing traffic accident numbers in the past decade, from *414,421* cases in 2010 to *567,516* cases in 2019 due to a variety of issues, and one of all is believed as the negligence of driver to the traffic sign precautions, which is concerning and a crucial matter that deserves Malaysian to take a look into. With the advancement of technology, autonomous driving technology has debuted to the public for a while, embedding with a smart algorithm enables it to do a certain level of real-time reactions to various road conditions, however, we find the gap in traffic sign detection accuracy that is existing in some of the autonomous driving technology nowadays. Hence, we propose a set of research on traffic sign detection to aid in stronger technology contribution that is beneficial to the latter.

Our motivation in the research is to aim for a long-term reduction in traffic accidents in the future, with the aid of robust traffic sign detection that embedded in, as a part of high accuracy and in-time responses to road condition analysis, to an excellent reaction to ever-changing road conditions by autonomous driving technology. The proposed traffic sign detection system is expected to recognize traffic signs in various colors, shapes, and directions at a specific distance and image quality.

**1.2 Project scope**

**Done by Cornelius Wong Qin Jun**

This project focuses on accurate traffic sign detection using the colors red, blue, orange and yellow, along with their corresponding shapes in different image conditions. The system uses color and shape segmentation to the model to effectively distinguish and recognize traffic signs from the background and noise. Feature extraction for the color and shape will be extracted such as BGR colour histogram, number of vertices and Hu moment value. The features obtained are utised for the machine learning by using Support Vector Machine(SVM), K- Nearest neighbor (K-NN), and random forest for do the classification to predict the classes of traffic sign.

**1.3 Project objective:**

**Done by Cornelius Wong Qin Jun**

- **to segment the traffic signs with different shapes and color**

The traffic sign in blue, red, orange, and yellow with the shape of triangle, square, octagon and circle can segment consistently in different image quality such as light intensity and with background noise.

- **to extraction features like BGR , number of vertices and Hu moment**

The color feature and shape feature can extracted correctly from the image with correct label for the purpose of the machine learning. The number of vertices and hu moments value obtained are correct despite translation, scale, and rotation.

- **to predict the traffic sign classes**

The system can detect the traffic signs by the trained model with correct labels such as red cicle, blue square and more.

**1.4 Impact and Significance**

Done by: Kok Tze Kang

Our project focused on creating an accurate, innovative and efficient traffic sign recognition system (TSR) in the small-time gap of academic research and limited computing resource to bring light on the impact of a good traffic sign detection system on public safety and transportation efficiency with improved modern machine learning techniques.

With this project, our TSR demonstrated novel color segmentation and training data to enhance the reliability of autonomous vehicles. This technology can result in better autonomous car detection systems, safer roads, and improved AI traffic management.

Our project demonstrated the significant result using materials from OpenCV library to produce an accurate and reliable traffics sign detection system and further enriching the traffic sign detection and recognition methods by exploring object detections under different challenging conditions caused by shapes and colors.

Next, by training strong machine learning algorithms: SVM, KNN and Random Forest, our traffic sign detection system is trained to recognize and categorize signs with accuracy in a variety of scenarios. Tested with various datasets, our TSR further minimizes the possibility of misinterpreting or overlooking important traffic indicators with considerations of the safety of the drivers. Our project shows the possibility of reducing unexpected encounters in which the traffic sign detection and recognition system can consistently and accurately detect traffic signs.

**1.5 Background Information**

**Done by: Chua Jing Xuan**

In all countries of the world, the important information about the road limitation and condition is presented to drivers as visual signals, such as traffic signs and traffic lanes. Traffic signs are an important part of road infrastructure to provide information about the current state of the road, restrictions, prohibitions, warnings, and other helpful information for navigation. Traffic signs come in a variety of shapes and colours. Disregarding or failing to notice these traffic signs may directly or indirectly contribute to a traffic accident. However, in adverse traffic conditions, the driver may accidentally or deliberately not notice traffic signs.

In these circumstances, if there is an automatic detection and recognition system for traffic signs, it can compensate for a driver’s possible inattention, decreasing a driver’s tiredness by helping him follow the traffic sign, and thus, making driving safer and easier. Traffic sign detection and recognition (TSDR) is an important application in the more recent technology

The TSDR system has received an increasing interest in recent years due to its potential use in various applications. However, a few challenges remain for a successful TSDR systems as the performance of these systems is greatly affected by the surrounding conditions that affect road signs visibility. Circumstances that affect road signs visibility are either temporal because of illumination factors and bad weather conditions or permanent because of vandalism of signs.

Hence, we aim to develop a reliable traffic sign detection system for autonomous vehicles using image processing techniques in C++ and OpenCV. The project includes segmenting each sign from the image, correctly identifying them. Predictions of the tested image labels are made using trained models. And then, the performances of models are evaluated using evaluation metrics such as accuracy, precision, recall and F1-score.

For this project, there are 7 chapters, which are included as follows: The introduction of the project is in Chapter 1. In chapter 2, there are literature reviews on traffic signs done by each team member. Chapter 3 is about System Methodology or System Model. Then, there is followed by the system design in Chapter 4. The system implementation done by each member in Chapter 5. Chapter 6 is the system evaluation and discussion. And the last chapter is about the conclusion and recommendation of the project.

**Chapter 2**

**Literature Review**

**CHAPTER 2 – LITERATURE REVIEW**

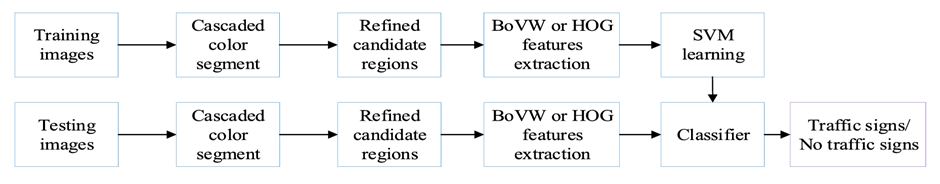
**Done by: Ang Chin Siang**

**Literature Review**

The literature review studies computer technology and hardware applications in traffic sign detection and recognition-related contributions. To achieve high accuracy, immediacy, and practicality in traffic sign detection and recognition, many precious studies have proposed different ideas and improvements. Some of them are built on top of the previously existing works. Usual methods to achieve traffic sign real-time detection and recognition are shape-based, color-based, and sorts of neural networks-related methods. Even deep learning comes into play to achieve more convenient and higher accuracy detection and recognition. Besides, there are lots of studies that used the Germany Traffic Sign Detection Benchmark (GTSDB) database, a well-known database with more than 50000+ colorful traffic signs, as the input to train and test the developed system.

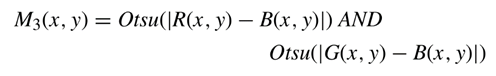
**2.1. Detection and Recognition of Horizontal and Vertical Chinese Text-based Characters on Traffic Sign**

In different countries, traffic sign always comes differently with varied shapes, colors, and contents on it. This can be found in China too. [1] Based on *J. Guo* *et al.*, there are traffic signs in China that are in blue and purely with white Chinese text, in both vertical and horizontal layouts. These two text layouts are different from other countries that horizontalize the context of the traffic signs, and thus cause higher challenges to detect the content on the traffic signs accurately. Besides, the trait of Chinese characters that have a loose connection between stroke and the main character's body, can be another challenge for a practical model. The following paragraphs review the outstanding work of this paper.



***Figure 2.1.1.1*** *Flowchart of the Chinese text traffic sign in the horizontal and vertical directions.*

To effectively differentiate the targets (traffic signs) from noises, *Cascaded Color Segmentation* has been proposed. This is a method that uses color to lock on the targets on the training images. The following figures show four phrases of refinements over the input images. *Figure 2.1.1.2* shows the method helped sort out objects with only a specific blue color (sky blue is removed), which is the traffic signs, with the threshold T = 90. *Figure 2.1.1.3* shows the method for further improvement to the sorting result gained. It works by filtering the dark-colored and extremely small objects of the previous output.



***Figure 2.1.1.2*** *Algorithm for blue object extraction with a specific threshold.*



***Figure 2.1.1.3.*** *Enhance the smoothness object detected.*



***Figure 2.1.1.4*** *Algorithm for the blue mask section extraction with a specific threshold.*

After locking on the targets, an algorithm, *HOG features*, was used to calculate the size of targets captured, known as region, to cull out multiple targets that are too small to provide information. The threshold of the region was set as Tarea = 225. The result of *HOG features* will be classified by the next algorithm, linear *SVM,* to label the captured image in “0” or “1”, to identify the existence of targets in the image. “0” implied the existence of target(s), but “1” does not.



***Figure 2.1.1.5.*** *Algorithm to filter Insignificant captured images*

With the existence of targets, the next step was recognizing the Chinese text on the targets. The smaller size of the image always aids in less computation for greater efficiency, *Figure 2.1.1.6* shows the method used in shrinking the image size. Due to the specialty of loose connection in Chinese characters, the “aspect ratio” was applied to calculate the closeness of each character, to better gather them for meaningful combination. *Figure 2.1.1.7* calculated the gap between characters, andcompared the ratio among characters, for pairing them up. *Figure 2.1.1.8.* shows theexample of character pairing.



***Figure 2.1.1.6.*** *Algorithm for shrinking the detected object**for higher efficiency in result computation.*

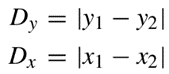


***Figure 2.1.1.7.*** *Algorithm for gap in Chinese character calculation.*

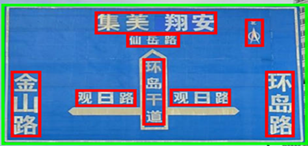


***Figure 2.1.1.8.*** *Result of character pairing.*

Besides recognizing a single character, forming characters into sentences was another challenge. Since there are vertical and horizontal layouts of sentences, an algorithm was required to recognize them. *Figure 2.1.1.9.* suggests the algorithm for the deviation between characters. The axis-x and axis-y are used to calculate the closeness of characters, to find the most reasonable pairs for a meaningful sentence. *Figure 2.1.1.10* shows a great result after applying the Deviation calculation algorithm.



***Figure 2.1.1.9.*** *Algorithm to calculate the deviation for word pairing.*



***Figure 2.1.1.10.*** *Result of deviation calculation algorithm*

Lastly, the experiment was conducted with 1,025 images as the training dataset, and 528 images as the testing set. With all proposed methods applied in the experiment, the result shows a high precision of **0.813**, **0.805**, and **0.523** in the corresponding short, medium, and high distances, with the launch of the *HOG features* algorithm, which the result is higher than the method, *Gonzálezs method*, proposed in earlier studies, with a precision of **0.818**, **0.692**, and **0.417**. As for the performance comparison among algorithms used in Chinese text detection, the proposed method also scored the highest precision of **0.942,** over Wang’s method with a precision of **0.898**, and Rong’s method with a precision of **0.755**.

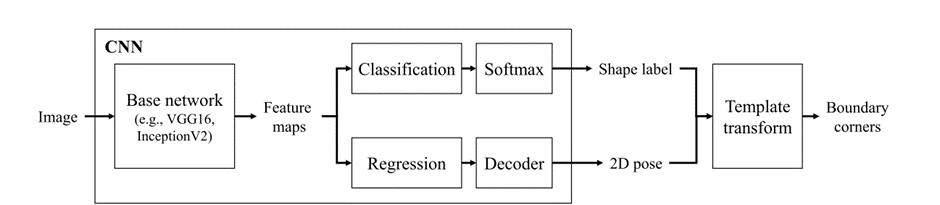
**2.2 Simultaneous Traffic Sign Detection and Boundary Estimation Using Convolutional Neural Network**

By: Cornelius Wong Qin Jun

This paper proposes an optimization for Convolutional Neural Network(CNN)-based traffic sign detection by using predicted 2D pose and shape class of traffic signs to estimate their location and precise boundary. The system projects the boundary of a template sign image onto the target sign in the input image to match its boundaries.

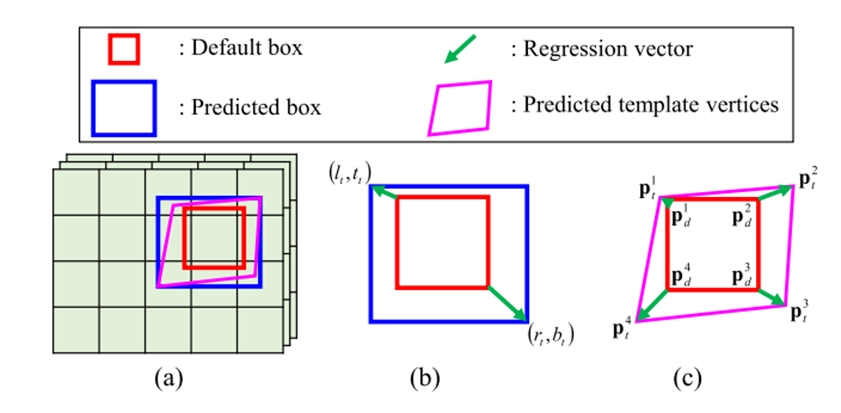
To fulfill the accuracy and speed requirements for real-time vehicle applications, VGG16 and InceptionV2 base networks, pre-trained on ImageNet, are chosen for their feature extraction capabilities. In [2], Single Shot MultiBox Detector (SSD) structure is used to add pose regression and shape classification layers, enabling reliable detection of traffic signs of different scales by performing detection on multiple feature maps.

The architecture is built on top of the SSD structure which allows to perform pose estimation and further converts into boundary estimation of traffic signs instead of bouncing box coordinates. Based on Figure 2.2.1, the input image is inputted into the base network. Series of convolution, non-linear activation, and pooling operations are undergone during feature map extraction. Then, the result will separated into 2 different convolutional layers which are the shape classification layer and pose regression layer for estimating the probabilities of the 2D pose and shape class. The output will be converted into shape label and 2D pose by Softmax and Decoder. At the end, the result which are 2D pose and shape class probability obtained can used to compute the boundary corners of the traffic sign [2].



