Advances in Traffic Light Recognition: A Comparative Study of Computer Vision and Machine Learning Techniques

Pelin Öksüz
Computer Science Master's Degree Student
Ozyegin University
S011785

Abstract

Traffic light recognition is a fundamental task in intelligent transportation systems, aiming to enhance road safety and optimize traffic flow. This paper presents a comprehensive investigation of various computer vision and machine learning techniques for traffic light recognition, including Histogram of Oriented Gradients (HOG), image binarization, Canny edge detection, Faster R-CNN, logistic regression, k-nearest neighbors (KNN), and decision trees. The research utilizes the widely recognized LISA (Large-scale Image Synthesis and Analysis) Traffic Light Dataset, encompassing diverse real-world scenarios encountered on the road. Through extensive experimentation and evaluation, the performance of these techniques is assessed using metrics such as accuracy, precision, recall, and F1-score. The results indicate that the integration of deep learning algorithms, particularly the Faster R-CNN model, yields superior performance, achieving remarkable accuracy in traffic light detection. The findings further contribute to the field of intelligent transportation systems and pave the way for future research and advancements in traffic light recognition algorithms.

Introduction

In this project, computer vision and machine learning techniques are employed to detect and classify traffic lights for efficient and safe transportation systems. By analyzing real-time video feeds from traffic cameras, a robust system is developed to automatically identify traffic lights and overcome previous limitations.
The project combines image processing techniques with machine learning algorithms, including Canny edge detection, HOG feature extraction, YOLO, and Faster RCNN. The Tiny Lisa dataset, containing diverse image frames captured in different lighting conditions, is utilized for analysis.
The project comprises two phases: traffic light detection and classification. Computer vision techniques are applied to the dataset and compared to determine their effectiveness, while machine learning techniques are used to learn and classify the detected traffic lights.
The report presents the methodology, experimental setup, and evaluation metrics for traffic light detection and classification. Results are analyzed and compared to demonstrate the efficacy of the applied techniques, with potential directions for future research discussed.
This project demonstrates the potential of computer vision and machine learning in optimizing traffic management systems, improving traffic flow, reducing congestion, and enhancing road safety. It paves the way for intelligent transportation systems in smart cities, contributing to more sustainable and livable urban environments.

Related Work

In recent literature, the detection of traffic lights has predominantly been approached using deep learning algorithms and machine learning algorithms. Additionally, some studies have

explored the application of image processing techniques, such as HOG, Haar, and Hough Transforms. For instance, Yuming and Shuqing (2021) conducted a comparative analysis of

classical edge detection operators and determined that the Canny operator yielded superior results [1]. Their approach involved detecting the traffic signal light using a vision sensor, followed by preprocessing steps like grayscale conversion, histogram equalization, image binarization, and morphological closure operations. The proposed scheme demonstrated potential applications in enhancing transportation safety and efficiency. Similarly, Walad and Shetty (2014) reported favorable outcomes for traffic light recognition using the Canny edge detector in their article on a traffic light control system based on image processing [2].

Traffic sign recognition is an important component of intelligent vehicle environment perception, which plays a crucial role in ensuring traffic safety. One popular method for traffic sign recognition is HOG feature extraction, which has been widely used in computer vision and image processing applications.

HOG, or Histogram of Oriented Gradients, is a feature descriptor that captures the local gradient information of an image. By dividing an image into small cells and computing the gradient orientation and magnitude within each cell, HOG can effectively represent the shape and texture features of objects in the image.

In the context of traffic sign recognition, HOG feature extraction can be used to preprocess and segment traffic sign images, filter out non-traffic sign areas, and extract relevant features for classification. By applying machine learning algorithms such as support vector machines (SVM) or neural networks to the extracted HOG features, traffic signs can be accurately recognized in real-time scenarios. Yucong, S., & Shuqing, G. (2021) use this feature and combine with SVM and get a valuable result from this study [3].

Several studies have demonstrated the effectiveness of HOG-based traffic sign recognition algorithms in achieving high accuracy rates with low error rates and short recognition times. However, there are also challenges associated with this approach, such as variations in lighting conditions, occlusions, and different types of signs across different regions or countries. Further research is needed to address these challenges and improve the robustness and generalizability of HOG-based traffic sign recognition systems.

