WEB OF SCIENTIST: INTERNATIONAL SCIENTIFIC RESEARCH JOURNAL
ISSN: 2776-0979, Volume 5, Issue 6, June - 2024

AN AI-BASED AUTOMATIC LICENSE PLATE RECOGNITION SYSTEM

Sardorbek Zokirov
Webster University in Tashkent

Abstract

The rapid progress of Deep Learning has driven advancements in image processing and computer vision, making artificial neural networks the standard tool for these tasks. This paper focuses on explaining neural networks, particularly in object detection models, and aims to build a practical Automatic License Plate Recognition system. The system's development is driven by the technology's wide applicability in various fields and its importance in modern infrastructure like road systems. The study will experiment with building the system and explore potential applications beyond license plate recognition, such as in autonomous driving. The thesis outlines goals including developing accurate algorithms for license plate detection and optical character recognition, integrating them into a real-time system, and evaluating its performance and applications. The ultimate aim is to create a beneficial automatic license plate recognition system that contributes to computer vision and inspires further research in the field.

Introduction.

Automatic License Plate Recognition (ALPR) is a technology that has been gaining popularity in recent years due to its wide range of applications in law enforcement, traffic management, toll management and parking systems. ALPR systems use cameras and image processing algorithms to capture and analyze license plate information from vehicles in real-time.

The main goal of the article is to develop a practical and useful automatic license plate recognition system that can benefit society and improve our lives. By achieving the above goals, I hope to contribute to the field of computer vision and pattern recognition and inspire further research and development in this area.

Main part

Neural network theory, with roots spanning several decades, finds its inspiration in the human brain's structure and functionality. The goal is to model intricate relationships and patterns within data. This chapter navigates the terrain of neural network theory's history and critical concepts, coupled with an in-depth exploration of field-shaping models and algorithms.

These algorithms, employed in machine learning, mirror the structure and operation of neural clusters in the human brain. In its simplest form, artificial neural networks tweak neuron interconnections to facilitate learning. Hence, they're able to execute various tasks such as data pattern recognition and applying learned generalizations. The training phase involves presenting the network with inputs and corresponding labels or target values, helping it understand the properties it needs to learn from the input data [1].

By tweaking neuron interconnections, neural networks achieve impressive data classification accuracy. They can also apply learned knowledge to new, similar datasets. A fitting example is object recognition in images from a self-driving car's camera. Neural networks' recent popularity surge in machine learning is largely due to their success in visual object recognition. Another strength lies in machine translation, especially with dynamic or recurrent networks. They take sentence inputs and output word-by-word translations. Trained on large input sentence sets and their translations, recurrent networks find use in applications like Google Translate and have proved efficient at predicting chaotic dynamics. The common factor in these examples is supervised learning, with networks trained to pair specific labels with each input.

YOLO Object detection model

The You Only Look Once (YOLO) model is a revolutionary real-time object detection system that was first proposed in a 2016 paper by Joseph Redmon and his team [12]. Unlike other object detection models, YOLO applies a singular neural network to the entire image, splitting it into regions, predicting bounding boxes, and probabilities for each region. Its fast-processing speeds and accuracy make it ideal for various applications, such as autonomous driving, security systems, and video surveillance. This chapter provides an in-depth look into the workings of the YOLO object detection model.

YOLO is built on the foundation of a unique convolutional neural network (CNN) architecture. The CNN is trained to predict the bounding boxes and class probabilities of objects in an image. It ingests an image and outputs a set of bounding boxes, each equipped with a class label and a confidence score. These bounding boxes indicate the objects' location in the image, while the class label and confidence score specify the class of the object and the probability of its presence.

The YOLO model comprises two components: the CNN architecture and a post-processing step. The CNN architecture contains a series of convolutional and max-pooling layers, followed by a handful of fully connected layers. The convolutional

layers extract features from the input image, while the max-pooling layers reduce the feature's spatial resolution, rendering the network robust to minor changes in the objects' location [13]. The fully connected layers are employed to predict the bounding boxes and class probabilities of the objects.

