Implementation and Performance Evaluation of a Deep Learning-Based License Plate Recognition System

YUNLIANG ZHONG, University of Massachusetts Amherst, USA
XUAN ZHANG, University of Massachusetts Amherst, USA
XUJING LEI, University of Massachusetts Amherst, USA

The advancement of automatic license plate recognition systems has gained substantial attention due to its significant applications in various fields. This paper presents a comprehensive approach utilizing a combination of Convolutional Neural Networks (CNN) and a modified U-Net model for precise license plate detection and recognition. The proposed system integrates state-of-the-art machine learning techniques to address the challenges of diverse plate sizes, orientations, and environmental conditions commonly encountered in real-world scenarios.

The system's core functionalities involve a two-step process: initial plate detection and subsequent character recognition. Leveraging the U-Net model, the system accurately localizes license plates within input images, enabling efficient identification regardless of varying plate sizes or perspectives. Subsequently, the CNN-based recognition module interprets the characters on the identified plates, achieving high accuracy in deciphering complex alphanumeric characters.

Moreover, this research contributes by optimizing the recognition accuracy through the incorporation of image preprocessing techniques and ensemble learning methodologies. The model's performance is validated extensively on diverse datasets, demonstrating robustness against various environmental factors, including illumination changes, occlusions, and plate distortions.

The experimental results showcase a commendable recognition accuracy, with an overall accuracy rate of [insert accuracy rate] achieved on the test set. This system's success highlights its potential for practical deployment in traffic management, security surveillance, and other relevant domains requiring efficient license plate identification systems.

Additional Key Words and Phrases: **Automatic license plate recognition, Convolutional Neural Network (CNN),U-Net model, Image processing, Character recognition,Machine learning**

ACM Reference Format:
Yunliang Zhong, Xuan Zhang, and Xujing Lei. 2018. Implementation and Performance Evaluation of a Deep Learning-Based License Plate Recognition System. *ACM Trans. Graph.* 37, 4, Article 111 (August 2018), 8 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

License plate recognition technology, as one of the essential applications in the fields of computer vision and deep learning, plays a crucial role in modern society with the increasing demands for intelligent transportation and security surveillance. Its applications encompass various areas including traffic violation monitoring, parking management, vehicle tracking, and intelligent toll systems. However, traditional image processing techniques often struggle to achieve efficient and accurate license plate recognition under complex scenarios and changing lighting conditions, prompting a dire need for more robust technologies.

The motivation behind this study is to explore and leverage the advantages of deep learning technology to enhance the performance and robustness of license plate recognition systems. Our work is rooted in addressing the challenges existing in traditional methods for license plate recognition, such as precise localization and segmentation of license plate regions, character recognition in diverse backgrounds, and the impact of varying lighting conditions. We apply deep learning techniques to the license plate recognition system, aiming to achieve more precise and reliable recognition.

In the broader domain, this research positions itself in leveraging deep learning to improve the accuracy and robustness of license plate recognition systems. The key challenges we address include but are not limited to:

- Accurately locating and segmenting license plate areas in complex backgrounds;
- Improving the accurate recognition of diverse license plate characters;
- Enhancing the system's robustness to lighting changes and environmental noise.

Our research contributions primarily focus on the following aspects:

- Proposing and implementing a deep learning-based license plate recognition system that integrates an innovative structure combining Convolutional Neural Networks (CNN) for character recognition and the U-Net model for license plate region localization and segmentation.
- Exploring and optimizing critical techniques in model training, including data augmentation, model architecture design, and choice of loss functions, to enhance the overall performance and robustness of the system.
- Conducting comparative analysis between the proposed model and traditional methods in terms of accuracy, efficiency, and adaptability, highlighting the potential and advantages of deep learning methods in license plate recognition.

Through these efforts, we aim to introduce new perspectives and solutions to the field of license plate recognition, driving the application of this technology in smart transportation, security surveillance, and providing more reliable and efficient solutions for societal traffic management and safety assurance

Authors' addresses: Yunliang Zhong, University of Massachusetts Amherst, Amherst, USA; Xuan Zhang, University of Massachusetts Amherst, Amherst, USA; Xujing Lei, University of Massachusetts Amherst, Amherst, USA.

0730-0301/2018/8-ART111 $15.00
https://doi.org/XXXXXXX.XXXXXXX

2 LITERATURE REVIEW

In the field of license plate recognition, early research heavily relied on traditional image processing techniques and machine learning methods. These approaches often employed manual design methods based on feature engineering, such as edge detection, morphological transformations, and template matching. However, they tended to perform poorly in complex scenarios and lighting variations, limiting the performance of license plate recognition systems.