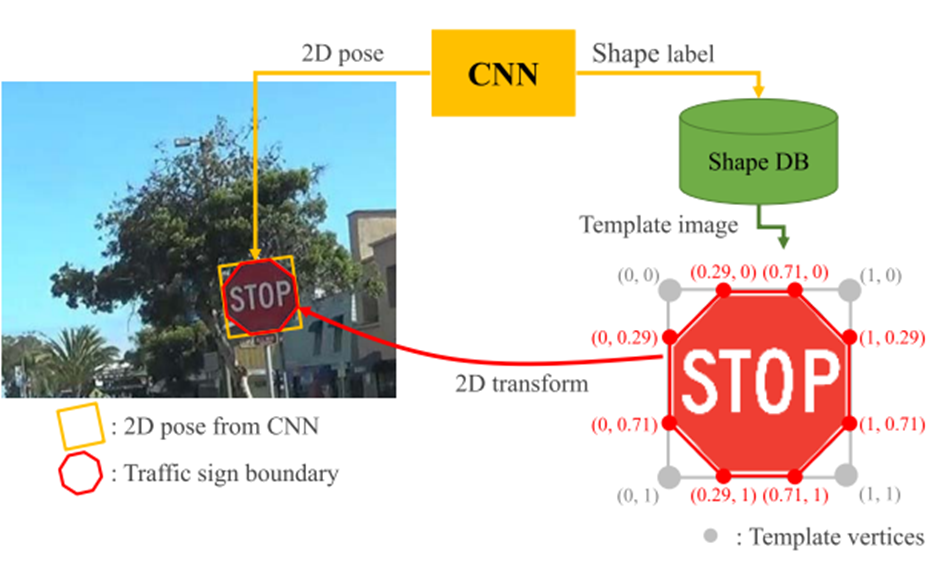
***Figure 2.2.1*** *Block diagram of system from [2]*

For 2D pose in traditional CNN object detectors, bounding box regression layers in Figure 2.2.2 (a) predict pose offsets relative to default box coordinates to locate objects. This is done by representing bounding boxes as 4-dimensional vectors, using the coordinates left(*l*), top(*t*), right(*r*), and bottom(*b*) that can be combined such as *(l, t, r, b*) = (*lt,tt,rt, bt*) − (*ld ,td ,rd , bd* ). In Figure 2.2.2 (b), the detection result can be obtained by adding two vectors (green arrows) at the (l,t) and *(r,b).* In the proposed method in [2], box regression is generalized to vertex regression to predict 2D poses using an 8-dimensional vector for four points on the target. These points called template vertices which are minimum bounding quadrilateral of the signs and are used as regression points to define boundaries (refer to figure 2.2.3) and their positions are calculated as the difference between the target and default box vertices. During the inference stage, the template vertices are estimated by adding the regression vector to the default box vertices, as shown in Figure 2.2.2 (c) [2].



***Figure 2.2.2*** *Box and pose prediction using regression vectors from [2]*

After obtaining the 2D pose of a traffic sign, the shape classification layer predicts the shape label to select the appropriate template image (e.g., octagon, diamond, rectangle). The coordinates of the template image are normalized. By using the predicted template vertices, a 2D transform matrix is computed. This matrix is then used to transform the boundary corners of the template image to the image coordinates, resulting in the precise boundary of the detected traffic sign. This can be shown in Figure 2.2.3. The grey dots are considered template vertices and the red dots are considered as boundary corners of traffic sign shape. In [2] states the circle shape can also be computed although circle does not have corners by observing the ellipse of major and minor axes but they do not focus on it as circle shape is not a common traffic sign in US. Hence, the boundary of shape had obtained.



***Figure 2.2.3*** *Boundary corner to image coordinate from [2]*

The advantage of the system in [2] is can effectively handle the blur and image that has been blocked by another object by using the predicted 2D poses and shape labels and computing the boundary corners of a traffic sign. Moreover, the system can suit low-power mobile platforms and still can run with a frame rate of 7 FPS by replacing the CNN-based network with InceptionV2. However, the system may have false detection when facing a similar appearance object with the traffic sign such as between a rectangle electronic board and a rectangle sign. To solve this problem will bring out another drawback of the system which requires huge training sets.

**Done by: Kok Tze Kang**

**2.3 Traffic Sign Detection based on Convolutional Neural Networks**

For colour-based detection method, a paper by [4] used RGB colour space thresholding for image segmentation from noise but faced sensitivity issues due to varying lighting conditions. Another technique by [5] employed YCbCr colour space for better handling of different lighting conditions using Y or luma for brightness, Cb blue minus luma and Cr for red minus luma. Input image was first converted from RGB colour space to YCbCR colour space and red was able to be detected. [6] and [7] utilized HSI colour space for comprehensive sign detection. It is a common technique using hue (H), saturation (S), and intensity(I) which was converted from RGB color space for better representations. However, these methods are sensitivity to lighting conditions like natural illumination and reflection and limitations in handling various sign colors and conditions. Therefore, the method used by the author converts RGV color space to (HSV) color space introduced by [8, 9]. HSV uses hue (H) for pixel color information, saturation (S) for purity of color, and value (V) for color brightness. Next, threshold technique is used to detect the red and blue color from input image in traffic signs by determining optimum threshold values using histogram thresholding, local thresholding and global thresholding. The method uses image histogram analysis on the three values of HSV for images of blue and red segmented signs. However, due to varying light conditions, YCbCr could be better considered.

For image filtering of false candidates by the detection, erosion for removal of pixels from the boundaries of an object and shrinking the size of the image, and dilation for adding pixels to the boundaries of objects in an image were used. Detected regions of interest (ROI) with unsatisfied bounding box size and ratio were manually removed. However, this requires extra data preprocessing, gaussian filtering for HSV color space could be considered to remove noise.

While for Shape-based feature extraction, Histogram of Oriented Gradients (HOG) technique was used for feature extraction of detected region and to analyze the gradient's orientation and distribution magnitude. The road sign blob is separated into smaller areas of matrixes of pixel to compute the HOG features [10]. To address the illumination variance issue, the pixels were then gathered into blocks and block normalization was applied to the final histograms. The process is shown in figure 3.1.

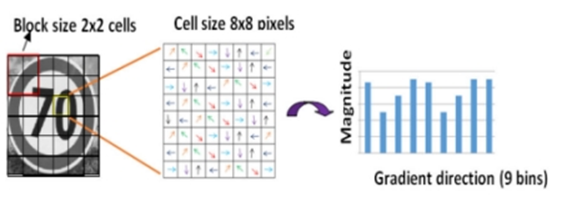


Figure 2.3.1 HOG feature extraction process by [3]

Machine Learning and Deep Learning Methods. Machine learning and Deep Learning require the two phase of training and testing of data using model training techniques which after evaluations and fine tuning, only the model achieves most suitable results from object detection. A popular technique as shown, linear support vector machine (SVM) method by[11] was used for classification robustness against intensity changes and rotation. SVM method uses supervised machine learning algorithm to classify data. For categorization of traffic sign, [12] used SVM to classify image using (HOG) features extraction and Radial Basis Function (RBF) kernel for capturing nonlinear relationships between data. In this paper, the author used SVM technique for image classification of shapes geometric shape recognition of the blobs that were produced from the segmentation stage. The advantage of using SVM technique identified by author is its resilience rotation, translations, and size. SVM can identify best hyperplane for best feasibility separation between classes or features. For non-linear separate planes, kernels were used to change the data to fit linearly.

With recognition of a notable achievement by [13] who achieved high accuracy and recognition rate compared to other with similar deep neural network using multi-column deep neural network based on CNN. In this paper, the author used multi-layer neural network (CNN) which specializes in visual patterns recognition by combining convolution, pooling and activation layers. CNN can extract different features from the same input image based on the size and number of layers of the kernel used. Three convolution stages, two fully connected layers, and a softmax layer make up the suggested CNN architecture. To add non-linearity to the network, full connection layers was used with rectifier linear unit (ReLU) as its activation function for convolutional layers. In addition, local response normalization (LRN) is used to normalize feature maps. The number of kernels, or "depth," and the stride, which stands for the number of kernels shifting steps, determine the size of the feature maps. Furthermore, for the case of overfitting, dropout technique by [14] was used as regularization technique for instilling “penalty” in machine learning. The final layer of the network applies a Softmax function to classify the 43 road sign types once it has gone through the fully linked layer.

The block diagram of the system is shown in figure 3.2 which include segmentation using HSV, filtering using erosion and dilation, and shape-based feature extraction using HOG for both testing and training datasets. After then the region of interest (ROI) will be used in classification step using linear SVM classifier between each features and CNN for pattern recognition stage in figure 3.3.

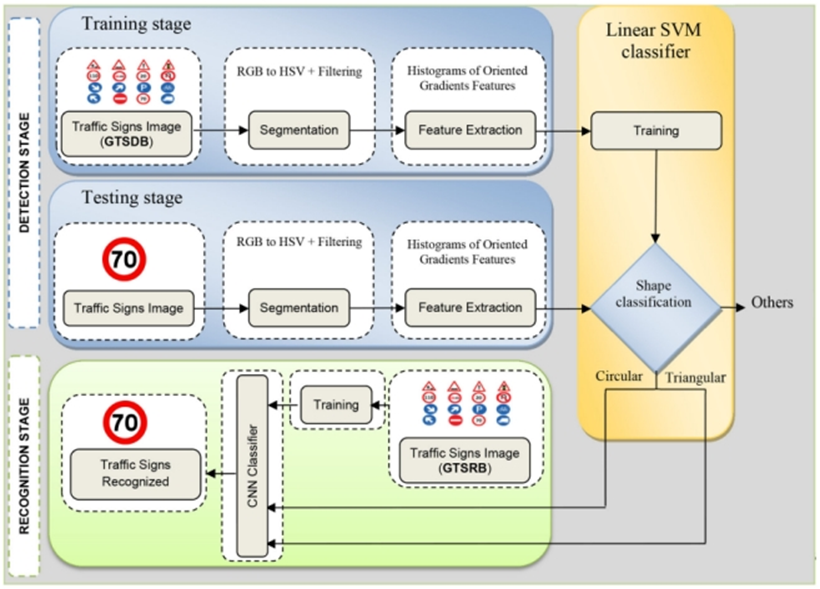


Figure 2.3.2 Block diagram traffic sign detection and classification by [3]

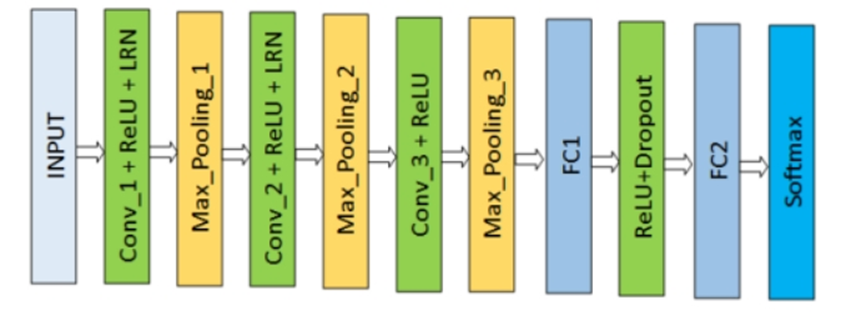


Figure 2.3.3 network structure or architecture of CNN by [3]

**2.4 Traffic sign detection and recognition (TSDR)**

**Done by: Chua Jing Xuan**

The paper discusses several key techniques in traffic sign detection and recognition (TSDR):

1. Detection: The initial stage in any TSDR system is locating potential sign image regions from a natural scene image input. Traffic signs usually have a strict colour scheme (red, blue, and white) and specific shapes (round, square, and triangular). These inherent characteristics distinguish them from other outdoor objects making them suitable to be processed by a computer vision system automatically, thus, allow the TSDR system to distinguish traffic signs from the background scene. Therefore, traffic sign detection methods have been traditionally classified:

* 1. Colour-Based Methods: The most common colour-based detection methods are Colour Thresholding, Region Growing, Colour Indexing and so on.

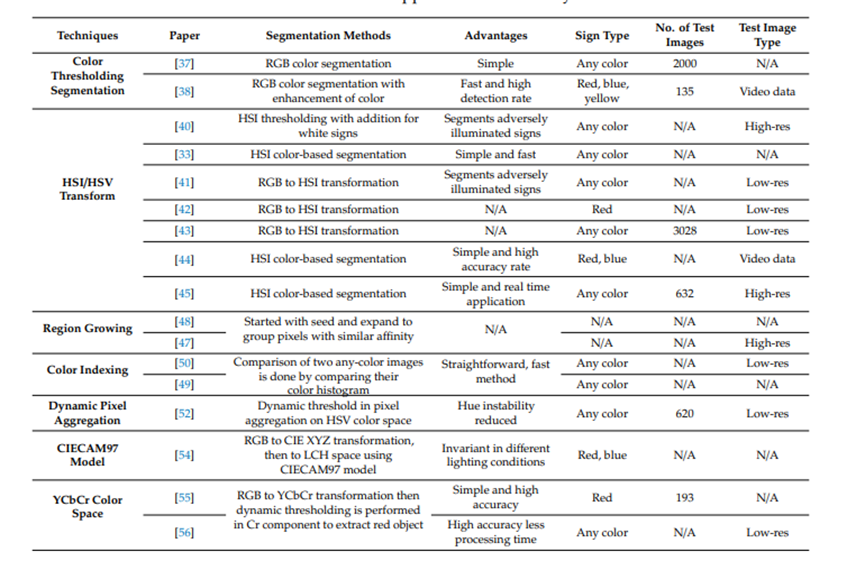


Table 2.4.1: Colour based approaches for TSDR system

* 1. Shape-Based Methods: Many shape-based methods are popular in TSDR system, such as Hough Transformation, Similarity Detection, Distance Transform Matching etc.

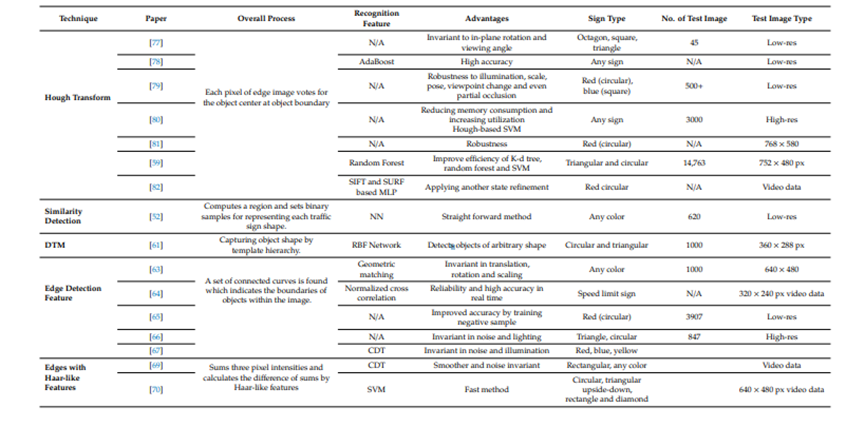


Table 2.4.2: Shape based methods for TSDR system

* 1. Hybrid Method：In the hybrid methods, either colour-based approaches take shape into account after having looked at colours, or shape detection is used as the main method but integrate some colour aspects as well.