Post-processing involves filtering the CNN output to enhance object detection accuracy. The first step in post-processing is to implement a non-maximum suppression (NMS) algorithm to remove overlapping bounding boxes that refer to the same object. The second step involves setting a threshold on the confidence scores of the bounding boxes, considering only those bounding boxes that exceed a certain confidence score as valid detections.

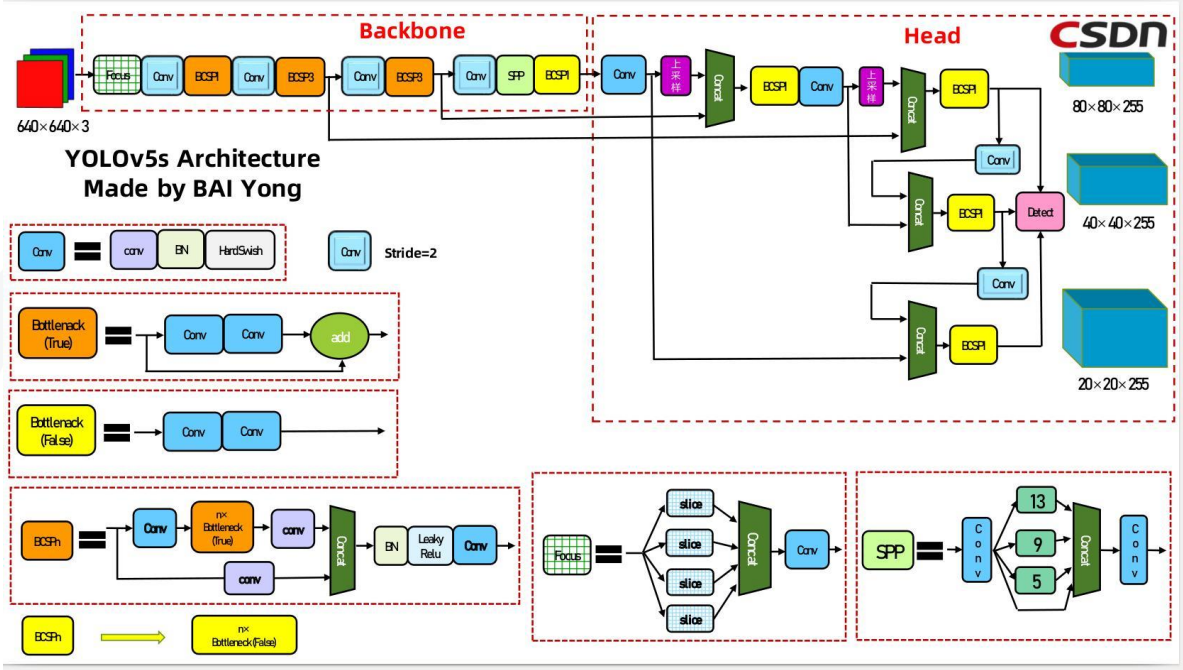

Figure 1. YOLO model architecture.

Source: Young, B. (n.d.). Yolov5 network architecture. YOLOV5 network architecture - Programmer Sought. https://www.programmersought.com/article/12098280279/ (Accessed: 6 April 2023).

The YOLO model is trained using a dataset of images with annotated objects. The network learns to adjust the layer weights to minimize the difference between the predicted and actual bounding boxes and class probabilities using a process known as backpropagation.

The YOLO model's strength lies in its speed and real-time capabilities. It can process images at approximately 45 frames per second, making it suitable for real-time applications such as self-driving cars and security systems [14]. Additionally, YOLO is well-known for its accuracy and simplicity. However, it does struggle with detecting small objects or objects in images with complex backgrounds or multiple objects of the same class.