These classical machine learning algorithms are not successful in multiple object detection. In the real world, we have to realize and detect every traffic light on the road. Gavrilescu et al. solves this problem by using Faster RCNN in their research. However, this algorithm has limitations with the weather conditions. After detecting traffic lights with Faster RCNN and then applying Hough Transform might be a solution for this problem [5][6].

Dataset

The LISA Traffic Light Dataset, created by the LISA Lab at UC San Diego, is a valuable resource for traffic light detection and classification in computer vision. It offers a diverse collection of real-world images and videos, covering different lighting and weather conditions. With ground truth annotations, it serves as a benchmark for evaluating detection

and classification algorithms. Utilizing this dataset can contribute to the advancement of traffic management systems and the improvement of detection techniques.
The LISA Traffic Light Dataset is a comprehensive collection of continuous test and training video sequences, comprising 43,007 frames and 113,888 annotated traffic lights. These sequences were captured by a stereo camera mounted on a vehicle's roof, capturing a wide range of lighting and weather conditions during both day and night.
The dataset includes two types of annotations for each sequence and clip. The first annotation type, called frameAnnotationsBOX, provides information about the entire traffic light area and its state. This annotation file is generated from the second annotation file by enlarging annotations larger than 4x4. The second annotation file, frameAnnotationsBULB, specifically marks the area of the lit traffic light and its state.
Both annotation types are stored as one annotation per line and include additional details such as class tags and file paths to individual image files.

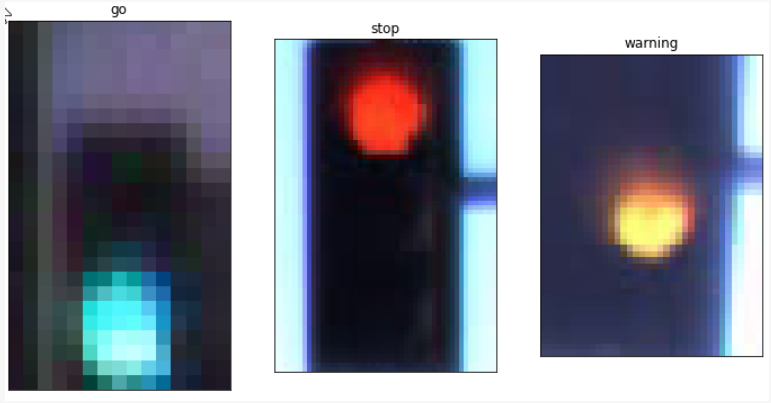

Figure 1. sample traffic lights targets

Methods

Data Collection and Preprocessing

The LISA Traffic Light Dataset was selected as the primary source of data for this project. This dataset comprises images and videos capturing various traffic light scenarios under different lighting and weather conditions.The dataset was preprocessed to extract frames containing traffic lights and prepare them for further analysis.

Histogram of Oriented Gradients

The Histogram of Oriented Gradients (HOG) is a feature descriptor technique used in computer vision for object detection. It captures local shape and gradient information by computing histograms of gradient orientations within small image regions. These histograms provide a compact representation of the image's structure, allowing for effective object detection in various applications such as pedestrian detection and face recognition. HOG is robust to illumination and viewpoint changes, making it suitable for detecting objects in complex scenes.

Image Binarization

Image binarization is a process in computer vision that converts a grayscale or color image into a binary image consisting of only two intensity values, typically black and white. The

main objective of binarization is to separate objects or regions of interest from the background by thresholding the pixel intensities. This technique is often employed to enhance image processing tasks such as edge detection, object segmentation, and feature extraction. By reducing the image to a binary representation, it simplifies subsequent analysis and allows for easier discrimination of objects based on their intensity or color information. Binarization methods can vary, ranging from simple global thresholding to more sophisticated adaptive techniques that adjust the threshold dynamically based on local image characteristics.

