SeAMK
SEINÄJOEN AMMATTIKORKEAKOULU
SEINÄJOKI UNIVERSITY OF APPLIED SCIENCES

Olga Shlenchak

Neural Network-Based License Plate Recognition

Thesis
Spring 2025
Bachelor of Engineering, Automation Engineering

SEINÄJOKI UNIVERSITY OF APPLIED SCIENCES

Thesis abstract

Degree Programme: Bachelor of Engineering

Specialization: Automation Engineering

Author: Olga Shlenchak

Title of thesis: Automated license plate recognition system

Supervisor: Raine Kauppinen

Year: 2025 Number of pages: 64 Number of appendices: 1

The purpose of this bachelor's thesis was to create a system for recognizing state registration numbers of cars.

The subject area was analyzed, and existing systems with similar functionality were reviewed. An informational-logic model of the system was developed: UML diagrams were created, algorithms were designed, and system architecture was documented using the Draw.io web application.

An automated system recognizing license plates (ASRN) was implemented using Python and Windows Forms in the PyCharm 2022 development environment, targeting Windows 10 and newer operating systems.

The ASRN was trained and validated using a dataset of license plate images. Performance evaluation on a test set of 1,000 photo frames showed an average recognition time of 93 milliseconds and an accuracy rate of 83%. In terms of accuracy, the developed system performs on par with commercial LPR solutions such as CVS Auto and Auto Trassir. The evaluation also included speed and robustness under challenging conditions such as low light and weather interference. While results demonstrate high efficiency, further training and optimization are planned to enhance overall accuracy and reliability.

[1] Keywords: Neural networks, two-dimensional image, symbol recognition, license plate number

TABLE OF CONTENTS

Pictures, Figures and Tables

Terms and Abbreviations

ASRN	Automated System Recognizing Numbers (license plates)
CA	Cellular Automaton
CNN	Convolutional Neural Network
DBMS	Database Management System
Fps	Frames Per Second
HD	High Definition
HSRN	Hardware-based Systems Recognizing Numbers (license plates)
ITS	Intelligent Transport Systems
KNN	K-nearest Neighbours algorithm
LPR	License Plate Recognition
OpenCV	Open-Source Computer Vision Library
OCR	Optical Character Recognition
PyTorch	Python Torch
RAS	Region of Interest Area
SSRN	Software-based Systems Recognizing Numbers (license plates)
SVM	Support Vector Machine
XOR	Exclusive OR (a type of logical operation)

1 Introduction

1.1 Background

The relevance of the development of an automated system that allows the recognition of state license plates on vehicles (hereinafter ASRN) is supported by several factors. The ASRN project creates safer traffic on the roads as the system identifies and then tracks vehicles by license plate number that have been involved in crimes, used by terrorists, or have exhibited other security threats. Automation is a solution that helps law enforcement authorities respond quickly to crimes to prevent them.

The implementation of ASRN eliminates problems related to poor traffic control, because with the monitoring function the whole traffic situation is controlled, traffic congestion is detected, and the vehicle flow becomes manageable. The system collects data on vehicles, tracking their location and movement, which optimizes the use and development of road infrastructure, distributes information on traffic jams, and improves road safety.

With the automation of ASRN, the processes become more precise and labour-intensive tasks are transferred to electronics. ASRN registers vehicles, studies traffic dynamics and identifies instances of traffic violations. Manual methods are not as efficient and fast, cost-effective or accurate as the ASRN.

Charges become more efficient thanks to the ASRN, as the payment for travel, or for the fact of parking is automatically processed. Consequently, payment by manual methods is not required, which is more comfortable for users and reliably increases the revenues of companies providing services to vehicle owners.

ASRNs make the first step towards the intellectualization of intelligent transport systems (ITS), as this link collects information about vehicles. This has a positive impact on traffic by increasing its safety and improving indicators of efficient and comfortable route planning.

The task of improving law enforcement is far from being an afterthought for the ASRS functions, as the system tracks data that law enforcement authorities use to issue fines or judge drivers based on the level of severity of their traffic violations. In particular, the ASRN could

productively trigger photo fixation for speeding or entering a prohibited area. Such functions form additional criteria for ensuring compliance with traffic rules and a safe route.

ASRN reduces the share of thefts, as wanted vehicles are tracked, their location is determined, and they are returned to their owners more often.

1.2 Goals

The purpose of the work has led to a number of tasks, including the review of software- and hardware-based license plate recognition systems, analysis of license plate recognition methods, analysis of the specifics of digital license plate processing, and the development of an algorithm. The final goal of the study is to develop an automated system for recognizing state license plates of cars.

1.3 Research Overview and Structure

The object of the study is the process of recognizing license plate numbers. The subject of the study is the automation of the process of recognizing license plate numbers. The methodological basis of the study includes the following methods: the method of analysis, synthesis, generalization, analogies. Theoretical and practical significance of the work lies in the possibility of using the results of the study in practical implementation of programs to automate the process of recognition of license plates.

The thesis begins with an introduction to License Plate Recognition (LPR) systems, detailing both software- and hardware-based approaches, followed by the development of an automated LPR algorithm, its implementation using modern technologies (Python, PyTorch, OpenCV, TensorFlow), and concludes with analysis, results, and supporting appendices.

2 LPR systems and methods

2.1 Software-based LPR systems

Since the 1990s, software-based license plate recognition systems, or SSRN, have been introduced globally (International Association of Chiefs of Police, 2019). The solution was in great demand due to its affordable cost and the ability to connect a wide variety of camera models.

The use of SSRN had certain disadvantages, as a powerful server required at high cost, otherwise the system did not recognize license plates (Du et al., 2013). In addition, the SSRN was connected in the local network and created a significant load. Any malfunctions on the server made the system inoperable.

Regarding the "SecurOS Auto" ASRN, we note that the system is widely used because of its main function: to recognize the license plate number. The software is a plugin that requires the SecurOS integration platform for installation. The software performs all recognition actions automatically (Intelligent Security Systems, n.d.). The recognized license plate number is checked for presence in the database, after which a decision is made, and the vehicle is logged. ASRN is compatible with stationary and mobile devices that provide surveillance, control and access regulations, which makes the "SecurOS" system an efficient and modern solution in guaranteeing the security of the territory (op. cit.).

The structure of the ASRN and the functions of its components in number recognition are shown in Figure 1.

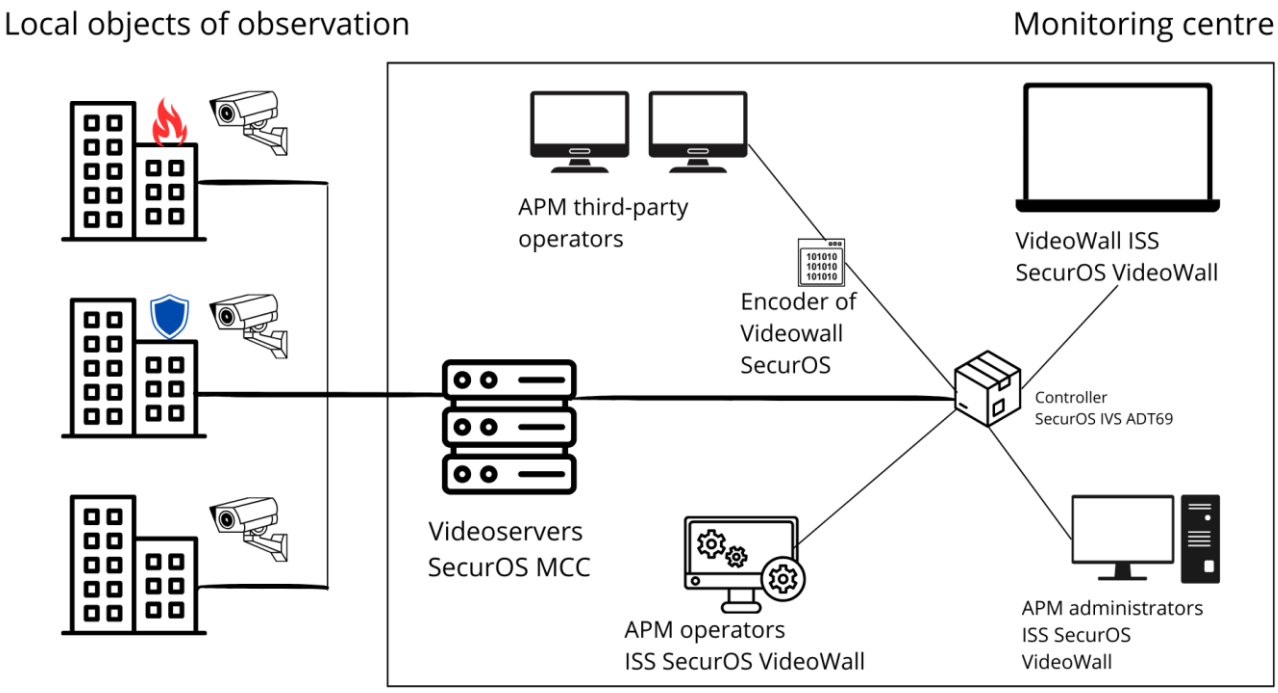


Figure 1 Composition and example of SecurOS Auto system operation

Many capabilities are recognized for ASRN (Shi, 2013, p. 105). The system can identify state registration plates by comparing them against a database containing data from 60 different states. Recognition is performed based solely on photographs, not video recordings. The system includes settings that allow for adjustments in accordance with updated license plate design standards.

All recognized license plates are automatically stored in a database. Each entry includes a comprehensive set of information: the date and time of detection, the location where the photo was captured by the local system's camera, data on the vehicle's speed and calculated route, as well as a link to the corresponding video footage (Intelligent Security Systems, n.d.). By entering a specific license plate number, the system provides access to all related detection files.

ASRN independently verifies recognized plate numbers against the database without requiring operator input. Based on the recognition results, the system is capable of initiating various signals or actions. It supports multi-level notification mechanisms, including alerts to the device operator (via sound, voice, or visual message), electronic communication with management (through messaging apps, email, or social networks), and notifications to external

authorities when a plate is associated with a specific database tag (e.g., departmental vehicle, top management, supplier’s vehicle, wanted or stolen vehicle).

Furthermore, ASRN can automatically generate a wide range of reports that summarize recognized license plate data or provide detailed responses to specific database search queries (Intelligent Security Systems, n.d.).

ASRN is designed with integration capabilities and can be made compatible with various types of equipment and systems. It supports both stationary and mobile devices that are used for surveillance, control, and accounting purposes. These include, but are not limited to, automated gates, barriers, fences, radars, weighing platforms, and payment terminals.

In addition to hardware integration, ASRN can also be incorporated into third-party software environments, such as systems for automated payment accounting or database management systems (DBMS).

A notable example of its integration potential is its compatibility with the "SecurOS Traffic Scanner" system (Security Info Watch, 2015). Through this integration, ASRN can automatically identify traffic violators and record instances of traffic violations. As a result, it enables comprehensive monitoring and management of the traffic situation within a given area.

The listed functions of "SecurOS Auto" ASRN prove the multitasking capability of the product, which effectively performs control of vehicles by license plate number. It improves the operation of vehicles of utility services, creates a security control, indicates the risks in the traffic situation of the territory. The system manages hardware and utilizes software products, even when they are developed by third parties.

