Xi'an Jiaotong-Liverpool University

西交利物浦大學

License Plate Recognition

By

Jiamin Jiang (2143135)

supervised by

Xiaobo Jin

SAT301 FINAL YEAR PROJECT

2025-05-06

Abstract

License plate recognition (LPR) plays a crucial role in intelligent transportation systems and urban management. In this final-year project report, I focus on building an efficient and lightweight LPR system based on traditional machine learning techniques. Specifically, the following work was carried out:
1. To eliminate noise and enhance key features, preprocess the image to achieve standardized input data and improve the stability of subsequent steps;
2. In order to accurately extract the license plate area from the responsible background, it is necessary to perform frame selection and rotation to reduce the calculation range and avoid irrelevant interference;
3. In order to provide accurate single character input for character recognition, the characters in the license plate are independently separated;
4. Use SVM model to recognize segmented characters;
Using a self-built image dataset for evaluation and the results showed that the accuracy of the system was 88.1 percent. The complete recognition of license plates under standard conditions has demonstrated the feasibility of using lightweight models for real-world license plate recognition tasks. In future work, the goal is to introduce deep learning methods such as Convolutional Neural Networks (CNN) and Transformer-based architectures to further improve recognition accuracy and enhance robustness against complex backgrounds, perspective distortions, and occlusions.

Keywords

Keywords: **license Plate Recognition, SVM, Machine Learning, OpenCV, Character Recognition**

Acknowledgements

I would like to thank my mentor, Dr. Xiaobo Jin, who provided me with a lot of help and support in terms of knowledge and methods in my graduation project. In addition, he also provided me with many useful suggestions and important ideas to help me complete my design. The article he presided over was of great significance to me in determining the research direction from the beginning. His suggestions helped me identify many useful methods and it was because of these that I was able to successfully complete my graduation project. In addition, I improved the evaluation of the recognition system to ensure a clear understanding of the project's implementation results under his suggestion.

I would also like to thank some of my friends because they gave me a lot of support and encouragement. When I was in trouble, they encouraged me with a lot of warmth and helped me overcome a series of difficulties.

Finally, I am very grateful to my family for always accompanying me and encouraging me, which is a very important part of my study journey. Their support and understanding are the key to my progress and persistence. I would like to express my sincere gratitude to everyone who has supported and helped me in this journey.

Contents

1 Introduction

1.1 Motivation, Aims and Objective Motivation

License Plate Recognition (LPR) technology plays a critical role in a wide range of applications such as traffic management, parking control and urban surveillance [1]. With the rapid development of intelligent transportation systems, the demand for lightweight, fast and accurate LPR solutions has increased significantly. Traditional LPR methods based on edge detection or template matching often suffer from poor robustness under varying environmental conditions, such as illumination changes, perspective distortion and background clutter [2]. Although deep learning-based methods have shown promising performance, they often require extensive computational resources and large-scale annotated datasets, which are not always feasible for lightweight deployment scenarios [3]. Therefore, there is a need to design a system that balances performance and efficiency, particularly for smaller datasets and real-time applications.

In this project, I explored a machine learning-based approach using Support Vector Machines (SVM) to achieve effective character recognition from license plates. Our system is designed to be simple, modular and extendable, making it suitable for both academic study and real-world applications.

The aim of this project is to develop a reliable and efficient license plate recognition system based on traditional machine learning techniques. The system should be able to recognize full license plates accurately under standard environmental conditions.

The specific objectives are as follows:

- Implement a preprocessing pipeline, including grayscale conversion and multiscale detection, to enhance plate localization performance.
- Adopt a multi feature fusion localization method, combined with contour detection and HSV color analysis, to accurately extract license plate areas in complex backgrounds.
- Develop an adaptive character segmentation algorithm based on histogram peak analysis and projection techniques to achieve high-precision single character separation.
- Build an efficient character recognition system that integrates SVM classifier and HOG feature extraction to accurately output readable license plate text information.
- Incorporate plate color recognition, distinguishing between blue, green, and yellow plates, to provide additional information.
- Develop a user-friendly graphical interface (GUI) using Tkinter, allowing users to easily upload images and view recognition results.

- Construct a labeled data set and evaluate system performance using metrics such as accuracy, precision, recall and F1 score.
- Analyze the recognition results, identify limitations, and propose directions for future improvements based on the experimental findings.

1.2 Literature Review

License Plate Recognition (LPR) has been a major research topic in the fields of computer vision and intelligent transportation systems for several decades. Generally, there are three main approaches to LPR: traditional methods based on image processing, deep learning-based methods, and end-to-end integrated frameworks [4].

1.2.1 The traditional method

Traditional license plate recognition (LPR) systems are based on classical computer vision techniques, utilizing primarily image processing algorithms provided by OpenCV [5]. These approaches typically follow a multistage pipeline, beginning with preprocessing steps such as Gaussian blurring, grayscale conversion, and morphological operations to enhance image quality. Edge detection techniques, including the Sobel and Canny operators, are then applied alongside HSV color space analysis to localize the license plate region [6]. Subsequent steps involve projection-based character segmentation and recognition using template matching or Support Vector Machine (SVM) classifiers [7].

The main advantages of such methods lie in their algorithmic transparency, low computational demands, and ease of deployment on low-power devices [8]. Without the need for large-scale datasets required by deep learning, development cycles are relatively short, making these systems well-suited for budget-constrained or latency-sensitive applications. However, their performance is often compromised under challenging conditions, such as varying illumination, complex backgrounds, or skewed plate angles [2]. As a result, robustness in unconstrained environments remains limited.

This project adopts a traditional approach combined with SVM for character recognition. More technical details regarding the implementation of methods and systems will be provided in the methodology section below.

1.2.2 The deep learning-based approach

With the development of deep learning, modern license plate recognition (LPR) systems increasingly adopt convolutional neural networks (CNNs) and object detection frameworks such as

YOLO (You Only Look Once), Faster R-CNN, and SSD (Single Shot MultiBox Detector). These methods generally treat license plate detection and character recognition as supervised learning tasks, offering greater flexibility and accuracy compared to traditional techniques.

A common architecture divides the LPR task into two separate stages: detection and recognition [9]. In the first stage, object detection models such as YOLO or Faster R-CNN are employed to localize license plates in an image, leveraging anchor box mechanisms to handle plates of varying sizes. In the second stage, CNN or CRNN-based models are used for character recognition, benefited from the strong feature extraction capabilities of deep networks. This modular structure allows independent optimization of each component and significantly enhances the recognition accuracy. The experimental results of calculating the CCPD dataset show that the average accuracy of this method reaches 99.4% [10]. Liu proposed that this algorithm significantly improves accuracy, averaging 7.7% in complex scenarios on the CCPD dataset [11].

Deep learning-based approaches demonstrate strong robustness in complex scenes, such as occlusions, blurring, skewed angles, and dynamic lighting conditions [12]. However, these systems require extensive labeled datasets for training, substantial computational power (especially GPU acceleration), and careful tuning of model parameters. Despite these demands, such architectures have been widely adopted in high-accuracy applications, including highway ETC systems and urban traffic enforcement. Nonetheless, the two-stage design introduces additional inference latency, requiring a balance between model accuracy and real-time performance.

1.2.3 An end-to-end integrated approach

End-to-end integrated approaches represent the most advanced direction in license plate recognition (LPR) technology. These methods consolidate the entire recognition pipeline from raw image input to final character output, within a single deep learning model. Representative architectures include LPRNet, YOLOv7 combined with CRNN, and recent implementations based on Vision Transformers [13].

Unlike traditional two-stage models, end-to-end systems eliminate the need for separate detection and recognition phases. By utilizing unified feature learning and joint optimization, these models reduce the risk of cross stage error accumulation. Architecturally, they often combine convolutional neural networks (CNNs) [14], recurrent neural networks (RNNs) and attention mechanisms to process the license plate as a character sequence without explicit segmentation. Transformer-based designs further enhance global context awareness, improving performance in cases of occlusion, distortion and visually complex backgrounds [15].

The primary advantages of end-to-end models include streamlined processing, minimal manual feature design and superior robustness under challenging conditions. The proposed method has the accuracy of 100% for vehicle detection, 100% for plate detection, and 99.37% for character recognition [16]. However, these systems require significant computational power, particularly high-end GPU resources, and involve models with millions of parameters. As such, deployment is most feasible in scenarios with substantial infrastructure support, such as cloud-based LPR services or perception systems in autonomous vehicles. This approach marks a key trend in the future development of license plate recognition technologies.

1.2.4 Comparison of SVM and Deep Learning Model Performance

To further verify the rationality of the selected approach, this study refers to representative research in the field of license plate recognition (LPR) in recent years. A horizontal comparison is conducted between traditional methods (e.g., SVM with HOG features) and deep learning-based methods (e.g., LPRNet, YOLO-LPR). According to the literature, traditional approaches relying on handcrafted features have certain recognition capabilities in controlled environments but show significantly degraded performance under challenging conditions such as varying lighting, camera angles, or occlusion. For instance, a classic method using SVM + HOG achieves only 84.7% recognition accuracy on the CCPD dataset, while LPRNet exceeds 95%, and YOLO-LPR outperforms both in terms of detection-recognition accuracy and speed [17, 18, 19].

Table 1: Comparison of typical license plate recognition methods on the CCPD dataset

Method	Feature Type	Model Architecture	Accuracy	End-to-End	Real-Time
SVM + HOG	Handcrafted	Binary Classifier	84.7%	No	Medium
LPRNet	Deep Features	CNN	95.3%	Yes	High
YOLO-LPR	Deep Features	YOLOv4 + CNN	96.8%	Yes	High

In addition to performance metrics, it is important to consider the hardware requirements of each method. Traditional methods like SVM + HOG can run efficiently on standard CPU environments, making them suitable for resource-constrained or embedded applications. In contrast, deep learning models such as LPRNet and YOLO-LPR generally require high-performance hardware (e.g., GPU or AI accelerators) to support real-time, end-to-end inference, especially when processing video streams.