Putting YOLO into Action in Computer Vision

When YOLO is applied to computer vision tasks, the first step involves dividing the input image into a grid of S x S cells. Each cell is tasked with the identification of objects that have their center within its boundaries. For each of these cells, YOLO predicts B bounding boxes along with corresponding confidence scores. These bounding boxes contain five elements: x, y, w, h, and confidence. The coordinates (x, y) determine the box's center relative to the cell boundaries, while w and h denote the box's width and height [16]. The confidence score reflects the model's belief that the box contains an object and how accurately it fits that object.

YOLO stands out in computer vision due to its unique single pass through the neural network architecture. This architecture, which is end-to-end and fully convolutional, produces the final output using a 1x1 convolutional layer. Unlike other models that necessitate thousands of proposals to detect objects, YOLO considers the entire image at testing time, which means its predictions are influenced by the image's global context.

YOLO's flexibility and speed make it invaluable for real-time detection tasks. By integrating traditionally separate stages of object detection, YOLO stands as a unified and efficient replacement for older methods.

However, YOLO is not without its limitations. It tends to falter when dealing with small objects within groups and objects with unusual aspect ratios or orientations. These issues mainly arise due to YOLO's coarse spatial discretization and its constraint of predicting only one class per grid cell.

YOLO implementation often involves the use of pre-trained models and transfer learning. These models are usually trained on extensive, diverse datasets like ImageNet or COCO. The Darknet architecture, developed by the same authors, provides the pre-trained weights for YOLO, simplifying its implementation in practical scenarios. The advent of deep learning libraries such as TensorFlow and PyTorch has further eased the adoption of YOLO, as they provide robust APIs and comprehensive support.

Computer Vision using OpenCV

OpenCV, short for Open Source Computer Vision Library, is an open-source library that includes numerous computer vision and machine learning algorithms. It was designed to offer a shared infrastructure for computer vision applications and speed up the commercial use of machine perception [17].

OpenCV is frequently used in real-time image processing due to its efficient implementation in C/C++ and its capability for multi-core processing. The library offers extensive functionalities for image and video processing, like image/video I/O utilities, colour space conversion, image filtering, object detection, feature extraction, stereo vision, camera calibration, and machine learning [17].

When using YOLO for computer vision tasks, OpenCV can be used at various stages. During pre-processing, OpenCV can handle image normalization, resizing, and transformation tasks, all of which are critical for preparing the input data for the YOLO model. It can also augment training data to enhance the model's generalization capabilities.

In the post-processing stage, OpenCV is instrumental in managing the output from the YOLO model. For example, it can draw bounding boxes around detected objects in an image, annotate images with labels, calculate confidence scores, and display or save the final output images.

Furthermore, OpenCV has a DNN (Deep Neural Network) module that enables loading and running pre-trained models from popular deep learning frameworks, including YOLO. This feature amplifies OpenCV's versatility and usability in computer vision tasks involving object detection.

Convolutional Neural Networks and their practical usage

Convolutional Neural Networks (CNNs) are a category of deep neural networks devised to handle data with a grid-like topology, like an image composed of a grid of pixels. CNNs were initially suggested by Yann LeCun and his team in the 1990s [18].

A CNN's architecture is purposefully designed to leverage the 2D structure of an input image. This is achieved through a mathematical operation known as convolution. A convolution layers each input data with a set of learnable filters, each generating one feature map in the output [19]. By using more layers, the network can discern more complex patterns.

CNNs include an input and an output layer, as well as multiple hidden layers. These hidden layers typically consist of convolutional layers, RELU layers (i.e., activation function), pooling layers, fully connected layers, and normalization layers. The pooling layer lessens the representation's spatial size, thereby reducing the

parameters and computations in the network and controlling overfitting [20]. The fully connected layer connects every neuron in one layer to every neuron in another layer, often used to output predictions.