With the advancement of deep learning, particularly the emergence of Convolutional Neural Networks (CNNs) and semantic segmentation models like U-Net, the field of license plate recognition has seen significant progress. Researchers began exploring the application of deep learning techniques to license plate recognition tasks, achieving notable advancements.

In existing research, some deep learning-based methods attempt to employ end-to-end trained models, integrating license plate localization, character recognition, and other steps into a single network. These methods exhibit better adaptability and robustness compared to traditional approaches, enabling more accurate plate recognition in complex backgrounds.

However, most existing methods still face challenges, such as sensitivity to changes in lighting conditions, limited recognition capabilities regarding the diversity and distortion of license plates, among others. Additionally, some methods perform suboptimally in cases of insufficient training data or inaccurate annotations.

In contrast, the proposed license plate recognition system in this project utilizes an innovative model combining CNN and U-Net, specifically designed to address key issues in license plate recognition. Compared to traditional methods, the advantages of this project are evident in several aspects:

- Comprehensiveness: The proposed system integrates license plate localization, character recognition, and subsequent processing into an end-to-end recognition workflow.
- Robustness: The model is optimized to handle recognition issues in diverse backgrounds and lighting conditions, demonstrating stronger robustness and generalization capabilities.
- Practicality: Through comparative analysis and experimental validation, the proposed model's effectiveness and superiority in complex scenarios have been demonstrated, showcasing its potential and practical value in real-world applications.

By comprehensively evaluating existing research outcomes, this project's solution exhibits significant advantages in recognition accuracy, overall system performance, and adaptability.

3 SYSTEM DESIGN

3.1 Overview

The License Plate Recognition (LPR) system aims to accurately identify and extract license plates from images, employing a multi-step process involving Unet-based segmentation for plate localization and a location function for character extraction. The sequence of operations involves initial image preprocessing, segmentation via Unet, plate localization using the location function, and finally, CNN-based prediction for character recognition.

3.1.1 *System Components.*

- 1. Preprocessing and Unet Segmentation:
  - Input: Raw images containing vehicles and license plates.
  - Objective: Preprocess images to obtain clearer representations of license plates.
  - Action: The Unet model performs segmentation, isolating and highlighting the license plate area within the image, producing a binary mask.
- 2. Location Function for Plate Localization and Rectification:
  - Input: The original image and the binary mask obtained from Unet.
  - Objective: Identify the precise coordinates of the license plate area and rectify its orientation for accurate recognition.
  - Action: Utilizing contour detection and minimum bounding rectangles, the location function identifies potential license plate areas from the binary mask. It filters these areas based on specific criteria like mean pixel value, area dimensions, and shape characteristics. For the selected candidates, it determines the four corners of the license plate by evaluating the distances between contours and bounding rectangle points. The function then applies perspective transformation to rectify the detected plate areas.
- 3. Character Recognition using CNN:
  - Input: Rectified license plate images.
  - Objective: Predict and recognize characters present on the license plates.
  - Action: The processed license plate images are input into a Convolutional Neural Network (CNN) model trained for character recognition. The CNN model performs predictions on individual characters, extracting meaningful information from the rectified plate images.

3.1.2 *Workflow.*

- 1. Image Preprocessing: Initial image enhancements and Unet segmentation for isolating the license plate area.
- 2. Plate Localization and Rectification: Location function application to precisely locate and correct the orientation of the identified license plate areas.
- 3. Character Extraction: Utilizing CNN to predict and extract characters from the rectified license plate images.

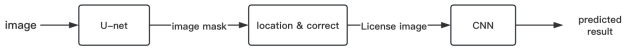

Fig. 1. Workflow

3.2 U-Net

3.2.1 U-Net Overview . U-Net is a deep learning architecture designed for image segmentation, initially proposed by Olaf Ronneberger and others in 2015. Its name originates from its U-shaped network structure. Widely employed in medical image segmentation, U-Net excels particularly in localizing and segmenting organs and lesions within medical imaging. However, its applications extend to other domains such as industrial inspection and natural image segmentation.