Table 2: Comparison of hardware requirements and inference performance

Method	Inference Time	Memory Usage	Hardware Platform	Reference
SVM + HOG	80 ms	$<$ 100 MB	CPU (i5-8400)	[17]
LPRNet	22 ms	$\sim$300 MB	GPU (GTX 1080 Ti)	[18]
YOLO-LPR	15 ms	$\sim$450 MB	GPU (RTX 2080)	[19]

From the comparison, it is evident that deep learning approaches offer significant advantages in terms of accuracy and robustness. However, their deployment requires careful consideration of hardware conditions. In practical systems, model compression and acceleration techniques (e.g., TensorRT, ONNX) can help adapt high-performance models to edge devices with limited resources.

2 Methodology and Results

License plate recognition (LPR) involves preprocessing, localization, segmentation, and recognition of characters from vehicle license plates. In this project, I implement a traditional, lightweight pipeline combining OpenCV-based image processing techniques with machine learning classifiers. This design ensures high explainability, low computational requirements, and modular expandability. Our goal is to maximize character recognition accuracy with minimal hardware dependency while preserving step-by-step controllability in the processing flow.

2.1 Support Vector Machine

2.1.1 Principle

Support Vector Machine (SVM) is a supervised learning method based on statistical learning theory, widely applied in classification, regression, and anomaly detection tasks [20]. It is particularly effective in scenarios involving small sample sizes and high-dimensional data. The core idea of SVM is to construct an optimal separating hyperplane that maximizes the margin between different classes, thereby enhancing the generalization ability of the model [21].

For a binary classification problem with a training set $(x_i, y_i)_{i=1}^n$, where $x_i \in \mathbb{R}^d$ and $y_i \in -1, +1$, the SVM aims to find a hyperplane defined by $w^T x + b = 0$ such that the following condition holds for all training samples:

$$y_i(w^T x_i + b) \geq 1, \quad \forall i = 1, \ldots, n \tag{1}$$

The margin between the two classes is defined as $\frac{2}{|w|}$, and maximizing this margin leads to the following convex quadratic optimization problem:

$$\min_{w,b} \quad \frac{1}{2}|w|^2 \quad \text{subject to} \quad y_i(w^T x_i + b) \geq 1 \tag{2}$$

This formulation is known as the hard-margin SVM, applicable to linearly separable datasets. To extend the model to handle linearly non-separable cases, slack variables $\xi_i \geq 0$ are introduced to allow misclassification. The soft-margin SVM problem then becomes:

$$\min_{w,b,\xi} \quad \frac{1}{2}|w|^2 + C\sum_{i=1}^{n} \xi_i \quad \text{subject to} \quad y_i(w^T x_i + b) \geq 1 - \xi_i \tag{3}$$

where $C > 0$ is the penalty parameter that controls the trade-off between maximizing the margin and minimizing the classification error [21].

To solve the above problem efficiently, SVM transforms it into its dual form using Lagrange multipliers α_i, resulting in the following objective:

$$\max_{\alpha} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j x_i^T x_j \tag{4}$$

subject to the constraints:

$$\sum_{i=1}^{n} \alpha_i y_i = 0, \quad 0 \leq \alpha_i \leq C, \quad \forall i \tag{5}$$

After solving for the optimal α_i, the final decision function is expressed as:

$$f(x) = \sum_{i=1}^{n} \alpha_i y_i x_i^T x + b \tag{6}$$

In practice, only a subset of the training samples with non-zero α_i values—termed support vectors—contribute to the decision function, significantly reducing computational complexity and improving interpretability [21].

For cases where the data is not linearly separable in the original input space, SVM employs the kernel trick. The idea is to implicitly map the input vectors into a higher-dimensional feature space $\phi(x)$, where linear separation may become possible, without explicitly computing $\phi(x)$. The dot product $x_i^T x_j$ in the dual form is replaced by a kernel function $K(x_i, x_j) = \phi(x_i)^T \phi(x_j)$. Commonly used kernel functions include:

Linear kernel: $K(x_i, x_j) = x_i^T x_j$

Polynomial kernel: $K(x_i, x_j) = (x_i^T x_j + c)^d$

Radial Basis Function (RBF): $K(x_i, x_j) = \exp(-\gamma |x_i - x_j|^2)$

Sigmoid kernel: $K(x_i, x_j) = \tanh(\kappa x_i^T x_j + \theta)$

Using kernels, the decision function generalizes to:

$$f(x) = \sum_{i=1}^{n} \alpha_i y_i K(x_i, x) + b \tag{7}$$

Hyperparameter tuning—such as selecting the kernel type, regularization parameter C, and kernel-specific parameters like γ—is essential for optimal performance. Techniques such as k-fold cross-validation are commonly applied to avoid overfitting and improve generalization.

Furthermore, since SVM is inherently a binary classifier, it must be extended to handle multiclass classification tasks. Two widely adopted strategies are one-vs-one (OvO) and one-vs-all (OvA), where multiple binary classifiers are trained and combined to produce the final multiclass decision [22].

2.1.2 Support Vector Machine in Character Recognition

Within license plate recognition (LPR) systems, Support Vector Machine (SVM) is often utilized in the *character recognition* stage. A complete LPR system typically comprises image preprocessing, license plate localization, character segmentation, and character recognition. The earlier stages generally rely on image processing techniques such as edge detection, morphological operations, or deep learning-based methods like YOLO, while character recognition is better suited for machine learning models. At this stage, SVM is responsible for classifying each segmented character image into its corresponding alphanumeric category [23].

In the character recognition phase, the task is to assign each segmented character image $x \in \mathbb{R}^d$ to one of several predefined classes (e.g., digits 0–9, letters A–Z, and certain Chinese characters), making this a *multiclass classification problem.* To adapt SVM for such problems, strategies like *one-vs-one* and *one-vs-all* are commonly used [22]. Each binary SVM classifier is trained to separate one class from the others, and final classification is performed based on voting or maximal decision function value.

Before classification, *feature extraction* is conducted on each character image. Typical features include raw pixel intensity values, *histograms of oriented gradients (HOG)*, and *local binary patterns (LBP).* Denoting a feature vector as $x \in \mathbb{R}^d$, and its corresponding label as $y \in \{-1, +1\}$, the SVM learns a decision function that takes the form:

$$f(x) = \operatorname{sign}\left(\sum_{i=1}^{n} \alpha_i y_i K(x_i, x) + b\right) \tag{8}$$

where α_i are the *Lagrange multipliers*, x_i are the support vectors from training data, y_i are the associated labels, b is the bias term, and $K(x_i, x)$ is the *kernel function* [21]. A commonly used kernel in character recognition is the *radial basis function (RBF)* kernel:

$$K(x_i, x_j) = \exp(-\gamma \|x_i - x_j\|^2) \tag{9}$$

which effectively maps input data into an infinite-dimensional space where linear separation becomes feasible [24]. The use of this *kernel trick* enables SVM to handle nonlinear classification problems without explicitly computing the high-dimensional mapping.

For multiclass recognition using one-vs-one or one-vs-all strategies, the classifier computes multiple decision functions:

$$f_k(x) = \sum_{i=1}^{n_k} \alpha_i^{(k)} y_i^{(k)} K(x_i^{(k)}, x) + b^{(k)} \tag{10}$$

and the predicted class label is determined by:

$$\hat{y} = \arg\max_k f_k(x) \tag{11}$$

Experimental results indicate that, compared to traditional template matching methods, SVM-based approaches exhibit stronger robustness to noise, font variations, and partial occlusions [23]. Moreover, SVMs are well-suited for character recognition tasks where labeled data is limited, thanks to their good generalization performance even with small training sets [25].

In certain practical applications, SVM is integrated with other methods, such as using *convolutional neural networks (CNNs)* to extract deep features $\phi(x)$, which are then classified using SVM. This *hybrid architecture* enhances accuracy while maintaining relatively low model complexity and training cost:

$$f(x) = \sum_{i=1}^{n} \alpha_i y_i K(\phi(x_i), \phi(x)) + b \tag{12}$$

This combination leverages the representational power of CNNs and the decision efficiency of SVMs, offering a robust solution particularly in real-world LPR systems under resource constraints.

In conclusion, due to its solid theoretical foundation, strong classification performance, and adaptability to high-dimensional and small-sample data, SVM plays a significant role in license plate recognition systems, particularly in the character recognition phase. Although deep learning models such as CNNs and Transformers have recently achieved leading performance in this field, SVM remains an efficient and reliable solution in resource-constrained scenarios or applications requiring high model explainability.

2.1.3 SVM Parameter Settings and Impact Analysis

In the training process of Support Vector Machines (SVM), two key hyper parameters play a decisive role in balancing model fitting and generalization, the penalty coefficient C and the kernel parameter γ. A larger value of C decreases the tolerance for training errors, leading the model to fit the data more closely and increasing the risk of overfitting. Conversely, a smaller C allows more classification errors, resulting in a smoother decision boundary but potentially causing underfitting.

For the Radial Basis Function (RBF) kernel, γ controls the influence range of each support vector. A larger γ leads to a more complex model that may capture noise in the data, while a smaller γ can overlook important local patterns.