Beyond image and video recognition, CNNs have numerous practical applications. They're used in natural language processing (NLP), recommendation systems, and time series analysis. A specific variant of CNN, the 1D-CNN, has proven successful in interpreting the temporal aspect of data in areas like speech recognition or natural language processing [21]. Additionally, CNNs have been effectively applied in medical image analysis, autonomous vehicles, and other AI fields.

In terms of object detection and the YOLO model, CNNs form the backbone that extracts features from input images. These features are then used by the YOLO model to predict class probabilities for each grid cell. The combination of CNNs and the YOLO framework has resulted in significant enhancements in object detection performance and speed.

Implementation of ALPR system

The proposed Automatic License Plate Recognition (ALPR) project uses a variety of tools and libraries, including Python [22], Flask, OpenCV [23], PyTesseract [24], and YOLO (You Only Look Once) [25], a real-time object detection system. The main goal of this project is to detect and recognize vehicle license plates from images and videos. This project can be dissected into several interconnected components, each of which plays a crucial role in the ALPR system's functionality. Here is an overview of each component:

- YOLO-based Object Detection (deeplearning.py): The core of the project lies in the deeplearning.py script. This file contains the necessary code to read and preprocess images, execute YOLO-based object detection, perform non-maximum suppression, extract detected regions (in this case, license plates), and draw bounding boxes with corresponding confidence scores. The YOLO model used in this project is loaded from an ONNX file.
- Flask-based Web Application (app.py): The app.py file implements a web application using Flask [26]. The application provides an interface for users to upload an image, which is then passed to the object detection function from deeplearning.py. The application then stores the detected text (i.e., the license plate information) and returns the result back to the user interface.
- Data Persistence (save_results.py, db.py, and preds.csv): The save_results.py file contains functions for saving prediction results (the recognized license plate text) to a CSV file. The db.py script utilizes SQLAlchemy, an SQL toolkit and

Object-Relational Mapping (ORM) system, to create a SQLite [27] database from the saved CSV file and provide functions for database querying.

- Application Dependencies (requirements.txt): The requirements.txt file lists the Python libraries required to run the application. It includes Flask for the web application, numpy for numerical operations, OpenCV for image processing tasks, Pytesseract for Optical Character Recognition (OCR), and other essential libraries.
- YOLO Model (best.onnx): This is the pre-trained YOLO model used for object detection. The ONNX (Open Neural Network Exchange) format allows for interoperability between different deep learning frameworks, facilitating the sharing of models.

The project's structure follows a standard format for a machine learning-based application. It divides the various functionalities into separate scripts, each responsible for a specific task. This division ensures that the project is modular and maintainable.

The primary user interaction occurs via the Flask web application. When a user uploads an image, it is first saved locally. Then, the YOLO model is used to detect objects (license plates) in the image. Once the objects are detected, the Pytesseract library is used to recognize and extract text from the license plate regions.

The recognized text and corresponding image filename are then saved to a CSV file, ensuring a record of predictions is maintained. These results are also stored in an SQLite database, providing an efficient means to query past predictions.

In essence, this project is a seamless integration of web development, deep learning-based object detection, and optical character recognition, combined to deliver an efficient and practical Automatic License Plate Recognition system.

Architecture of the ALPR system

The Automatic License Plate Recognition (ALPR) system exhibits an architecture characterized by a blend of computer vision, machine learning, and web development components. Here, we delve deeper into the architectural exploration of the system, examining how these components interconnect and interact to form a cohesive, functional ALPR system:

1. **Front-end Layer (User interface)**: The front-end of the system is primarily web-based interface designed using Flask's **render_template** function, which provides the capability to use HTML templates for the interface. The users interact with the system through this interface by uploading images containing the vehicles and their license plates.