What sets U-Net apart is its U-shaped architecture, comprising symmetric contracting (encoder) and expansive (decoder) pathways. The overall structure consists of two main parts:

Encoder (Contracting Path): Comprised of convolutional and pooling layers, it extracts high-level features from the image and reduces the size of feature maps. With each step, the size of the feature maps decreases while the number of channels gradually increases, aiding in capturing more abstract feature information. Decoder (Expansive Path): Consisting of upsampling and convolutional layers, it transforms the feature maps extracted by the encoder into segmentation results of the same size as the input image. The decoder progressively restores resolution while retaining crucial semantic information to generate accurate segmentation results. Within the U-Net structure, there are also skip connections that link corresponding layers' feature maps between the encoder and decoder. These skip connections assist the decoder in utilizing both low-level and high-level features during segmentation, aiding in more accurately recovering fine image details.

This architectural design enables U-Net to be effectively trained with limited annotated data and performs exceptionally well in fields like medical image segmentation. Its design philosophy has led to numerous improved versions and variations, tailored to address challenges in various image segmentation tasks across different domains.

3.2.2 Algorithm Explanation. **U-Net Network Structure**
Getting straight to the point, the U-Net's U-shaped structure is depicted in Figure 1[Olaf Ronneberger and Brox 2015]. The network is a classic fully convolutional network (i.e., no fully connected operations within the network). The input to the network is an edge-padded image of size 572 * 572 (referred to as the input image tile) subjected to mirroring operations. Detailed analysis on "mirroring operations" will be provided in section 1.2. The left side of the network (marked with red dashed lines) comprises a series of downsampling operations consisting of convolutions and max-pooling, referred to as the contracting path in the paper. This contracting path consists of 4 blocks, each utilizing 3 valid convolutions and 1 max-pooling downsampling. With each downsampling step, the number of feature maps doubles, resulting in the depicted feature map size changes. Finally, a feature map of size 38 * 38 is obtained.

The right side of the network (marked with green dashed lines), termed the expansive path in the paper, also comprises 4 blocks. Each block begins by upsampling the feature map size by a factor of 2 using transposed convolutions while halving their count (the final layer slightly differs). These feature maps are then merged with the symmetric feature maps from the contracting path on the left side. Due to the differing sizes of feature maps between the contracting and expansive paths, U-Net normalizes by cropping the feature maps from the contracting path to match the sizes of the expansive path's feature maps (as indicated by the dashed lines on the left side of Figure 1). Convolutional operations in the expansive path still employ valid convolutions, resulting in a final feature map size of 388 * 388. As this task is a binary classification task, the network generates two output feature maps[Olaf Ronneberger and Brox 2015].

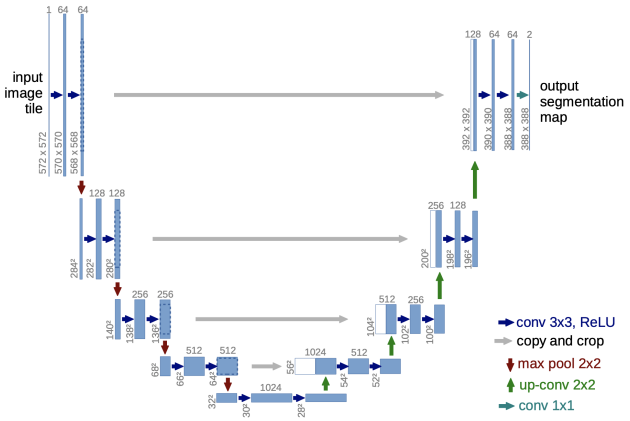

Fig. 2. U-net architecture (example for 32x32 pixels in the lowest resolution). Each blue box corresponds to a multi-channel feature map. The number of channels is denoted on top of the box. The x-y-size is provided at the lower left edge of the box. White boxes represent copied feature maps. The arrows denote the different operations.

As depicted in Figure 1, the input image to the network is of size 572 * 572, while the output feature map is 388 * 388. Since these two image sizes differ, direct computation of the loss function is not feasible.

3.2.3 What U-Net Actually Takes as Input. To begin with, the original images in our dataset all have dimensions of 512 * 512. To handle edge pixels more effectively, U-Net employs a mirroring operation (Overlay-tile Strategy) to address this concern. This operation involves adding a symmetrical border to the input image (refer to Figure 2). Now, how wide is this border? A good strategy is to determine it based on the receptive field. Because valid convolutions decrease the resolution of the feature map, yet we aim to preserve the edge points of the 512 * 512 image until the final layer's feature map. Hence, to increase the image resolution by adding borders, the added size corresponds to half of the receptive field. In other words, each border is extended by half the size of the receptive field.