To strike a balance between underfitting and overfitting, a grid search combined with cross-validation was employed. The parameter ranges were set as $C \in \{0.1, 1, 10, 100, 1000\}$ and $\gamma \in \{0.0001, 0.001, 0.01, 0.1, 1\}$. The optimal combination was found to be $C = 10$ and $\gamma = 0.01$, which yielded a character recognition accuracy of approximately 88.5%.

2.1.4 Model Training and Evaluation

The license plate recognition system is constructed using traditional machine learning methods, with the core component being a character recognition model based on Support Vector Machine (SVM). Model training is divided into two parts: one for recognizing Chinese province abbreviations and the other for recognizing alphanumeric characters. The training images are sourced from the `train/charsChinese/` and `train/chars2/` directories, respectively.

All training images undergo geometric correction and grayscale conversion before feature extraction. Histogram of Oriented Gradients (HOG) features are extracted and used as input to the SVM classifier. The SVM model employs the Radial Basis Function (RBF) kernel to perform nonlinear classification. Upon completion of the training process, the models are saved as `svmchinese.dat` and `svm.dat`.

During the evaluation stage, the system reads prediction results from a CSV file and calculates overall and subtype (i.e., Chinese characters and alphanumeric strings) performance metrics. These include accuracy, precision, recall, and F1-score, computed based on true positives (TP), false positives (FP), and false negatives (FN). Trend and statistical outcomes are visualized using graphical charts.

This training and evaluation pipeline integrates lightweight image feature extraction with an efficient classifier design, enabling the system to maintain robust recognition performance under limited data conditions.

2.2 Implementation Details

2.2.1 Image Preprocessing

Gaussian Blur

Principle: Gaussian Blur is an image blurring technique based on the Gaussian function. It works by performing a convolution operation, where each pixel is replaced by the weighted average of its neighboring pixels [26]. The weights are determined by the Gaussian distribution, which has a bell-shaped curve. The central pixel has the highest weight, and the weights decrease as the distance from the center increases. This process smooths the image, effectively reducing noise and fine details, resulting in a blurred effect.

In the system: The function `cv2.GaussianBlur(image, (blur_size, blur_size), 0)` is used to smooth the image before Canny edge detection to reduce the effect of noise on subsequent edge extraction.

Listing 1: Gaussian Blur

```
blur_size = self.current_config["blur_size"]
if blur_size > 0:
    image = cv2.GaussianBlur(image, (blur_size, blur_size), 0)
```

Morphological Operations

Principle: Morphological operations are a set of image processing techniques that process images based on their shapes or structures. These operations primarily work on binary images, applying a structuring element to manipulate the image's shape [27]. Common morphological operations include dilation (expanding boundaries), erosion (shrinking boundaries), opening (erosion followed by dilation), and closing (dilation followed by erosion). These operations are used to remove noise, fill gaps, and enhance or suppress specific image features, making them particularly useful for tasks like object detection, segmentation, and feature extraction.

In the system: The operation `cv2.morphologyEx(gray_image, cv2.MORPH_OPEN, kernel)` is applied to remove noise, and `addWeighted` is used afterward to enhance the contrast of the license plate region for better binarization and edge extraction.

Listing 2: Morphological Operations

```
kernel = np.ones((20, 20), np.uint8)
opened_image = cv2.morphologyEx(gray_image, cv2.MORPH_OPEN, kernel)
opened_image = cv2.addWeighted(gray_image, 1, opened_image, -1, 0)
```

Grayscale Conversion

Principle: Grayscale conversion is a process of transforming a color image into an image that only contains shades of gray, removing all color information. This is typically achieved by calculating the luminance or intensity of each pixel based on its color components (red, green, and blue). A common approach is to use a weighted sum of the RGB channels, where the green channel has the highest weight due to its perceived brightness contribution to human vision [28]. The resulting grayscale image reflects the brightness variations in the original image, allowing for simpler analysis while maintaining essential visual information.

In the system: `cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)` is used to convert the input image to grayscale, serving as input for morphological processing, Canny edge detection, and character extraction.

Listing 3: Grayscale Conversion

```
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
```

Canny Edge Detection *Principle*: Canny Edge Detection is a multi-step algorithm used to detect edges in an image. The method involves several key stages:

1. **Noise Reduction**: The image is first smoothed using a Gaussian filter to reduce noise, which could interfere with edge detection. This step helps to prevent false edge detection caused by small variations in pixel values.

2. **Gradient Calculation**: The gradient of the image is computed using convolution with derivative filters (typically the Sobel operator) in both horizontal and vertical directions. This step helps identify regions of rapid intensity change, which are potential edges.

3. **Non-Maximum Suppression**: After the gradient calculation, the algorithm performs non-maximum suppression to thin out the edges. This step removes pixels that are not part of the edge by comparing the gradient magnitude with neighboring pixels along the gradient direction.

4. **Edge Tracing by Hysteresis**: Finally, the algorithm uses two thresholds to decide which edges are strong enough to be considered valid. Pixels with gradient magnitudes above the high threshold are marked as strong edges, while those below the low threshold are marked as weak edges. Weak edges that are connected to strong edges are retained, while others are discarded.

Canny Edge Detection is widely used in computer vision tasks due to its ability to detect clear and precise edges while minimizing the impact of noise [29].

In the system: `cv2.Canny(threshold_image, 100, 200)` extracts edge contours from the license plate image for further localization.

Listing 4: Canny Edge Detection

```
edge_image = cv2.Canny(threshold_image, 100, 200)
```

2.2.2 License Plate Localization

Contour Detection

Principle: Contour detection is an essential technique in image processing that focuses on identifying the boundaries of objects within an image. It typically involves computing the gradient of the image to highlight regions of rapid intensity change, followed by applying edge detection algorithms like the Canny edge detector. The detected edges are then linked to form continuous contours, which represent the object boundaries. This process is crucial for tasks like object recognition and image segmentation, allowing for better shape analysis and feature extraction [30].

In the system: Using `cv2.findContours()` to obtain all edge regions, and selecting rectangles with an area greater than `MIN_PLATE_AREA` and an aspect ratio between 2 and 5.5 as candidates.

Listing 5: Contour Detection

```python
contours, _ = cv2.findContours(opened_edges, cv2.RETR_TREE, cv2.
    CHAIN_APPROX_SIMPLE)
contours = [cnt for cnt in contours if cv2.contourArea(cnt) >
    MIN_PLATE_AREA]
```

Color Space Analysis

Principle: Color space analysis involves representing colors in a structured format for various image processing tasks. Common color spaces include RGB (Red, Green, Blue), which is used for display systems, HSV (Hue, Saturation, Value), which is useful for color-based segmentation, and CIELAB, which aims to model human vision. Different color spaces are chosen depending on the specific application, with conversions between them enabling more efficient image manipulation and analysis [31].

In the system: `cv2.cvtColor(plate_img, cv2.COLOR_BGR2HSV)` converts the image, then the hue histogram is analyzed to determine the color type of the plate (blue, yellow, green), guiding color feature extraction.

Listing 6: Color Space Analysis

```python
plate_hsv = cv2.cvtColor(plate_img, cv2.COLOR_BGR2HSV)
for row in range(rows):
    for col in range(cols):
        H = plate_hsv.item(row, col, 0)
        S = plate_hsv.item(row, col, 1)
        V = plate_hsv.item(row, col, 2)
        if 11 < H <= 34 and S > 34:
            yellow += 1
        elif 35 < H <= 99 and S > 34:
            green += 1
        elif 99 < H <= 124 and S > 34:
            blue += 1
```

Rotation Correction

Principle: Rotation correction is a crucial process in image processing aimed at aligning images that are tilted or rotated due to camera angle or other factors. The process typically involves detecting the angle of rotation, often using techniques such as the Hough Transform, which identifies the dominant orientation of edges or lines in the image. Once the angle is determined, the image is rotated back to its correct orientation using geometric transformations like affine or perspective transforms. This step is important for tasks such as document scanning, license plate recognition, and object tracking, where accurate alignment is essential for further analysis [32].

In the system: Using `cv2.getAffineTransform()` and `cv2.warpAffine()` with three corner points to transform the plate to a horizontally aligned image.

Listing 7: Rotation Correction

```
angle = 1 if -1 < rect[2] < 1 else rect[2]
rect = (rect[0], (rect[1][0] + 5, rect[1][1] + 5), angle)
box = cv2.boxPoints(rect)
M = cv2.getAffineTransform(src_points, dst_points)
warped = cv2.warpAffine(original_image, M, (width, height))
```

2.2.3 Character Segmentation

Histogram-based Wave Detection *Principle*: Histogram-based wave detection is a technique used in image processing to identify patterns or waves in images based on the intensity distribution of pixel values. The approach involves analyzing the image's histogram to detect periodic changes or variations in intensity that correspond to wave-like patterns. By applying methods like Fourier Transform or other statistical techniques, the image's frequency components can be analyzed to detect these waves. This method is particularly useful in fields like medical imaging, texture analysis, and object detection, where understanding periodic structures is crucial for accurate interpretation.

In the system: Uses `np.sum(binary_plate, axis=1)` for row projection and `axis=0` for column projection; `find_histogram_peaks()` detects character regions.

Listing 8: Histogram-based Wave Detection

```
row_hist = np.sum(binary_plate, axis=1)
min_row = np.min(row_hist)
avg_row = np.sum(row_hist) / row_hist.shape[0]
row_thresh = (min_row + avg_row) / 2
row_peaks = find_histogram_peaks(row_thresh, row_hist)
```

Waveform Analysis

Principle: Waveform analysis is a technique used to examine the structure and characteristics of signals, typically by analyzing their amplitude, frequency, and phase over time. In image processing, waveform analysis can be applied to study periodic patterns, such as those found in textures or repeated features. Methods like Fourier Transform are often used to decompose the signal into its frequency components, allowing for the detection of underlying periodic structures. This analysis is useful in applications such as speech recognition, medical diagnostics, and vibration analysis, where understanding the periodicity of a signal is crucial for accurate interpretation.