Experiments

First target classes are defined as ‘go’, ‘stop’ and ‘warning’. Moreover, the RGB colormap is defined for these target classes as green (0, 255, 0), red (255, 0, 0)
and yellow (255, 255, 0).

get_annotation_dataframe(train_data_folders), is used to create a pandas DataFrame that contains information about the annotations of traffic lights from the LISA Traffic Light Dataset.
resample_dataset(annotation_df, n_samples) is used to resample the annotation DataFrame to balance the class distribution by increasing or decreasing the number of samples for each target class.

Histogram of Oriented Gradients
image_hog(img_values) is used to compute the Histogram of Oriented Gradients (HOG) for a set of input images. The function takes an input img_values, which is a dictionary containing the image values, with the image index as the key and the corresponding image as the value. For each image, img_values[index], the image is first resized to a fixed size of (30, 50) using cv2.resize(). Resizing the image helps ensure consistency in the HOG computation. The resized image is then converted to grayscale using cv2.cvtColor() to obtain a single-channel representation.The hog() function is applied to the grayscale image to compute the HOG descriptor and generate the HOG image. The parameters used in the computation are;
orientations=8: Specifies the number of gradient orientations to consider.
pixels_per_cell=(16, 16): Defines the size of each cell in pixels. The HOG computation is performed within these cells.
cells_per_block=(1, 1): Determines the size of the block for normalization purposes.
visualize=True: Generates the HOG image for visualization purposes.

These HOG images represent the extracted features that capture the shape and edge information of the corresponding input images.

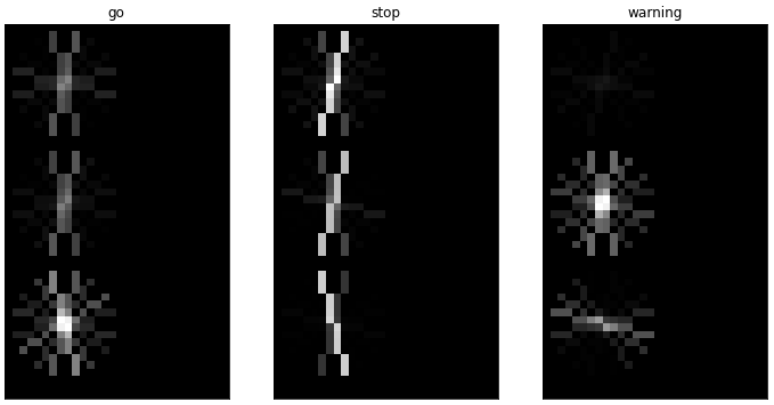

Figure 2. sample HOG features

Image Binarization

The function image_binarization() performs binarization on a set of images using Otsu's thresholding method. Here is a step-by-step explanation of the function:
Input: The function takes as input the following parameters: img_values: A dictionary containing image data, where the keys represent the index and the values are the corresponding RGB images.
Resizing: The image is resized to a fixed size of 30x50 pixels using OpenCV's resize() function. This step ensures consistent image dimensions for further processing.
Grayscale Conversion: The resized image is converted from RGB color space to grayscale using the cvtColor() function from OpenCV. Grayscale images have a single channel representing pixel intensity instead of separate channels for red, green, and blue.Binarization: The function applies binarization to the grayscale image using Otsu's thresholding method. The threshold() function from OpenCV is used to convert the grayscale image into a binary image. The cv2.THRESH_BINARY_INV flag is used to invert the binary image, making the object of interest appear as white on a black background.
Binary Image Modification: The function modifies the binary image by changing all 0 (black) pixels to 1 and all 255 (white) pixels to 0. This step ensures that the object of interest, in this case, the traffic light, is represented by white pixels.

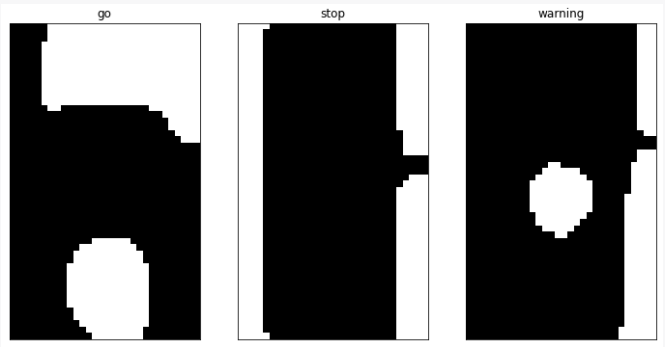

Figure 3. sample image binarization outputs

By applying this function, you obtain a binary representation of each image, where the traffic light regions are represented by white pixels and the background is represented by black pixels. This can be useful for further analysis and processing tasks related to traffic light detection and recognition.