After integration with ASRN the equipment does not lose its full functionality. Video and audio data are being exchanged, telemetry is under control and management, records are being made and are being processed in the database.

Interest in the 'SecurOS Auto' ASRN is driven by its support modules for integrating hardware to match recognized numbers. For example, the auto detector provides voice notifications to announce the recognized number and can also categorize the number into specific lists based on the recognition.

User audio messages are entered into the ASRN to output along with license plate numbers recognized based on specific criteria. In real-time mode, the ASRN reports the recognized license plate number to an external database and indicates whether the license number is included in any categorized vehicle lists.

SecurOS Auto ASRN provides a range of significant advantages for organizations, particularly in terms of facility security and logistics management. Its main function is the protection against unauthorized vehicle entry. The system enables continuous monitoring of vehicle movements within the facility, enhancing internal control and operational oversight.

SecurOS Auto ASRN also facilitates the tracking of vehicles involved in the import and export of goods, allowing for verification against accompanying documentation. This ensures greater transparency and accountability in logistics operations.

The system automatically logs all vehicles passing through checkpoints, recording detailed data such as arrival and departure times. It also enables analysis of duration of stays, whether the vehicles belong to the enterprise or to third-party organizations, both on-site and off-site.

Another important feature is the ability to monitor the execution of transport-related tasks by checkpoint personnel, ensuring adherence to operational protocols. Furthermore, security processes are automated through differentiated access permissions, allowing for precise control over which vehicles may pass through various checkpoints.

In reports, ASRN generates data on license plate numbers of vehicles that moved past the cameras in the territory according to various criteria (Figure 2).

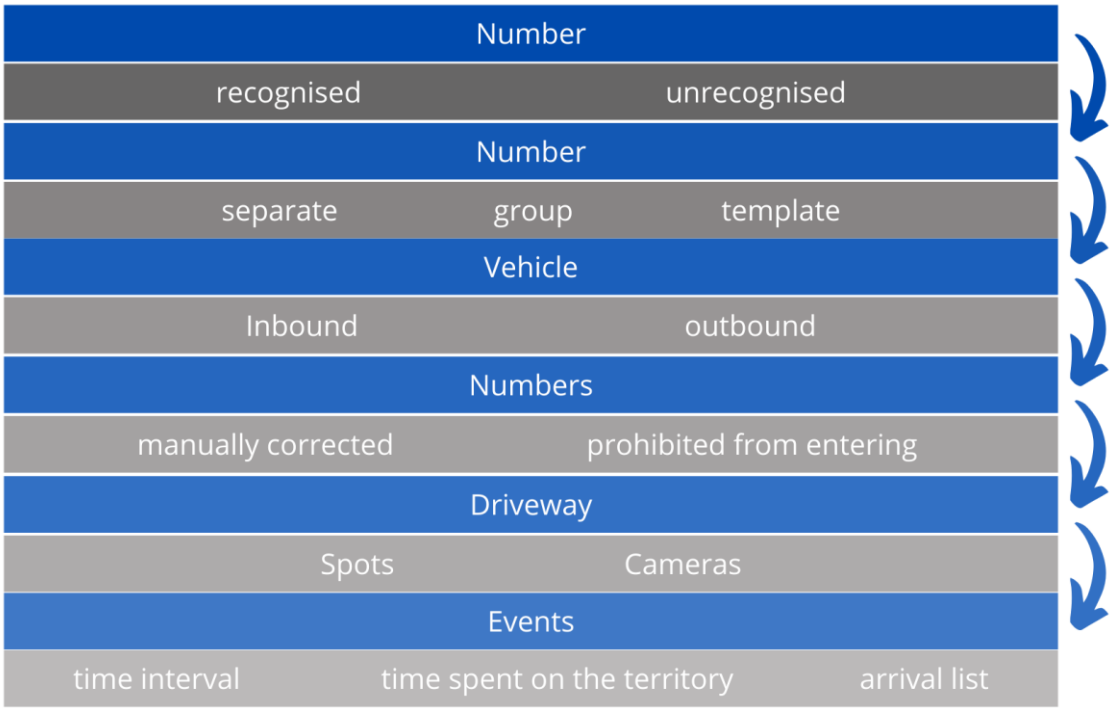


Figure 2 Options for outputting ASRN reports by different criteria

2.2 Hardware-Based LPR Systems

Automatic license plate number recognition functions can also be performed by hardware-based license plate recognition systems (hereinafter referred to as 'HRSN'). Several developments in this category are currently popular.

HSRN Access is offered by the manufacturer Nedap AVI, which has ensured that this development stands out from similar products due to its wide range of functionality (Nedap Identification Systems, n.d). For instance, in the Access HSRN, license plates are recognized by a complete set supplied by the manufacturer: a video camera node works in tandem with infrared lighting, and the collected information is processed on-site in a processing unit. The device is compactly designed with a hermetic case that reflects excess solar energy. For control and data transfer, the device connects to a local network with a web interface.

The license plate number can be recognized very accurately due to the camera's sensitive sensor and the high-resolution signal it receives (Nedap Identification Systems, n.d). The recognition algorithm is also designed logically and effectively. The camera is set up to rec-

ognize license plate numbers on vehicles approaching the device from a distance of 4-6 meters. The device is configured via a web interface not only immediately after installation in the ASRN (this initial setup is called primary configuration) but also during operation. This allows for checking records in the event registers, identifying errors in recognized license plate numbers, and recording them in the log.

Access HSRN manages license plate recognition for vehicles numbered according to state standards (Nedap Identification Systems, n.d). The database contains license plate formats for the European region, as well as Russian and Belarusian license plates. If a license plate number does not correspond to regional standards, the system will flag the error and issue a signal. Additionally, checkpoint staff must inspect the vehicle, as the system will not allow access. Such suspicious vehicles are recorded in the HSRN Access log.

The product has a large memory capacity, allowing vehicle numbers to be categorized into different access levels at the checkpoints. One such category is the whitelist, which includes vehicles known to the organization, such as company-owned vehicles, staff vehicles, those belonging to suppliers, and vehicles used by emergency or state services. In contrast, the blacklist contains vehicle numbers that are either unknown to the system or associated with individuals who have previously committed rule violations. This classification mechanism enhances security and streamlines access control procedures across the facility. The HSRN Access system can control a gate barrier or an electronic lock, as it has a relay output built in.

In HSRN Access, the device's own memory is provided by an internal card (Nedap Identification Systems, n.d). This memory stores every recognized license plate number and a photo of the vehicle, highlighting the device's relevance as a solution for autonomously controlling vehicle access to areas such as parking lots or industrial facilities. The access control system follows the typical scheme and adheres to the standard card reading mechanism, which is not additionally integrated on the controller side.

Another example of an HSRN system is Hikvision by the manufacturer of the same name. This product is presented to consumers as a leading solution among video surveillance hardware developers (Hikvision, n.d.). The Hikvision HSRN is a network camera with advanced video analytics capabilities. Designed for professional use, it handles a variety of tasks efficiently.

The camera captures 3-megapixel resolution images, with clarity enhanced by the built-in smart codec (Hikvision, n.d.). All of the camera's video traffic is high quality, transmitted as an HD stream at 50 fps. The camera has numerous functions to address the challenges of modern video analytics. It can detect and identify intruders crossing restricted barriers. The system is also capable of counting the number of individuals entering a specific area. It also can track the movement of vehicles in relation to the camera's position. Another important feature is the ability to distinguish and count faces.

In the 'Hikvision' HSRN, the camera's advantage lies in its built-in hardware module for recognizing license plate numbers. These advanced developments by 'Hikvision' ensure that frames are processed quickly and efficiently.

This ASRN module performs a number of functions. It conducts video analytics of vehicle data, creates, and manages black- and whitelists. The module is easily and quickly customizable and can export lists and reports to Excel files. It provides license plate numbers and their photos from the database. The system recognizes license plates both day and night, for vehicles moving at speeds up to 120 km/h. It records entry, exit times, and calculates vehicle stay duration. Additionally, it generates various reports.

In the Hikvision HSRN camera, memory is built-in and supported by a card, allowing the database with numbers to be formed directly on the device (Hikvision, n.d.). This enables the system to operate autonomously, without the need for an operator at the checkpoint or elsewhere in the protected facility.

The structure of the 'Hikvision' ACP includes the following configuration blocks. It consists of a unit with video recording functions, at least four cameras for video recording, accumulators for camera power supply, and memory stored on a hard disk.

The Hikvision HSRN generates reports on vehicle movements over specified intervals. It displays vehicles and their license plate numbers along with photos, dates, and periods of stay in the territory, including passages near the camera. The Hikvision HSRN also has a search function to find license plate numbers and import all related movement data, including photos, videos, and timestamps, into an Excel file (Hikvision, n.d.).

2.3 LPR methods

The process of license plate recognition (LPR) occurs either in software or hardware ASRN. In general, the solutions follow similar algorithms and conduct the necessary processing of photos or videos. LPR is carried out using various methods and algorithms that allow for the recognition of a vehicle's license plate.

The LPR process involves multiple stages and is multi-step (Figure 3).

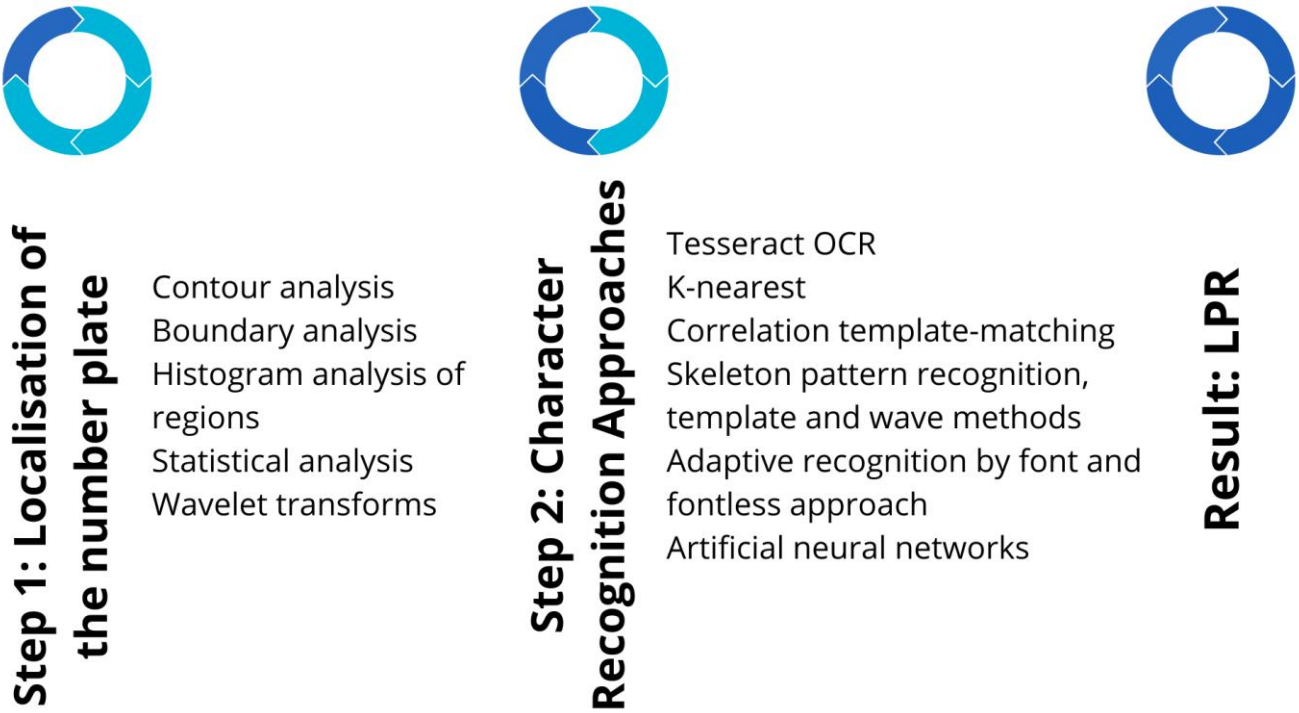


Figure 3 Stages and steps in car license plate recognition

Stage 1 involves localizing the license plate through a preliminary search. This can be accomplished in several ways:

1) Contour analysis extracts boundaries and shapes, then analyses them. LPR starts by searching for a rectangular contour, but it only triggers on well-defined contours with even borders and high-resolution photos. The camera can only recognize the license plate number without interference of fog, precipitation, or dust.