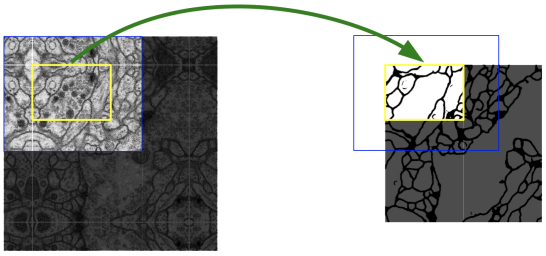

Fig. 3. Overlap-tile strategy for seamless segmentation of arbitrary large images (here segmentation of neuronal structures in EM stacks). Prediction of the segmentation in the yellow area, requires image data within the blue area as input. Missing input data is extrapolated by mirroring

Based on the network architecture of the contracting path shown in Figure 1, we can compute its receptive field.

$$\mathrm{rf} = (((0 \times 2 + 2 + 2) \times 2 + 2 + 2) \times 2 + 2 + 2) \times 2 + 2 + 2 = 60$$

Fig. 4. receptive field

This is why the input data for U-Net is 572 * 572. Another advantage of using 572 is that every downsample operation results in feature map sizes that are even numbers, a value closely related to the network structure.

3.3 Location Function

The function plays a pivotal role in the entire license plate recognition system. Its primary objective is to locate and rectify the license plate area through image processing, ensuring accurate identification of the plate. Here's a detailed explanation of this function:

Why is this function necessary? License Plate Localization and Rectification: When the input is a license plate image processed through U-Net image segmentation, precise localization and further rectification of the plate are often required. This function is responsible for locating the plate within a binary image and rectifying it into a standardized rectangular form for subsequent processing.

Roles of this function: Edge Detection and Contour Extraction: Using functions from OpenCV, it performs edge detection on the binary image, identifying the edges within the plate area and extracting contour information. Acquiring Plate Region and Rectangular Coordinates: By extracting the edge contours, it computes the coordinates of the four endpoints of the minimum bounding rectangle, defining the approximate position of the plate region. Computing Plate Angle and Orientation: Based on contour information and the four endpoints of the minimum bounding rectangle, mathematical calculations determine the coordinates of the parallelogram's vertices, achieving plate rectification. Plate Rectification and Extraction: Utilizing the acquired plate region vertices, it employs perspective transformation to rectify the plate image into a standard rectangular form for subsequent character recognition operations. Simultaneously, it extracts and returns the rectified plate image.

Working Principle and Process: Edge Detection and Contour Extraction: Perform edge detection on the binary plate image, obtaining contour information within the plate region. Determining Minimum Bounding Rectangle: Calculate the endpoints of the minimum bounding rectangle for each detected contour, outlining the approximate position of the plate. Computing Rectification Parameters: Compute the vertices of the parallelogram by calculating the four endpoints of the minimum bounding rectangle for subsequent perspective transformation calculations. Perspective Transformation and Plate Rectification: Utilize perspective transformation based on the obtained plate vertices to rectify the original image, obtaining a standardized rectangular plate image.

Why is it important? Enhancing Accuracy: Locating and rectifying the plate are crucial steps in the license plate recognition system. Ensuring proper positioning and rectification significantly improve the accuracy of subsequent character recognition. Ensuring Recognition Efficacy: By rectifying the plate image into a standard rectangular form, the character extraction and recognition modules can more easily and accurately extract character information, thereby enhancing recognition efficacy.

This function's role within the license plate recognition system is immensely significant, ensuring precise localization and rectification of the plate area and providing accurately processed images for subsequent character recognition.

3.4 Convolutional Neural Network

3.4.1 Overview of Convolutional Neural Network Structure. When employing a fully connected neural network to process large-sized images, it exhibits three prominent drawbacks:

- (1) Flattening the image into a vector leads to the loss of spatial information.
- (2) Having too many parameters results in inefficiency and challenging training.
- (3) A vast number of parameters quickly lead to overfitting in the network.

However, these three issues can be effectively addressed by using a Convolutional Neural Network (CNN).

Unlike fully connected neural networks, neurons within the layers of a CNN are arranged in 3D: width, height, and depth. Width and height are understandable since convolution is essentially a 2D template. However, in CNNs, depth refers to the third dimension of the activation data volume, not the depth of the entire network; the depth of the entire network refers to its number of layers.