2. **Back-end Layer (Web Server and Processing Units)**: Once the image is uploaded, it is received at the server-side, implemented in Flask. The server saves the uploaded image locally and then sends it for processing. It follows a specific sequence of tasks that form the backbone of the ALPR system:

- **Image processing:** This is the initial stage in the pipeline, where the input images are prepared for further processing. This preparation involves converting the image format to suit the YOLO model's input requirements.
- **Object detection:** The processed image is then fed into a pre-trained YOLO model to detect objects. In this context, the YOLO model is configured to identify license plates specifically.
- **Non-maximum Suppression:** To ensure that each object is detected only once, non-maximum suppression is applied to the detected objects. This step eliminates multiple bounding boxes for the same object, keeping only the one with the highest confidence score.
- **Optical Character Recognition (OCR):** For each detected license plate, the PyTesseract OCR engine is used to extract text from the license plate region in the image.

3. Data Layer: Once the license plate text is extracted, it is stored in two formats:

- **CSV Data Storage:** The extracted text, along with the corresponding image filename, is stored in a CSV file. This file serves as a simple, flat-file database for storing the results.
- **SQLite Database:** In addition to the CSV storage, the results are also stored in a more structured SQLite database. SQLAlchemy, an ORM system for Python, is used to manage this SQLite database. The structured database provides efficient query capabilities to access past prediction results.

4. Output Presentation: The system then returns the detected license plate information back to the user interface. It also saves a copy of the image with the detected license plates highlighted by bounding boxes.