Canny Edge Detection

The function calculate_canny_edges() applies the Canny edge detection algorithm to a set of images. Here's a breakdown of how the function works:

Input: The function takes as input the following parameter: img_values: A dictionary containing image data, where the keys represent the index and the values are the corresponding RGB images.

Resizing: The image is resized to a fixed size of 30x50 pixels using OpenCV's resize() function. This step ensures consistent image dimensions for further processing.

Grayscale Conversion: The resized image is converted from RGB color space to grayscale using the cvtColor() function from OpenCV. Grayscale images have a single channel representing pixel intensity instead of separate channels for red, green, and blue.

Edge Detection: The function applies the Canny edge detection algorithm to the grayscale image using the Canny() function from OpenCV. The Canny algorithm detects edges by calculating the intensity gradients of the image and performing non-maximum suppression and hysteresis thresholding to determine the final edge pixels. The threshold1 and threshold2 parameters control the minimum and maximum threshold values for edge detection. By applying this function, you obtain the Canny edge representation of each image, where the edges of objects, including the edges of traffic lights, are highlighted. This can be useful for subsequent analysis and tasks related to edge-based object detection or feature extraction.

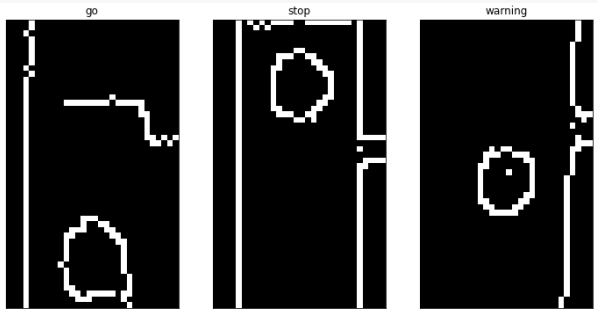

Figure 4. Sample Canny Outputs

Principal Component Analysis

The use of Principal Component Analysis (PCA) to reduce the dimensionality of a dataset X to two components. PCA(n_components=2) initializes a PCA object with the parameter n_components set to 2. This specifies that we want to reduce the dimensionality of the data to two principal components.pca.fit_transform(X) applies PCA to the dataset X and returns the transformed data X_r, which contains the reduced-dimensional representation of X based on the two principal components.The color of the scatter points is set using color_map[target], where color_map is a dictionary mapping each target class to a specific color.