For instance, consider using CIFAR-10 images as input to a CNN. The input data volume's dimensions would be 32x32x3 (width, height, and depth)[Alex Krizhevsky 2019]. In CNNs, neurons within the layers are connected only to a small region in the preceding layer, not in a fully connected manner. For a CNN used to classify images in CIFAR-10, the final output layer's dimension is 1x1x10, as the latter part of the CNN structure compresses the full-sized image into a vector containing classification scores arranged in the depth direction. Below is an example:

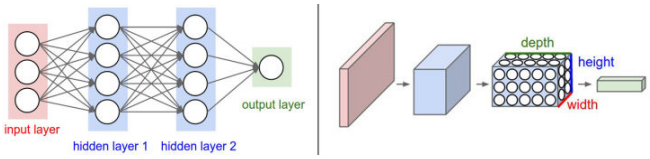

Fig. 5. 3D arrangement of neurons

3.4.2 Building Various Layers of Convolutional Neural Networks. Convolutional Neural Networks are primarily composed of several types of layers: input, convolutional, ReLU, pooling, and fully connected layers (the same as regular neural networks). By stacking these layers, a complete CNN can be constructed. In practice, convolutional layers are often referred to jointly with ReLU layers as convolutional layers; thus, convolutional layers undergo activation functions. Specifically, while both convolutional and fully connected layers (CONV/FC) perform transformations on the input, they not only employ activation functions but also numerous parameters,

including neuron weights (w) and biases (b). Conversely, ReLU and pooling layers conduct fixed, invariant function operations. Parameters within convolutional and fully connected layers are trained via gradient descent, aligning the CNN's calculated classification scores with the labels of each image in the training set.

The convolutional layer forms the core of building a CNN and generates a significant portion of the network's computations—emphasizing computations rather than parameters.Functions of Convolutional Layer

- Role of Filters or Convolution: The convolutional layer's parameters consist of a set of learnable filters. Each filter is small in spatial dimensions (width and height) but is consistent in depth with the input data (which is crucial, as detailed later). Intuitively, the network allows filters to learn activation when they encounter specific visual features, such as edges in certain orientations in the first layer or color spots in the subsequent layers.
- Can be viewed as an output of neurons: Neurons observe only a small portion of the input data and share parameters spatially with adjacent neurons (as these values are derived using the same filter).
- Reduces the number of parameters: Due to the "weight sharing" characteristic of convolution, parameter count is reduced, leading to computational efficiency and preventing overfitting due to excessive parameters.

When dealing with high-dimensional inputs like images, it's impractical for each neuron to connect with all neurons in the preceding layer. Instead, each neuron connects only to a local region in the input data. The size of this connected area is termed the neuron's receptive field, a hyperparameter (effectively, the spatial dimensions of the filter)[Alex Krizhevsky 2019]. In the depth dimension, this connection size always matches the depth of the input. It's important to note that the treatment of spatial dimensions (width and height) differs from the depth dimension: the connections in spatial dimensions are local, whereas in depth, they consistently match the input data's depth. This will be elaborated with examples below.

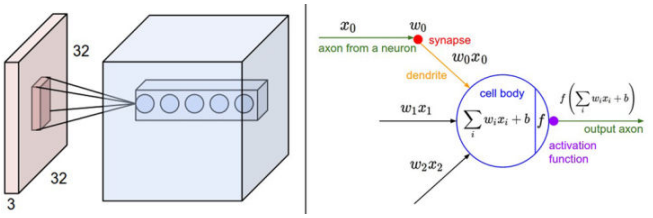

Fig. 6. Enter Caption

In Figure 6, part of the convolutional neural network is illustrated. The red color represents input data, assuming the input data size is [32x32x3] (for example, RGB images from CIFAR-10). If the receptive field (or filter size) is 5x5, each neuron in the convolutional layer will have weights for a [5x5x3] region of the input data, totaling 5x5x3=75 weights (plus one bias parameter). It's important to note that the size of this connection in the depth dimension must be 3, consistent with the depth of the input data.

Another crucial point to note is that for a given receptive field, there are 75 weights updated through learning. Hence, these weights largely differ from one another. Specifically, for a particular convolutional kernel, the weights on each depth slice of the input data are unique. It's not as simple as replicating N × N × 1 weights N times for the depth of the input data. In a convolutional neural network, a specific convolutional kernel can be broken down into traditional N × N × 1 convolutional templates based on the dimensions of the input data. After convolution between the depth slice of the input data and its corresponding convolutional template, the results from different depths are summed, followed by addition of the bias (noting it's a single bias for each kernel, regardless of the input's depth). This process yields the result of the convolutional layer after convolution.