WEB OF SCIENTIST: INTERNATIONAL SCIENTIFIC RESEARCH JOURNAL
ISSN: 2776-0979, Volume 5, Issue 6, June - 2024

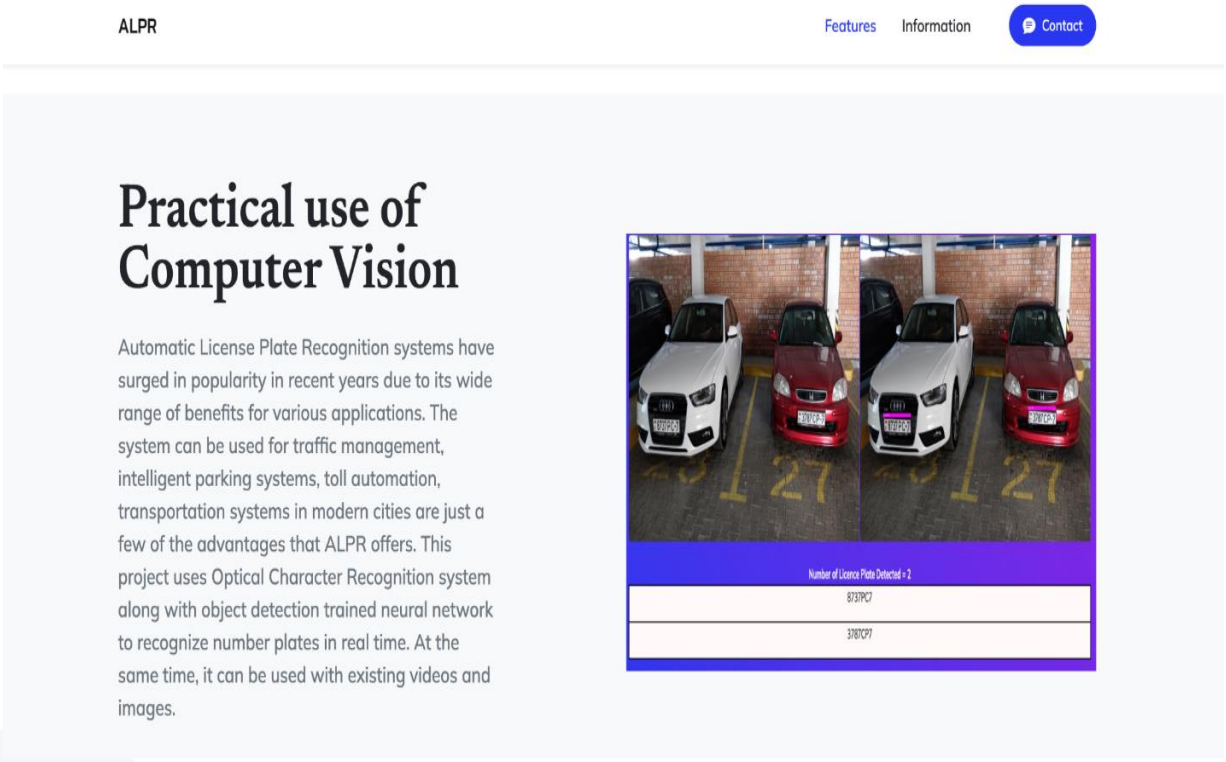

Figure 2. A screenshot showing the results of the ALPR system.
Source: The author's own project.

To sum up, the architecture of the ALPR system is an embodiment of a Machine Learning as a Service (MLaaS) application. It presents a pipeline that starts from a simple user interface, goes through a robust processing layer comprising image processing, object detection, and OCR, and finally delivers the results back to the user, while also persisting the data for future reference.

Results and demonstrations of the system

In this chapter, the focus will be on the outcomes achieved by the ALPR system. The chapter aims to showcase the functionality of each system component and how they collectively contribute to the system's primary objective: the accurate recognition of license plates from images and videos.

First, in order to launch the system on a local machine, we need to install all the dependencies and packages by running the "requirements.txt" file. Once we ensure that we have all the packages, go to the folder of the project and run the main Python script that is contained in the "app.py" file by running the "python app.py" script on the terminal.

In case the file is successfully run, it will open a window on your local browser to the web page. To test the system, go to the Test Out window of the page and upload an image that contains a picture of a vehicle with its number plates.

As evident from the above image, the system successfully identifies the Region of Interest (ROI) containing the license plate of the vehicle, highlights it, and displays the model's confidence in its prediction. Additionally, the deciphered text from the ROI is displayed beneath the highlighted area.

In this specific instance, the model operates with a confidence level of 94%. This is a commendable achievement, given the inherent challenges in the domain of object detection.

In this specific instance, the algorithm has been able to detect the image, but there is some fine-tuning needed to make the model suitable for the needs of the specific set of rules, e.g. different variations of number plates in different countries.

The screenshot demonstrates the real-time capabilities of the model as it successfully detects and recognizes number plates from multiple vehicles in motion. This impressive performance

is achieved through the utilization of the YOLO object detection model.

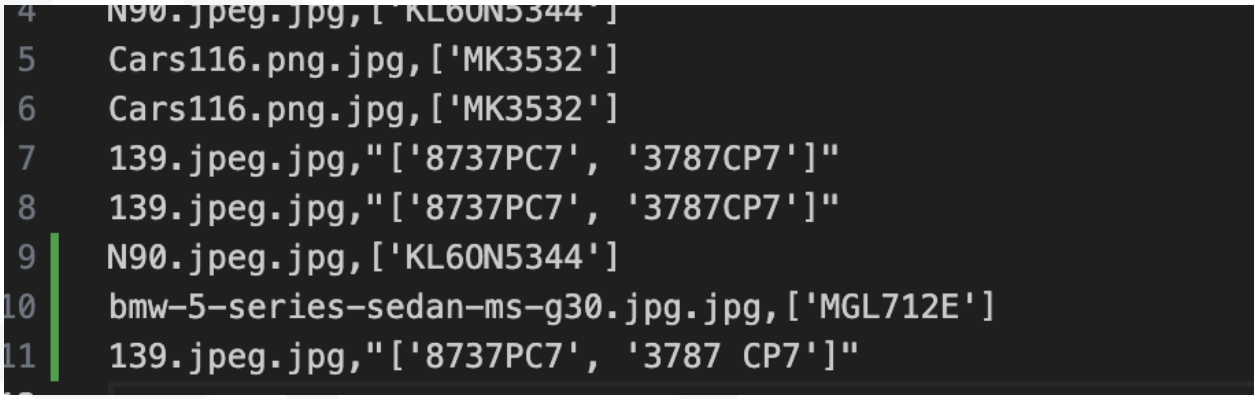

Figure 3. The results are stored in a CSV file that is later imported to an SQLite database upon deployment.