During contour analysis, the image is filtered, borders are detected, and contours are highlighted for further analysis. However, this method has limited practicality due to challenges such as vehicle movement and unclear license plates caused by weather factors, interference, or dirt on the plate.

2) Analysing part of the boundaries as a LPR method yields high and stable results, making the application more practical. The ASRN partially analyses the number by recognizing contours and then labelling each read vertical line. If the straight lines are close together, with minimal Y-axis shift and correct distance correlation, it is assumed that the number lies within the space bounded by these lines.

3) The principle of histogram analysis of areas is a popular LPR method. It assumes that the area containing the number and its components will have different frequency characteristics compared to other areas. The resulting image is analyzed to identify and label each region with high-frequency characteristics. The image is projected along the Y-axis (or sometimes the X-axis), and the projection must align with the contours of the number. A drawback of this method is that the background of the number may include inscriptions or other details, which can lead to false positives in LPR. Therefore, a clear frame from a close distance is required.

4) Methods such as statistical analysis and classifiers are introduced to address the limitations of earlier approaches. Early methods often struggle with LPR when numbers are dusty or covered in dirt, as these conditions prevent clear boundary definition and hinder the collection of accurate statistics for processing.

Applying classifiers is more reliable. This group of methods analyzes the vehicle photo to identify typical number contours in a given area, such as the ratio of lines, location of dots, and color gradients. This LPR method works even under complex conditions and interference.

LPR in a trained Haar cascade is based on the Viola-Jones method. Scientists have developed Haar features, dividing the vehicle's license plate region into rectangles with subareas, also rectangles, but with different properties. Algorithms often start by identifying the boundary of the number. Some methods, however, do not detect the boundary directly but still reference this element during analysis.

The efficiency of this method is low if the number plate is not dusty, but the frame is light, chrome-plated, and the vehicle body has a white enamel coating. This is because there are fewer such vehicles, and the system cannot learn effectively from them. More often, LPR occurs with vehicles having contaminated or dirty license plates. The quality of training in such cases is not satisfactory, making the application of the method less effective.

5) LPR process involves detecting and extracting the license plate using wavelet transformations. The process can be outlined in the following sequential steps: first, the image is processed using wavelet transforms. Then, the RAS data is transformed into the wavelet domain for filtering. Finally, the license plate is estimated and extracted from the processed image.

Wavelet transforms offer efficient processing by analyzing the image in different frequency components (Gonzalez & Woods, 2014, p. 1104). In the context of LPR, the number on the plate is considered the useful signal, while other image details are considered noise. The principle of wavelet transformation combined with filtering allows for effective processing even with low contrast images, such as those with headlights directly facing the camera.

The scale for wavelet transformation is selected based on the size of the license plate area and the frequency of alternating bands (black and white) in the image. For instance, a scale of 2 is sufficient when using a Gaussian wavelet of the 6th order.

Wavelet transforms come in many types due to their discrete and continuous properties. High-quality RAS results are typically obtained using symmetric wavelets. Discrete wavelets, especially those from the coiflets family, have proven to be highly effective, while continuous wavelets from the Gaussian family have also shown efficiency.

In vehicle photos with license plates, wavelet transforms capture all rows in the image, resulting in a one-dimensional rather than two-dimensional spectrum. LPR relies on detecting alternating bands—black and white—on the plate, which is achieved when wavelet coefficients show well-differentiated values. The black band in the wavelet transform appears as a plateau that then sharply drops.

LPR using wavelet images depends on this characteristic. The system not only detects the plateau but also checks the symmetry and steepness of the slopes on both sides. Simple threshold clipping based on wavelet coefficient data is ineffective because it fails to address the high values of the coefficients and the grating data introduced in LPR (Wu et al., 2009, p.507-510).

To improve boundary detection, LPR sometimes uses the second derivative of the wavelet coefficients, which often has higher values (Hill, Pearce, & Stromberg, 2020, p.8). To compute the second derivative, the wavelet image pixels are binarized and then clustered using a cellular automaton (CA).

In binarization, all pixels in the wavelet image are categorized into grayscale levels: approximately 70% of pixels closer to black are set to 0, while the top 30% closer to white are set to 1. A white CA cell will "survive" if it matches its eight neighboring cells. The rules are designed to preserve merged single pixels in the license plate region ("survive") while eliminating isolated single pixels ("die out"). CA processes the wavelet image so that single pixels lack a specific shape or orientation and are arranged randomly. The CA calculates the area of the cluster (with stable and specified pixel count) and compares it with the length and width proportions (Gonzalez & Woods, 2014).

The image then moves to Stage 2, where LPR is carried out through character recognition on the license plate. This can be achieved using various approaches:

1) Tesseract OCR Software: Tesseract is a popular OCR software known for its openness and ability to automatically recognize both character and text data. It is compatible with all major operating systems and is recognized for its high accuracy and stable performance. However, Tesseract struggles with RAS if the license plate is dusty, damaged, or deformed. Its accuracy is notably lower under these conditions, with recognition rates in real-world scenarios not exceeding 20-30%.
2) The K-nearest neighbor (KNN) approach, while considered primitive, often performs better in license plate recognition (LPR) compared to some support vector machine (SVM) projects and neural network solutions.

The KNN processing mechanism works as follows. First, a database with image records is created to categorize numbers into classes without errors. Then, the proportion of inter-character distance is established. After binarizing the image, an XOR operation is used, which is considered standard and optimal. LPR is performed character-by-character by calculating the distance between each character in the image and characters in the database. The number k, reflecting the nearest neighbors, may include symbols from several classes. However, a symbol is assigned to the class with the majority of its neighbors.

Despite its advantages, the KNN approach has weaknesses. Computing distances, binarizing images, and applying XOR operations can be time-consuming and challenging. Binarization can make symbols unrecognizable, and interference from factors such as precipitation or dust can reduce efficiency. Nevertheless, an extensive database with symbols in various conditions provides a solid theoretical foundation for the effectiveness of the KNN approach in LPR. The KNN algorithm is not only simple and transparent but also has few obstacles during debugging and tuning. This simplicity allows LPR systems to be adjusted for optimal results with minimal effort (Vizilter, Zheltov, & Knjaz, 2018, p.464).

The correlation method is another approach used in LPR to recognize images and classify them into categories. Each class is described in Cartesian space by specific features and characterized as a reference area. During the LPR process, the recognized object is compared with the class that shows the maximum similarity to the reference template. This approach is also known as 'template matching' or 'LPR via matched templates,' emphasizing the process of comparing input image characters with patterns stored in the system database. In this method, license plates are assumed to be in a consistent font, and noise from photo capture or interference is modeled using a Gaussian distribution. Prior to image binarization, there is no information on signal amplitude, which complicates the adjustment of brightness levels relative to the symbols.

LPR involves calculating the covariance between the input signal and a hypothetical signal. The formula used is:

cov (X,Y) E [(X EX) (Y EY)],

where: X is the input signal, Y represents the hypothetical signal, E denotes the expected value (mathematical expectation). Offsets and rotations may also be accounted for in the calculation.

When LPR selects the most accurate symbol, it accounts for rotation and bias by evaluating each symbol against a set of hypotheses. The photo of the symbol with the maximum covariance, covering all hypotheses, will lead to the recognition of the symbol, including adjustments for bias and slope. However, similar symbols on the license plate (e.g., "o"/"c" or "v"/"u") can create recognition challenges. This issue is addressed by incorporating a weight matrix of coefficients for each symbol. To overcome uncertainty, an intensive search for possible options may be necessary, which can justify the use of an adaptive approach.

The correlation method has its strengths. It provides reliable LPR results, as noise variants do not significantly degrade recognition quality. It can effectively read numbers from plates even when they are covered in dust, dirt, scuffs, or deformations due to the consistent font used. However, this method has its challenges. The primary drawback of the correlation method is the computational cost associated with the required operations.

LPR using recognizable skeletal images is performed through a process known as skeletonization or thinning. To extract each character from the plate as a skeleton, the following methods are used. Shchepin's method treats each number as consisting of two contours: the outer contour and the inner contour. The process starts by identifying the topmost point on the left side. The algorithm then traverses the contour point by point, evaluating each point based on its 8 neighbors. Points that are not endpoints and whose removal does not affect the connectivity of the neighboring points are deleted. The algorithm proceeds by analyzing each point, comparing it with its 8 neighboring points to decide whether to delete it, and then moving to the next point, ensuring that each new point is adjacent to the previous one. This process continues until all points in each layer are deleted, leaving a skeleton of the symbols made up of the undeleted points.

Another method is template-based skeletonization. In this method, templates are used to remove redundant points. The area is correlated with the template, and deletion begins with the central black pixel. The image is processed multiple times to ensure that no extraneous dots remain. The LPR process prepares the symbol skeleton by focusing on key points, which are the connections formed by the edges of the symbol and their endpoints.

The skeletonization process using the template method involves several steps. First, a new stack is created for the number to be recognized. The coordinates of elements, including edges (with their start and end points) and all branching points of the skeleton, are recorded. Then, a point from the skeleton with specific coordinates is selected and recorded. The process enters a loop where a point is chosen from the stack. An edge is constructed from the selected point, either leading to a branching point or an endpoint on the skeleton. The path from the selected point to either the endpoint or the branching point with a known edge is written into an array. The new branch should point to the edges connecting at the recorded point, which is updated as a sequence of edges in the array. The branching point is recorded in the stack. Finally, consistency is verified with the conditions specified in the third step. This process ensures that the skeletonization of the number is accurately captured and represented. The stack describes the skeleton in a general and approximate manner, primarily focusing on preprocessing tasks such as removing short lines and closely spaced triodes.

Another approach, known as the wave method, involves launching a spherical wave along the license plate number and analyzing its movement. The process begins with skeletonization, creating a simplified, thin representation of the number. A spherical wave is then launched along the skeletonized image of the number. The movement and displacement of the wave's center of mass are analyzed systematically. Based on the displacement of the center of mass, new waves are generated to track the lines of the number. These tracked segments can be smoothed to improve accuracy. During this process, the skeleton is optimized. The wave's motion helps in tracking the lines by examining how the wave travels from the center of each segment and stops at the segment's extremes. However, the initial bitmap image obtained from this method may introduce distortions, especially when the symbols are small. The optimization process addresses these distortions by refining the skeleton. It describes the segments as edges in a sequence, which is useful for LPR. Any vagueness in the skeleton edges is analytically resolved by evaluating the deviation of each edge from a straight line. Small deviations are corrected by merging the edges into a cohesive whole. Additionally, the skeleton is optimized in branching zones, where the wave transforms into half-waves, improving the accuracy of the representation.