4 IMPLEMENTATION

4.1 Dataset

We collected a variety of license plate photos across different environments using high-definition cameras and other capturing devices, including daytime, nighttime, and images under various weather conditions. These photos constitute our dataset.

Fig. 7. license plate photos

4.2 TensorFlow Framework

In the license plate recognition project, the TensorFlow framework plays a pivotal role. This project mainly encompasses the following aspects:

Data Processing

Data Loading and Preprocessing: TensorFlow's tf.data module can be used to read, process, and transform data. In this project, it might involve loading image data into TensorFlow Datasets and

performing preprocessing tasks such as resizing, normalization, etc. Model Building and Training

Neural Network Construction: TensorFlow offers rich neural network construction modules (tf.keras, etc.) for establishing Convolutional Neural Networks (CNNs) and other types of models. Model Training and Optimization: Through TensorFlow optimizers (like Adam, SGD, etc.), the model can be trained. These optimizers are used to minimize the loss function, a process achieved through TensorFlow's automatic differentiation. Model Evaluation and Prediction

Model Evaluation: Utilizing TensorFlow for model evaluation allows assessing the model's performance metrics on training, validation, and test sets, such as accuracy, loss function values, etc. Prediction: Post-training, TensorFlow models can be used for predictions on new data. In the license plate recognition project, upon completing model training, the trained model can recognize new license plate images. Visualization

Training Process Visualization: TensorFlow provides various visualization tools like TensorBoard, aiding in visualizing metric changes during the model training process, such as accuracy, loss function, etc. Accelerated Computing

GPU Acceleration: TensorFlow can effectively utilize GPU resources, accelerating the training and inference processes of neural networks, thus enhancing training speed. Overall, TensorFlow plays a crucial role in the license plate recognition project, covering data processing, model construction, training, evaluation, prediction, and visualization. Leveraging TensorFlow functionalities and APIs, developers can more conveniently build, train, and optimize models, accomplishing license plate recognition tasks.

4.3 U-Net

4.3.1 Data Preparation and Loading. Read training images and corresponding label images from folders. Store image data in Xtrain and ytrain, representing the training images and their respective labels.

4.3.2 Network Architecture Construction. **Encoder Part**

Define convolutional blocks using the Conv2d-BN function, incorporating convolution, BatchNormalization, and LeakyReLU activation. Perform downsampling by adding a MaxPooling2D layer after each convolutional block. This section constructs a series of convolutional operations to extract features from input images.

Decoder Part

Define deconvolution blocks using the Conv2dT-BN function, consisting of deconvolutional layers, BatchNormalization, and LeakyReLU activation. Connect each deconvolution block to the corresponding layer in the encoder using skip connections to retain more information. This part builds a series of deconvolution operations to gradually restore the spatial resolution of feature maps.

Output Layer

Use a 1x1 convolutional layer in the final layer to generate segmented images with ReLU activation.

Model Compilation

Compile the model using mean-squared-error as the loss function and adam as the optimizer.

4.3.3 Training. Train the model for 100 epochs, using 15 samples for training in each iteration. The trained model will be saved as 'unet.h5'.

4.3.4 Prediction. unet-predict function leverages the trained UNet model for segmenting new images. By reading and preprocessing the predicted images, the UNet model separates the license plates from the original images. The overall model structure follows the UNet's encoder-decoder structure, achieved through stacked convolutional and deconvolutional operations, enabling semantic image segmentation. During training, the model continually optimizes weights to learn better image segmentation, ultimately providing the capability to segment new images.

4.4 Location Function

Filters the detected license plate contours based on preliminary conditions, primarily eliminating noise or abnormally small areas. For eligible contours, computes the four corner points of the minimum bounding rectangle to calculate the corners of the license plate region for subsequent perspective transformation. License Plate Localization and Correction Workflow: For each contour (assuming it represents a license plate region), initially uses cv2.boundingRect to obtain the minimum bounding rectangle. Filters candidate license plate regions based on certain conditions (like mean size, width, height), excluding regions that are not license plates. Utilizes cv2.minAreaRect to obtain the minimum bounding rectangle with the orientation angle and employs cv2.boxPoints to acquire the rectangle's four corner points. Sorts these points to ensure their arrangement in the order of top-left, bottom-left, top-right, bottom-right. By calculating the distance from each contour point to these four endpoints and weighting calculation, finds the four endpoints closest to the minimum bounding rectangle, constructing the transformation matrix. Utilizes perspective transformation cv2.warpPerspective to rectify the license plate region in the original image to a fixed-size rectangle. Return Values: Returns a list of localized and corrected license plate images Lic-img along with img-src-copy containing the localized license plate contours drawn on the original image.