1. Tracking: Methods for tracking detected signs across frames in a video sequence to improve detection reliability and robustness. Techniques include:
   1. Kalman Filter: Predicts the position of the traffic sign in the next frame.

Advantage:

* For avoiding incorrect assignment, rule-based approach utilizing combined distance direction difference is used.
* Takes less time in tracking and verifying.
* Used stereo parameters to reduce the error of stereo measurement.

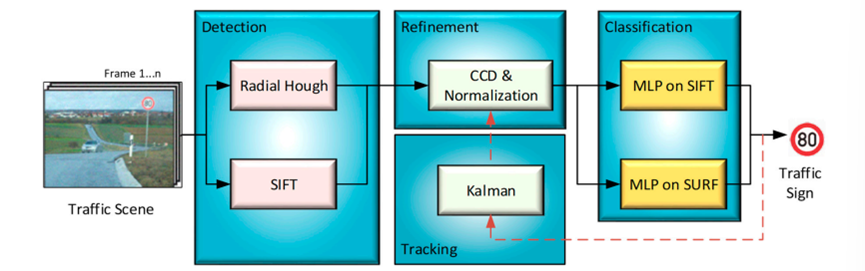


Figure 1: An example of a TSDR system includes tracking process based on Kalman filter.

* 1. Particle Filter: Uses a set of weighted particles to represent the distribution of the target state.

Advantage：

* Fast and advanced method, high detection and tracking rate.

1. Classification: After localizing regions of interest (ROIs), classification techniques are used to determine the content of detected traffic signs. This involves reading the inner part of the sign using classifier methods. Several methods are outlined, including:
   1. Template Matching: A common method in image processing and pattern recognition, uses pre-defined templates to search the whole image pixel by pixel, Fast and straightforward.
   2. Decision Tree Method: Examples show high accuracy (90-96%), Used with HOG and SVM features.
   3. Support Vector Machine (SVM): Separates data into two categories using a hyperplane, Highly accurate and fast, Effective for detecting speed limit signs.

The Block diagram of the traffic sign recognition system:

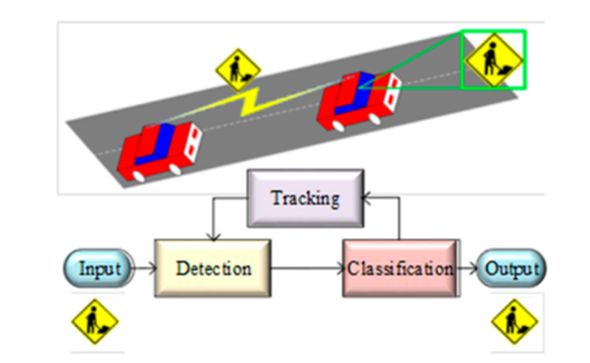


Figure 2: Procedure of the Traffic Sign Recognition System

This image represents the general procedure of a Traffic Sign Detection and Recognition (TSDR) system.

Input: A picture or a video frame taken by a camera installed on a car usually serves as the system's initial input.

Detection: During this phase, the system analyses the input to find possible locations for traffic signs. This entails locating areas of interest in the image that might contain traffic signs.

Tracking: After identifying a traffic sign, the system may employ tracking methods to trace the sign's motion across several frames, guaranteeing precise monitoring of the sign even when the vehicle or the sign moves.

Classification: The system classifies a possible traffic sign to identify its exact kind for example: stop sign, speed restriction, pedestrian crossing after detecting and tracking it.

Output: Lastly, the system generates the traffic sign that has been identified. This may be utilised for a few purposes, including driver assistance programs and autonomous vehicles.

**2.5 Comparing 4 methods:**

**Done by: Cornelius Wong Qin Jun**

For the first method in 2.1, the strength is effective differentiation and recognition of traffic signs with different colors and sizes. However, it is very complex to due the loose stroke connections of Chinese characters. For the second method in 2.2, its strength is able to handle the blurred or blocked image with a lower power device. However, it easily gets false detection with an object similar to a traffic sign and needs a huge dataset. For the third method in 2.3, its strength is using different of color spaces (YCbCr, HSI, HSV) and histogram analysis to improve traffic sign detection under different lighting conditions. However, there is a potential overfit in the deep learning model and causing tuning and training become complicated. For the fourth method, its strength is Kalman Tracking improves the robustness of traffic sign detection across video frames. However, the Kalman Tracking may increase the complexness of computational and increase the time for processing.

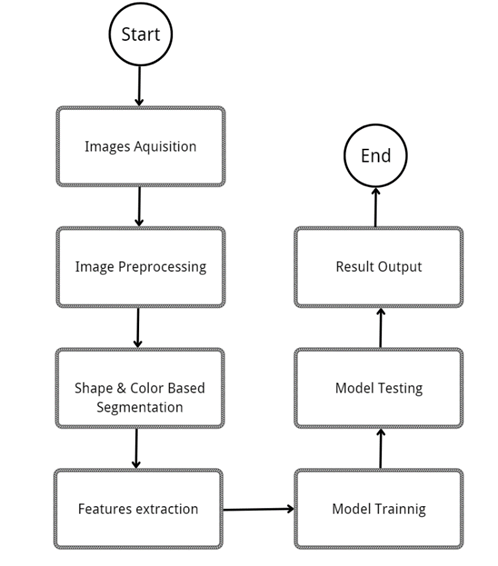
In the end, we chose the method in 2.1 as it can recognize colour well and does not complex the computation like CNN which fits our goal to do red, yellow, and blue and shape traffic sign for real-time traffic sign detection for autonomous car.

**Chapter 3**

**System Methodology/Approach OR System Model**

**3.1 System Design Diagram/Equation**

Done by: Ang Chin Siang

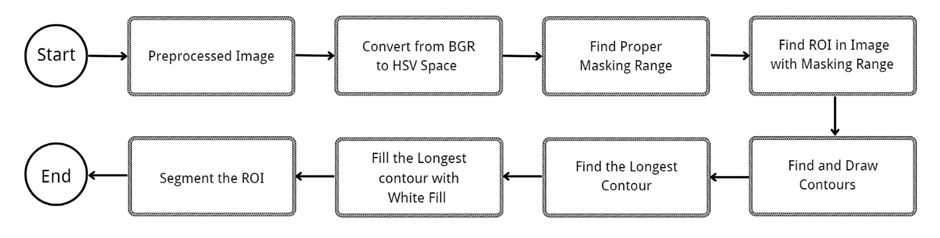


*Figure 3.1 Design Diagram of Traffic Sign Detection and Recognition Model*

Figure 3.1 shows the design diagram of the traffic sign detection and recognition model, starting with image acquisition to final result output. There are in total seven steps in a round for the model to output the result, segmented images, and accuracy scores of the model. The model starts with image acquisition from a traffic sign image file that is imported into the model. Next, to ensure a nice region of interest (traffic sign) to be extracted from the raw image, image preprocessing is done on the raw image, such as brightness adjustment, contrast adjustment, image size adjustment, and color channel adjustment. After the image preprocessing process, the image is ready to go for traffic sign detection and segmentation based on both colors and shapes found on the image. The general colors of a traffic sign are yellow, blue, and red; hence, color segmentation is targeting the vivid colors of these three. Shape-based segmentation is also helpful in complementing color-based segmentation by capturing edges and vertices found on a shape of traffic sign. One is able to see a prototype of a segmented traffic sign from a raw image after the segmentation process. The next step is to extract the traits of a traffic sign for a model training purpose used for traffic sign recognition. Features such as shape and color are the conspicuous features of a traffic sign and are helpful in recognition by a model. With all relevant features of a traffic sign collected, the next step is to train a model for traffic sign recognition. There are several genres of machine models, such as KNN, SVM, Random Forest, and so on, for traffic sign classification. To access the excellence of a model’s performance, testing sessions are carried out soon after the model training session. Datasets are usually separated into two sets for training purposes and testing purposes, respectively. Ultimately, the final process is accessing the performance of a trained model by analyzing the accuracy score that the model produces. The following sections will dive deeper into each step in the system design diagram.

**3.2 Color Segmentation of a Traffic Sign**

Done by: Ang Chin Siang

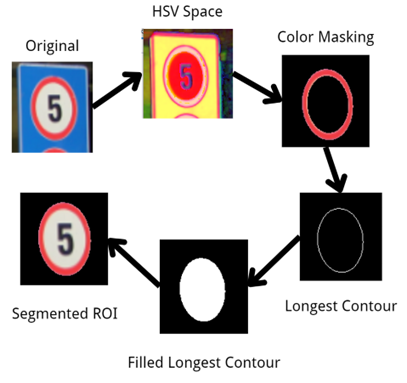
*Figure 3.2.1 Processes of color segmentation of a traffic sign*

Color segmentation is one of the many object detection and segmentation methods. Color segmentation provides a useful way for traffic sign segmentation; this is all thanks to the vivid color fill in a traffic sign, which makes it stand out of other items around it in terms of color context. Figure 3.2.1 shows processes involved in the color segmentation of a traffic sign.

Study shows HSV color space is more robust in color segmentation due to its traits consisting of hue, saturation, and value, separating brightness and making object segmentation a lot easier without the effects brought by a lighting condition change. Hence, a preprocessed image is first converted to HSV color space for masking range analysis. Finding a proper range of color space can vary the masking result and lead to better segmentation of a traffic sign. Here we find a proper color range for red, yellow, orange, and blue for the corresponding traffic sign color segmentation.

After a successful masking process, contour drawing is then performed around all segmented regions, including the region of interest (ROI), and noises. However, before the step is realized, points (coordinates) are required for contour drawing. Points that are found on an image are collected and are connected to draw a rough shape as the results found after the masking process. There is usually one ROI and numerous noises captured, and therefore a sieve is required to find the ROI. The ROI is mostly the longest contour found on the contour drawing; hence, in the next step, we use some algorithm to find out the largest region of the drawn contour; this is efficient in overcoming all the noises.

Once the longest contour is found and drawn, the next step is filling the inside of the bounded shape line with white fills. The reason for the white color fill is to help in later color image segmentation of ROI since white color has the highest value (255), which accommodates all colors in the color space. Finally, with all the criteria prepared, the ROI can be extracted from the raw image by using the white-color traffic sign prototype we obtained in the previous step. This is done by combining the color image with white fills, and the results will be the color image since the color image has a lower value.



*Figure 3.2.2 Steps of Color Segmentation of a Traffic Sign*

**3.2.2 Multicolor Masking Process on an Image**

Done by: Ang Chin Siang



*Figure 3.2.3 Flowchart showcase of color masking processes*

Figure 3.2.3 shows the detailed flows of the color segmentation processes. The process starts with the raw image as the process target; yellow\_mask, blue\_mask, red\_mask, and orange\_mask are the masking means used to find the ROI. The model will perform 4 color masking before proceeding to the next phrase. After that, each color masking result is used in performing contour finding and drawing. The longest contour is the shortlisted from all captured contours. White filling is done to fill up the inside of the longest contour region. All filled-longest regions will undergo checking to ensure the number of pixels is greater than 1500 (traffic signs are usually bigger than this size) or otherwise discarded. All the shortlisted, white-filled longest contours will be filtered to find the biggest pixel value to be the final prototype. The final result will be used for color segmentation in conjunction with the original image to get the final colorful segmented result.

**3.3 Feature extraction process**

Done by: Cornelius Wong Qin Jun

3.3.1 Overall feature extraction flow

The extracted features are both color-based and shape-based. The BGR features capture the color information of the traffic sign image by analyzing the Blue (B), Green (G), and Red (R) channels. This helps in identifying the dominant color within the image, which is essential for recognizing the type of traffic sign based on its color. The shape-based features are the number of vertices and the hu moment value. The number of vertices identifies the shape type such as 3 for triangle, 4 for square and more than 4 can be a circle due to the curve. The hu moment value can identify the shape information as it is invariant to position, orientation, or size changes. This is effective to consistently recognition across different shape conditions.

The feature extraction process flow is illustrated in Figure 3.3.1.1. First, the traffic sign image is loaded and segmented based on its color. The segmented mask, corresponding to its color, and the image are then used to compute the BGR features. This is done by dividing each BGR channel into 50 bins and calculating a histogram for each channel within the given mask. After calculating the histograms, the values are normalized, and a vector of BGR features is returned. Next, the shape features are computed. These include the number of vertices and the 7 Hu moment values. The number of vertices and Hu moments are extracted by finding the longest contour within the segmented mask. Once the number of vertices, Hu moment values, and color are obtained, the traffic sign is labeled by comparing them with reference Hu moment values for different shapes and colors. The labels are 0 for red triangle, 1 for red square, 2 for red octagon, 3 for red circle, 4 for yellow triangle, 5 for yellow square, 6 for yellow circle, 7 for blue triangle, 8 for blue square, 9 for blue circle and 10 for others. 10 is represented as fail not match these shapes and colours The label list in shown in table 3.3.1.2. Both the number of vertices and Hu moment values are stored in a vector, which is later combined with the BGR features. This combined feature vector, along with the label, is stored as part of the training dataset. The process repeats for all images in the dataset until feature extraction is completed.