The adaptive recognition method is applied by different approaches considering font dependency. In the font approach, or font-dependent LPR, recognition is performed on the data of

font work — measurements and analyses, which are compared with font etalons from the database. LPR is achieved and improved with training of the algorithm so that the font program recognizes the number, but only in the font studied by it. LPR needs to be trained to recognize all types of fonts so that the scope of application of HSRN is as wide as possible. The weaknesses in the font-dependent LPR approach arise from a number of significant factors. If not trained, the algorithm does not recognize a font, and all fonts are recognized by their databases and blocks (Saha, 2018). However, the font approach is important, because the number is recognized quickly, as its font is described to the details of each character and learned by the algorithm. In this case, the LPR occurs to be accurate and reliable.

The principles of the font-independent approach are to measure and analyze font properties with reference to font size. For the font-independent approach, weaknesses include reduced quality and accuracy compared to font-dependent approaches, and calculations for the LPR reliability coefficient reflect its low value. However, non-font-dependence comes with many advantages. LPR is universal — the algorithm handles any font with any kegel. It is also technological, as the algorithm is quickly trained, including through automated methods.

In the context of LPR, neural networks operate by processing input data through a set of interconnected neurons. The neural network transforms an input vector into an output vector using learned weights. The range of LPR problems that a neural network can solve is extensive, provided the network is properly trained. During training, neurons interact with standard reference data and adjust based on this input. When a signal that deviates from the benchmark is presented, the network compares it to known patterns, recognizes similarities, and updates the database with new benchmarks. A neural network trained on a diverse set of input signals, including those with deviations from benchmarks, can achieve high learning rates and robust performance (Sarfraz, Ahmed, & Ghazi, 2010).

3 Automated LPR

3.1 Approach

Automating the recognition of number plates, which serve as the means of state identification for vehicles, is enabled by ASRN. However, its development is a complex process with significant challenges, as a state-of-the-art method must integrate advanced data processing, machine learning principles, databases, and a computer vision unit.

In ASRN, the starting point is data preprocessing, which often involves dealing with issues such as uncorrected noise, low contrast, and unsegmented characters. To segment characters, methods based on connected components have been tested and found effective (Rao, Rao, Babu, & Goplani, 2013). Gradient map methods have also shown efficacy, while convolutional neural networks (CNNs) are increasingly favored for character segmentation (Jo et al., 2020).

Once the characters are segmented, they are recognized using classification algorithms implemented in ASRN. Pattern recognition methods (e.g., correlation with a reference), along with other techniques such as statistical and probability distribution methods, aid in the accurate recognition of the symbols.

Currently, the most promising approach for ASRNs is the use of neural networks. CNNs are particularly effective because they perform highly accurate processing by evaluating data through hierarchical features, which are then extracted for analysis in the ASRN process (Lecun & Bengio, 1995, pp. 255–258).

The implementation of modern strategies in the latest versions of ASRN has significantly improved the reliability of LPR. Number grammar rules restrict characters to certain combinations, ensuring accuracy (Špaňhel, Herout, & Zemčík, 2016). Checksum verification confirms that LPR is conducted with sufficient reliability. And multistage recognition, which uses a cascade of classifiers, enhances accuracy through iterative processing.

ASRN requires algorithms capable of handling data captured under challenging conditions, such as varying vehicle number plate illumination, adverse weather, plate contamination, or unique vehicle design features. To address these challenges, specialized methods are

needed, including normalizing photos, correcting perspectives, and training the algorithm with a comprehensive database that covers a wide range of LPR complexities.

ASRNs are developed using efficient algorithms, with evaluation criteria primarily based on accuracy (including both overall LPR accuracy and symbol accuracy) and processing speed (time characteristics of the LPR procedure).

The improvement of ASRNs involves advancing algorithms through deep learning, with a shift towards synthetic images and processing methods. This approach aims to make LPR less dependent on external conditions, ensuring more reliable and flexible results.

A theoretical understanding of current ASRNs provides a general algorithmic perspective on LPR. The task of recognizing number plates on a car should be approached step by step, following the sequence of operations outlined in Figure 4.

Figure 4 General scheme of solving the car number plate recognition problem

3.2 Algorithm design

The LPR algorithm is structured to include several pre-processing steps that ultimately result in the recognition of a car's number plate. The characterization of LPR for ASRN purposes is illustrated in Figure 5.

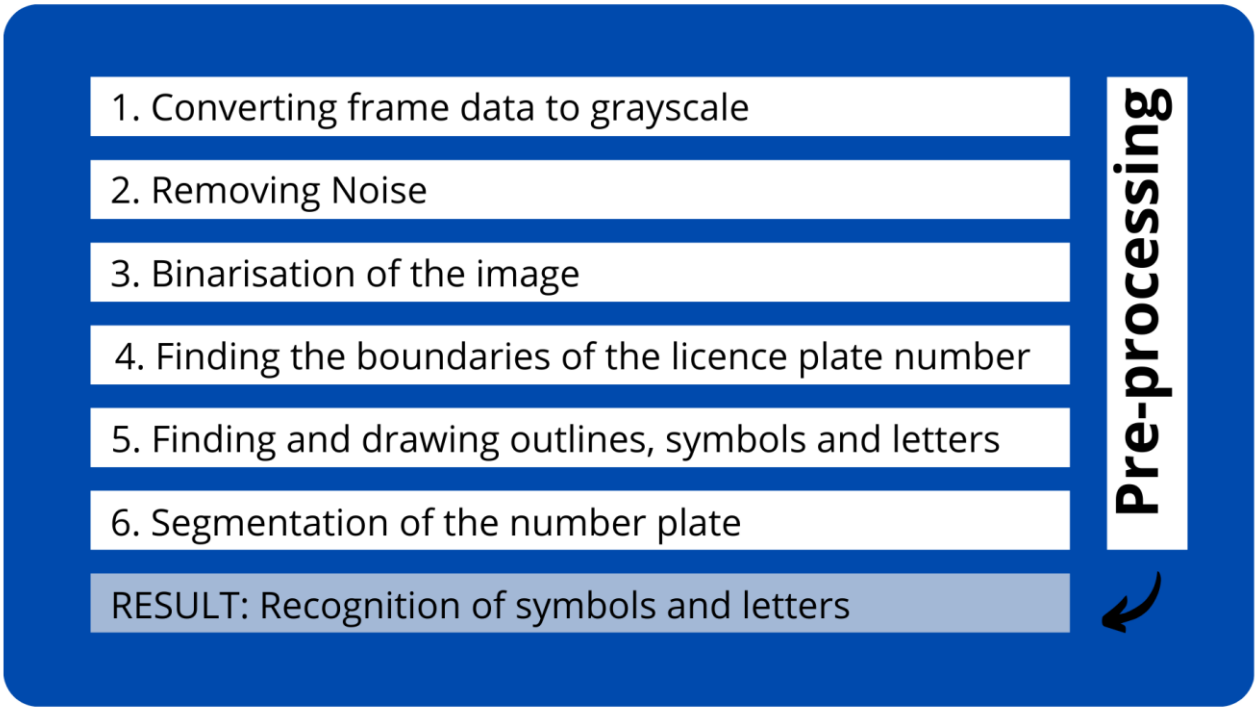


Figure 5 Structural diagram of the recognition process

In ASRN, the LPR algorithm relies on a strictly ordered sequence of operations to ensure accurate symbol recognition (Marzuki et al., 2024). Initially, the car plate data is transformed into a grayscale image. This is followed by noise reduction and removal procedures, enhancing image clarity. The next step is image binarization, which prepares the image for contour analysis. Once binarized, the boundaries of the license plate are defined by detecting its contours. These contours include both symbolic and alphabetic characters, which are then identified and organized for search-based processing. The segmented license plate is subsequently analyzed, and the LPR process concludes with the final recognition of all characters.

3.3 License plate processing

The LPR procedure enables the reading of car license plates from photographs. It is implemented through computer vision, which is currently employed in numerous applications for

the digitalization of urban and industrial environments and transport infrastructure. These applications include intelligent systems (IS) for parking management, road control and monitoring, and the creation of various security levels, including national security.

LPR is an algorithm that uses digital processing procedures to capture a photograph, extract, and recognize the designations on the plate—both symbolic and alphabetic.

In the preprocessing stage of number plate processing, LPR performs several operations. To improve sharpness, the algorithm enhances the boundaries of each symbol, making recognition clearer. Illumination normalization is applied to balance poor lighting conditions, ensuring the image is clear. Binarization is achieved by converting the photographs to black and white, so the symbols stand out against a white background.

Segmentation operations break the image down into individual symbols, forming two line directions: horizontal projections, which order symbols within lines and indicate gaps between lines, and vertical projections, which separate symbols within each line.

To recognize characters, each symbol is extracted from the segmented portion and compared to a reference, such as a letter or digit. In ASRN, character recognition can be achieved through pattern matching, neural networks, or machine learning algorithms.

In the post-processing stage, noise is eliminated, and character recognition is corrected. Noise elimination is crucial to prevent false positives and ensure accurate recognition. The correction process is also positive, allowing ASRN to remember common LPR errors and correct incorrect recognitions.

Photographs in grayscale mode are converted to color, with the entire color palette rendered in shades of gray. This conversion is based on a reference table that uses samples with varying brightness levels of white.

Traditionally, photographs are converted into staggered images, where a uniform series of optical densities forms neutral gray fields (Picture 6). This mode is commonly used by colorists and color scientists and is important for recognizing the tonal quality of photographic and scanning materials. It is widely used in photographic printing, photocopying, and publishing.

Picture 1 Converting an image to grayscale

The brightness of the photograph needs to be evaluated, which is done computationally using the formula:

$$Y' = 0{,}299 * R + 0{,}114 * G + 0{,}587 * B, \quad (1)$$

where Y' represents the brightness level of the photograph (OpenCV, n.d.). The variables in the formula are characterized by numerical point values in different color channels:

R represents the red channel,

G represents the green channel,

B represents the blue channel.

To remove extraneous noise from the photo, filtering is applied. The Gaussian filter (Picture 2) is particularly effective for removing "salt and pepper" noise, smoothing the contours, and preparing the image for further contour analysis.

Picture 2 Elimination of extraneous noise

The photo is then sent for binarization (Picture 3). Binarization converts the frame, which is currently in grayscale mode (although color processing is also possible), into a monochrome image. This process simplifies each recognizable character by converting all points in the

photograph to either dark or light. Binarization creates a binary object from the photograph, similar in properties to a barcode or text.

Picture 3 Binarisation of the car plate image

Simultaneously, the photograph is prepared for contour detection. The contours of the symbols, highlighted by the binarization process, are detected using the Canny edge detector, which is more effective than the Sobel filter (Jain & Patel, 2018). Picture 4 shows the results of the contour detection on the processed car license plate photograph.