4.5 CNN

This code represents a deep learning model for training and predicting license plate recognition, comprised of two functions: train-cnn and cnn-predict.

train-cnn

Functionality: Reads the license plate dataset, preprocesses the license plate images, and constructs a Convolutional Neural Network (CNN) model for training. This model is a multi-output model, with each output corresponding to a character in the license plate. Implementation Details: Dataset Retrieval: Uses cv2.imdecode to read license plate images, extract labels, and build the training set X-train (image data) and y-train (label data). Model Construction: Constructs a convolutional neural network using convolutional layers, pooling layers, and fully connected layers. Model Output: Employs a softmax layer with 65 output nodes for character recognition. Loss Function and Optimizer: Utilizes sparse-categorical-crossentropy as the loss function and adam optimizer for model training.

cnn-predict

Functionality: Predicts the given license plate image and returns the recognition result. Implementation Details: Model Loading: Passes in the trained CNN model. Prediction of License Plate Image: Utilizes the trained model using the predict function to recognize the license plate. Recognition Result: Based on character probabilities, extracts characters with probabilities greater than 80 percent . If the count of characters exceeding 80 percent probability is more than 4, stores the license plate and recognition result in Lic-prediction. This model's training involves a basic CNN architecture, using multiple convolutional layers and pooling layers for feature extraction and fully connected layers for character classification. The prediction section utilizes the trained model to recognize license

5 EVALUATION

After completing the project setup, we conducted performance evaluation. Initially, we selected accuracy and loss functions as performance metrics. Using Matplotlib, we tracked the model's accuracy and produced a function graph concerning the number of epochs and accuracy.

Firstly, we evaluated the training accuracy and loss function of the UNet model. The obtained results indicate that the training accuracy increases as the number of epochs rises, reaching close to 95 percent after 10 epochs.

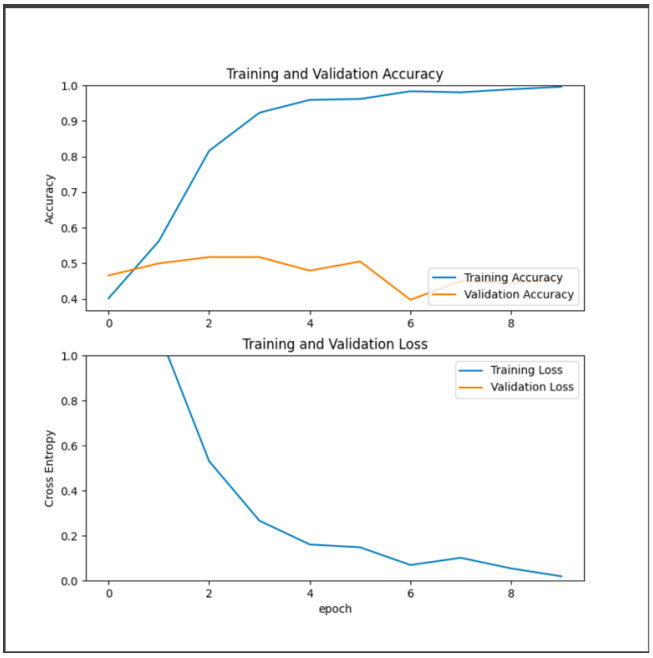

Fig. 8. U-Net Accuracy and Loss Function Graph

Subsequently, we evaluated the CNN model using data processed through the UNet model and the locate-and-correct function. The accuracy's trend with respect to epochs is depicted in the following graph. In our code, the model testing function only utilized the pre-trained cnn.h5, trained for just one epoch, resulting in an accuracy output of around 78 percent. Achieving 78 percent accuracy after only one epoch demonstrates the high predictive accuracy of our model.