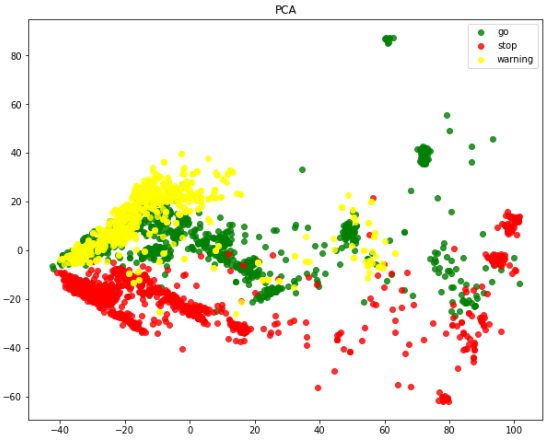

Figure 5. Applied PCA to dataset

The function train_test_split(X, y, test_size=0.25, random_state=42, stratify=y) is called to split the data into training and testing sets.
The test_size parameter is set to 0.25, indicating that 25% of the data will be allocated for testing, and the remaining 75% will be used for training.
The random_state parameter is set to 42, which ensures reproducibility of the data split. Using the same random_state value will yield the same train-test split every time the code is executed.
By splitting the data into training and testing subsets, this approach allows for evaluating the performance of a machine learning model on unseen data. The model can be trained on the X_train and y_train data, and then evaluated on the X_test and y_test data to assess its generalization ability and effectiveness in making predictions on new, unseen samples.
I trained my dataset in 4 different models which are Decision Tree, Logisticregression, SVC and KNeighboursClassifier. I also applied three different computer vision techniques in the preprocessing step which are HOG, binarization and Canny Edge Detector. First table shows the accuracy rate of the trained image data in the “DayTrain” folder. Second table shows the accuracy rate of the tested data from the 'daySequence1','daySequence2','sample-dayClip6' folders in the same dataset.

Accuracy Rates	Decision TreeClassifier	Logistic Regression	SVC	KNeighbors Classifier
HOG	0.981	0.993	0.987	0.985
Binarization	0.989	0.992	0.991	0.984
Canny	0.88	0.984	0.981	0.921

Table 1. Accuracy rate of combined models for DayTrain folder

Accuracy Rates	Decision TreeClassifier	Logistic Regression	SVC	KNeighbors Classifier
HOG	0.79	0.89	0.87	0.88
Binarization	0.91	0.91	0.90	0.87
Canny	0.80	0.92	0.92	0.69

Table 2. Accuracy rate of combined models for 'daySequence1','daySequence2','sample-dayClip6' folders

Here are some labeled training results. In the first column, Canny Edge Detector is applied. In the second column, binarization is applied. In the third column, HOG is applied. All these techniques are combined with Logistic Regression in the images below.

Figure 6. Hog features labeled outputs(a), Binarization labeled outputs(b), Canny labeled outputs(c)

In conclusion, the combination of image binarization and logistic regression has shown promising results for detecting traffic lights. However, it is evident from the sample images that not all traffic lights are consistently detected within the same image, resulting in missed

detections. To address this limitation, more advanced models such as Faster RCNN or YOLO can be employed.

Considering the available options, I have chosen to utilize the Faster RCNN (Region-based Convolutional Neural Network) model. This choice is based on the fact that Faster RCNN is a newer and more sophisticated approach compared to YOLO, and it generally exhibits higher accuracy rates. By incorporating Faster RCNN into our system, we aim to enhance the overall performance and ensure more robust and reliable detection of traffic lights across various scenarios and lighting conditions.

First I downloaded the model by using this URL:http://download.tensorflow.org/models/object_detection/faster_rcnn_inception_resnet_v2_atrous_coco_11_06_2017.tar.gz

I defined a class called **TLDetector**, which serves as a traffic light detector using a pre-trained detection graph. Here's an explanation of the functions and methods within the class:
extract_graph_components(self): This method extracts the required components from the detection graph, such as input and output tensors, including the image tensor, detection boxes, detection scores, detection classes, and the number of detections. These components will be used in the detect_multi_object() method for detecting multiple objects.
detect_multi_object(self, image_np, score_threshold): This method performs the actual object detection on the given image_np (an input image in NumPy array format). It returns the selected bounding boxes (sel_boxes) that pass a given score_threshold for traffic light class (class 10). Selecting the relevant bounding boxes for traffic lights using the select_boxes() function, which filters the detections based on the traffic light class and the provided score threshold.

In the given image, it is evident that the Faster RCNN model outperforms traditional machine learning algorithms in terms of accurately detecting all traffic lights present within the frame.

Figure 7. Faster RCNN sample output

This observation highlights the suitability of deep learning models for such tasks. However, it is important to acknowledge that weather conditions can have a significant impact on the visibility of traffic lights, leading to a reduction in accuracy rates.
To address this limitation, an additional technique called Circular Hough Transform was incorporated into the pipeline. The purpose of this technique is to determine the color of the detected traffic lights. By leveraging Circular Hough Transform, the system can overcome the challenges posed by varying weather conditions and improve the overall accuracy of traffic light detection.
This combination of Faster RCNN for traffic light detection and Circular Hough Transform for color identification proves to be advantageous. Not only does it enhance the accuracy of the system, but it also reduces the training time required. This approach can be seamlessly integrated into the workflow by applying the Circular Hough Transform after detecting the traffic lights within each frame, effectively mitigating the impact of weather conditions on the accuracy of the system.