In the system: Initial peaks (for province and city codes) are merged, and special characters (like '·') are ignored to improve segmentation.

Listing 9: Horizontal Projection

```
row_hist = np.sum(binary_plate, axis=1)
min_row = np.min(row_hist)
avg_row = np.sum(row_hist) / row_hist.shape[0]
row_thresh = (min_row + avg_row) / 2
row_peaks = find_histogram_peaks(row_thresh, row_hist)
largest_peak = max(row_peaks, key=lambda x: x[1] - x[0])
binary_plate = binary_plate[largest_peak[0]:largest_peak[1]]
```

Listing 10: Vertical Projection

```
col_hist = np.sum(binary_plate, axis=0)
min_col = np.min(col_hist)
avg_col = np.sum(col_hist) / col_hist.shape[0]
col_thresh = (min_col + avg_col) / 5
col_peaks = find_histogram_peaks(col_thresh, col_hist)
if col_peaks[0][1] - col_peaks[0][0] < max_width / 3:
    col_peaks.pop(0)
```

Listing 11: Peak Detection

```
def find_histogram_peaks(threshold, histogram):
    rising_edge = -1
    is_peak = False
    peaks = []
    for i, x in enumerate(histogram):
        if is_peak and x < threshold:
            if i - rising_edge > 2:
                peaks.append((rising_edge, i))
                is_peak = False
        elif not is_peak and x >= threshold:
            is_peak = True
            rising_edge = i
    return peaks
```

Projection-based Segmentation

Principle: Projection-based segmentation is an image processing technique used to segment objects in an image by analyzing the projection of pixel intensities along different directions, typically horizontal or vertical. This method involves projecting the image's pixel values onto a set of lines and then analyzing the resulting histograms or profiles to identify regions of interest. Peaks or valleys in the projection profiles correspond to boundaries of objects or foreground/background separation. This approach is particularly useful for applications like document layout analysis, text extraction, and medical imaging, where accurate segmentation of structured objects is crucial [33].

In the system: Character blocks are cropped based on projections and resized to a standard size for classification.

Listing 12: Projection-based Segmentation

```
col_hist = np.sum(binary_plate, axis=0)
min_col = np.min(col_hist)
avg_col = np.sum(col_hist) / col_hist.shape[0]
col_thresh = (min_col + avg_col) / 5
col_peaks = find_histogram_peaks(col_thresh, col_hist)
```

2.2.4 Character Recognition

HOG Feature Extraction

Principle: HOG (Histogram of Oriented Gradients) feature extraction is a technique used to describe the local shape and structure of an image by capturing the distribution of gradient orientations. It is particularly effective in object detection tasks, such as human detection, because it is invariant to small changes in position, scale, and orientation. The process begins by dividing the image into small cells, typically 8x8 or 16x16 pixels. For each cell, the gradient magnitude and orientation are computed using derivative filters, such as the Sobel operator, to highlight edges and texture details. These gradients are then grouped into orientation bins, and a histogram of these orientations is formed.

Next, the histograms of neighboring cells are combined into larger blocks, typically 2x2 or 3x3 cells per block, to normalize the histograms and reduce the effects of lighting variations. The final feature vector is obtained by concatenating the normalized histograms from all blocks in the image. This feature vector represents the image in terms of the gradient distribution across different regions, making it suitable for classification tasks when combined with machine learning algorithms like Support Vector Machines (SVM).

HOG is widely used in computer vision for tasks such as pedestrian detection, vehicle recognition, and facial recognition, due to its robustness to lighting changes and ability to capture essential structural features [34].

In the system: In HOG feature extraction, I adopted the classic parameter configuration and made appropriate optimizations. The cell size is set to 8×8, the block size is 2×2 cells (i.e., 16×16 pixels), and the block stride is 8 pixels. The number of orientation bins is set to 9, and L2-Hys normalization is used. These parameter combinations were referenced from the original design by Dalal and Triggs, and were verified through comparative experiments on our license plate dataset. Specifically, I tried several different cell sizes (e.g., 8×8, 16×16) and numbers of orientations (e.g., 6, 9, 12), and the current combination demonstrated the best performance in terms of recognition accuracy and computational efficiency. Each character image is divided into four 10x10 blocks. HOG descriptors are computed and concatenated into a 64-dimensional feature vector, then normalized as SVM input.

Listing 13: HOG Feature Extraction

```python
def extract_hog_features(digit_images):
    feature_vectors = []
    for img in digit_images:
        grad_x = cv2.Sobel(img, cv2.CV_32F, 1, 0)
        grad_y = cv2.Sobel(img, cv2.CV_32F, 0, 1)
        magnitude, angle = cv2.cartToPolar(grad_x, grad_y)
        bins_count = 16
        quantized_angles = np.int32(bins_count * angle / (2 * np.pi))

        angle_blocks = (quantized_angles[:10, :10], quantized_angles
            [10:, :10],
                        quantized_angles[:10, 10:], quantized_angles
                            [10:, 10:])
        mag_blocks = (magnitude[:10, :10], magnitude[10:, :10],
                      magnitude[:10, 10:], magnitude[10:, 10:])

        histograms = [np.bincount(b.ravel(), m.ravel(), bins_count) for
            b, m in zip(angle_blocks, mag_blocks)]
        combined_hist = np.hstack(histograms)

        epsilon = 1e-7
        combined_hist /= combined_hist.sum() + epsilon
        combined_hist = np.sqrt(combined_hist)
        combined_hist /= norm(combined_hist) + epsilon

        feature_vectors.append(combined_hist)
    return np.float32(feature_vectors)
```

SVM Classification

Principle: As the introduction to SVM classification has already been provided earlier, it will not be repeated here.

In the system: The Chinese character model (`province_model`) and the alphanumeric character model (`char_model`) are trained separately. Final predictions are concatenated into the license plate string.

Listing 14: SVM Classification

```python
class SVMModel(MachineLearningModel):
    def __init__(self, C=1, gamma=0.5):
        self.model = cv2.ml.SVM_create()
        self.model.setGamma(gamma)
        self.model.setC(C)
        self.model.setKernel(cv2.ml.SVM_RBF)
        self.model.setType(cv2.ml.SVM_C_SVC)

    def train_model(self, samples, responses):
        self.model.train(samples, cv2.ml.ROW_SAMPLE, responses)

    def predict_chars(self, samples):
        result = self.model.predict(samples)
        return result[1].ravel()
```

2.2.5 Training and Evaluation

Training Workflow: The training process is implemented within the `PlateRecognizer` class, where the `train_svm()` function is invoked to execute model training. The system automatically traverses the specified training directories—one for Chinese characters and one for alphanumeric characters. Each image is loaded and converted to grayscale, followed by skew correction using the `correct_skew()` function. Subsequently, uniform-dimensional HOG (Histogram of Oriented Gradients) feature vectors are extracted via the `extract_hog_features()` method. Two separate SVM models are trained using the `SVMModel.train_model()` method and saved for later inference.

Listing 15: Train letters and Numbers

```python
training_images = []
training_labels = []
for root, dirs, files in os.walk("train\\chars2"):
    char_code = ord(os.path.basename(root))
    for filename in files:
        char_img = cv2.imread(file_path, cv2.IMREAD_GRAYSCALE)
        training_images.append(char_img)
        training_labels.append(char_code)
training_images = list(map(correct_skew, training_images))
training_features = extract_hog_features(training_images)
training_labels = np.array(training_labels)
self.char_model.train_model(training_features, training_labels)
```

Listing 16: Training Chinese Characters

```
for root, dirs, files in os.walk("train\\charsChinese"):
    pinyin = os.path.basename(root)
    province_id = PROVINCE_MAPPING.index(pinyin) + PROVINCE_START_ID + 1
    for filename in files:
        char_img = cv2.imread(file_path, cv2.IMREAD_GRAYSCALE)
        training_images.append(char_img)
        training_labels.append(province_id)
self.province_model.train_model(training_features, training_labels)
```

Evaluation Workflow: The evaluation process is handled in the `Evaluate.py` script via the `evaluate_and_plot_plate_recognition(csv_path)` function. The system loads the recognition results from a given CSV file and computes character-level evaluation metrics per license plate. True positives (TP), false positives (FP), and false negatives (FN) are calculated for each character prediction. The evaluation metrics—including accuracy, precision, recall, and F1-score—are reported for the overall dataset, as well as separately for Chinese characters and alphanumeric strings. In addition, Matplotlib is used to generate trend plots and comparison charts to provide comprehensive visual analysis of recognition performance.

2.3 Results

2.3.1 Main Results

The license plate recognition system was evaluated using a test set consisting of 100 license plate images, containing a total of 705 characters (100 Chinese characters and 605 letters and numbers). In the test set, the distribution of the number of Chinese characters on the license plate is shown in the Fig. 1. Most of the pictures were taken during the day, from the front of the license plate. There are also images taken at night or at a steep angle in the dataset, and the recognition results obtained from these images have a higher error rate compared to other images.