Picture 4 Contour search result

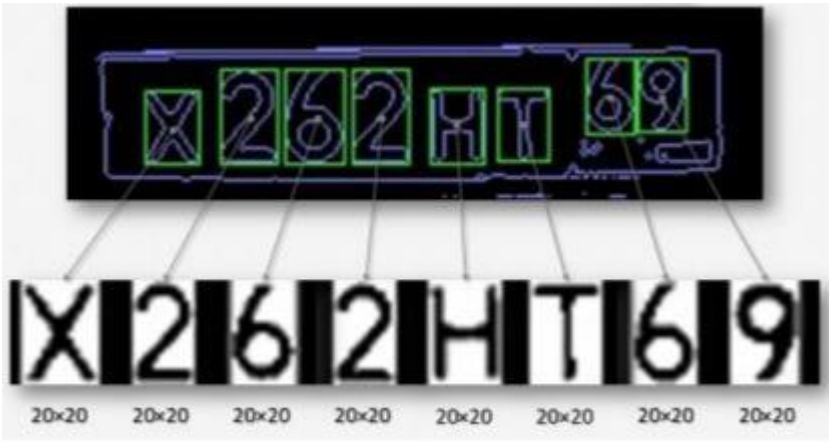
For segmentation, each detected closed contour is outlined with red rectangles. These rectangles define the areas to be segmented. Picture 5 displays these rectangles around the detected contours.

Picture 5 Result of closed contour extraction

However, only some of these areas are relevant for LPR. Given the known proportions of length and width for plate fonts, only the areas containing numeric and alphabetic symbols should be considered. Picture 6 shows the photo with the selected target areas, which will be evaluated for recognition by the ASRN algorithm.

Picture 6 Result of selecting target areas for recognition

Then the coordinates are taken from the rectangles to segment the frame of the photo with the number plate by these segments (shown in green), to cut out the segments before recognition. Picture 7 demonstrates the segments from the photograph of the vehicle number plate.



Picture 7 Result of segment extraction

After extraction, each segment should be resized to 20×20 pixels to standardize its dimensions. The resized segments are then stored in a temporary folder, from which the ASRN will retrieve them for recognition. Additionally, the segments are numbered sequentially from 1 for the left-most to 8 for the right-most, or 9 if the region contains a three-digit designation.

3.4 Comparison algorithm development

In the developed ASRN, we propose comparing segments using an algorithm that applies LPR operations according to the diagram structure outlined in paragraph 2.2. We will use photos of cars where the license plate has not yet been isolated from the background (Picture 8) to demonstrate the problem statement for the algorithm.

Picture 8 Highlighted number plate

This process involves a classifier formed using Haar primitives, for which values are calculated based on all features (Viola & Jones, 2001). The algorithm should be trained with a set of photos that clearly show and frame the license plate area. It will then search for primitives and describe the license plate features by calculating their values. After the calculations, each value is recorded as an entry in an XML file. The Haar cascade should be trained using the utility provided in the OpenCV package.

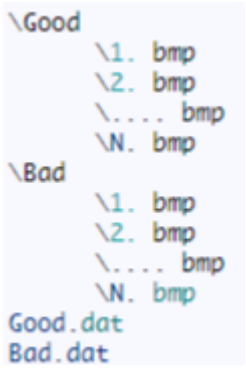
The training materials for the ASRN consist of photos of vehicles with the following characteristics: vehicles in real-world photos, ensuring the sample closely resembles the LPR object. The license plate recognizer needs to be trained using real vehicle photos that include interference, noise, and varying illumination conditions (Viola & Jones, 2001, p.511-518). Additionally, create a sample of negative photos where it is impossible to recognize the license plate due to its absence. All photos should be taken outdoors, under various conditions that reflect the anticipated recognition scenarios.

The LPR detector will produce consistently high results if the training photos are prepared with an equal number of positive and negative examples, approximately 500 in each group. The ASRN detector trains faster than a face detector, which typically requires around 30-40 thousand examples for each group. Although increasing the volume and diversity of the training sample lengthens the process, it also enhances the efficiency of the ASRN.

The photo sample is prepared during the training preparation stage. We use the Picture Cropper program to process the sample. Vehicle photos, along with their corresponding license plates, need to be manually saved and categorized into positive and negative samples.

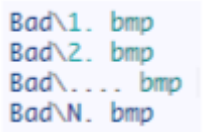
To train the LPR algorithm, place the photo examples into designated folders: the "Good" folder for positive photos, and the "Bad" folder for negative (non-compliant) photos. The OpenCV environment, following general naming conventions, does not allow the use of dots, spaces, or special characters in the names of files containing example photos.

A description file that outlines the characteristics of the examples is placed in the examples folder. The conventionally established filenames are ‘Good.dat’ and ‘Bad.dat’. The hierarchy of files and folders should be consistent, as illustrated in an example (Picture 9).



Picture 9 Example of a hierarchy

The structure of the example description files varies. For negative photo examples, the paths are listed relative to specific frames (Picture 10).



Picture 10 Structure of non-compliant objects

However, the positive examples file requires a more detailed format. The path, photo location, and size values must be specified (Picture 11).

```
Good \0.bmp 1 0 0 414 148
Good \1.bmp 1 0 0 568 164
Good \....bmp 1 0 0 440 144
Good \N.bmp 1 0 0 590 182
```

Picture 11 Structure of non-compliant objects

The entries include the following information:

‘Good\0.bmp’ - the path to the photo in the description file

‘1’ - the number of positive examples in the photo

‘0 0 414 148’ - the coordinates of the rectangle where the car license plate number is located in the photo

If a photo contains multiple license plates, the entry should include all relevant details: ‘Good\0.bmp 2 100 200 50 50 300 300 300 25 25 25’. Ideally, each example should be on its own photo and have corresponding coordinates.

To begin processing photos, start by launching Picture Cropper and opening a folder containing one type of example or the other. Each photo in the program is cropped manually. Picture Cropper then creates the Good.dat file and its associated records. The Good folder is used to store the cropped frames, which are now ready for training.

Once the folder and database of examples are formed, you can proceed with training the algorithm. The training process begins by accessing the directory containing the Good.dat file via the command line. To launch the set of examples with pictures, use the following command in the console: opencv_createsamples (Picture 12).

```
opencv_createsamples.exe -info Good.dat -vec samples.vec -w 180  -h 40
```

Picture 12 Creating a set of compliant images

The command entries are interpreted as follows:

‘Good.dat’ is the file containing descriptions of the photo samples. The file path should be either fully specified or referenced relative to the location of the opencv_createsamples.exe file on the disk.

‘-vec samples.vec’ specifies the file where the database of examples is stored after conversion to a common format. The path should link this file to the opencv_createsamples.exe program.

‘-w 180 -h 40’ indicates the size of the template, ensuring that the proportions match those of a car license plate. The aspect ratio of Russian license plates is approximately 4:1. It is crucial for the template to be compact in size, verified by the visual clarity of the license plate in the example photo. If the template size is too large, the training process will take longer.

The program should generate a samples.vec file, which saves the photos in BMP format with a proportional size of 4:1 (w:h). It is important to verify that the samples.vec file is present in the folder.

The task of calculating the final cascade is performed by the program opencv_traincascade.exe. This program, along with opencv_createsamples.exe, should be placed in the same folder. Typically, training a cascade with 500-1000 examples takes between 12 to 24 hours; in this particular case, it took nearly 24 hours to train the algorithm.

The training command is shown in Picture 13.

```
w:\learn>opencv_traincascade.exe -data haarcascade -vec samples.vec -bg Bad.dat -numStages 16
 -minhitrate 0.990 -maxFalseAlarmRate 0.5 -numPos 400 -numNeg 500 -w 180 -h 40 -mode ALL -pre
calcValBufSize 256 -precalcIdxBufSize 256
```

Picture 13 Training cascade

‘-data haarcascade’ is the address of the folder where to put the obtained results. The command begins from the root folder of the program. The following elements should already be prepared:

vec samples.vec - specifies the path to the file containing the positive image samples.

bg Bad.dat - provides the path to the description file containing the negative samples.

numStages 16 - indicates the number of cascade levels used in training. More training levels can improve accuracy but will increase the time required. Typically, 16 to 20 levels are sufficient.

minhitrate 0.999 - sets the hit rate coefficient, which measures training quality. This value reflects the proportion of error-free detections out of the total number of detections. A score of 0.999 means that in a similar sample, the target could be missed up to 1 - 0.999 = 0.1% of the time. However, with a high hit rate, the ASRN is more likely to produce false alarms. Proper sample preparation generally allows for a coefficient between 0.99 and 0.999. If the sample is weak (e.g., containing a small number of photos, or photos with interference and noise), the value should be set lower.

maxFalseAlarmRate 0.4 - sets the acceptable level of false alarms.

numPos 400 - specifies the number of positive samples. The minimum coefficient, minhitrate, can cause high rejection rates, so it is standard to set numPos at 80% of the total sample.

numNeg 500 - specifies the number of negative examples.-w 180 -h 40 - defines the size of the primitives in the negative examples.

mode ALL - indicates whether to use the complete or incomplete set of Haar primitives.

precalcValBufSize 256 -precalcIdxBufSize 256 - sets the amount of memory allocated for the process.

The algorithm will conclude by generating the cascade.xml file in the haarcascade folder. This file contains the trained cascade, which is used to recognize objects (Baggio, 2012).

Once the search for the license plate is complete, the next step is to recognize the plate itself. This involves performing normalization for both tilt angle and scale.

The rotation of the license plate can vary within a range from -10° to +10°. During processing, the algorithm will incrementally rotate the image by 0.1° for each new frame, independent of previous results. The algorithm considers all possible rotations and selects the one that best matches the correct orientation, identifying it as the recognized number.

The maximum tilt angles are illustrated in Picture 14.

Picture 14 Maximal angles

In the photo, we need to computationally determine the lower boundary. The algorithm selects the corner whose boundary is higher than the others as the desired corner. Regarding the scale of the photo, which is close to the rotation, a corridor of 1-3 with a step of 0.1 is set.

The known angle of the license plate prompts us to search for the boundaries, and we identify the lower boundary by analyzing the brightness histogram. Testing has shown that it is unnecessary to use the brightness histogram to find the upper boundary, as this method fails 50% of the time. Therefore, the Haar cascade will be trained on the symbols and their upper bounds to determine the location of the upper boundary of the plate. However, in some cases, the Haar cascade may be inconclusive, particularly if the photo has low resolution. In these instances, an alternative algorithm is used to find the boundary by applying a brightness histogram.

When cropping a car license plate, the upper and lower boundaries are identified, but the side boundaries remain undefined. The brightness histogram method can help here, especially if the car's color is recognized by the algorithm. White or light-colored enamel interferes with binarization, causing the photo's left and right edges to appear white. Conversely, if the

enamel is dark or black, the edges appear black, while the plate itself has a white background. To address this, we construct hypotheses to find the border on both light and dark cars. The hypothesis that results in a border closest to the center of the license plate is considered the most accurate.

Machine learning relies on the quality of the feature vector, but a CNN requires a well-designed network architecture. For our CNN we use the following notation for its layers:

input: The layer receiving the photo pixels.

conv: Convolutional layer.

pool: Subsampling (pooling) layer.

fully-conn: Fully connected layer.

output: The layer that provides the predicted class for the photo.

To classify a photo, we apply the architecture depicted in Figure 6.