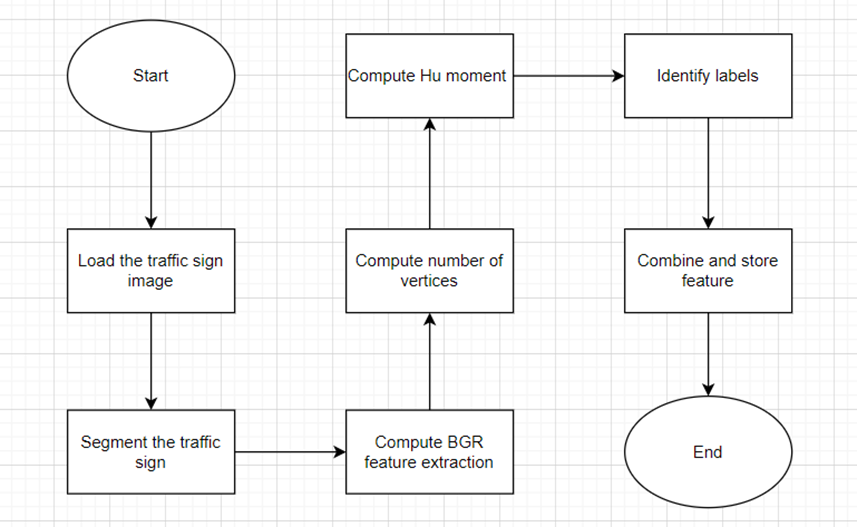


Figure 3.3.1.1 feature extraction process flow

|  |  |
| --- | --- |
| Labal | type |
| 0 | red triangle |
| 1 | red square |
| 2 | red octagon |
| 3 | red circle |
| 4 | yellow triangle |
| 5 | yellow square |
| 6 | yellow circle |
| 7 | blue triangle |
| 8 | blue square |
| 9 | blue circle |
| 10 | others |

Table 3.3.1.2 Label list

3.3.2 Colour feature extraction process

Done by: Kok Tze Kang

After segmentation is completed, the current computed mask and original image will then be passed as arguments to the function extractBGRFeatures(mask, image) and the color feature extraction workflow is shown below:

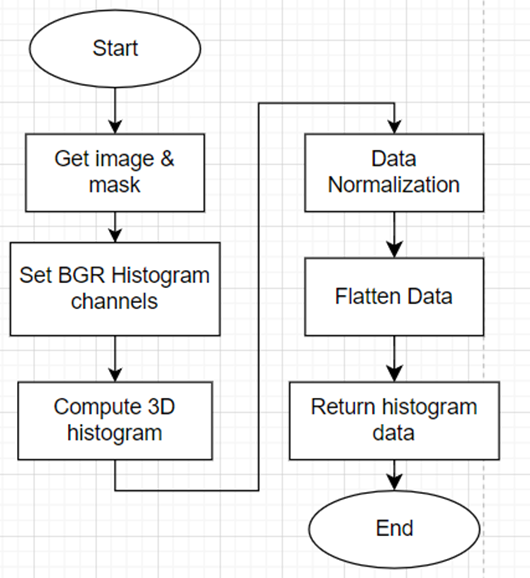


Figure 3.3.2.1. BGR color Feature extraction workflow

The input image, typically in BGR format. It contains the color information from which the features will be extracted. The binary mask passed has non-zero pixels to define the region of interest (ROI) in the image where only pixels within the masked region are processed when calculating the color histogram. Next, the 3 Blue, Green, Red (BGR) histogram channels will then be set up for 50 bins per channel for formatting how color information is binned with range value for each of the color channels (0 to 256) for 8-bit images and BGR channels (0 for Blue, 1 for Green, 2 for Red) for histogram calculation. Thirdly, for computation of 3D histogram, calcHist() function will be called and image and mask passed will be used to compute a 3D histogram for the ROI based on the BGR color channels created and stored in hist matrix which is in (50x50x50) 3D histogram matrix. Then, the computed hist is then normalized to scale between 0 and 1 for comparison across different images and masks without being affected by image size or light condition. The normalized 3D hist will then be flattened to 1D feature vector into a list of float values and finally be returned to main interface for machine learning training.

3.3.3 Shape feature extraction flow

Done by: Cornelius Wong Qin Jun

Figure 3.3.3.1 illustrate the shape feature extraction flow. The mask of the dominant color that detected by the system will be input to the extractVertices and extractHuMoments function. Next, the contour of the external outline will be found out and identifies longest contour. The longest contour index will be saved to retrieve the longest contour for applying computation. The longest contour will undergo applypolyDP to get the number of vertices and hu moment calculation at there corresponding function. The values will return and then save to a shape feature vector.

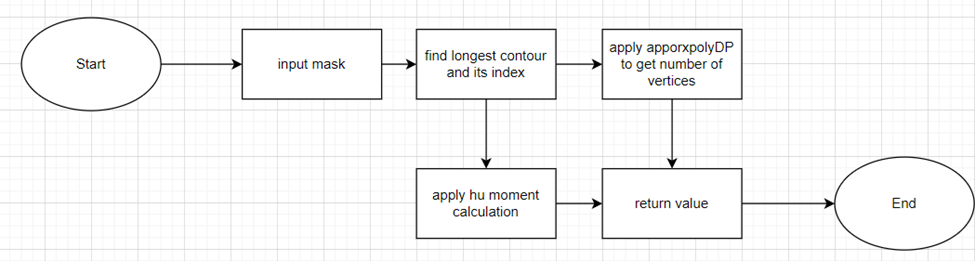


Figure 3.3.3.1 shape feature extraction flow

**3.4 Classification**

**3.4.1 Support vector machines**

Done by: Chua Jing Xuan

Support Vector Machines (SVM) are supervised learning models with associated learning algorithms that analyse data used for classification and regression analysis. Its core idea is to find an optimal hyperplane that best separates the data points from different classes. The SVM algorithm is often reported to achieve better results than other classifiers. Because of this, SVM is perfect for traffic sign recognition, as signs can be categorised into several groups according to unique attributes. Its capacity to handle high-dimensional data, like HOG feature extraction, is an important quality in TSR because of the effects of lighting and other environmental conditions. Additionally, SVM can be modified for multiclass classification, which will allow it to recognise a variety of traffic signs with accuracy. Its capacity to quickly identify the best possible hyperplane effectively distinguishes classes, even when there are minute visual variations in traffic signs. It gets chosen as one of the classifiers as a result.

The performance of the SVM classifier relies on the choice of the regularization parameter C and the kernel parameter. Together they are known as the hyperplane parameter. During the training phase, SVM builds a model, maps the decision boundary for each class, and specifies the hyperplane that separates the different classes. Increasing the distance between the classes by increasing the hyperplane margin helps increase the classification accuracy.

**3.4.2 K-Nearest Neighbor (KNN)**

Done by: Chua Jing Xuan

KNN is a straightforward, instance-based learning technique that uses the feature space's k-nearest neighbours' majority label to classify a given data point. KNN is a suitable option for real-time traffic sign recognition since it is simple to comprehend and apply. New instances of the data can be classed based on their similarity to previously labelled data. Although K-NN is also capable of handling multiclass problems, when multiple classes are involved, its inherent processes for optimising class separation may not be as strong, which could result in less accurate classifications.

A diagram of a class

Description automatically generated

Figure 3.4.1. A simple pictorial overview of the K-Nearest Neighbour (KNN) algorithm

Since K-NN bases its predictions on the closest neighbours, noisy neighbours can result in an inaccurate classification, K-NN is therefore more susceptible to noise, particularly if the noisy data points are close to the test data.

**3.4.3** **Random Forest**

Done by: Ang Chin Siang

Random Forest is one of the powerful classifiers that is made of decision trees. Users of random forests can decide on the number of decision trees in them to achieve the optimum performance. Like some other classifiers, the random forest consists of general parameters such as the number of estimators, depth of the tree, number of iterations, and so on. The idea of combining decision trees is to make it a higher accuracy and prevent the performance from overfitting situations (results being too perfect). The reasons for random forest being chosen here are several. First, the random forest classifier fits well to both regression and classification tasks effectively, which is suitable for traffic sign recognition in our paper. Besides, it is capable of making multiclass classification, which is the case in traffic sign recognition based on colors and shapes. Next, the robustness of comprehending feature importance in a random forest makes it further capable of analyzing some complex feature inputs such as color histogram, HOG features, number of vertices, and Hu Moment of a shape. Finally, random forest trees are competent in handling non-linear relationships, which can be our circumstances, consisting of 9 features in a traffic sign recognition process.

The following shows the idea of how a random forest works:

* **Bootstrap Sampling**: Creates multiple subsets of the original training data by randomly sampling with replacement.
* **Decision Trees**: Constructed by randomly selecting a subset of random feature sampling at each split, the tree is then diverse.
* **Voting**: Predictions that are done by each split decision tree. The majority rule is applied for the final prediction label.

**Parameter**

The parameters involved in random forest tree construction in our paper are Maximum depth, Maximum category, and Maximum iteration.

|  |  |
| --- | --- |
| Parameters | Descriptions |
| Maximum depth | 7 levels (decision trees can split furthest to 7 levels) |
| Maximum category | 15 Maximum category (Maximum 15 distinct class labels of categorical data feature) |
| Maximum iteration | 100 Maximum iteration (100 iteration of a random forest process) |

Table 3.4.1 Random Forest Parameters Setting

**CHAPTER 4** **SYSTEM IMPLEMENTATION**

**4.1 Hardware**

Laptop

|  |  |
| --- | --- |
| Brand | Huawei Matebook 14s |
| Operating System | Windows 10 Home – 64-bit |
| Processor | 11th Gen IntelI CoreI i5-11300H @ 3.10GHz 3.11 GHz |
| RAM | 8.00 GB LPDDR 4322MHz |
| GPU | Intel ® IRIS Xe ® Graphic |
| Hard Disk | 512GB SSD |

*Table 4.1 Hardware specification*

**4.2 Software**

1. Microsoft Visual Studio Community 2022

Version: 17.10.5

1. OpenCV

Version: 4.9.0

Release Date: 2023-12-28

**4.3 Color Segmentation Operation**

Done by: Ang Chin Siang

Color segmentation plays an important role in the later traffic sign recognition success rate. A good object segmentation captures the ROI (traffic sign) on an image by using respective color masking and cancels out all unnecessary noises to obtain an intact segmented traffic sign result. The following shows the steps of color segmentation, methods used, and segmentation results that are performed in this paper:

|  |  |  |
| --- | --- | --- |
| No | Descriptions | Result |
| 1 | *Figure 4.3.1 Raw Image Preprocessing Process Function*     * Raw images undergo preprocessing with a function shown in Figure 4.3.1. * Preprocessed images will be turned into HSV color space. | *Figure 4.3.2 Preprocessed Image* |
| 2 | *Figure 4.3.3 Four Color Masking Function*     * Four color masking processes (Red, Blue, Yellow, Orange) to Figure 4.3.2, by four color masking functions shown in Figure 4.3.3. * To obtain the ROI of traffic signs based on their conspicuous color. | *Figure 4.3.4 Red Color Masked Result* |
| 3 | *Figure 4.3.5 Longest Contour Finding and Drawing Function*     * The longest contour of Figure 4.3.4 is found and drawn by the *ContourDrawingAndFilling* function. * Longet contour brings the highest possibility of containing the target (traffic sign). * Figure 4.3.6 is then filled with white fills to obtain the prototype of the traffic sign for a later color segmentation process. | *Figure 4.3.6 Longest Contour of Color Masked Result*    *Figure 4.3.7 White-Filled Longest Contour Result* |
| 4 | *Figure 4.3.8 Color Segmentation of a Traffic Sign Using AND Operator*   * Figure 4.3.7 is used to segment the colorful traffic sign region by using *AND* operator shown in Figure 4.3.8, and store in “Mask” variable. * Figure 4.3.9 shows the final outcome of segmentation. | *Figure 4.3.9 White-Filled Longest Contour Result* |

Table 4.3.1 Color Segmentation Operation

**4.4 Demo feature extraction**

**Done by Cornelius Wong Qin Jun**

In the system, three functions are used for feature extraction: extractBGRFeatures for color features, and extractVertices and extractHuMoments for shape features. After the segmentation done, the mask obtained and image will used for BGR color histogram extraction. The extracted values are returned, stored in vector, and the total number of values is displayed to the user. Figure 4.4.1 illustrates this process. The figure shows a total of 2916 values in the BGR extraction for a red traffic sign image, with red being the dominant color based on the percentage of pixels. Once the dominant color is identified, the corresponding mask is selected and passed to the shape feature extraction functions. This logic is shown in Figure 4.4.2, where the mask for the dominant color is chosen for further processing.

Next, the shape feature extraction is performed to compute the number of vertices and the seven Hu moment values. The result is shown in Figure 4.4.3. The longest contour in the image is identified, and the number of vertices and Hu moments are computed using the approxPolyDP algorithm and Hu moment calculation, respectively. For this particular image, the number of vertices obtained is 7, and the 7 Hu moment values are displayed. These values are then used for labeling by comparing the color, number of vertices, and Hu moment values with reference shape Hu moments. The label assigned to this image is 3, representing a red circle. The reference Hu moment values are obtained from a perfectly segmented shape, as shown in Figure 4.4.4, and are used in the labelDetermine function to comparing and determine the label. Figure 4.4.5 shows the logic if else for assigning and comparing with the reference hu moment values of shape.

At the end of the feature extraction process, the number of vertices and the 7 Hu moment values are combined into a shape feature vector. This vector is then appended to the BGR color feature vector to form the complete feature set for the image. Both the combined feature set and the corresponding label are stored for use in the machine learning training phase. These feature vectors, along with their labels, will later be used to train the model for traffic sign classification.

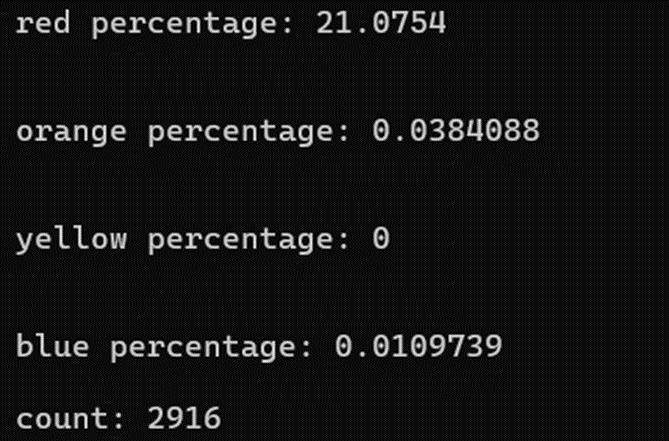


Figure 4.4.1 The number of values obtained in BRG feature extraction of an image



Figure 4.4.2 Mask chosen by dominant color

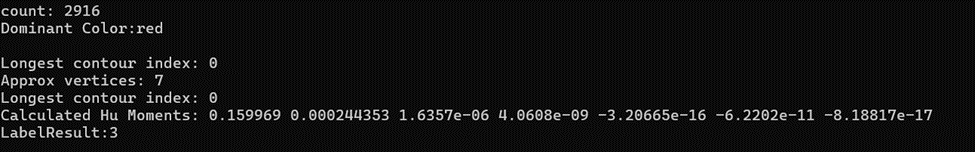


Figure 4.4.3 result of shape feature extraction (number of vertices and Hu moments)

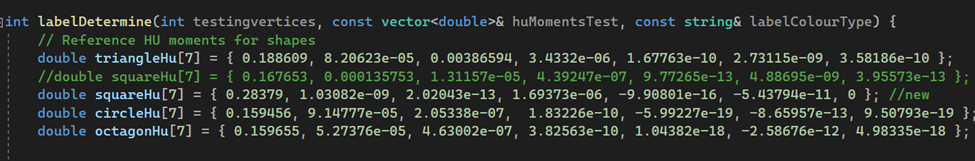


Figure 4.4.4 Hu moments reference value of each type of shape in labelDetermine function

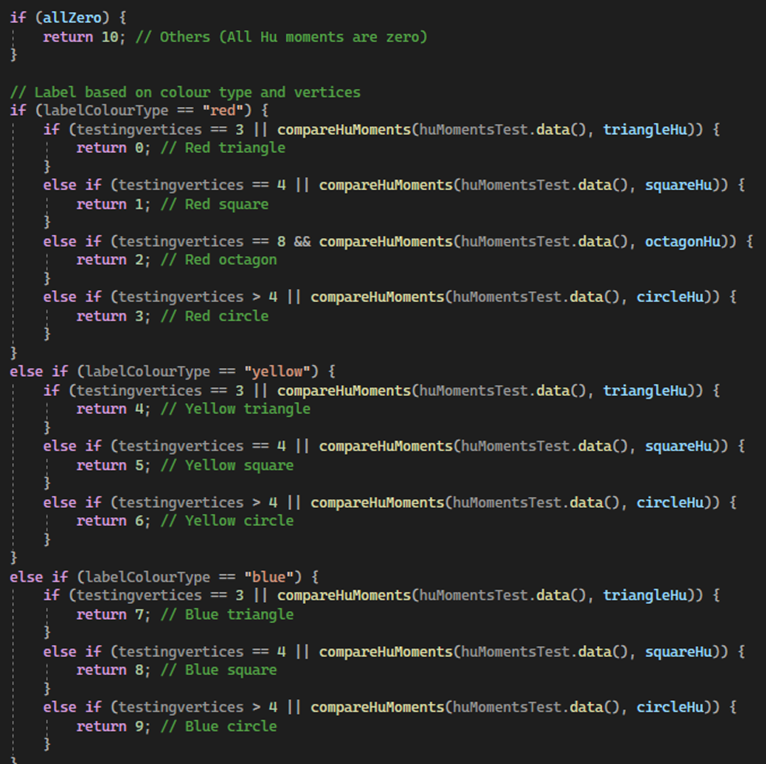


Figure 4.4.5 Logic assigning label and comparing reference Hu moment of shape

**4.5 Demo Training classifier and Evaluation of classifier**

Done by: Kok Tze Kang

After having the feature extraction flatten and put into vector float of trainingData and determined label y in corresponding vector integer labels, splitting of training and testing data is done by 80% data for training models and 20% data for testing models. The data is shuffled using seed of 52 for same data splitting conditions for repeated validation and correction.