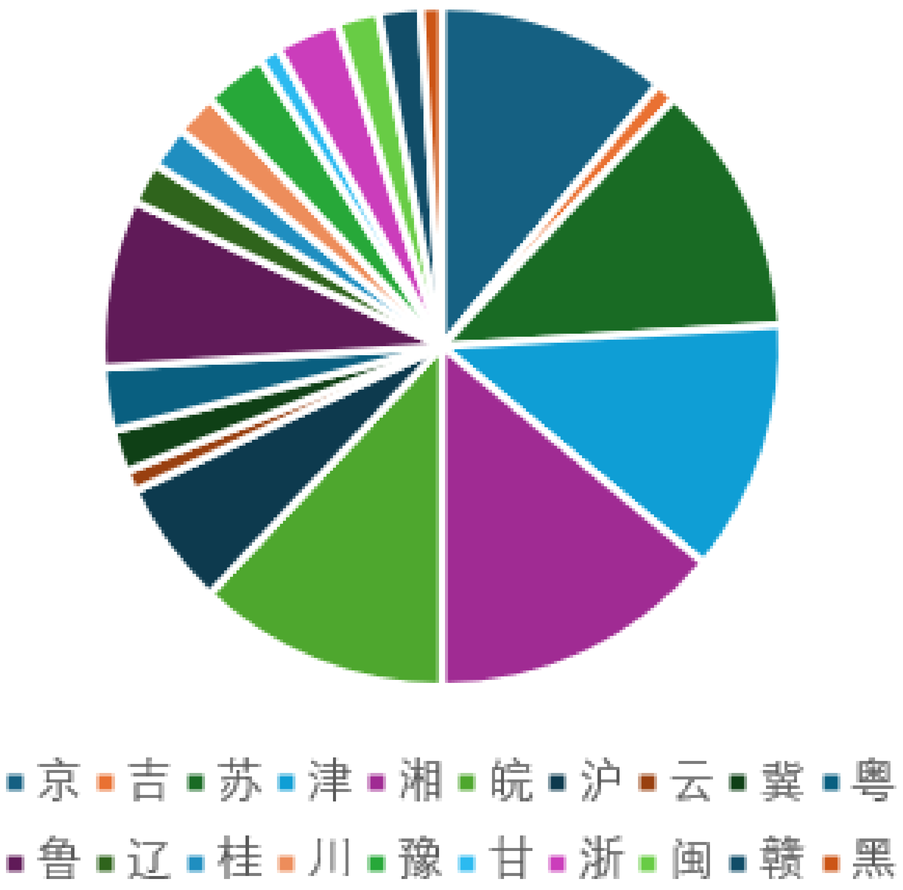


Figure 1: Actual license plates

The model's performance was measured using four standard metrics: accuracy, precision, recall, and F1 score. These results are presented in Tab. 3. The overall recognition accuracy reached 88.09%, with a precision of 88.46%, recall of 88.59%, and an F1 score of 88.52%. These figures indicate strong recognition capabilities in most scenarios, particularly in identifying letters and numbers. Fig. 2 illustrates how these metrics varied as more test data are introduced.The system also realizes the recognition of license plate colors. The recognizable colors include green, yellow and blue. The recognition accuracy can reach 100%.

Table 3: Assessment results for all characters, Chinese characters, letters and numbers

Character Type	Accuracy	Precision	Recall	F1 Score
All Characters	0.8809	0.8846	0.8859	0.8852
Chinese Characters	0.5800	0.5800	0.5859	0.5829
Letters & Numbers	0.9306	0.9352	0.9352	0.9352

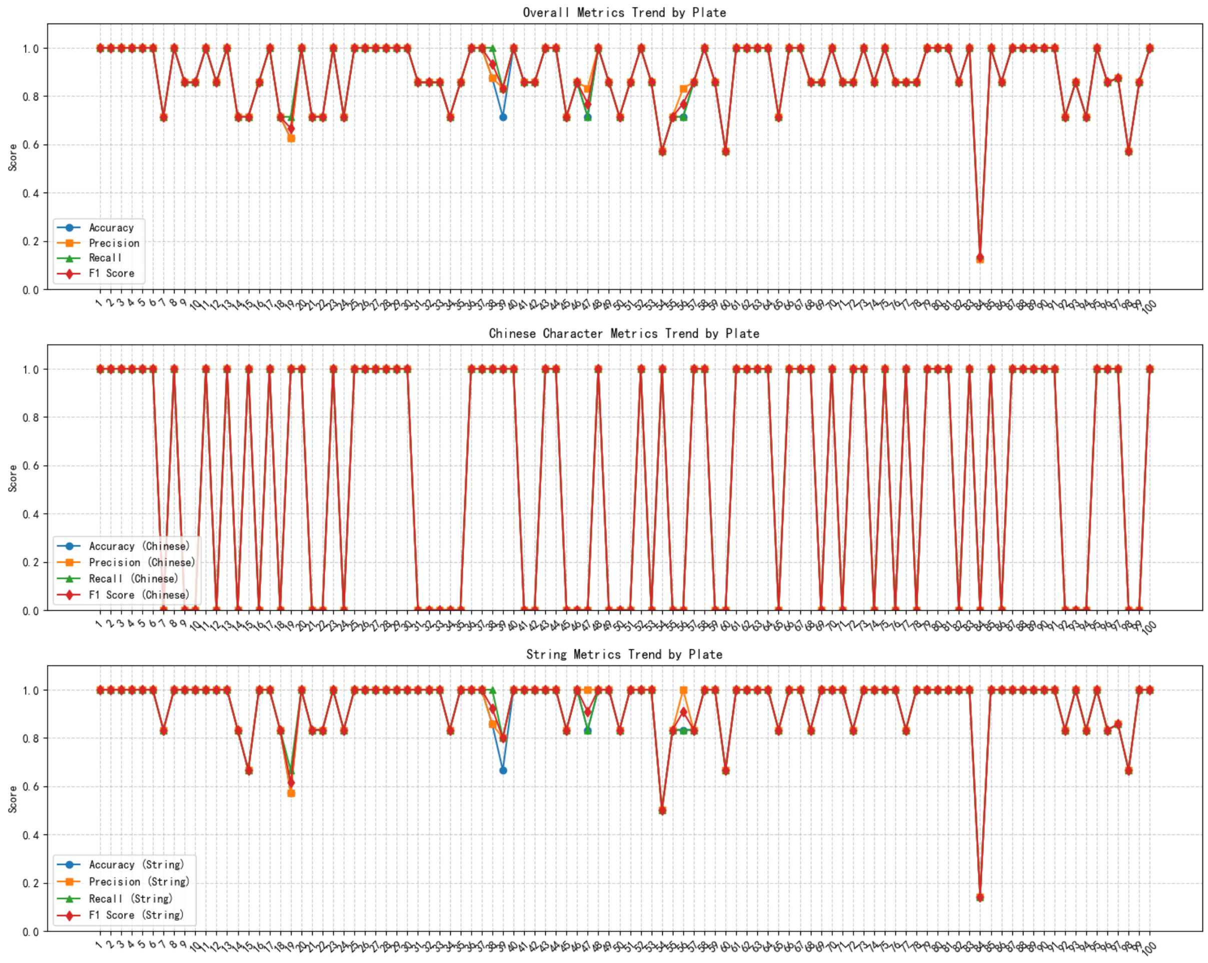
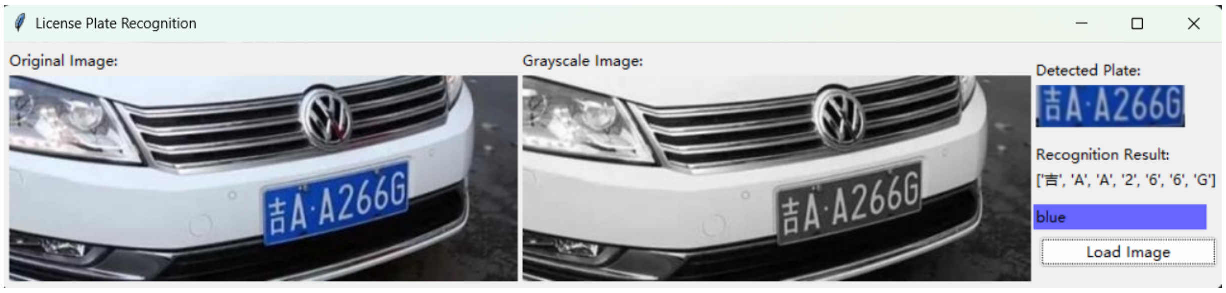


Figure 2: Evaluation curves for different characters

To better illustrate the system's recognition effectiveness, several successful recognition cases are shown in Fig. 3, including correctly identified characters and color classification results. These examples demonstrate the system's robustness under varied lighting conditions, angles, and resolutions.



Recognition Result:
['吉', 'A', 'A', '2', '6', '6', 'G']
blue

Figure 3: Examples of successful license plate and color recognition

1. Overall Performance

Out of 705 characters, 621 were correctly recognized while 84 were misclassified, indicating a high level of recognition accuracy. As shown in Fig. 4 , most license plates achieved precision, recall, accuracy, and F1 scores close to or at 0.9. Noticeable drops were observed only in a few cases, such as plates numbered 54 and 84. These declines were typically associated with issues like low image quality, light issue or partial occlusion. For example, Fig. 5 shows examples of license plates captured under excessively dark or bright lighting, both of which negatively impact recognition accuracy.