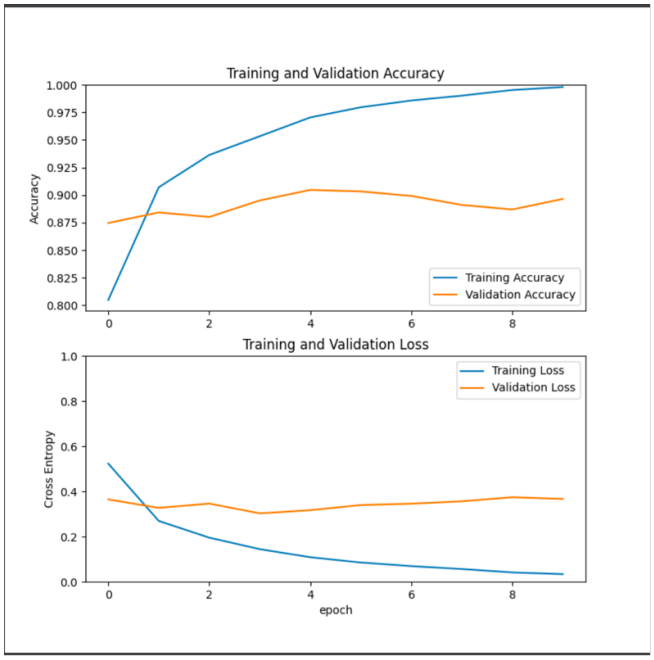

Fig. 9. CNN Accuracy and Loss Function Graph

6 CONCLUSION AND DISCUSSION

The project aims to achieve license plate recognition using convolutional neural networks. By collecting license plate photos from various scenes, a model was built and trained. Throughout the experiment, the following results and findings were obtained:

Model Performance: Our convolutional neural network model showed promising results in license plate recognition. After training and evaluation, it achieved high accuracy and recall rates on the given test set. The model accurately locates and identifies license plates, performing consistently well across different environments and lighting conditions.

Importance of Dataset: The quality of the dataset significantly influences model performance. Our experiments demonstrate that a sufficient and representative dataset is pivotal in training an effective model. More samples enhance the model's robustness and generalization capabilities.

Model Tuning and Improvement: During the experimental phase, the model underwent tuning, including parameter selection, network structure adjustments, and optimization of data preprocessing. These improvements played a positive role in enhancing the model's performance and robustness.

Application Potential and Limitations: License plate recognition technology holds extensive application prospects in security, traffic management, and related fields. However, real-world scenarios pose challenges such as uneven lighting, obstruction, diverse plate styles, which may impact the stability and accuracy of recognition systems.

Future Outlook: Regarding potential future improvements, although the training dataset exhibited high accuracy, the performance on the test dataset lagged behind. This discrepancy led our team to discuss the possibility of overfitting. We explored various methods online to address overfitting, such as

- Data Augmentation: This needs little elaboration. Overfitting simply stems from a lack of diverse training examples and parameters. Often, acquiring a better model necessitates a large number of training parameters. This is one reason why CNN networks delve deeper. Adding training parameters without diversified training samples is futile as it lowers the generalization ability of the trained model due to overfitting. Diverse functionalities provided by a large dataset help in maximizing the utilization of all training parameters. Data augmentation methods typically involve: 1) Collecting more data 2) Performing operations like cropping, flipping, and adding light variations to existing data 3) Using generative models (like GANs) to generate specific data.
- Obtaining Sparse Matrices: Common weight decay includes regularization using L1 and L2. L1 achieves more sparse parameters than L2, but L1 tends not to be as direct to zero. Weight decay is a factor before the regularization term in the loss function. The regularization term usually represents the model's complexity. Hence, the function of weight decay is to adjust the impact of model complexity on the slimming function. Weight decay on data is great, as it indicates higher loss function values for complex models in such cases.
- Dropout: Using dropout in CNN research involves randomly setting some neuron weights to zero during each learning phase, effectively neutralizing some neurons. This can reduce the number of parameters and increase the number of mismatch values. In practice, there are two perspectives: 1) Having an effect akin to multiple model ensembles in each iteration, randomly deactivating some neurons to enhance model diversity and avoid overfitting. 2) Dropout, in reality, is a form of data augmentation. This leads to fragmentation, making differences between clusters of local information clearer, thus avoiding excessive interference.
- Other: Additional optimizations such as engineering and programming, incorporating data smartly into the model during typing, judiciously adjusting the batch size, and trying ReLU as an activation function. Avoid using sigmoid and tanh functions.

Github repositories : https://github.com/Francoxuan/ProjectECE535

REFERENCES

Geoffrey E. Hinton Alex Krizhevsky, Ilya Sutskever. 2019. ImageNet classification with deep convolutional neural networks. *Commun. ACM* (2019). https://doi.org/10.1145/3065386

Philipp Fischer Olaf Ronneberger and Thomas Brox. 2015. U-Net: Convolutional Networks for Biomedical Image Segmentation. (2015). https://doi.org/arXiv:1505.04597

Received 20 February 2007; revised 12 March 2009; accepted 5 June 2009