Then, data for training is used to do model training for three classifiers: SVM, k-NN, and Random Forest using the training dataset. Each classifier was trained using the extracted features from the images. During training, the models learned to map feature vectors to the correct traffic sign labels. For the parameter setting explanation for the 3 classifiers:

SVM:

* Kernel: Linear for simple decision boundaries.
* Type: C-SVC, with a regularization parameter C = 1.0 to balance margin maximization and misclassification.
* Termination: Max iterations of 100 or epsilon (1e-6) to ensure efficient training.

k-NN:

* k: Set to 3, meaning the model considers the number of neighbours k closest neighbors for classification.
* Distance Metric: Euclidean distance to determine the nearest neighbors.

Random Forest:

* Max Depth: 7 to limit the depth of each tree to prevent overfitting.
* Max Categories: 15 for handling of multiclass classification.
* Termination: Max iterations of 100 or epsilon (0.01) ensuring efficient training.

The following figure shows the process flow of training the 3 trained classifiers and performance evaluation result:

A screen shot of a number

Description automatically generated

Figures 4.5.1. The accuracy of the 3 trained classifiers: SVM, k-NN, and Random Forest.

a) training b) testing

A screenshot of a computer

Description automatically generated A screenshot of a computer

Description automatically generated

Figures 4.5.2. The confusion matrix of the 3 trained classifiers: SVM, k-NN, and Random Forest for (a) training data and (b) testing data

A screenshot of a computer program

Description automatically generated

Figures 4.5.3. The precision, recall and F1 score for 10 (training data) classes of the 3 trained classifiers: SVM, k-NN, and Random Forest.

A screen shot of a computer program

Description automatically generated

Figures 4.5.4. The precision, recall and F1 score for 10 testing data classes of the 3 trained classifiers: SVM, k-NN, and Random Forest.

4.6 Implementation issues and Challenges

Done by Cornelius Wong Qin Jun

The first issue is the presence of similar colors in the background. When the traffic sign has a color similar to the background, the mask captures both, altering the original shape of the segmented traffic sign. As shown in Figure 4.6.1, the red noise in the background is the same color as the traffic sign, leading the system to capture and segment it along with the sign. This distorts the shape and prevents a proper circular segmentation, which in turn affects the computation of the number of vertices and Hu moment values, resulting in incorrect labeling.

The second issue is the dominant color problem. This occurs when the background contains a significant portion of colors like red, blue, orange, or yellow—the same colors used for traffic sign segmentation. If the background color dominates, the system evaluates the larger percentage of color and applies the corresponding mask. In such cases, the background, rather than the traffic sign, is detected. In Figure 4.6.2, the blue background color is more dominant than the red traffic sign, causing the system to segment the blue background instead of the red sign. The white region shows the segmented region which is not traffic sign.

The third issue is the longest contour problem, which is most common with yellow signs. In Figure 4.6.3, a triangle-shaped traffic sign is segmented, but the longest contour index is -1, indicating that no contour was found. This prevents the calculation of the number of vertices and Hu moment values, as these are based on the longest contour which shown in Figure 4.6.4. This problem is less frequent with red and blue signs because these colors are more stable and less affected by lighting conditions. Yellow, being a combination of red and green, is more susceptible to lighting changes, which often breaks the contours and disrupts the shape feature computation.

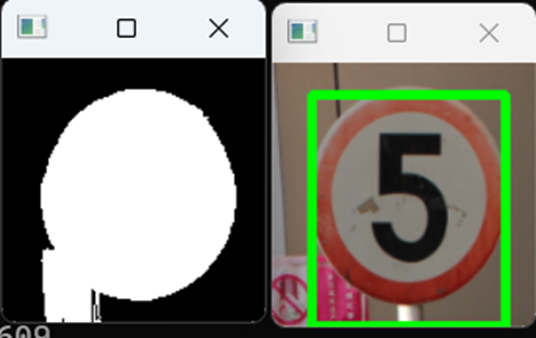


Figure 4.6.1 Similar background color with traffic sign

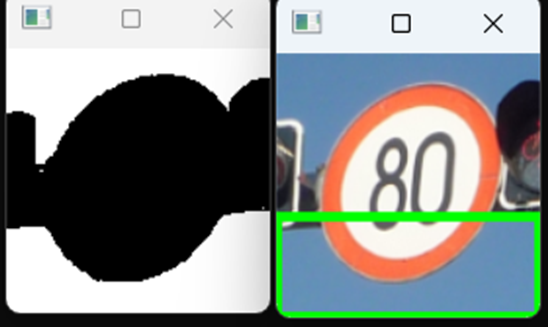


Figure 4.6.2 Background color as the dominant color and captured

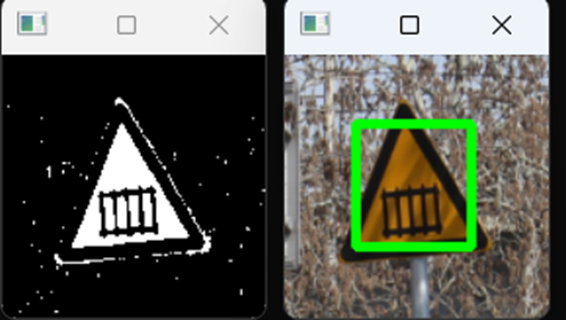


Figure 4.6.3 Longest contour problem in yellow

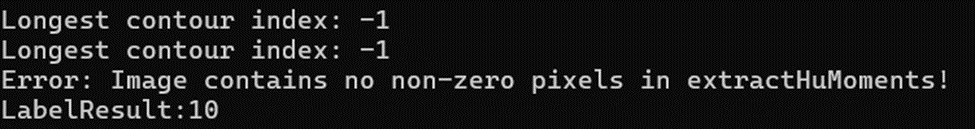


Figure 4.6.4 Longest contour detect fail in yellow

**CHAPTER 5**

**SYSTEM EVALUATION AND DISCUSSION**

**5.1 Comparison of classifier results**

Done by: Chua Jing Xuan

Table 5.1 shows that SVM performs substantially better in terms of overall accuracy than K-NN and Random Forest when comparing the three classifiers. SVM attains 68.75% accuracy, K-NN reaches 62.5% accuracy, and random attains the lowest accuracy of all three at just 50%. This discrepancy demonstrates how well SVM performs when utilising the same features to get precise predictions. Furthermore, when compared to the other classifiers, SVM performs better in terms of precision, recall, and F1 score. KNN does well in classes 3 and 4, but badly in classes 1 and 7. It is less accurate overall than SVM. Random does well in the third class, but not so well in the fourth or seventh. Its overall accuracy is the lowest, which can be the result of overfitting or difficulties differentiating between certain groups. This comparison shows that SVM is the best performing classifier in this dataset, followed by KNN, and finally Random Forest.

A screenshot of a computer program

Description automatically generated

Figure 5.1 The result of precision, recall and F1 Score

|  |  |  |  |
| --- | --- | --- | --- |
| **Classifier** | **SVM** | **KNN** | **Random Forest** |
| Accuracy (%) | 68.75 | 62.5 | 50 |
| Class 1 Precision (%) | 100 | 0 | 0 |
| Class 1 Recall (%) | 100 | 0 | 0 |
| Class 1 F1 Score (%) | 100 | 0 | 0 |
| Class 3 Precision (%) | 100 | 72.73 | 94.18 |
| Class 3 Recall (%) | 100 | 94.18 | 94.18 |
| Class 3 F1 Score (%) | 100 | 82.06 | 94.18 |
| Class 4 Precision (%) | 100 | 91.67 | 70 |
| Class 4 Recall (%) | 100 | 100 | 63.64 |
| Class 4 F1 Score (%) | 100 | 95.65 | 66.67 |
| Class 7 Precision (%) | 0 | 0 | 0 |
| Class 7 Recall (%) | 0 | 0 | 0 |
| Class 7 F1 Score (%) | 0 | 0 | 0 |
| Class 9 Precision (%) | 100 | 88.24 | 58.62 |
| Class 9 Recall (%) | 100 | 88.24 | 100 |
| Class 9 F1 Score (%) | 100 | 88.24 | 73.91 |

Table 5.1 Classifiers performance comparison

**Chapter 6**

**CONCLUSION AND RECOMMENDATION**

**6.1 Conclusion**

Done by: Chua Jing Xuan

In conclusion, the aim of this project was to employ machine learning techniques with OpenCV to create a reliable and effective system for detecting and recognising traffic signs. The system's objective was to correctly identify and categorise traffic signs in a range of backdrop and lighting situations. Utilising machine learning models in conjunction with colour and shape-based feature extraction approaches, the system showed encouraging results in the classification of traffic signals with various colours and shapes.

SVM, KNN, and Random Forest are the three machine learning classifiers that were examined; of these, SVM demonstrated the highest accuracy and dependability, surpassing the others in terms of F1 score, precision, and recall. In addition to efficiently handling colour segmentation for traffic signs with the colours red, blue, yellow, and orange, the system was also able to extract pertinent shape information, including vertices and Hu moments, which improved classification accuracy.

However, in certain situations, issues including illumination fluctuations and background colour interference impacted the system's functionality, especially with regard to yellow traffic signs. In spite of these obstacles, the project succeeded in achieving its goals by developing a system that can recognise traffic signs in real time, which is essential for improving driver safety and the efficiency of autonomous driving technology.

Overall, by putting forth a system that combines conventional image processing methods with machine learning models, the study advances the field of computer vision-based traffic sign recognition and opens the door to further advancements in practical applications.

**6.2 Recommendation**

Done by: Kok Tze Kang

For future direction and research, our suggestion would be the usage of Chinese character AI model. An experiment in a previous paper about traffic signs with Chinese characters show a failed recognition of some Chinese characters. This is expected a false negative result for the model to recognize the item detected as not a Chinese character which can cause a wrong candidate of item detection, and lead to misbehavior for an autonomous vehicle.

Therefore, we proposed an idea of using a Chinese character AI model to aid in more accurate and efficient character detection and recognition. By implementing an AI model that has been trained to recognize 5000+ Chinese characters that cover most of the daily used characters, it is expected to bring significant improvement in Chinese character detection and recognition by introducing new feature extraction to enrich the dataset and improve the robustness of the traffic sign detection and recognition system for the future research direction.

**Reference**

[1] J. Guo, R. You, and L. Huang, “Mixed Vertical-and-Horizontal-Text traffic sign detection and recognition for Street-Level scene,” *IEEE Access*, vol. 8, pp. 69413–69425, Jan. 2020, doi: 10.1109/access.2020.2986500.

[2] 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.

[3] A. Hechri and A. Mtibba, “Two-Stage Traffic Sign Detection and Recognition Based on SVM and Convolutional Neural Networks,” *IET Image Processing*, Dec. 2019.