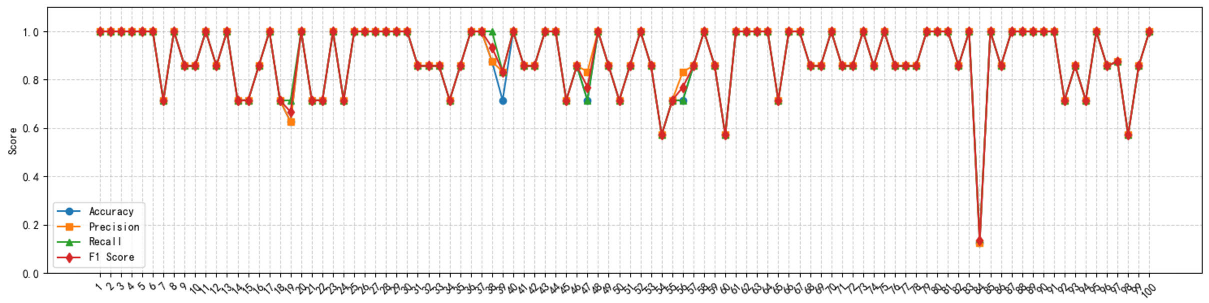


Figure 4: Evaluation curves for all characters

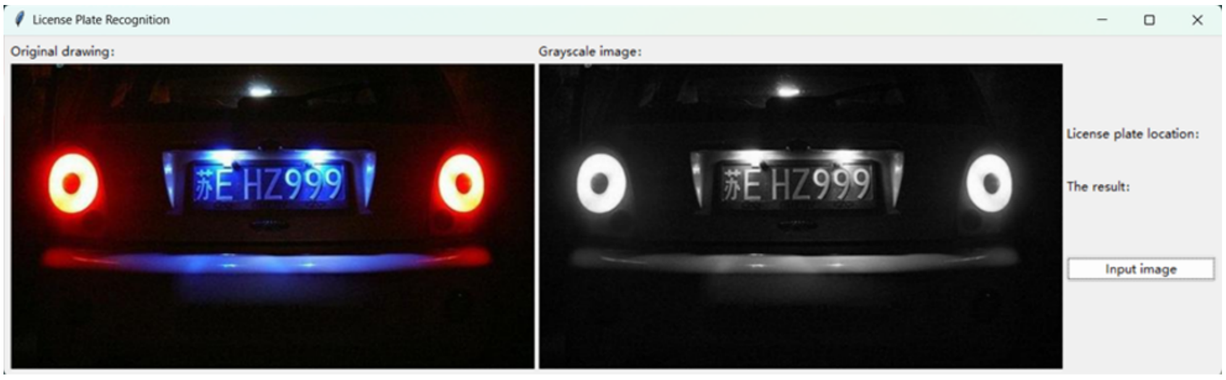
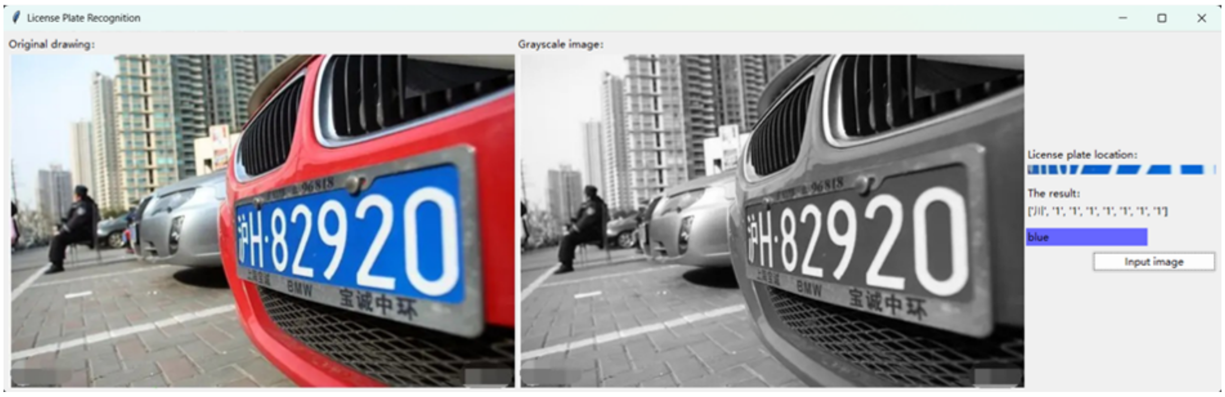


Figure 5: Recognition failures caused by poor lighting conditions

In addition, during the recognition process, if there are multiple lines of characters on the same license plate or multiple license plates in the same image, it will affect the recognition results. As shown in Fig. 6, a tilted shooting angle can also distort character shapes and spacing, leading to segmentation and recognition errors.



```
['川', '1', '1', '1', '1', '1', '1', '1']
```

Figure 6: Recognition errors caused by tilted camera angles

2. Poor Performance in Chinese Character Recognition

The recognition accuracy for Chinese characters is significantly lower than the overall level, reaching only 58.00%, with precision and recall at 58.00% and 58.59% respectively, and an F1 score of just 58.29%. As shown in Fig. 7, performance for Chinese characters fluctuates considerably compared to results of all character recognition. For several license plates, the F1 score for Chinese characters drops to zero, indicating complete failure in handling certain characters. This issue is particularly prominent in nearly half of the plates, highlighting Chinese character recognition as the primary weakness of the system. And another issue was discovered during the evaluation process, which is that recognition errors or direct recognition failures may occur when Chinese characters other than provinces appear in the license plate.

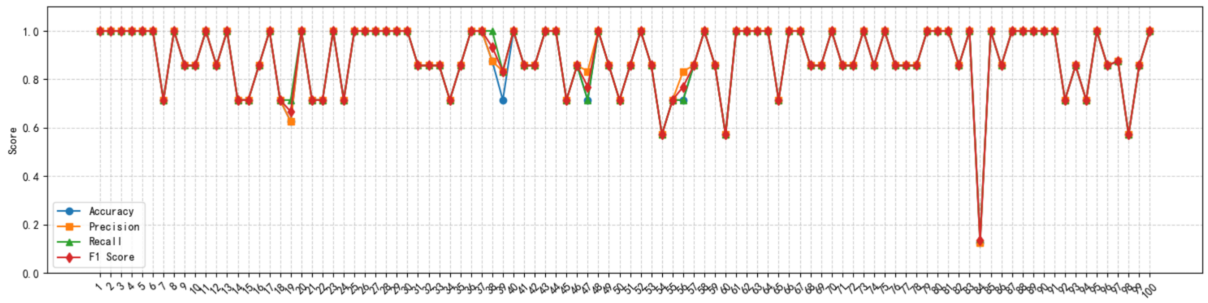


Figure 7: Evaluation curves for Chinese characters

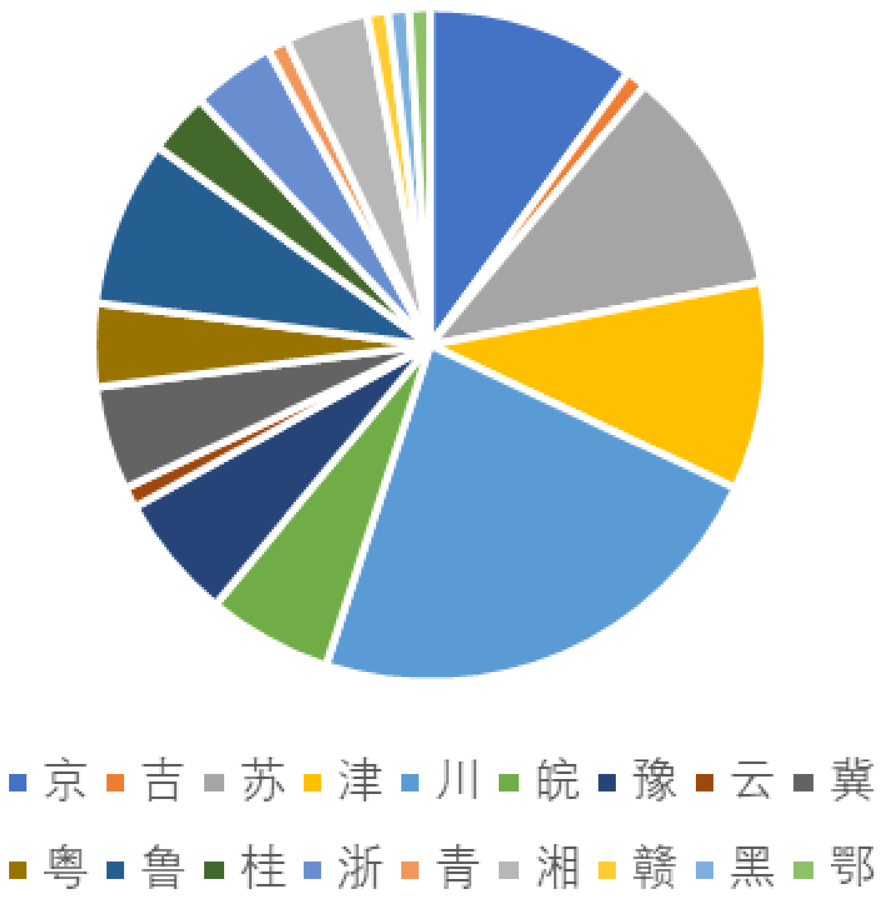
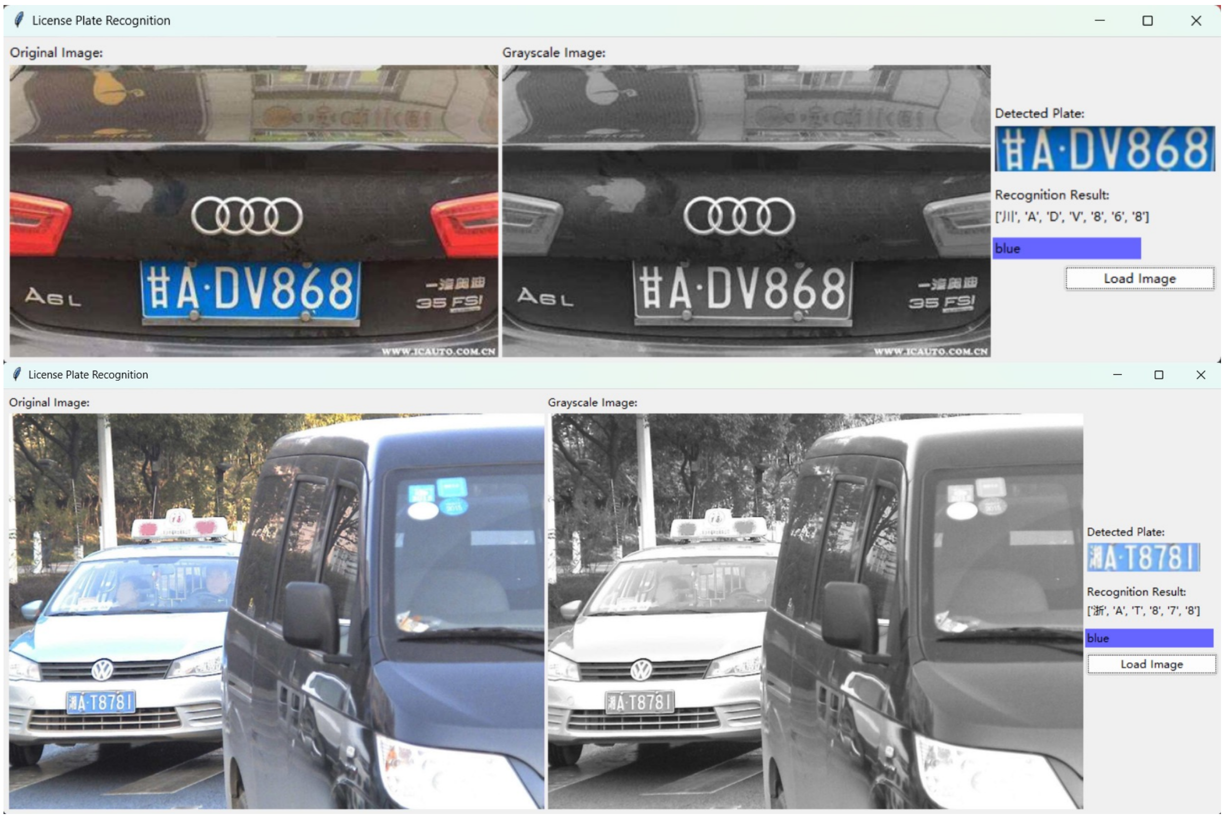