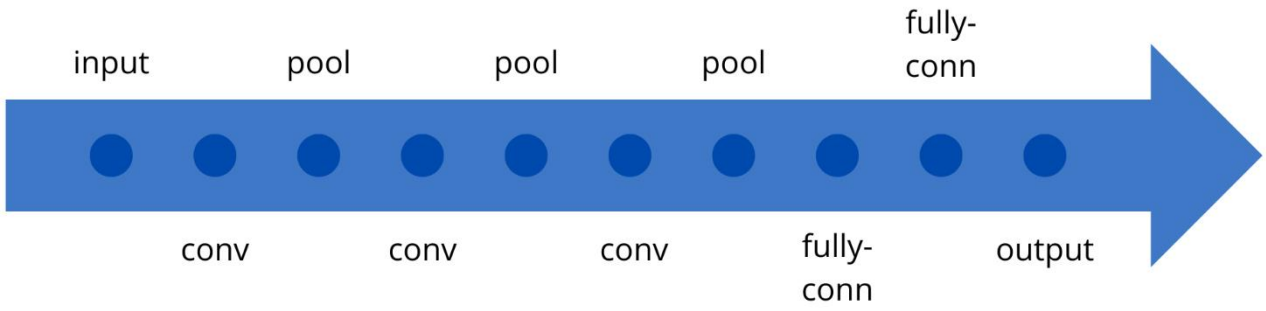


Figure 6 Neural network architecture

The number of layers can vary significantly (conv -> pool), but using fewer than two layers in the CNN will generally lead to failure. The number of fully connected layers should be at least one or more.

4 LPR software design and implementation

4.1 Technologies

ASRN is written in the Python programming language, utilizing the OpenCV, PyTorch, and TensorFlow libraries. Photos are preprocessed using the OpenCV library, and characters are recognized through a call to TensorFlow.

4.1.1 Python

Neural networks in Python are popular, despite the availability of many other programming languages. Python allows developers to create complex algorithms in a short amount of time. It is characterized by its simplicity, brevity, and expressiveness. Additionally, Python has a powerful mechanism for interoperability with C/C++, enabling fast computations (Norvig, 2024). This makes it possible to create both simple and complex neural networks in Python.

Neural networks are often small programs, but they frequently need to be modified to optimize architecture, data preprocessing, and other parameters. Therefore, issues with legacy code are minimal, while the need for rapid development is high. Building neural networks in Python is a solution that meets these requirements better than using C++ or Java (Foster, 2014).

Python is an ideal programming language for data processing. With Python it’s possible to read, analyze, and process large amounts of data, which is crucial for creating neural networks that require extensive data for training. Python has many libraries that greatly accelerate and simplify the processing of large datasets.

Python also has an extensive range of libraries for machine learning and neural networks. These machine learning libraries augment Python's basic functions, allowing the creation and training of neural networks using off-the-shelf solutions. Most of these libraries come with comprehensive documentation and have active communities, which greatly assist in the process of training neural networks.

4.1.2 PyTorch

PyTorch is a framework for the Python programming language designed specifically for machine learning (PyTorch, n.d.). It includes a set of tools for working with models and is used in natural language processing, computer vision, and other related fields. The framework is based on Torch, a library for the Lua language that is designed for mathematics and machine learning. However, its specialty lies in adopting the style and programming philosophy of Python.

PyTorch is open-source and freely available. A whole ecosystem has been built around the framework, consisting of many libraries for different purposes, making it a comprehensive and powerful tool for solving machine learning problems.

There are many solutions for machine learning, each differing in their approach. PyTorch has several important features that set it apart from the rest. One of PyTorch's key features is its dynamic computation graph, which adds convenience. A neural network or machine learning model is represented as a graph—a structure consisting of vertices and the connections between them. These vertices represent the 'neurons,' while the edges represent the 'synapses' that store and transmit information. The vertices store formulas, and the edges store weights—numerical values that are recalculated at each step.

In most other available solutions, the computation graph can only be modified before compilation. Once the program is compiled, the developer can only run a forward or backward pass through the graph. If the task is complex, this approach can significantly complicate debugging.

In PyTorch, the approach is different: the graph is not a fixed structure but is dynamically constructed during computation. With each pass, the graph is rebuilt according to new conditions. This gives developers more flexibility and allows them to implement more complex and non-standard solutions.

The second key feature is automatic differentiation. Finding the derivative is a challenging task for a computer. Sometimes, machine learning specialists solve it manually by calculating the derivative on paper. Another approach is to use approximations, but this carries a high

risk of losing accuracy due to rounding. Some languages attempt to find the derivative according to specific rules, but this can result in cumbersome and uninformative expressions instead of an exact result.

Automatic differentiation addresses this problem. The essence of this solution is to express a large and complex function through smaller, ‘base’ functions. If the values of the base functions are known at a certain point, both the value and the derivative of the complex function can be automatically calculated.

PyTorch supports automatic differentiation. Combined with dynamic computation, the framework constructs the graph and calculates weights in real-time, requiring the developer to write only a few lines of code. This is both convenient and enables the development of more complex projects.

Another reason to use PyTorch is its support for CUDA. CUDA is a technology that allows calculations to be run on both the CPU and the GPU. Deep learning is resource-intensive, as models require significant computational power during both training and execution. This is why graphics processors (GPUs) have become essential—they are powerful and well-suited for handling complex calculations.

Typically, all computations can be run either on the CPU or the GPU, and some frameworks offer separate versions for each. However, moving calculations from one platform to another can be cumbersome and inconvenient in these cases.

This is not the case with PyTorch. Thanks to CUDA support, computations can be easily moved to the GPU or even created directly on it. With just a single line of code, user can push a calculation to the GPU, perform complex computations, and then push it back to the CPU. All of this is accessible directly from the interpreter. However, there is a limitation: CUDA works only with NVIDIA graphics cards, and not all models are supported. Therefore, not every computer configuration is suitable for its use.

The framework allows the user to create even complex models and solve a wide range of problems. This is possible due to direct access to mathematical functions and the ability to write your own. Some solutions limit your ability to build non-standard models, but PyTorch is

not one of them. On the contrary, it is well-suited for developing complex and non-standard models. Additional advantages include CUDA support and optimization tools.

Thanks to its dynamic computation graph, PyTorch avoids some of the disadvantages found in alternative frameworks. Changes to the graph can be made in the process, simplifying debugging and making it easier to modify the neural network when necessary.

For beginners, one of the key strengths of PyTorch is its detailed, extensive, and clear documentation, which covers all the important aspects. Everything a user needs is well-documented, so if problems arise, user can always refer to the up-to-date documentation.

While PyTorch was once less popular than some other frameworks, it has recently caught up with TensorFlow in terms of usage and references. The framework has a large ecosystem and an active, supportive community that helps newcomers and continually introduces user-friendly development solutions.

However, PyTorch has some disadvantages that should be considered. PyTorch does not have built-in tools for creating graphs, charts, or other visualization methods, so user needs to rely on third-party solutions. The same applies to deploying PyTorch software on production servers to provide end users with access to the product. PyTorch lacks built-in tools for easy deployment on servers, requiring the use of external standards, protocols, and tools. As a result, professional work with PyTorch often requires the use of additional libraries. This means that a developer needs to be proficient in not just one tool, but several, in order to fully complete tasks.

4.1.3 OpenCV

OpenCV (Open Source Computer Vision Library) is an open-source library for computer vision algorithms, machine learning, and image processing (OpenCV, n.d.). It is written in C++ but is also available for Python, JavaScript, Ruby, and other programming languages. OpenCV runs on various platforms, including Windows, Linux, macOS, iOS, and Android.

OpenCV can be used wherever computer vision is needed, and it also supports machine learning applications. The library uses vectors, scalars, matrices, and ranges to store and

manipulate images, enabling mathematical transformations, image navigation, and many other operations.

With OpenCV, users can manipulate images as in a graphic editor: cropping, zooming in or out, and rotating. Programmers often use these features for preliminary image preparation before decoding, such as trimming unnecessary parts. Also, images can be converted to grayscale or completely black and white, which is important for recognition algorithms that work with desaturated images. Users can also adjust the color tone, blur, smooth, or geometrically modify the image. OpenCV uses object outlines, color segmentation, and built-in recognition methods that can be adjusted based on the object and the sensitivity of the algorithm. Newer versions of the library support not only images but also video processing. They can read video clips using codecs, analyze the content, and track movements and elements within the footage.

OpenCV for Python, JavaScript, Ruby, and other programming languages is widely used worldwide, including by companies like Google and Microsoft. As a result, there is an active community around the library, making it valuable for both beginners and experts. OpenCV documentation is available in multiple languages.

OpenCV is distributed under a free license for both educational and commercial use. The library's open-source code is accessible to any programmer, providing greater flexibility in working with OpenCV and allowing users to understand how specific functions are implemented.

The library is faster than large-scale, resource-intensive software for mathematical calculations, such as Matlab. Therefore, it is ideal for situations where quick image processing is required. Thanks to its high speed and cross-platform nature, OpenCV is well-suited for real-time image processing. This capability opens up a range of possibilities, from creating beauty filter software for social networks to developing systems that automatically raise alarms when suspicious activity is detected on cameras.

However, if an error occurs, it can be difficult to pinpoint where it originated within OpenCV. This can make debugging challenging for beginners. Additionally, OpenCV is designed for large-scale platforms, so if user runs it on a microcontroller or single-board computer, performance may be low.

4.1.4 TensorFlow

TensorFlow is a machine learning library, part of a suite of technologies that enables the training of artificial intelligence to solve various problems (TensorFlow, n.d.). The library was originally developed for Python and is most commonly used with it. TensorFlow is free, with open-source code available on GitHub, and it is actively supported by a community of enthusiasts.

TensorFlow operates with tensors—multidimensional data structures in vector space, commonly used in linear algebra and physics. This is where the library gets its name. In TensorFlow, tensors are used to describe graph paths, with vertices representing mathematical operations.

The library can run on the power of a standard central processing unit (CPU) or leverage the power of a graphics processing unit (GPU), with the mode being switchable in the code. There is also a specialized tensor processing unit (TPU) developed by the TensorFlow team, which can be accessed via Google Cloud services.

TensorFlow offers a high level of abstraction. The library is designed so that users don't have to worry about the technical implementation of abstract concepts. Instead, users can focus on describing the program logic and the mathematics, while TensorFlow handles the underlying calculations. This simplifies development and allows users to concentrate on the most important tasks.

TensorFlow is a popular library, which means many questions can be easily answered within the community. This active community not only advances technology but also creates new products and add-ons related to TensorFlow, and provides extensive documentation and tutorials. As a result, getting started with TensorFlow and maximizing its potential is relatively easy.

However, there are some downsides. TensorFlow is a Google product, and the company is known for its own technology standards. The first version of the library was developed for internal use, and even today, TensorFlow can sometimes exhibit non-obvious behavior, which can make debugging more challenging. These difficulties can be mitigated by thoroughly studying the documentation and using additional debugging tools.

When TensorFlow is used with a graphics processor (GPU), it can consume all of the GPU's memory, potentially reducing performance. For instance, if multiple models are running on different frameworks, TensorFlow may monopolize the GPU memory, leading to errors in the other models. To avoid this, it's advisable to manually limit the library's memory consumption.

4.2 LPR Software

The architecture for ASRN should be designed so that all components necessary for LPR are interconnected and account for the system's limitations and requirements. Typically, the ASRN architecture illustrates each module, their hierarchical relationships, and how they pair with one another, as shown in Figure 7.