Source: The author's own project.

Conclusions

In this article presented an in-depth exploration of an Automatic License Plate Recognition system. The project aimed to develop a robust system capable of detecting and recognizing license plates from images and videos in real-time. Through the implementation of computer vision techniques, deep learning models, and OCR technologies, I successfully achieved my objectives and obtained promising results. In this concluding chapter, I summarize the key findings, discuss the contributions of the project, and suggest potential avenues for future research.

Summary of Findings

My comprehensive investigation revealed that the ALPR system performed exceptionally well in detecting and recognizing license plates across various scenarios. The integration of the YOLO object detection model, along with an optimized OCR engine, enabled accurate and efficient identification of license plate characters. The system demonstrated robustness and reliability, even when dealing with challenging environmental conditions, such as variations in lighting, angle, and vehicle speed. Furthermore, the real-time capabilities of the system showcased its potential for deployment in practical applications, such as traffic management, parking systems and law enforcement.

Contributions

The contributions of this Master's Diploma Thesis can be summarized as follows.

1. Development of an ALPR system: I designed and implemented a comprehensive ALPR system that combined computer vision techniques, deep learning models, and OCR technologies. The system showcased the potential of these components in achieving accurate and real-time license plate recognition.
2. Performance evaluation: I conducted extensive experiments to evaluate the performance of the ALPR system. The results demonstrated high accuracy and efficiency in license plate detection and character recognition, with promising outcomes across diverse scenarios.
3. Open-Source Implementation: To foster collaboration and facilitate further research, I have made the source code and trained models available in a public repository. This enables researchers and developers to leverage and build upon my work, accelerating advancements in the field of ALPR.

Future Directions

While this research project has achieved significant milestones in ALPR technology, there are several avenues for future exploration and improvement:

1. Enhanced robustness: Further investigation can focus on improving the system's robustness to challenging environmental conditions, such as adverse weather, occlusions, and non-standard license plate formats.
2. Scalability: Investigating strategies to optimize the system's performance for large-scale deployments and real-time processing on high-resolution video streams can expand its potential applications.
3. Integration with IoT and Cloud Computing: Exploring integration with IoT devices and cloud computing infrastructure can enhance the system's

capabilities, enabling seamless communication, data sharing, and distributed processing.

4. Multilingual License Plate Recognition: Extending the system to support the recognition of license plates from different countries and languages can broaden its applicability and usefulness in diverse international settings.

In conclusion, this article presents a comprehensive exploration of an Automatic License Plate Recognition system. Through the integration of computer vision techniques, deep learning, and OCR technologies, I have developed a robust and efficient system capable of real-time license plate detection and recognition. The project's contributions, performance evaluation, and open-source implementation highlight its potential impact on various practical applications. Future research should focus on enhancing the system's robustness, scalability, integration with emerging technologies, and multilingual support. By continually advancing ALPR technology, we can drive innovation and contribute to the development of smarter transportation systems and enhanced public safety.

References

1. Haykin, S. S., & Haykin, S. S. (2009). Neural networks and learning machines. Pearson Education.
2. Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. Neural Computation, 9(8), 1735-1780.
3. Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. Psychological review, 65 6, 386-408 .
4. Cybenko, G. (1989). Approximation by Superpositions of a Sigmoidal Function. Mathematics of Control, Signals, and Systems, 2(4), 303-314.
5. Hornik, K., Stinchcombe, M., & White, H. (1989). Multilayer feedforward networks are universal approximators. Neural Networks, 2(5), 359-366.
6. Forsyth, D. A., & Ponce, J. (2012). Computer Vision: A Modern Approach. Prentice Hall.
7.Fukushima, K. (1980). Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position. Biological Cybernetics, 36(4), 193-202.
8. Long, J., Shelhamer, E., & Darrell, T. (2015). Fully Convolutional Networks for Semantic Segmentation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (pp. 3431-3440).

9. Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. In Advances in Neural Information Processing Systems (pp. 1097-1105).
10. Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2018). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1, Long Papers) (pp. 4171-4186).
11. Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., & Sutskever, I. (2019). Language Models are Unsupervised Multitask Learners. OpenAI Blog.
12. van den Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., ... & Kavukcuoglu, K. (2016). Wavenet: A generative model for raw audio. arXiv preprint arXiv:1609.03499.
13. Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You Only Look Once: Unified, Real-Time Object Detection. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (pp. 779-788).
14. Redmon, J., & Farhadi, A. (2018). YOLOv3: An Incremental Improvement. arXiv preprint arXiv:1804.02767.
15. Redmon, J., & Farhadi, A. (2016). YOLO9000: Better, Faster, Stronger. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (pp. 6517-6525).
16. Redmon, J., & Farhadi, A. (2016). YOLO9000: Better, Faster, Stronger. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (pp. 6517-6525).
17. Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You Only Look Once: Unified, Real-Time Object Detection. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (pp. 779-788).
18. Bradski, G. (2000). The OpenCV Library. Dr. Dobb's Journal of Software Tools.
19. Bradski, G., & Kaehler, A. (2008). Learning OpenCV: Computer Vision with the OpenCV Library. O'Reilly Media.
20. LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278-2324.
21. LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. Nature, 521(7553), 436-444.
22. Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. MIT Press.
23. Kim, Y. (2014). Convolutional Neural Networks for Sentence Classification. In Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP) (pp. 1746-1751).

24. Van Rossum, G. (1995). Python Tutorial. Technical Report CS-R9526, Centrum voor Wiskunde en Informatica (CWI).
25. OpenCV (2021). Welcome to OpenCV-Python Tutorials's documentation! Available at: https://docs.opencv.org/4.5.2/d6/d00/tutorial_py_root.html (Accessed: 6 April 2023).
26. PyTesseract (2021). pytesseract/pytesseract: An Optical Character Recognition (OCR) tool for python. That is a wrapper for Google Tesseract-OCR. Available at: https://github.com/madmaze/pytesseract (Accessed: 6 April 2023).
27. Redmon, J. and Farhadi, A. (2016). You only look once: Unified, real-time object detection. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 779-788).
28. Flask (2021). Welcome to Flask. Available at: https://flask.palletsprojects.com/en/2.1.x/ (Accessed: 5 April 2023).
29. SQLAlchemy (2021). SQLAlchemy - The Database Toolkit for Python. Available at: https://www.sqlalchemy.org/ (Accessed: 5 April 2023).
30. Fielding, R. T., Gettys, J., Mogul, J. C., Frystyk, H., Masinter, L., Leach, P. J., & Berners-Lee, T. (1999). Hypertext Transfer Protocol -- HTTP/1.1. IETF RFC 2616.
31. Historia del perceptron timeline. Timetoast timelines. (n.d.). https://www.timetoast.com/timelines/historia-del-perceptron (Accessed: 2 February 2023).
32. Saha, S. (2022, November 16). A comprehensive guide to Convolutional Neural Networks-the eli5 way. Medium. https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53 (Accessed: 2 February 2023).
33. Fig. 1. Artificial Neural Network Architecture. - researchgate. (n.d.). https://www.researchgate.net/figure/Artificial-neural-network-architecture_fig1_346012838 (Accessed: 4 February 2023).
34. Young, B. (n.d.). Yolov5 network architecture. YOLOV5 network architecture - Programmer Sought. https://www.programmersought.com/article/12098280279/ (Accessed: 6 April 2023).