[4] Deshmukh, V.R., Patnaik, G.K, Patil, M.E.: ‘Real-time traffic sign recognition system based on colour image segmentation’, Int. J. Comput. Appl., 2013, 83, (3), pp. 30–35

[5] Hechri, A., Mtibaa, A.: ‘Automatic detection and recognition of road sign for driver assistance system’. Proc..16th IEEE Mediterranean Electrotechnical Conf. (MELECON ‘12), Yasmine Hammamet, Tunisia, March 2012, pp. 888– 891

[6] Shaposhnikov, W.D., Shaposhnikov, D.G.N.L., Golovan, E.-V., et al.: ‘Road sign recognition by single positioning of space-variant sensor’. Proc.15th Int. Conf. on Vision Interface, Calgary, Canada, 2002, pp. 213–217

[7] Vitabile, S., Gentile, A., Sorbello, F.: ‘A neural network based automatic road signs recognizer’. Proc. Int. Joint Conf. on Neural Networks, Honolulu, HI, USA, USA, 12–17 May 2002

[8] Fleyeh, H.: ‘Road and traffic sign color detection and segmentation -A fuzzy approach’. Proc. Conf. on Machine Vision Applications, Tsukuba Science City, Japan, 16–18 May 2005

[9] Rashmi, H., Shashidha, M., Kumar, G.P.: ‘Automatic tracking of traffic signs based on HSV’, Int. J. Eng. Res. Technol. (IJERT), 2014, 3, (5), pp. 914–917

[10] Soetedjo, A., Somawirata, I.K.: ‘An efficient algorithm for implementing traffic sign detection on low cost embedded system’, Int. J. Innov. Comput., Inf. Control, 2014, 14, (1), pp. 1–14

[11] Qin, F., Fang, B., Zhao, H.: ‘Traffic sign segmentation and recognition in scene images’. Proc. Chinese Conf. on Pattern Recognition (CCPR), Chongqing, China, 21–23 October 2010

[12] Yang, Y., Luo, H., Xu, H., et al.: ‘Towards real-time traffic sign detection and classification’, IEEE Trans. Intell. Transp. Syst., 2016, 17, (7), pp. 2022–2031

[13] Cireşan, D., Meier, U., Masci, J., et al.: ‘Multi-column deep neural network for traffic sign classification’, Neural Netw., 2012, 32, pp. 333–338

[14] Srivastava, N., Hinton, G., Krizhevsky, A., et al.: ‘Dropout: A simple way to prevent neural networks from overfitting’, J. Mach. Learn. Res., 2014, 15, pp. 1929–1958

[15] Chang, X.; Yang, Y. Semisupervised feature analysis by mining correlations among multiple tasks. IEEE Trans. Neural Netw. Learn. Syst. 2017, 28, 2294–2305

[16] N. Daim, “36.3 million vehicles in Malaysia,” *NST Online*, Dec. 06, 2023. [Online]. Available: <https://www.nst.com.my/news/nation/2023/12/987062/363-million-vehicles-malaysia>

[17] “Ministry of Transport Malaysia.” <https://www.mot.gov.my/en/land/safety/road-accident-and-facilities>

**Appendix:**

#include <opencv2/opencv.hpp>

#include <opencv2/highgui/highgui.hpp>

#include <opencv2/imgproc.hpp>

#include <iostream>

#include <stdlib.h>

#include <string>

#include "Supp.h"

using namespace cv;

using namespace std;

using namespace cv::ml;

void Preprocess(Mat& srcI, Mat& hsv, Mat& Large, Mat win[], Mat legend[], int noOfImagePerCol, int noOfImagePerRow, vector<Mat>& hsvChannels, Mat& hsvEnhanced, Mat& enhancedImage)

{

cv::resize(srcI, srcI, cv::Size(135, 135), 0, 0, cv::INTER\_LINEAR);

createWindowPartition(srcI, Large, win, legend, noOfImagePerCol, noOfImagePerRow);

cvtColor(srcI, hsv, COLOR\_BGR2HSV);

split(hsv, hsvChannels);

double saturationFactor = 1.1, brightnessFactor = 1.7; // Adjust this factor to increase or decrease saturation

hsvChannels[1] \*= saturationFactor;

hsvChannels[2] \*= brightnessFactor;

// Clip the values to stay within valid range [0, 255]

hsvChannels[1] = min(hsvChannels[1], 255);

merge(hsvChannels, hsvEnhanced);

cvtColor(hsvEnhanced, enhancedImage, COLOR\_HSV2BGR);

}

Mat redMask(Mat hsvEnhanced)

{

Mat redMask1, redMask2, result;

//split(hsvEnhanced, threeImages); // split image into its BGR components

//// Below get red dominant regions / points

//redMask1 = (threeImages[0] \* 1.5 < threeImages[2]) &

// (threeImages[1] \* 1.5 < threeImages[2]);

////cvtColor(redMask1, result, COLOR\_GRAY2BGR); // show result of red color

Scalar RedLower(0, 90, 50);

Scalar RedLower2(8, 255, 255);

Scalar RedUpper(165, 110, 50);

Scalar RedUpper2(179, 255, 255);

inRange(hsvEnhanced, RedLower, RedLower2, redMask1); //create mask for Red traffic sign

inRange(hsvEnhanced, RedUpper, RedUpper2, redMask2);

bitwise\_or(redMask1, redMask2, result);

return result;

}

Mat OrangeMask(Mat hsvEnhanced)

{

Mat mask;

Scalar OrangeLower(10, 110, 110);

Scalar OrangeUpper(30, 255, 255);

inRange(hsvEnhanced, OrangeLower, OrangeUpper, mask); //create mask for Orange traffic sign

return mask;

}

Mat YellowMask(Mat hsvEnhanced)

{

Mat mask;

Scalar YellowLower(27, 120, 100);

Scalar YellowUpper(38, 255, 255);

inRange(hsvEnhanced, YellowLower, YellowUpper, mask); //create mask for yellow traffic sign

return mask;

}

Mat BlueMask(Mat hsvEnhanced)

{

Mat mask;

Scalar BlueLower(90, 110, 70);

Scalar BlueUpper(125, 255, 255);

inRange(hsvEnhanced, BlueLower, BlueUpper, mask); //create mask for Orange traffic sign

return mask;

}

void ContourDrawingAndFilling(Mat& canvasGray, vector<vector<Point> > contours, Mat& canvasColor, vector<Scalar> colors, Point2i center, Mat win[], Mat win3[], Mat srcI, vector<Mat>& LongestFill)

{

int max = 0, index = 0;

for (int i = 0; i < contours.size(); i++) { // We could have more than one sign in image

//canvasGray = 0;

canvasColor = Scalar(0, 0, 0);

if (max < contours[i].size()) { // Find the longest contour as sign boundary

max = contours[i].size();

index = i;

}

drawContours(canvasColor, contours, i, colors[i]); // draw Color boundaries

canvasColor.copyTo(win[2]);

canvasGray = 0;

drawContours(canvasGray, contours, index, 255); //draw the longest contour

cvtColor(canvasGray, win[4], COLOR\_GRAY2BGR);

Moments M = moments(canvasGray);

center.x = M.m10 / M.m00;

center.y = M.m01 / M.m00;

floodFill(canvasGray, center, 255);

cvtColor(canvasGray, win[5], COLOR\_GRAY2BGR);

//If longest contour fill with size of pixels > 3000 only considered containing traffic sign

if (countNonZero(canvasGray) > 3000 && countNonZero(canvasGray) < 11001)

{

Mat temp, Longest;

cvtColor(canvasGray, temp, COLOR\_GRAY2BGR);

//Extract the captured sign

Longest = srcI & temp;

Longest.copyTo(win[6]);

Longest.copyTo(win3[4]);

cvtColor(temp, temp, COLOR\_BGR2GRAY);

//copy longest fill image to vector

LongestFill.push\_back(temp);

}

}

}

void ResultPrint(Mat win[], Mat& mask4, Mat& mask3, Mat& mask2, Mat& mask, Mat srcI)

{

Mat Red, Yellow, Orange, Blue;

cvtColor(mask4, mask4, COLOR\_GRAY2BGR); // Blue\_mask

cvtColor(mask3, mask3, COLOR\_GRAY2BGR); // Yellow\_mask

cvtColor(mask2, mask2, COLOR\_GRAY2BGR); // Orange\_mask

cvtColor(mask, mask, COLOR\_GRAY2BGR); // Red\_mask

Blue = srcI & mask4; //Blue\_mask result

Yellow = srcI & mask3; //yellow mask result

Orange = srcI & mask2; //orange mask result

Red = srcI & mask; //red mask result

Red.copyTo(win[0]);

Yellow.copyTo(win[1]);

Orange.copyTo(win[2]);

Blue.copyTo(win[3]);

}

void Morphology(Mat& Result, Mat PreprocessImage) //Not Used

{

Mat kernel = getStructuringElement(MORPH\_RECT, Size(3, 3));

morphologyEx(PreprocessImage, Result, MORPH\_OPEN, kernel);

}

vector<float> extractBGRFeatures(const Mat& mask, const Mat& img) {

vector<float> histogram;

Mat hist;

int histSize[] = { 50, 50, 50 };

float bRanges[] = { 0, 256 };

float gRanges[] = { 0, 256 };

float rRanges[] = { 0, 256 };

const float\* ranges[] = { bRanges, gRanges, rRanges };

int channels[] = { 0, 1, 2 };

calcHist(&img, 1, channels, mask, hist, 3, histSize, ranges, true, false);

normalize(hist, hist, 0, 1, NORM\_MINMAX, -1, Mat());

histogram.insert(histogram.end(), hist.begin<float>(), hist.end<float>());

return histogram;

}

void ImageColor(Mat finalImage, vector<string>& color)

{

Mat hsv;

vector<Mat> hsvChannels;

double pixels[4], percentage[4], Percentage;

int index;

string Color[4]{ "red","orange","yellow","blue" };

cvtColor(finalImage, hsv, COLOR\_BGR2HSV);

split(hsv, hsvChannels);

double saturationFactor = 1.1, brightnessFactor = 1.7; // Adjust this factor to increase or decrease saturation

hsvChannels[1] \*= saturationFactor;

hsvChannels[2] \*= brightnessFactor;

// Clip the values to stay within valid range [0, 255]

hsvChannels[1] = min(hsvChannels[1], 255);

merge(hsvChannels, finalImage);

//cvtColor(finalImage, finalImage, COLOR\_HSV2BGR);

Mat mask = redMask(finalImage);

pixels[0] = countNonZero(mask);

mask = OrangeMask(finalImage);

pixels[1] = countNonZero(mask);

mask = YellowMask(finalImage);

pixels[2] = countNonZero(mask);

mask = BlueMask(finalImage);

pixels[3] = countNonZero(mask);

for (int i = 0; i < 4; i++)

{

percentage[i] = pixels[i] / (finalImage.cols \* finalImage.rows) \* 100;

cout << endl << Color[i] << " percentage: " << percentage[i] << endl << endl;

}

Percentage = percentage[0];

index = 0;

for (int i = 0; i < 3; i++)

if (percentage[i + 1] > Percentage)

{

Percentage = percentage[i + 1];

index = i + 1;

}

color.push\_back(Color[index]);

}

// feature extraction number of vertices

int extractVertices(const Mat& image) {

/\* //this is commend out for testing purpose because cannot run if in my fyp1 code

Mat gray;

cvtColor(image, gray, COLOR\_BGR2GRAY);

threshold(gray, gray, 127, 255, THRESH\_BINARY);

\*/

vector<vector<Point>> contours;

findContours(image, contours, RETR\_EXTERNAL, CHAIN\_APPROX\_NONE); // image is replaced from gray for testing

int longestContourIndex = -1;

double maxContourLength = 0;

// find logest contour

for (size\_t i = 0; i < contours.size(); i++) {

double contourLength = arcLength(contours[i], true);

if (contourLength > maxContourLength) {

maxContourLength = contourLength;

longestContourIndex = (int)i;

}

}

cout << "Longest contour index: " << longestContourIndex << endl;

if (longestContourIndex != -1) {

vector<Point> approx;

approxPolyDP(contours[longestContourIndex], approx, arcLength(contours[longestContourIndex], true) \* 0.04, true);

cout << "Approx vertices: " << approx.size() << endl;

return approx.size();

}

return 0;

}

vector<double> extractHuMoments(const Mat& img) {

/\* // this is commend out for testing as it cannot run in my fyp1 code

Mat gray;

cvtColor(img, gray, COLOR\_BGR2GRAY);

\*/

vector<vector<Point>> contours;

findContours(img, contours, RETR\_EXTERNAL, CHAIN\_APPROX\_NONE); // image is replaced from gray for testing

int longestContourIndex = -1;

double maxContourLength = 0;

// find logest contour

for (size\_t i = 0; i < contours.size(); i++) {

double contourLength = arcLength(contours[i], true);

if (contourLength > maxContourLength) {

maxContourLength = contourLength;

longestContourIndex = (int)i;

}

}

cout << "Longest contour index: " << longestContourIndex << endl;

//test

if (countNonZero(img) == 0) {

cout << "Error: Image contains no non-zero pixels in extractHuMoments!" << endl;

return vector<double>(7, 0);

}

if (longestContourIndex == -1) {

cout << "Error: longest contour index == -1!" << endl;

return vector<double>(7, 0);

}

Moments m = moments(contours[longestContourIndex]); //img is replaced from gray for testing

//test

if (m.m00 == 0) {

cout << "Warning: Moment calculation failed (m.m00 is zero)" << endl;

return vector<double>(7, 0);

}

double hu[7];

HuMoments(m, hu);

// Debug: Print Hu moments

cout << "Calculated Hu Moments: ";

for (int i = 0; i < 7; i++) {

cout << hu[i] << " ";

}

cout << endl;

vector<double> huFeatures(hu, hu + 7);

return huFeatures;

}

// Compare hu moment by shape in labelDetermine function

bool compareHuMoments(const double hu[7], const double shapeHu[7], double threshold = 0.001) {

for (int i = 0; i < 7; i++) {

if (fabs(hu[i] - shapeHu[i]) > threshold) {

return false;

}

}

return true;

}

int labelDetermine(int testingvertices, const vector<double>& huMomentsTest, const string& labelColourType) {

// Reference HU moments for shapes

double triangleHu[7] = { 0.188609, 8.20623e-05, 0.00386594, 3.4332e-06, 1.67763e-10, 2.73115e-09, 3.58186e-10 };

//double squareHu[7] = { 0.167653, 0.000135753, 1.31157e-05, 4.39247e-07, 9.77265e-13, 4.88695e-09, 3.95573e-13 };

double squareHu[7] = { 0.28379, 1.03082e-09, 2.02043e-13, 1.69373e-06, -9.90801e-16, -5.43794e-11, 0 }; //new

double circleHu[7] = { 0.159456, 9.14777e-05, 2.05338e-07, 1.83226e-10, -5.99227e-19, -8.65957e-13, 9.50793e-19 };

double octagonHu[7] = { 0.159655, 5.27376e-05, 4.63002e-07, 3.82563e-10, 1.04382e-18, -2.58676e-12, 4.98335e-18 };

// Check if all Hu moments are zero

bool allZero = true;

for (double val : huMomentsTest) {

if (val != 0) {

allZero = false;

break;

}

}

if (allZero) {

return 10; // Others (All Hu moments are zero)

}

// Label based on colour type and vertices

if (labelColourType == "red") {

if (testingvertices == 3 || compareHuMoments(huMomentsTest.data(), triangleHu)) {

return 0; // Red triangle

}

else if (testingvertices == 4 || compareHuMoments(huMomentsTest.data(), squareHu)) {

return 1; // Red square

}

else if (testingvertices == 8 && compareHuMoments(huMomentsTest.data(), octagonHu)) {

return 2; // Red octagon

}

else if (testingvertices > 4 || compareHuMoments(huMomentsTest.data(), circleHu)) {

return 3; // Red circle

}

}

else if (labelColourType == "yellow") {

if (testingvertices == 3 || compareHuMoments(huMomentsTest.data(), triangleHu)) {

return 4; // Yellow triangle

}

else if (testingvertices == 4 || compareHuMoments(huMomentsTest.data(), squareHu)) {

return 5; // Yellow square

}

else if (testingvertices > 4 || compareHuMoments(huMomentsTest.data(), circleHu)) {

return 6; // Yellow circle

}

}

else if (labelColourType == "blue") {

if (testingvertices == 3 || compareHuMoments(huMomentsTest.data(), triangleHu)) {

return 7; // Blue triangle

}

else if (testingvertices == 4 || compareHuMoments(huMomentsTest.data(), squareHu)) {

return 8; // Blue square

}

else if (testingvertices > 4 || compareHuMoments(huMomentsTest.data(), circleHu)) {

return 9; // Blue circle

}

}

return -1; // Return -1 if no match (this shouldn't happen if logic is correct)