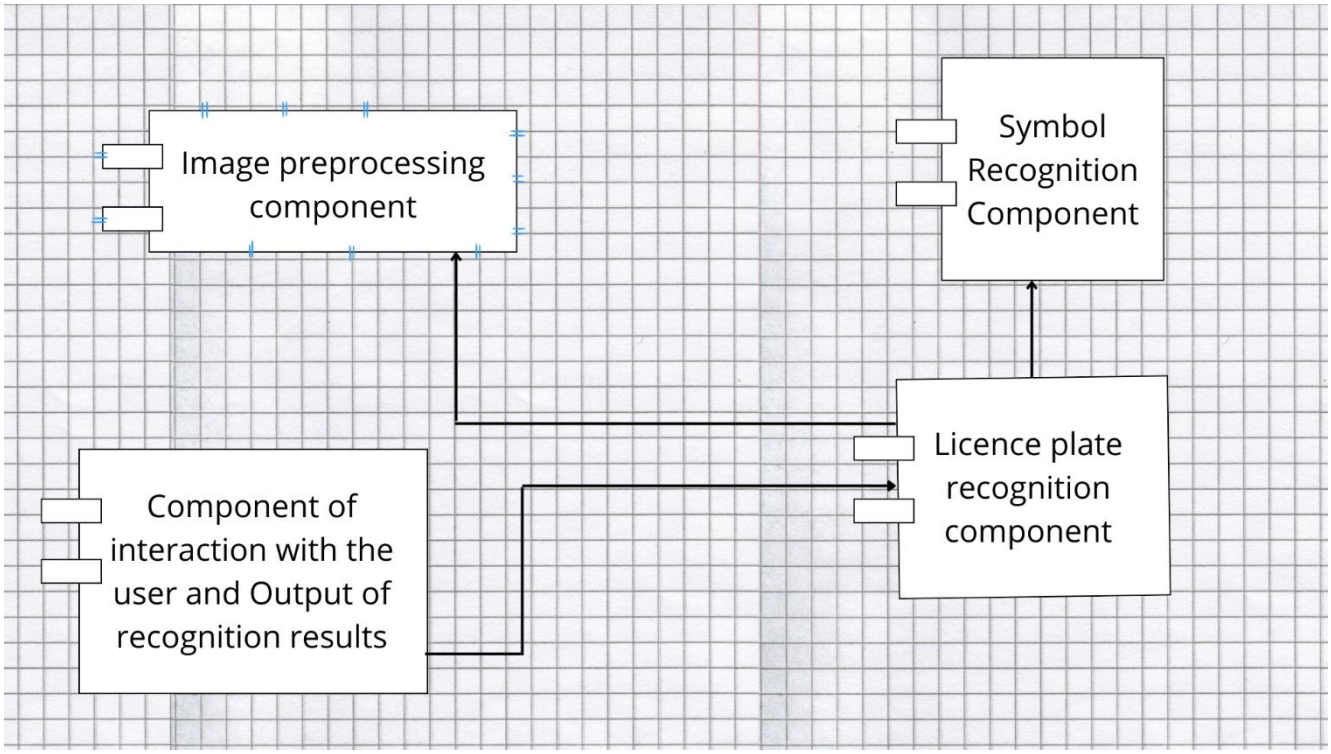


Figure 7 Diagram of software components

Since ASRN is a specialized program, the architecture design is closely tied to clear input data, the specification. The software product development is guided by all the requirements outlined in the specification, while also considering the functional architecture from the perspective of ASRN (Murygin, 2011).

In this ASRN, the architecture includes the following set of components. A dedicated component for user interaction and LPR output. The user initiates the ASRN, specifies a photo for

LPR, and the processed result is displayed on the screen. Also, a component specifically for LPR, which recognizes the vehicle's license plate from the photo. Next, a separate component for photo preprocessing, which converts a color or black-and-white frame to grayscale, removes noise, and binarizes the frame. And a final component for character recognition, which processes the prepared and segmented photos.

From a development perspective, ASRN’s microarchitecture is comprehensively addressed, balancing the interrelationship of architectural components with trade-offs in performance, power consumption, and project cost. ASRN is built on general principles of computer architecture, but it incorporates the latest technologies to ensure effective and stable microarchitecture, considering the project's functional requirements.

In ASRN, symbols are recognized through a query to the TensorFlow library. Character recognition is controlled by a variable, and initialization is performed to set the characters in a "white list"—digits 0-9 and all letters of the Cyrillic and Latin alphabets. The input receives a segmented character, which is ultimately recognized at the output. This operation compiles the recognized characters into a string, enabling ASRN to perform LPR.

When the software starts, a path to the image is specified, and upon activation, the processing and LPR tasks begin. The input data is the image of the license plate. Once the LPR process is complete, ASRN displays a window showing the recognized number.

The evaluation of ASRN's LPR accuracy is addressed through several sequential steps. The first task involves collecting photos in preparation for testing. This requires assembling a diverse set of photos where the license plate is captured under both favorable and unfavorable conditions, such as varying lighting, different angles, precipitation, fog, on different vehicles, and at different times of day. These photos should be collected under real-world conditions.

To annotate the photos, the license plate area and the license plate number itself must be manually identified and recorded. ASRN uses these annotations as reference records, against which the accuracy of LPR is measured.

The training process involves adjusting ASRN using a training set of photos, guided by machine learning algorithms. Once the ASRN is fine-tuned, it is tested on a separate validation set to optimize performance.

To determine accuracy, a test set of photos is then processed by the ASRN. Accuracy is evaluated as the fraction of LPR results that match the reference records. The LPR system is considered highly accurate if this fraction reaches 80%.

In addition to general accuracy, the evaluation includes informative metrics such as LPR speed (the number of plates recognized per second) and resistance to interference (e.g., performance under low light, midday sun, or adverse weather conditions).

The analysis of ASRN's results highlights potential areas for improvement. Additional tuning and retraining may be necessary to enhance accuracy by incorporating a broader set of photos. To achieve consistent accuracy, new cycles of tests and training are planned, involving iterations where new photos and ASRN adjustments are reevaluated.

In the developed ASRN, the evaluation process involved inputting 1,000 photo frames, resulting in an accuracy value of 83%. On average, the LPR process requires 93 milliseconds per recognition. Comparative characteristics of ASRN products available on the market, based on the LPR accuracy criterion, are presented in Table 1.

Table 1 Comparison of software accuracy

System name	Recognition accuracy (A), %
Auto-Inspector	94,19
Auto Trassir	83,07
CVS Auto	82,09
MegaCar	81,06
Overseer Traffic	79,89
AutoUragan 3.3.2	89,69

AutoUragan 3.4	90,45
IntegraVideoAuto	81,77
Potok	86,38
TeleWizardAuto	51,33
Autonumber plate recognitor (PyTorch)	83

The ASRN under consideration performs LPR tasks in 93 milliseconds with an average accuracy of 83%.

5 Conclusion

The thesis addresses the design of an automatic vehicle state registration number recognition system effectively. The primary objective was to develop an automated system for recognizing state license plates of cars, and the project has succeeded in doing so using a complete solution approach that covered analysis, design, implementation, and testing.

The project began with a thorough study of software-based and hardware-based automated systems recognizing license plates. This review played a key role in identifying the strengths and limitations of existing solutions and was taken as a standard while designing the new system. Various techniques such as contour analysis, histogram analysis, and neural networks were evaluated during this phase. Among them, the neural network approach proved the most effective and was therefore selected for final implementation.

Following analysis, the ASRN system was designed and developed with a focus on efficiency and accuracy. The system architecture incorporated pre-processing steps for processing input images, character recognition software for obtaining the license plate number, and post-processing software tools for cleaning the results. Development utilized Python, PyTorch, OpenCV, and TensorFlow, leveraging the features of the tools for machine learning and computer vision tasks. This specific combination of technologies contributed significantly to the improvement of the system's performance and flexibility.

Subsequently, the system was tested on a test dataset of 1,000 photo frames. The results of the tests included an average recognition time of 93 milliseconds and an accuracy rate of 83%. These rates are comparable with commercial ASRN solutions such as CVS Auto and Auto Trassir, thereby validating the effectiveness of the implemented system.

Although the current system yields promising outcomes, there are several areas of research and potential improvements for the future. For example, by increasing the amount of data to train on, particularly if the data set includes a greater diversity of license plates photographed under different weather, lighting, and angles. Furthermore, experimenting and optimizing a variety of neural network architectures may lead to better performance. Techniques such as transfer learning and fine-tuning pre-trained models are promising approaches for this purpose.

The second most important area is integrating the ASRN system with external databases. This would significantly increase its value for applications such as traffic monitoring, security, and automated toll or fee collection. For example, integrating the system with vehicle registration databases would allow real-time checks on license plates and identification of unauthorized vehicles. Making APIs or similar interfaces available would also simplify data exchange and enhance integration with existing infrastructure.

Further, optimizing the system for real-time deployment is necessary for field use in traffic monitoring systems. Optimizing this way would involve reducing latency to allow the system to process images fast enough to maintain pace with traffic movement. Deploying using edge devices or embedded systems would allow reducing reliance on centralized servers, thereby improving response times.

Finally, user interface and experience should be considered to make the system more usable. Designing an operator interface that is user-friendly with features such as instant feedback, adjustable parameters, and expanded reporting can make a real difference in user satisfaction and operating efficiency. Visual tools that graphically display recognition results and system performance metrics would also help operators monitor the operation of the system and feel any issues that need correction.

By addressing these improvement areas and exploring new technological frontiers, the system can be further developed to cater to the evolving requirements of traffic management, security, and other real-world scenarios.

BIBLIOGRAPHY

Baggio, D. L. (2012). *Mastering OpenCV with practical computer vision projects: Step-by-step tutorials to solve common real-world computer vision problems for desktop or mobile, from augmented reality and number plate recognition to face recognition and 3D head tracking*. Packt Publishing.

Brunelli, R. (2009). *Template Matching Techniques in Computer Vision*. John Wiley & Sons.

Du, S., Ibrahim, M., Shehata, M., & Badawy, W. (2013). Automatic license plate recognition (ALPR): A state-of-the-art review. *IEEE Transactions on Circuits and Systems for Video Technology*, 23(2), 311–325. https://doi.org/10.1109/TCSVT.2012.2203741

Foster, E. (2014, December). *A comparative analysis of the C++, Java, and Python languages*. Keene State College.

Gonzalez, R. C., & Woods, R. E. (2014). *Digital image processing* (4th ed.). Pearson.

Hill, E. J., Pearce, M. A., & Stromberg, J. M. (2020). Improving automated geological logging of drill holes by incorporating multiscale spatial methods. *Mathematical Geosciences, 52(8)*, 1105–1125. https://doi.org/10.1007/s11004-020-09883-2

Hikvision. (n.d.). *HSRN system*. Retrieved April 13, 2025, from https://www.hikvision.com/en/

Hoiem, D., Chodpathumwan, Y., & Dai, Q. (2012). Diagnosing error in object detectors. In A. Fitzgibbon, S. Lazebnik, P. Perona, Y. Sato, & C. Schmid (Eds.), *Computer vision – ECCV 2012: 12th European conference on computer vision, Florence, Italy, October 7–13, 2012, proceedings, part I* (pp. 340–353). Springer. https://doi.org/10.1007/978-3-642-33712-3_25

Intelligent Security Systems. (n.d.). *SecurOS® Auto: License plate recognition solution*. Retrieved April 13, 2025, from https://issivs.me/solutions/securos-video-analytics/securos-auto/

International Association of Chiefs of Police. (2019). Expansions in ALPR technology. *Police Chief Magazine*. https://www.policechiefmagazine.org/product-feature-expansions-in-alpr-technology/

Jain, S., & Patel, R. (2018). Performance analysis of Canny and Sobel edge detection algorithms in image mining. *International Journal of Advanced Research in Computer Science, 9*(6), 130–134.