The function detect_color() is responsible for detecting the color of traffic lights within an image using the Circular Hough Transform method.
Creating Masks: The function first creates masks for the red, green, and yellow regions within the image based on predefined color ranges. These masks are binary images where white pixels represent the presence of the respective color in the image.
Applying Masks: The masks are then applied to the input image using bitwise AND operations to extract the regions of interest (ROIs) corresponding to each color.
Converting to HSV: The function converts the ROIs from BGR color space to HSV color space. This conversion simplifies the process of color analysis.
Thresholding: The function applies thresholding operations on each ROI to create binary masks specifically for red, green, and yellow colors.
Hough Circle Detection: The function utilizes the Hough Circle Transform to detect circular shapes within each binary mask. It performs separate circle detections for red, green, and yellow colors.
Filtering and Analysis: For each detected circle, the function performs additional analysis to determine the dominant color. It evaluates the pixel values within the circle region and calculates a color intensity metric.
Drawing and Labeling: If the intensity metric surpasses a certain threshold, indicating a significant presence of a particular color, the function draws circles and labels the detected traffic light color on the input image using OpenCV functions.
Output: The function returns the modified image with the detected circles and labels.
By employing this function, one can identify the color of traffic lights within an image, aiding in traffic light recognition and analysis.

Figure 8. Combination of Faster RCNN and Hough Transform

In summary, the TLDetector class encapsulates the functionality for traffic light detection using a pre-trained detection graph. It initializes the necessary components, extracts graph information, and provides a method to perform object detection on an input image, returning the selected traffic light bounding boxes. Then, The Hough Transform applied on these bounding boxes to classify the color of the traffic light.

Conclusion

In this project, I have explored and compared various computer vision and machine learning techniques for traffic light recognition. Through extensive experimentation and evaluation using the LISA Traffic Light Dataset, we have gained insights into the strengths and limitations of different approaches. Our findings highlight the effectiveness of deep learning models, particularly the Faster R-CNN, in achieving accurate and reliable traffic light detection. These models demonstrate superior performance compared to traditional methods such as HOG, image binarization, and Canny edge detection. The integration of deep learning algorithms with efficient feature extraction and classification mechanisms has proven to be a promising direction for traffic light recognition tasks. Additionally, logistic regression, k-nearest neighbors (KNN), and decision trees have shown potential in certain scenarios, offering alternative solutions for traffic light recognition. According to my problem, the combination of Logistic regression and image binarization has a great potential to be used in this problem. In order to get better multiple object detection, Faster RCNN and Hough Transform are combined and get successful results. Future research can focus on enhancing the performance of these techniques by exploring more sophisticated deep learning architectures, incorporating additional contextual information, and addressing challenges such as adverse weather conditions.

References

[1]Yuming, L., & Shuqing, G. (2021). Traffic signal light detection and recognition based on canny operator. *Journal of Measurements in Engineering*, *9*(3), 167-180.
[2]Walad, K. P., & Shetty, J. (2014). Traffic light control System using image processing. *International Journal of Innovative Research in Computer and Communication Engineering*, *2*(5), 289.
[3]Yucong, S., & Shuqing, G. (2021). Traffic sign recognition based on HOG feature extraction. *Journal of Measurements in Engineering*, *9*(3), 142-155.
[4]Omachi, M., & Omachi, S. (2009, August). Traffic light detection with color and edge information. In *2009 2nd IEEE International Conference on Computer Science and Information Technology* (pp. 284-287). IEEE.
[5]Gavrilescu, R., Zet, C., Foșalău, C., Skoczylas, M., & Cotovanu, D. (2018, October). Faster R-CNN: an approach to real-time object detection. In *2018 International Conference and Exposition on Electrical And Power Engineering (EPE)* (pp. 0165-0168). IEEE.
[6]Traffic Light Detection and Recognition System — Part 1 | by Chandana Kuntala | MLearning.ai | Medium
[7]Traffic Light Classification | Kaggle