Figure 8: Detected license plates

The main factors contributing to the low performance include:

- Insufficient sample size: A limited and uneven distribution of Chinese character samples leads to inadequate model training.
- Structural complexity: Dense strokes and diverse forms make Chinese characters more prone to background interference or segmentation errors.
- High similarity: Characters like "Zhe" and "Xiang" or "Gan" and "Chuan" in Fig. 9, are subtle differences and easily confused.



Detected Plate:

Recognition Result:
['川', 'A', 'D', 'V', '8', '6', '8']

blue

Detected Plate:

Recognition Result:
['浙', 'A', 'T', '8', '7', '8']

blue

Figure 9: Examples of misidentification due to similar-looking Chinese characters

3. Stable and High-Precision Recognition of Letters and Digits

In comparison, the recognition of letters and digits demonstrates strong stability. The accuracy reaches 93.06%, with both precision and recall at 93.52%, resulting in an F1-score of 93.52%. As shown in Fig. 10, all four metrics consistently remain at high levels across the majority of license plates, indicating strong generalization and robustness in this sub-task. Notably, nearly all metrics remain above 0.9, highlighting the model's effectiveness in recognizing characters with clear structure and regular shapes.

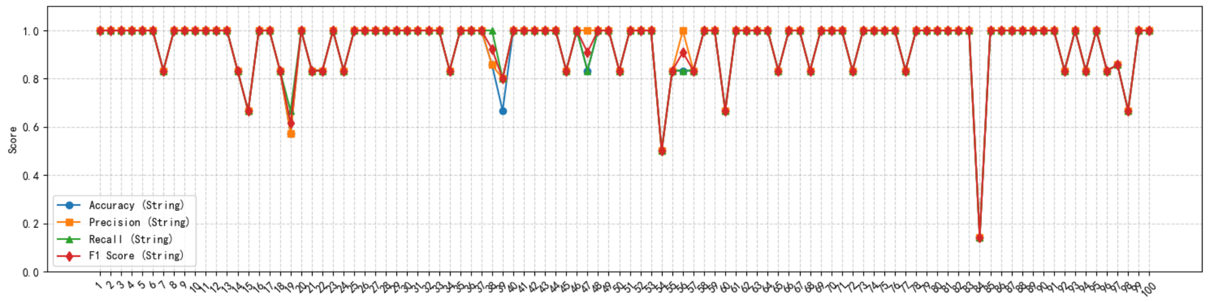


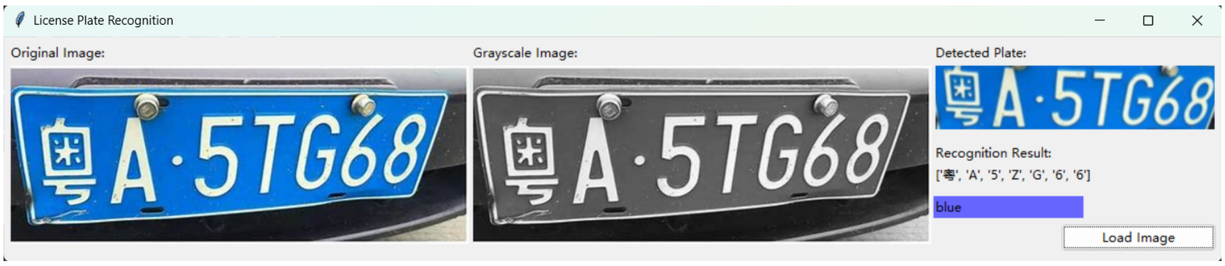
Figure 10: Evaluation curves for letters and digits characters

This performance advantage is primarily attributed to:

- Simple character structure: Significant variation and well-defined shapes among characters make feature extraction more effective.
- Abundant training data: A comprehensive range of letters and digits in the training set enhances generalization.
- Effective feature extraction: The method used for feature extraction responds particularly well to this type of character.

4. Error Distribution and Consistency

An analysis of error types reveals that the number of false positives (FP = 81) and false negatives (FN = 80) is roughly balanced for both Chinese characters and alphanumeric ones. This indicates a relatively even error distribution, without a strong tendency toward over-detection or omission. At the character level, common confusion pairs are frequently observed, such as “6” and “8”, or “T” and “Z”. These misclassifications typically occur in areas with low resolution or blurred edges. As illustrated in Fig. 11, these visually similar characters are often misidentified due to shape overlap and environmental noise.



Recognition Result:
['粤', 'A', '5', 'Z', 'G', '6', '6']
blue

Figure 11: Examples of misidentification due to similar-looking letters and digits

5. Plate-Level Recognition Trend Analysis

The three metric charts collectively indicate that the system’s performance on full license plate strings aligns closely with its character-level recognition. As shown in Fig. 4, most license plates achieve an F1-score above 0.9 for the entire string, indicating correct recognition of all characters. However, in certain cases, errors in Chinese character recognition result in complete string recognition failure, dropping the accuracy to 0.6 or lower, such as plates numbered 54, 60, and 84. This highlights that the system’s overall robustness is heavily influenced by the accuracy of the initial (Chinese) character.

2.3.2 Discussion

The results indicate that the system performs exceptionally well in recognizing letters and digits, demonstrating solid practical potential, particularly in scenarios involving stable character structures and high-quality images. However, the recognition of Chinese characters remains a significant bottleneck, currently the main limitation on overall system performance.

Strengths

- **High accuracy on standard characters:** The system demonstrates strong recognition capabilities for 26 English letters and 10 digits, with stable performance across multiple test cases, suitable for most industrial applications.

- **Well-structured recognition pipeline:** The entire process from character detection to recognition is clear and robust, effectively handling standardized license plate images.

- **Balanced error distribution:** False positives and false negatives are evenly distributed, suggesting that the system's decision-making logic is relatively stable.

Comparison with Existing Methods Recent approaches based on deep neural networks, such as CNNs, CRNNs, and Transformer-based architectures, have demonstrated superior performance in license plate recognition tasks. For instance, in the work by Shi et al. [35], a CRNN-based model achieved an overall recognition accuracy of 94.7% on public Chinese license plate datasets, with an F1-score of 91.2% for Chinese characters. In contrast, the proposed system, which follows a traditional step-wise architecture, achieves significantly lower F1-scores for Chinese characters—typically below 80%. Furthermore, end-to-end models eliminate the need for character segmentation, which enhances robustness against noisy backgrounds and image distortions. While the proposed system performs comparably on alphanumeric characters, it struggles with diverse fonts and complex environments.

Challenges

- **Low Chinese character recognition accuracy:** The performance is inconsistent, with lower F1-scores, and a single misclassified character—often the province code—can lead to failure in recognizing the entire plate.

- **Limited feature representation:** Traditional handcrafted features (e.g., HOG) are insufficient for representing complex Chinese characters, making the system vulnerable to small-scale deformations and rotations.

- **Strong dependency on segmentation:** The step-wise design causes recognition accuracy to be highly sensitive to errors in character localization or segmentation.