}

// SVM Training

Ptr<SVM> trainSVM(const vector<vector<float>>& featureVectors, const vector<int>& labels) {

Mat trainData = Mat(featureVectors.size(), featureVectors[0].size(), CV\_32F);

Mat trainLabels = Mat(labels.size(), 1, CV\_32S);

for (int i = 0; i < featureVectors.size(); i++) {

for (int j = 0; j < featureVectors[i].size(); j++) {

trainData.at<float>(i, j) = static\_cast<float>(featureVectors[i][j]);

}

trainLabels.at<int>(i, 0) = labels[i];

}

Ptr<SVM> svm = SVM::create();

svm->setKernel(SVM::LINEAR);

svm->setType(SVM::C\_SVC);

svm->setC(1.0);

svm->setTermCriteria(TermCriteria(TermCriteria::MAX\_ITER, 100, 1e-6));

svm->train(trainData, ROW\_SAMPLE, trainLabels);

return svm;

}

// k-NN Training

Ptr<KNearest> trainKNN(const vector<vector<float>>& featureVectors, const vector<int>& labels) {

Mat trainData = Mat(featureVectors.size(), featureVectors[0].size(), CV\_32F);

Mat trainLabels = Mat(labels.size(), 1, CV\_32S);

for (int i = 0; i < featureVectors.size(); i++) {

for (int j = 0; j < featureVectors[i].size(); j++) {

trainData.at<float>(i, j) = static\_cast<float>(featureVectors[i][j]);

}

trainLabels.at<int>(i, 0) = labels[i];

}

Ptr<KNearest> knn = KNearest::create();

knn->setDefaultK(3); // Set the number of neighbors

knn->train(trainData, ROW\_SAMPLE, trainLabels);

return knn;

}

//random forest

Ptr<RTrees> trainForest(const vector<vector<float>>& featureVectors, const vector<int>& labels) {

Mat trainData = Mat(featureVectors.size(), featureVectors[0].size(), CV\_32F);

Mat trainLabels = Mat(labels.size(), 1, CV\_32S);

for (int i = 0; i < featureVectors.size(); i++) {

for (int j = 0; j < featureVectors[i].size(); j++) {

trainData.at<float>(i, j) = static\_cast<float>(featureVectors[i][j]);

}

trainLabels.at<int>(i, 0) = labels[i];

}

// Create and configure the random forest model

Ptr<ml::RTrees> randomForest = ml::RTrees::create();

randomForest->setMaxDepth(7); // Set maximum depth of the tree

randomForest->setMaxCategories(15); // Max number of categories (useful for categorical variables)

randomForest->setPriors(Mat()); // Priors of each class

randomForest->setTermCriteria(TermCriteria(TermCriteria::MAX\_ITER, 100, 0.01));

// Train the random forest

randomForest->train(trainData, ml::ROW\_SAMPLE, trainLabels);

return randomForest;

}

// Split data into train and test

void splitData(const vector<vector<float>>& data, const vector<int>& labels, float trainRatio,

vector<vector<float>>& trainData, vector<vector<float>>& testData, vector<int>& trainLabels, vector<int>& testLabels)

{

vector<int> indices(data.size());

for (int i = 0; i < indices.size(); ++i) {

indices[i] = i;

}

// Shuffle the indices using seed for validation

unsigned int seed = 52;

srand(seed);

random\_shuffle(indices.begin(), indices.end());

int trainSize = static\_cast<int>(trainRatio \* data.size());

for (int i = 0; i < trainSize; ++i) {

trainData.push\_back(data[indices[i]]);

trainLabels.push\_back(labels[indices[i]]);

}

for (int i = trainSize; i < indices.size(); ++i) {

testData.push\_back(data[indices[i]]);

testLabels.push\_back(labels[indices[i]]);

}

// cout << train

}

// Function to calculate precision, recall, and F1 score for each class

void calculateMetrics(const vector<int>& trueLabels, const vector<int>& predictedLabels, int numClasses) {

vector<int> truePositives(numClasses, 0);

vector<int> falsePositives(numClasses, 0);

vector<int> falseNegatives(numClasses, 0);

// Calculate TP, FP, FN for each class

for (int i = 0; i < trueLabels.size(); i++) {

int trueLabel = trueLabels[i];

int predictedLabel = predictedLabels[i];

if (trueLabel == predictedLabel) {

truePositives[trueLabel]++;

}

else {

falsePositives[predictedLabel]++;

falseNegatives[trueLabel]++;

}

}

// Print precision, recall, and F1 score for each class

for (int i = 0; i < numClasses; i++) {

double precision = (truePositives[i] + falsePositives[i] > 0) ?

static\_cast<double>(truePositives[i]) / (truePositives[i] + falsePositives[i]) : 0;

double recall = (truePositives[i] + falseNegatives[i] > 0) ?

static\_cast<double>(truePositives[i]) / (truePositives[i] + falseNegatives[i]) : 0;

double f1Score = (precision + recall > 0) ? 2 \* (precision \* recall) / (precision + recall) : 0;

cout << "Class " << i << ": "

<< "Precision = " << precision

<< ", Recall = " << recall

<< ", F1 Score = " << f1Score

<< endl;

}

}

// Function to calculate confusion matrix

Mat calculateConfusionMatrix(const vector<int>& trueLabels, const vector<int>& predictedLabels, int numClasses) {

Mat confusionMatrix = Mat::zeros(numClasses, numClasses, CV\_32S);

for (int i = 0; i < trueLabels.size(); i++) {

int trueLabel = trueLabels[i];

int predictedLabel = predictedLabels[i];

confusionMatrix.at<int>(trueLabel, predictedLabel)++;

}

return confusionMatrix;

}

// Function to visualize the confusion matrix

void visualizeConfusionMatrix(const Mat& confusionMatrix, const string& windowName) {

int cellSize = 50;

int numClasses = confusionMatrix.rows;

// Create an image large enough to hold the confusion matrix

Mat visualMatrix = Mat::zeros(numClasses \* cellSize, numClasses \* cellSize, CV\_8UC3);

// Normalize confusion matrix to [0, 255] for visualization

Mat normalizedConfusion;

normalize(confusionMatrix, normalizedConfusion, 0, 255, NORM\_MINMAX, CV\_32F);

// Draw each cell of the confusion matrix

for (int i = 0; i < numClasses; i++) {

for (int j = 0; j < numClasses; j++) {

int value = static\_cast<int>(normalizedConfusion.at<float>(i, j));

Rect cellRect(j \* cellSize, i \* cellSize, cellSize, cellSize);

// Color intensity based on the value

Scalar color(value, 0, 255 - value); // Red to Green gradient

// Draw the cell

rectangle(visualMatrix, cellRect, color, FILLED);

// Add text to show the actual value in each cell

string label = to\_string(confusionMatrix.at<int>(i, j));

putText(visualMatrix, label, Point(j \* cellSize + 10, i \* cellSize + 30), FONT\_HERSHEY\_SIMPLEX, 0.8, Scalar(255, 255, 255), 2);

}

}

// Display the matrix in a window

imshow(windowName, visualMatrix);

waitKey(0);

}

int main()

{

String imgPattern("Inputs/Traffic signs/\*.png");

vector<string> imageNames;

int const noOfImagePerCol = 5, noOfImagePerRow = 2;

Mat threeImages[3], srcI, thresh, canvasColor, canvasGray, gaussian, hsv, test, blur;

vector<Scalar> colors;

int t1, t2, t3, t4, index, max = 0;// used to record down the longest contour

RNG rng(0);

Point2i center;

//store feature and label

vector<vector<float>> trainingData;

vector<int> labels;

Mat Result, Result\_win[1 \* 5], legend3[1 \* 5];

Mat Red\_large, Orange\_large, Yellow\_large, Blue\_large, win[4][noOfImagePerCol \* noOfImagePerRow], legend[4][noOfImagePerCol \* noOfImagePerRow];

for (int i = 0; i < 300; i++) {

for (;;) {

t1 = rng.uniform(0, 255); // blue

t2 = rng.uniform(0, 255); // green

t3 = rng.uniform(0, 255); // red

t4 = t1 + t2 + t3;

// Below get random colors that is not dim

if (t4 > 255) break;

}

colors.push\_back(Scalar(t1, t2, t3));

}

cv::glob(imgPattern, imageNames, true);

//cout << imageNames.size();

for (int j = 0; j < imageNames.size(); j++)

{

srcI = imread(imageNames[j]);

//cvtColor(srcI, srcI, COLOR\_GRAY2BGR);

if (srcI.empty()) { // found no such file?

cout << "cannot open image for reading" << endl;

return -1;

}

//create hsv channels to contain split hsv traits

vector<Mat> hsvChannels;

Mat hsvEnhanced, enhancedImage, Red\_mask, Orange\_mask, Yellow\_mask, Blue\_mask;

//Used to find contour that contains Traffic Sign but not others

vector<vector<Point> > Contour, Contour2, FinalContour;

//store all longestFill image

vector<Mat> LongestFill;

//store shortlisted final image (Mask or LongestFill)

vector<Mat> CounterAndMask;

//Preprocess picture and Create windows to display

Preprocess(srcI, hsv, Red\_large, win[0], legend[0], noOfImagePerCol, noOfImagePerRow, hsvChannels, hsvEnhanced, enhancedImage);

Preprocess(srcI, hsv, Orange\_large, win[1], legend[1], noOfImagePerCol, noOfImagePerRow, hsvChannels, hsvEnhanced, enhancedImage);

Preprocess(srcI, hsv, Yellow\_large, win[2], legend[2], noOfImagePerCol, noOfImagePerRow, hsvChannels, hsvEnhanced, enhancedImage);

Preprocess(srcI, hsv, Blue\_large, win[3], legend[3], noOfImagePerCol, noOfImagePerRow, hsvChannels, hsvEnhanced, enhancedImage);

//Preprocess(srcI, hsv, newFrame, Mor\_win, legend2, noOfImagePerCol, noOfImagePerRow, hsvChannels, hsvEnhanced, enhancedImage);

Preprocess(srcI, hsv, Result, Result\_win, legend3, 1, 5, hsvChannels, hsvEnhanced, enhancedImage);

Red\_mask = redMask(hsvEnhanced);

Orange\_mask = OrangeMask(hsvEnhanced);

Yellow\_mask = YellowMask(hsvEnhanced);

Blue\_mask = BlueMask(hsvEnhanced);

//Vector stores all masked results (4 colors Masks results)

vector<Mat> Masks{ Red\_mask, Orange\_mask, Yellow\_mask, Blue\_mask };

ResultPrint(Result\_win, Blue\_mask, Yellow\_mask, Orange\_mask, Red\_mask, srcI);

vector<vector<Point>> contours;

//create canvases for line drawing

canvasColor.create(srcI.rows, srcI.cols, CV\_8UC3);

canvasGray.create(srcI.rows, srcI.cols, CV\_8U);

for (int i = 0; i < Masks.size(); i++)

{

findContours(Masks[i], contours, RETR\_EXTERNAL, CHAIN\_APPROX\_NONE);

ContourDrawingAndFilling(canvasGray, contours, canvasColor, colors, center, win[i], Result\_win, srcI, LongestFill);

//Red

putText(legend[0][0], "Original", Point(5, 11), 1, 1, Scalar(250, 250, 250), 1);

putText(legend[0][1], "", Point(5, 11), 1, 1, Scalar(250, 250, 250), 1);

putText(legend[0][2], "Color contours", Point(5, 11), 1, 1, Scalar(250, 250, 250), 1);

putText(legend[0][3], "Gray contour", Point(5, 11), 1, 1, Scalar(250, 250, 250), 1);

putText(legend[0][4], "FillContour", Point(5, 11), 1, 1, Scalar(250, 250, 250), 1);

putText(legend[0][5], "Longest contour", Point(5, 11), 1, 1, Scalar(250, 250, 250), 1);

putText(legend[0][6], "LongFillContour", Point(5, 11), 1, 1, Scalar(250, 250, 250), 1);

putText(legend[0][7], "Red Mask", Point(5, 11), 1, 1, Scalar(250, 250, 250), 1);

putText(legend[0][8], "OrangeResult", Point(5, 11), 1, 1, Scalar(250, 250, 250), 1);

putText(legend[0][9], "YellowResult", Point(5, 11), 1, 1, Scalar(250, 250, 250), 1);

//Orange

putText(legend[1][0], "Original", Point(5, 11), 1, 1, Scalar(250, 250, 250), 1);

putText(legend[1][1], "", Point(5, 11), 1, 1, Scalar(250, 250, 250), 1);

putText(legend[1][2], "Color contours", Point(5, 11), 1, 1, Scalar(250, 250, 250), 1);

putText(legend[1][3], "Gray contour", Point(5, 11), 1, 1, Scalar(250, 250, 250), 1);

putText(legend[1][4], "FillContour", Point(5, 11), 1, 1, Scalar(250, 250, 250), 1);

putText(legend[1][5], "Longest contour", Point(5, 11), 1, 1, Scalar(250, 250, 250), 1);

putText(legend[1][6], "LongFillContour", Point(5, 11), 1, 1, Scalar(250, 250, 250), 1);

putText(legend[1][7], "Orange Mask", Point(5, 11), 1, 1, Scalar(250, 250, 250), 1);

putText(legend[1][8], "OrangeResult", Point(5, 11), 1, 1, Scalar(250, 250, 250), 1);

putText(legend[1][9], "YellowResult", Point(5, 11), 1, 1, Scalar(250, 250, 250), 1);

//Yellow

putText(legend[2][0], "Original", Point(5, 11), 1, 1, Scalar(250, 250, 250), 1);

putText(legend[2][1], "", Point(5, 11), 1, 1, Scalar(250, 250, 250), 1);

putText(legend[2][2], "Color contours", Point(5, 11), 1, 1, Scalar(250, 250, 250), 1);

putText(legend[2][3], "Gray contour", Point(5, 11), 1, 1, Scalar(250, 250, 250), 1);

putText(legend[2][4], "FillContour", Point(5, 11), 1, 1, Scalar(250, 250, 250), 1);

putText(legend[2][5], "Longest contour", Point(5, 11), 1, 1, Scalar(250, 250, 250), 1);

putText(legend[2][6], "LongFillContour", Point(5, 11), 1, 1, Scalar(250, 250, 250), 1);

putText(legend[2][7], "Yellow Mask", Point(5, 11), 1, 1, Scalar(250, 250, 250), 1);

putText(legend[2][8], "OrangeResult", Point(5, 11), 1, 1, Scalar(250, 250, 250), 1);

putText(legend[2][9], "YellowResult", Point(5, 11), 1, 1, Scalar(250, 250, 250), 1);