Jo, J., Koo, H. I., Soh, J. W., & Cho, N. I. (2020). Handwritten text segmentation via end-to-end learning of convolutional neural networks. *Multimedia Tools and Applications, 79*(43–44), 32137–32150. https://doi.org/10.1007/s11042-020-08979-z

LeCun, Y., & Bengio, Y. (1995). Convolutional networks for images, speech, and time-series. In M. A. Arbib (Ed.), *The handbook of brain theory and neural networks* (pp. 255–258). MIT Press.

Marzuki, P., Wong, Y. C., Hamid, N., Nur Alisa, A., Ibrahim, M., & Syafeeza, A. (2024). Advanced automated number plate recognizer using machine learning. *International Journal of Computer Applications, 177*(30), 1–7.

Murygin, K. (2011). Segmentation of symbols on the car license plate images. *Proceedings of the International Workshop on Intelligent Information Systems*. https://ibn.idsi.md/sites/default/files/imag_file/Composed_31_Murygin.pdf

Nedap Identification Systems. (n.d.). *Nedap license plate reader*. Retrieved April 13, 2025, from https://www.nedapidentification.com/

Norvig, P. (2024, January 10). Interfacing Python with C/C++ for performance: My experience with enhancing Python's performance by interfacing it with the raw power of C/C++ code. *Paul Norvig's Guides*. https://www.paulnorvig.com/guides/interfacing-python-with-cc-for-performance.html

OpenCV. (n.d.). *Color conversions. OpenCV Documentation*. Retrieved April 13, 2025, from https://docs.opencv.org/3.4/de/d25/imgproc_color_conversions.html

OpenCV. (n.d.). *Introduction to OpenCV. OpenCV*. Retrieved April 14, 2025, from https://docs.opencv.org/4.x/d1/dfb/intro.html

PyTorch. (n.d.). *PyTorch documentation*. Retrieved April 14, 2025, from https://pytorch.org/docs/stable/index.html

Rao, K. V. R., Rao, N. B., Babu, R., & Goplani, D. (2013, September). Connected component analysis for character feature extraction. In *Proceedings of the National Conference on Innovations in Electrical, Electronics and Computer Science Engineering (IEECSE-2013)* (Vol. ISBN 978-81-31703-83-2). Bangalore, India.

Saha, S. (2018, April 29). A review on automatic license plate recognition system. In *Students' Technical Article Competition: PRAYAS-2018*, Department of Electronics & Communication Engineering, MCKV Institute of Engineering, Liluah, Howrah.

Sarfraz, M., Ahmed, M., & Ghazi, S. A. (2010). Artificial neural networks based vehicle license plate recognition. *Procedia Computer Science, 3*, 1033–1037. https://doi.org/10.1016/j.procs.2010.12.169

Shi, S. (2013). *Emgu CV essentials: develop your own computer vision application using the power of Emgu CV*. Packt Publishing.

Špaňhel, J., Herout, A., & Zemčík, P. (2016). Algorithmic and mathematical principles of automatic number plate recognition systems. *Acta Electrotechnica et Informatica, 16*(4), 3–8. https://doi.org/10.15546/aeei-2016-0032

Security Info Watch. (2015, April 10). ISS introduces SecurOS TrafficScanner. *Security Info Watch*. Retrieved April 13, 2025, from https://www.securityinfowatch.com/video-surveillance/press-release/12066065/intelligent-security-systems-iss-introduces-securos-trafficscanner

TensorFlow. (n.d.). *About TensorFlow. TensorFlow*. Retrieved April 14, 2025, from https://www.tensorflow.org/about

Viola, P., & Jones, M. J. (2001). Rapid object detection using a boosted cascade of simple features. In *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)* (Vol. 1, pp. 511–518). IEEE. https://ieeexplore.ieee.org/document/990517

Vizilter, Y., Zheltov, S., & Knjaz, V. (2018). *Processing and analysis of digital images with examples in LabVIEW and IMAQ Vision*. Book on Demand Ltd.

Wu, M.-K., Wei, J.-S., Shih, H.-C., & Ho, C.-C. (2009). 2-level-wavelet-based license plate edge detection. In *Proceedings of the Fifth International Conference on Information Assurance and Security (IAS 2009)* (pp. 507–510). IEEE. https://doi.org/10.1109/IAS.2009.132

APPENDICES

Appendix 1. Listing of program modules

Appendix 1. Listing of program modules

--Module of Training and Number Finding in the image

```
import torch

import ultralytics

from ultralytics import YOLO

from PIL import Image

import cv2

import warnings

warnings.filterwarnings("ignore")


def crop_detected_area(image_path: str) -> None:

    model = YOLO(r"C:\Users\olga\Desktop\Torch\car_plate\runs\detect\car_plate_detection2\weights\best.pt")  # Make sure you have the correct model loaded

    results = model(image_path)

    original_image = Image.open(image_path)


    for result_idx, result in enumerate(results):

        for bbox_idx, (xmin, ymin, xmax, ymax) in enumerate(result.boxes.xyxy):

            xmin, ymin, xmax, ymax = map(int, (xmin, ymin, xmax, ymax))

            cropped_image = original_image.crop((xmin, ymin, xmax, ymax))

            cropped_image.save(f'detected_area.jpg')
```

```
        return


def learn_model():
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")


    try:
        model_path = r"yolov8n.pt"
        model = YOLO(model_path)
        model.to(device=device)
        print(f'The YOLO model was successfully loaded from the path: {model_path}')
        print(f'Device for YOLO: {model.device}')


    except Exception:
        print('An error occurred while loading the model')


    ultralytics.checks()
    print(torch.cuda.is_available())


    model.train(
        data="config_file.yaml",
        name="car_plate_detection",
```

```
        imgsz=640,
        epochs=1,
        batch=8)

--Module of License plate symbols' recognition
def make_predict_car_number(image_path: str):
    image0 = cv2.imread(image_path, 1)
    image_height, image_width, _ = image0.shape
    image = cv2.resize(image0, (1024, 1024))
    image = image.astype(np.float32)
    paths = './model_resnet.tflite'

    interpreter = tf.lite.Interpreter(model_path=paths)
    interpreter.allocate_tensors()
    input_details = interpreter.get_input_details()
    output_details = interpreter.get_output_details()
    X_data1 = np.float32(image.reshape(1, 1024, 1024, 3))
    interpreter.set_tensor(input_details[0]['index'], X_data1)
    interpreter.invoke()

    detection = interpreter.get_tensor(output_details[0]['index'])
```

```
img = image0
img2 = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)  # Конвертация BGR --> RGB
img3 = img[:, :, :]
box_x = int(detection[0, 0, 0] * image_height)
box_y = int(detection[0, 0, 1] * image_width)
box_width = int(detection[0, 0, 2] * image_height)
box_height = int(detection[0, 0, 3] * image_width)

if np.min(detection[0, 0, :]) >= 0:
    image = img3[box_x:box_width, box_y:box_height, :]
    grayscale = rgb2gray(image)
    edges = canny(grayscale, sigma=3.0)
    out, angles, distances = hough_line(edges)
    _, angles_peaks, _ = hough_line_peaks(out, angles, distances, num_peaks=20)
    angle = np.mean(np.rad2deg(angles_peaks))
    if 0 <= angle <= 90:
        rot_angle = angle - 90
    elif -45 <= angle < 0:
        rot_angle = angle - 90
    elif -90 <= angle < -45:
```

```
    rot_angle = 90 + angle
if abs(rot_angle) > 20:
    rot_angle = 0


rotated = rotate(image, rot_angle, resize=True) * 255
rotated = rotated.astype(np.uint8)
rotated1 = rotated[:, :, :]
minus = np.abs(int(np.sin(np.radians(rot_angle)) * rotated.shape[0]))
if rotated.shape[1] / rotated.shape[0] < 2 and minus > 10:
    rotated1 = rotated[minus:-minus, :, :]


lab = cv2.cvtColor(rotated1, cv2.COLOR_BGR2LAB)
l, a, b = cv2.split(lab)
clahe = cv2.createCLAHE(clipLimit=3.0, tileGridSize=(8, 8))
cl = clahe.apply(l)
limg = cv2.merge((cl, a, b))
final = cv2.cvtColor(limg, cv2.COLOR_LAB2BGR)


paths = './model1_nomer.tflite'


interpreter = tf.lite.Interpreter(model_path=paths)
```

```
interpreter.allocate_tensors()

# We get the input and output tensors

input_details = interpreter.get_input_details()

output_details = interpreter.get_output_details()


img = final

img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

img = cv2.resize(img, (128, 64))

img = img.astype(np.float32)

img /= 255


img1 = img.T

img1.shape

X_data1 = np.float32(img1.reshape(1, 128, 64, 1))

interpreter.set_tensor(input_details[0]['index'], X_data1)

interpreter.invoke()

net_out_value = interpreter.get_tensor(output_details[0]['index'])

pred_texts = decode_batch(net_out_value)


print(pred_texts)

# Drawing of the found box with text
```

```
        cv2.rectangle(img2, (box_y, box_x), (box_height, box_width), (255, 0, 0), thick-
ness=2)

        # cv2.putText(img2, pred_texts[0], (box_y, box_x), cv2.FONT_HERSHEY_COM-
PLEX, 0.8, (0, 255, 0), 2, cv2.LINE_AA)

        plt.imshow(img2)

        plt.xticks([]), plt.yticks([])

        plt.show()


        return str(pred_texts)


    else:

        print('No car found in the image')


-- Interface module

import tkinter as tk

from tkinter import filedialog

from PIL import Image, ImageTk

from recognition import make_predict_car_number


def select_image():

    file_path = filedialog.askopenfilename()

    if file_path:
```

```
        try:
            image = Image.open(file_path)
            image.thumbnail((300, 300))
            photo = ImageTk.PhotoImage(image)
            label_image.config(image=photo)
            label_image.image = photo
            global selected_image
            selected_image = image
        except Exception as e:
            result_text.set("Error: " + str(e))
            selected_image = None


def recognize():
    if selected_image:
        try:
            result = make_predict_car_number(selected_image.filename)
            result_text.set("Number on the image: " + result)
        except Exception as e:
            result_text.set("Error of recognition: " + str(e))
    else:
        result_text.set("First, select an image")
```

```
root = tk.Tk()

root.title("License plate recognition ")


root.update_idletasks()

min_width = root.winfo_reqwidth()

root.minsize(min_width, 0)


label_image = tk.Label(root)

label_image.pack(pady=10)


button_select = tk.Button(root, text="Choose an image", command=select_image)

button_select.pack(pady=5)


button_recognize = tk.Button(root, text="Recognize", command=recognize)

button_recognize.pack(pady=5)


result_text = tk.StringVar()

result_text.set("The result of the recognition will be here ")

label_result = tk.Label(root, textvariable=result_text)

label_result.pack(pady=10)
```

```
selected_image = None


root.mainloop()
```