Suggested Improvements

- **Enhancing Chinese character samples:** Use data augmentation techniques such as font variation, noise simulation, and illumination perturbation to expand the training set and improve model generalization.
- **Adopting deep learning models:** Replace the traditional SVM with CNN, CRNN, or Transformer-based recognizers to enhance feature extraction and robustness.
- **Optimizing segmentation strategy:** Introduce deep semantic segmentation or attention mechanisms to improve character localization and reduce segmentation errors.
- **Incorporating rule-based post-processing:** Apply rule-based verification using dictionaries of Chinese province abbreviations and valid character combinations to correct recognition errors and improve usability.

2.4 Conclusions

In conclusion, the current system can fulfill the needs of low real-time license plate recognition scenarios. However, for high-precision and high-robustness applications such as traffic monitoring and toll collection, further improvements are required, particularly in handling Chinese characters and stabilizing the overall end-to-end structure.

3 Conclusion and Future Work

3.1 Conclusion

In this project, a modular license plate recognition (LPR) system was developed using traditional computer vision techniques and machine learning. OpenCV was used for image preprocessing and character segmentation, while character recognition was performed using a support vector machine (SVM) classifier with histogram of oriented gradient (HOG) features. The system was designed to prioritize computational efficiency, explainability, and adaptability in low-resource environments.

The final system achieved an overall character recognition accuracy of 88.09%, with strong performance in alphanumeric characters (93. 06% accuracy, F1 score of 0.9352). In contrast, Chinese character recognition remained a challenge, with a lower F1 score of 0.5829. This disparity was primarily attributed to data imbalance, character complexity, and limitations in feature representation.

A detailed failure analysis identified several key issues:

- Segmentation errors due to overlapping or touching characters.
- Misclassifications between structurally similar characters.
- Limited robustness under varied lighting conditions, image skew, and noise.

Despite these challenges, the traditional pipeline demonstrates clear advantages in explainability and ease of debugging, making it suitable for lightweight applications and educational environments. However, for deployment in real-world dynamic environments, further improvements are necessary.

3.2 Future Work

To address the identified limitations, the following directions are proposed:

1. **Model Enhancement**
    - Replace the SVM with a CNN or CRNN-based classifier to enable richer feature learning, particularly for complex Chinese characters.
    - Introduce attention mechanisms or Transformer-based models to capture long-range dependencies between character strokes.

2. **Data Expansion**

    - Increase the volume and diversity of Chinese character samples.
    - Incorporate training data exhibiting greater variability in lighting, rotation, and occlusion.

3. **End-to-End Integration**

    - Transition to a unified deep learning framework (e.g., YOLOv7 for plate detection combined with CRNN for character recognition) to reduce reliance on manual rules and limit error propagation.

In summary, the project establishes a robust foundation for character recognition within LPR systems using classical techniques, while also highlighting clear avenues for advancement through deep learning. The findings and implementation strategies serve as a practical baseline for future research and real-world applications.

References

[1] Q. Liu, Y. Liu, S. L. Chen, T. H. Zhang, F. Chen, and X. C. Yin, "Improving multi-type license plate recognition via learning globally and contrastively," *IEEE Transactions on Intelligent Transportation Systems*, vol. 25, no. 9, pp. 11 092–11 102, Sept. 2024.

[2] S. Chen, Z. Li, X. Du, and Q. Nie, "Eand-lprm: Enhanced attention network and decoding for efficient license plate recognition under complex conditions," *Algorithms*, vol. 17, no. 6, p. 262, Jun. 2024.

[3] S. Tejaswi, T. Babu, and K. Tejaswi, "A comparative analysis of image processing and deep learning techniques for license plate recognition in difficult environments," in *2024 International Conference on IoT Based Control Networks and Intelligent Systems (ICICNIS)*. IEEE, Dec. 2024, pp. 1225–1231.

[4] M. Satsangi, M. Yadav, and P. S. Sudhish, "License plate recognition: A comparative study on thresholding, ocr and machine learning approaches," in *2018 International Conference on Bioinformatics and Systems Biology (BSB)*, 2018, pp. 1–6.

[5] H. Chen, X. Shi, M. Liu, and C. Chen, "A chinese license plate recognition system based on opencv for complex environments," *Procedia Computer Science*, vol. 243, pp. 1265–1272, 2024.

[6] D. Zheng, Y. Zhao, and J. Wang, "An efficient method of license plate location," *State Key Laboratory of Intelligent Technology and Systems, Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China*, June 2005, received 11 May 2004, Revised 4 April 2005, Available online 22 June 2005.

[7] S. M. Kazi and B. G. Kodge, "Svm-based vehicle number plate detection and recognition," in *Proceedings of the 2nd International Conference on Cognitive and Intelligent Computing: ICCIC 2022, Volume 1*, ser. Cognitive Science and Technology, A. Kumar, G. Ghinea, and S. Merugu, Eds. Singapore: Springer Nature Singapore, 2023, pp. 485–491, 27–28 December, Hyderabad, India.

[8] P. Kulkarni, A. Khandebharad, D. Khope, and P. U. Chavan, "License plate recognition: A review," in *2012 Fourth International Conference on Advanced Computing (ICoAC)*, 2012, pp. 1–8.

[9] H. Li and C. Shen, "Reading car license plates using deep convolutional neural networks and LSTMs," *IEEE Transactions on Intelligent Transportation Systems*, vol. 17, no. 9, pp. 2654–2666, 2016.

[10] Z. Xu, W. Yang, A. Meng, N. Lu, H. Huang, C. Ying, and L. Huang, "Towards end-to-end license plate detection and recognition: A large dataset and baseline," in *Proceedings of the European Conference on Computer Vision (ECCV)*. Munich, Germany: Springer, Sep. 2018.

[11] Z. Liu and Y. Zhu, "Vehicle license plate recognition in complex scenes," in *2020 IEEE 5th International Conference on Intelligent Transportation Engineering (ICITE)*, 2020, pp. 235–239.

[12] S. M. Silva and C. R. Jung, “License plate detection and recognition in unconstrained scenarios,” in *European Conference on Computer Vision (ECCV)*, 2018, pp. 580–596.

[13] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, “An image is worth 16x16 words: Transformers for image recognition at scale,” in *International Conference on Learning Representations (ICLR)*, 2021. [Online]. Available: https://openreview.net/forum?id=YicbFdNTTy

[14] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[15] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo, “Swin transformer: Hierarchical vision transformer using shifted windows,” *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 10 012–10 022, 2021.

[16] J. Pirgazi, M. M. P. Kallehbasti, and A. G. Sorkhi, “An end-to-end deep learning approach for plate recognition in intelligent transportation systems,” *Wireless Communications and Mobile Computing*, vol. 2022, p. 3364921, 2022, academic Editor: Mohammad R Khosravi.

[17] C. N. Anagnostopoulos, I. E. Anagnostopoulos, I. D. Psoroulas, V. Loumos, and E. Kayafas, “License plate recognition from still images and video sequences: A survey,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 9, no. 3, pp. 377–391, 2008.

[18] S. Zherzdev and A. Gruzdev, “Lprnet: License plate recognition via deep neural networks,” *arXiv preprint arXiv:1806.10447*, 2018.

[19] G.-S. Hsu, J.-H. Chen, and Y.-L. Chen, “End-to-end license plate detection and recognition with cnn,” in *2017 IEEE International Conference on Multimedia and Expo (ICME)*. IEEE, 2017, pp. 1517–1522.

[20] D. Nasien, S. S. Yuhaniz, and H. Haron, “Statistical learning theory and support vector machines,” in *2010 Second International Conference on Computer Research and Development*, 2010, pp. 760–764.

[21] C. Cortes and V. Vapnik, “Support-vector networks,” *Machine Learning*, vol. 20, no. 3, pp. 273–297, 1995. [Online]. Available: https://doi.org/10.1007/BF00994018

[22] A. Fitzgibbon, “Simultaneous linear estimation of multiple view geometry and lens distortion,” in *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, vol. 1, 2001, pp. I–I.

[23] D. Ridel, E. Rehder, M. Lauer, C. Stiller, and D. Wolf, “A literature review on the prediction of pedestrian behavior in urban scenarios,” in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, 2018, pp. 3105–3112.

[24] B. Schölkopf and A. J. Smola, *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*, ser. Adaptive Computation and Machine Learning. The MIT Press, 2001. [Online]. Available: https://doi.org/10.7551/mitpress/4175.001.0001

[25] V. N. Vapnik, *The Nature of Statistical Learning Theory*, 2nd ed. Springer, 1999, https://doi.org/10.1007/978-1-4757-3264-1.

[26] R. E. Twogood and F. G. Sommer, "Digital image processing," *IEEE Transactions on Nuclear Science*, vol. 29, no. 3, pp. 1075–1086, 1982.

[27] J. Serra, *Image Analysis and Mathematical Morphology.* Academic Press, 1982, vol. 1.

[28] X. Zeng, "Highway license plate recognition system based on hough transform and svm," *A. Traffic Engineering*, vol. 3, pp. 175–180, 2022.

[29] J. Canny, "A computational approach to edge detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 8, no. 6, pp. 679–698, 1986.

[30] D. Marr, *The Theory of Vision.* MIT Press, 1980.

[31] J. Schanda, *Color Science and Technology: A Handbook for Scientists and Engineers.* Springer, 2007.

[32] D. Shepard, "Image registration techniques: A survey," *Computer Vision, Graphics, and Image Processing*, vol. 43, no. 3, pp. 160–179, 1995.

[33] J. Ma and K. Chen, "Projection-based segmentation and feature extraction," *Journal of Visual Communication and Image Representation*, vol. 17, no. 3, pp. 600–615, 2006.

[34] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2005, pp. 886–893.

[35] B. Shi, X. Bai, and C. Yao, "An end-to-end trainable neural network for image-based sequence recognition and its application to scene text recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017.