//Blue

putText(legend[3][0], "Original", Point(5, 11), 1, 1, Scalar(250, 250, 250), 1);

putText(legend[3][1], "", Point(5, 11), 1, 1, Scalar(250, 250, 250), 1);

putText(legend[3][2], "Color contours", Point(5, 11), 1, 1, Scalar(250, 250, 250), 1);

putText(legend[3][3], "Gray contour", Point(5, 11), 1, 1, Scalar(250, 250, 250), 1);

putText(legend[3][4], "FillContour", Point(5, 11), 1, 1, Scalar(250, 250, 250), 1);

putText(legend[3][5], "Longest contour", Point(5, 11), 1, 1, Scalar(250, 250, 250), 1);

putText(legend[3][6], "LongFillContour", Point(5, 11), 1, 1, Scalar(250, 250, 250), 1);

putText(legend[3][7], "Blue Mask", Point(5, 11), 1, 1, Scalar(250, 250, 250), 1);

putText(legend[3][8], "OrangeResult", Point(5, 11), 1, 1, Scalar(250, 250, 250), 1);

putText(legend[3][9], "YellowResult", Point(5, 11), 1, 1, Scalar(250, 250, 250), 1);

//Results

putText(legend3[0], "Red", Point(5, 11), 1, 1, Scalar(250, 250, 250), 1);

putText(legend3[1], "Yellow", Point(5, 11), 1, 1, Scalar(250, 250, 250), 1);

putText(legend3[2], "Orange", Point(5, 11), 1, 1, Scalar(250, 250, 250), 1);

putText(legend3[3], "Blue", Point(5, 11), 1, 1, Scalar(250, 250, 250), 1);

putText(legend3[4], "Longest", Point(5, 11), 1, 1, Scalar(250, 250, 250), 1);

Mat Mask;

srcI.copyTo(win[0][0]);

srcI.copyTo(win[1][0]);

srcI.copyTo(win[2][0]);

srcI.copyTo(win[3][0]);

Mask = Red\_mask & srcI;

Mask.copyTo(win[0][7]);

Mask = Orange\_mask & srcI;

Mask.copyTo(win[1][7]);

Mask = Yellow\_mask & srcI;

Mask.copyTo(win[2][7]);

Mask = Blue\_mask & srcI;

Mask.copyTo(win[3][7]);

}

vector<string> color;

//Processes all shortlisted Mask and LongestFill images to find the real traffic sign

for (int i = 0; i < std::max(Masks.size(), LongestFill.size()); i++)

{

//4 Masks images filter Model

if (i < Masks.size())

{

//image with at least 1500 pixel only possible to contain traffic sign

if (countNonZero(Masks[i]) > 1500 && countNonZero(Masks[i]) < 11001)

{

findContours(Masks[i], Contour, RETR\_EXTERNAL, CHAIN\_APPROX\_NONE);

//filter all noise which are not traffic sign

for (int y = 0; y < Contour.size(); y++)

{

Rect boundingRect = cv::boundingRect(Contour[y]);

if (cv::contourArea(Contour[y]) > 1500 && cv::contourArea(Contour[y]) < 11001)

{

//Traffic sign shape usually close to square

if (boundingRect.x < boundingRect.y \* 1.2 || boundingRect.x \* 1.2 > boundingRect.y)

{

//Shortlisted images are pushed into vector

CounterAndMask.push\_back(Masks[i]);

FinalContour.push\_back(Contour[y]);

}

}

}

}

}

//LongestFill images filter Model

if (i < LongestFill.size())

// cout << endl <<"pixel: " << countNonZero(LongestFill[i]) << endl;

if (LongestFill.size() != 0)

{

//image with at least 1500 pixel only possible to contain traffic sign

if (countNonZero(LongestFill[0]) > 1500 && countNonZero(LongestFill[0]) < 11001)

{

findContours(LongestFill[0], Contour2, RETR\_EXTERNAL, CHAIN\_APPROX\_NONE);

for (int z = 0; z < Contour2.size(); z++)

{

Rect boundingRect2 = cv::boundingRect(Contour2[z]);

if (cv::contourArea(Contour2[z]) > 1500 && cv::contourArea(Contour2[z]) < 11001)

{

//Traffic sign shape usually close to square

if (boundingRect2.x < boundingRect2.y \* 1.2 || boundingRect2.x \* 1.2 > boundingRect2.y)

{

//Shortlisted images are pushed into vector

CounterAndMask.push\_back(LongestFill[0]);

FinalContour.push\_back(Contour2[z]);

}

}

}

}

LongestFill.erase(LongestFill.begin() + 0); //Erase the image from vector Whether shortlisted or not (Not repeatedly analyse the same image)

}

// cout << endl << "total pic: " << CounterAndMask.size() << endl;

if (CounterAndMask.size() > 1)

{

int value;

//Pick only one image from all shortlisted image for final Showcase

for (int i = 0; i < (CounterAndMask.size()); i++)

{

value = countNonZero(CounterAndMask[0]);

//if Image 1 > Image 2, then Image 2 is deleted

if (value >= countNonZero(CounterAndMask[1]))

{

CounterAndMask.erase(CounterAndMask.begin() + 1);

FinalContour.erase(FinalContour.begin() + 1);

}

//if Image 2 > Image 1, then Image 1 is deleted

else

{

value = countNonZero(CounterAndMask[1]);

CounterAndMask.erase(CounterAndMask.begin() + 0);

FinalContour.erase(FinalContour.begin() + 0);

}

}

}

// cout << endl << "contour:" << FinalContour.size() << endl;

}

Mat GreenBox, temp;

srcI.copyTo(GreenBox);

if (FinalContour.size() > 0)

{

//Draw Green box around detected components (Traffic sign)

cv::rectangle(GreenBox, boundingRect(FinalContour[0]), cv::Scalar(0, 255, 0), 3);

// imshow("Shortlisted Image", CounterAndMask[0]);

vector<float> bgrFeatures = extractBGRFeatures(CounterAndMask[0], srcI);

vector<double> combinedFeatures(bgrFeatures.begin(), bgrFeatures.end());

int count = 0;

for (int i = 0; i < combinedFeatures.size(); i++) {

if (combinedFeatures[i] != 0)

cout << "non-zero color feature: " << combinedFeatures[i] << endl;

count += 1;

}

cvtColor(CounterAndMask[0], temp, COLOR\_GRAY2BGR);

temp = srcI & temp;

ImageColor(temp, color);

cout << "count: " << count << endl << "Dominant Color:" << color[0] << endl << endl;

int masktype = 0;

if (color[0] == "red") {

masktype = 0;//red

}

else if (color[0] == "orange") {

masktype = 1;//orange

}

else if (color[0] == "yellow") {

masktype = 2;//yellow

}

else {

masktype = 3;//blue

}

vector<float> shapeFeature;

int NumberOfVertices = extractVertices(Masks[masktype]);

vector<double> HuResult = extractHuMoments(Masks[masktype]);

shapeFeature.push\_back(static\_cast<float>(NumberOfVertices));

// Loop through the 7 Hu moments and push them one by one into shapeFeature

for (size\_t i = 0; i < HuResult.size(); ++i) {

shapeFeature.push\_back(static\_cast<float>(HuResult[i]));

}

//comnbine bgr and shape

vector<float> combinedBgrShapeFeature;

// Combine bgrFeature and shapeFeature

combinedBgrShapeFeature.insert(combinedBgrShapeFeature.end(), bgrFeatures.begin(), bgrFeatures.end());

combinedBgrShapeFeature.insert(combinedBgrShapeFeature.end(), shapeFeature.begin(), shapeFeature.end());

// Push the combined feature vector to trainingData

trainingData.push\_back(combinedBgrShapeFeature);

//save into trainingData

if (masktype == 1)

color[0] = "yellow";

int Number = labelDetermine(NumberOfVertices, HuResult, color[0]);

cout << "LabelResult:" << Number << endl << endl;

//store into labels

labels.push\_back(Number);

}

//GreenBox = srcI & GreenBox;

// imshow("GreenBox Image", GreenBox);

waitKey();

destroyAllWindows();

}

//==========================================

// Split data into 80% train, 20% test

vector<vector<float>> trainData, testData;

vector<int> trainLabels, testLabels;

splitData(trainingData, labels, 0.8, trainData, testData, trainLabels, testLabels);

cout << "training start............................" << endl;

cout << "Waiting............................" << endl;

Ptr<ml::SVM> svm;

Ptr<ml::KNearest> knn;

Ptr<ml::RTrees> randomForest;

//training model =======================================================================

svm = trainSVM(trainData, trainLabels);

knn = trainKNN(trainData, trainLabels);

randomForest = trainForest(trainData, trainLabels);

cout << "training success............................" << endl;

// Evaluate on the test data================================================================

vector<int> svmPredictions, knnPredictions, randomPredictions;

for (int i = 0; i < testData.size(); i++) {

int predictedLabel = static\_cast<int>(svm->predict(testData[i]));

svmPredictions.push\_back(predictedLabel);

}

for (int i = 0; i < testData.size(); i++) {

int predictedLabel = static\_cast<int>(knn->predict(testData[i]));

knnPredictions.push\_back(predictedLabel);

}

for (int i = 0; i < testData.size(); i++) {

int predictedLabel = static\_cast<int>(randomForest->predict(testData[i]));

randomPredictions.push\_back(predictedLabel);

}

// Calculate accuracy for each model

auto calculateAccuracy = [](const vector<int>& trueLabels, const vector<int>& predictedLabels) -> double {

int correct = 0;

for (size\_t i = 0; i < trueLabels.size(); ++i) {

if (trueLabels[i] == predictedLabels[i]) {

++correct;

}

}

return static\_cast<double>(correct) / trueLabels.size() \* 100.0;

};

if (!svmPredictions.empty()) {

double accuracySVM = calculateAccuracy(testLabels, svmPredictions);

cout << "Accuracy of SVM: " << accuracySVM << "%" << endl;

}

if (!knnPredictions.empty()) {

double accuracyKNN = calculateAccuracy(testLabels, knnPredictions);

cout << "Accuracy of k-NN: " << accuracyKNN << "%" << endl;

}

if (!randomPredictions.empty()) {

double accuracyRF = calculateAccuracy(testLabels, randomPredictions);

cout << "Accuracy of Random Forest: " << accuracyRF << "%" << endl;

}

//=====================================

// Define the number of classes (e.g., 10 classes for traffic signs)

int numClasses = 10;

// Predict on training data for SVM, k-NN, Random Forest

vector<int> svmTrainPredictions, knnTrainPredictions, randomForestTrainPredictions;

// Predict using SVM

for (int i = 0; i < trainData.size(); i++) {

int predictedLabel = static\_cast<int>(svm->predict(trainData[i]));

svmTrainPredictions.push\_back(predictedLabel);

}

// Predict using k-NN

for (int i = 0; i < trainData.size(); i++) {

int predictedLabel = static\_cast<int>(knn->predict(trainData[i]));

knnTrainPredictions.push\_back(predictedLabel);

}

// Predict using Random Forest

for (int i = 0; i < trainData.size(); i++) {

int predictedLabel = static\_cast<int>(randomForest->predict(trainData[i]));

randomForestTrainPredictions.push\_back(predictedLabel);

}

// ==========================================

// Calculate Confusion Matrix for Training Data

cout << "======================================" << endl;

cout << "Confusion Matrix and Metrics for Training Data==============" << endl;

// SVM Training Confusion Matrix

Mat svmTrainConfusionMatrix = calculateConfusionMatrix(trainLabels, svmTrainPredictions, numClasses);

cout << "\nConfusion Matrix:\n" << svmTrainConfusionMatrix << endl;

// k-NN Training Confusion Matrix

Mat knnTrainConfusionMatrix = calculateConfusionMatrix(trainLabels, knnTrainPredictions, numClasses);

cout << "\nConfusion Matrix:\n" << knnTrainConfusionMatrix << endl;

// Random Forest Training Confusion Matrix

Mat randomForestTrainConfusionMatrix = calculateConfusionMatrix(trainLabels, randomForestTrainPredictions, numClasses);

cout << "\nConfusion Matrix:\n" << randomForestTrainConfusionMatrix << endl;

// ==========================================

// Calculate Precision, Recall, and F1 Score for Training Data

cout << "======================================Training" << endl;

cout << "Precision, Recall, and F1 Score for Training Data" << endl;

// SVM Metrics for Training Data

cout << "SVM (Training) Metrics: " << endl;

calculateMetrics(trainLabels, svmTrainPredictions, numClasses);

// k-NN Metrics for Training Data

cout << "k-NN (Training) Metrics: " << endl;

calculateMetrics(trainLabels, knnTrainPredictions, numClasses);

// Random Forest Metrics for Training Data

cout << "Random Forest (Training) Metrics: " << endl;

calculateMetrics(trainLabels, randomForestTrainPredictions, numClasses);

cout << "======================================Testing" << endl;

// Calculate metrics for SVM

cout << "SVM (Testing) metrics: " << endl;

calculateMetrics(testLabels, svmPredictions, numClasses);

// Calculate metrics for k-NN

cout << "k-NN (Testing) metrics: " << endl;

calculateMetrics(testLabels, knnPredictions, numClasses);

// Calculate metrics for Random Forest

cout << "Random Forest (Testing) metrics: " << endl;

calculateMetrics(testLabels, randomPredictions, numClasses);

//confusion matrix

cout << "Confusion Matrix and Metrics for Test Data==============" << endl;

// Calculate and visualize confusion matrix for SVM

Mat svmConfusionMatrix = calculateConfusionMatrix(testLabels, svmPredictions, numClasses);

cout << "\nConfusion Matrix:\n" << svmTrainConfusionMatrix << endl;

// Calculate and visualize confusion matrix for k-NN

Mat knnConfusionMatrix = calculateConfusionMatrix(testLabels, knnPredictions, numClasses);

cout << "\nConfusion Matrix:\n" << knnTrainConfusionMatrix << endl;

// Calculate and visualize confusion matrix for Random Forest

Mat rfConfusionMatrix = calculateConfusionMatrix(testLabels, randomPredictions, numClasses);

cout << "\nConfusion Matrix:\n" << rfConfusionMatrix << endl;

return 0;

}

**PLAGIARISM CHECK RESULT**

