National Research University

Higher School of Economics

Faculty of Computer Science

Computer Vision: Software for License Plate Verification

Made by students:

Tomaily Tatiana, Gorelskii Roman, Sergazin Iskander

Course title: Computer Vision

Supervisor: Kopylov Ivan

Moscow, 2025

Introduction

Description of the problem

There are many private areas, where there is a controlling of vehicles which are allowed to enter this area. Manual verification is often slow, error-prone, and inconvenient. Our project solves this problem by developing an automatic license plate recognition system that detects a car’s license plate and checks if it exists in a pre-approved database. If the plate is found, the system confirms access with a "yes" response; otherwise, it denies entry.

Application

The system responds with a simple "yes" or "no" indicating whether the vehicle is authorized or not. The license plate database is managed and updated through a Telegram bot, which allows for convenient and user-friendly data collection.

This system is designed to automate vehicle access control and useful for residential and parking areas, or some private events. Now the system is basically MVP, and potentially, it can be integrated with automated gates, allowing only authorized vehicles to pass through them without human intervention. Potential users include residential complexes, corporate offices, and event organizers seeking efficient access control.

Problem definition

The main goal of this project is to automatically recognize vehicle license plate numbers from camera footage or images and check whether the detected number exists in a predefined access list. The data used includes images from surveillance cameras, and the core methods are Optical Character Recognition (OCR) for license plate detection, image preprocessing techniques to capture plates from camera and database lookup techniques. In the future, the system should be assisted with physical barriers and cameras to allow vehicles to enter.

Technology stack

The stack includes:

- Python
- OpenCV
- Telegram Bot Api
- OCR method implementation

Project stages

The project stages include:

- Review of existing solutions
- Data collection
- Bot implementation

- License plate detection and recognition
- Implementation of system matching the plate numbers with database and returning a response

Review of Existing Solutions

Approaches already used

Automatic License Plate Recognition (ALPR) systems are widely used and typically have license plate detection: traditional methods using OpenCV with contour detection or edge detection (e.g., Canny) and deep learning-based object detection models like YOLO (You Only Look Once), SSD, or Faster R-CNN for more robust and accurate localization. Also, character segmentation is used through thresholding. Reading the characters from the detected plate – optical character recognition (OCR) – include Tesseract OCR (open-source engine), EasyOCR and deep learning-based CRNN (Convolutional Recurrent Neural Networks) models.

Reason of the proposed methods chosen

In our project, we use a lightweight and fast license plate OCR pipeline inspired by the Fast Plate OCR approach. The reasons for our method choices are:

- **Efficiency and Speed:** The model is optimized for real-time processing, which is crucial for integration with systems like automatic gates.
- **Simplicity:** The solution is easy to deploy and maintain, especially when with a user-friendly Telegram bot for managing access lists.
- **Accuracy in Diverse Conditions:** Deep learning-based OCR provides better generalization across different lighting and camera angles compared to traditional methods.
- **Scalability:** The architecture allows future enhancements, such as adding support for regional formats or improving the detection model using YOLO or other modern detectors.

Data Preparation

Description of the dataset and Visualization

Source: For training and testing the license plate recognition system, we used an open-source dataset from the Fast Plate OCR GitHub repository[1]. This dataset contains images of vehicles with visible license plates

Quantity: 2874 for training and 764 for validation

Format: images in .jpg format, Argentina plates

[1] https://github.com/ankandrew/fast-plate-ocr

Figure 1. The example of the data image

Data preprocessing

First of all, only the license plates are in training so there was no read in detecting them and segmenting from other objects. The augmentation technique is used to get better predictions because of generalisation: random shifts, scaling, brightness and contrast changes with monitor blur used. The example of before/after is in the figure[2].

Figure 2. Data before and after augmentation

Model Development

Neural network used

For solving the problem we have decided to use an existing Fast Plate OCR methodology [1]. The neural network used takes as input a grayscale image of the plate and then normalizes it

between [0,1]. Once the preprocessing of the image is complete the output is passed to the backbone which consists of three consecutive [block_pool_conv, block_bn, block_bn] blocks that increase the number of channels from 32 to 1024. The block_pool_conv is a layer which consists of a pooling layer (either max_pooling or average_pooling) for feature extraction, followed by a CNN layer and then batch_normalization which is passed through an activation function. The block_bn layer is similar to the black_pool_conv layer but omits the pooling layer. The backbone outputs 1024 channels which are passed into the head consisting of a global pooling layer, a dropout layer and parallel linear layers with softmax activation (of size= len(vocab)). This outputs elements from a dictionary of possible license plate characters which are concatenated to provide the final license plate optical character recognition. A diagram of the architecture is provided in fig3 and a diagram of the Black Pool Conv and Block BN is provided in fig4 (for more details please see the code repository) .

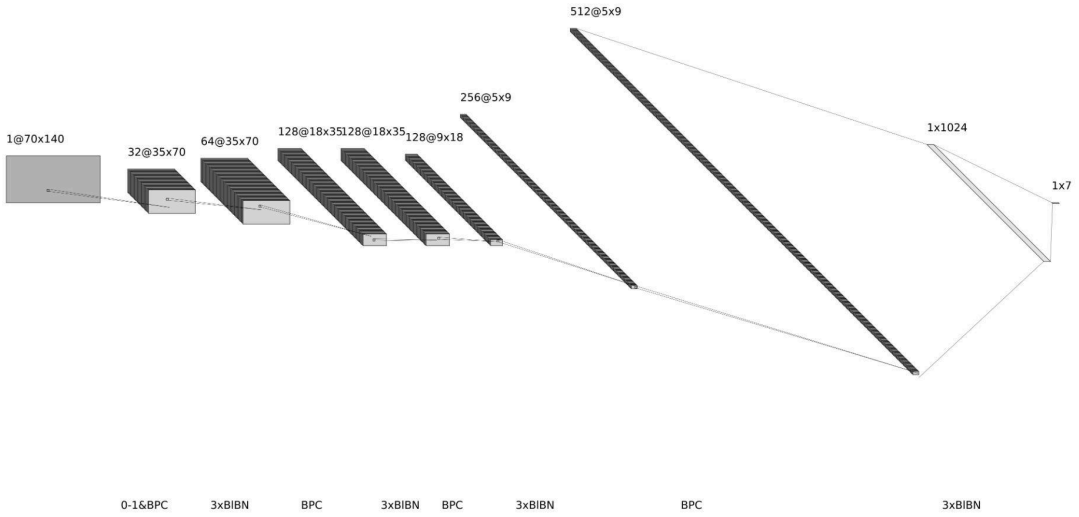

Figure 3. Neural Network Architecture

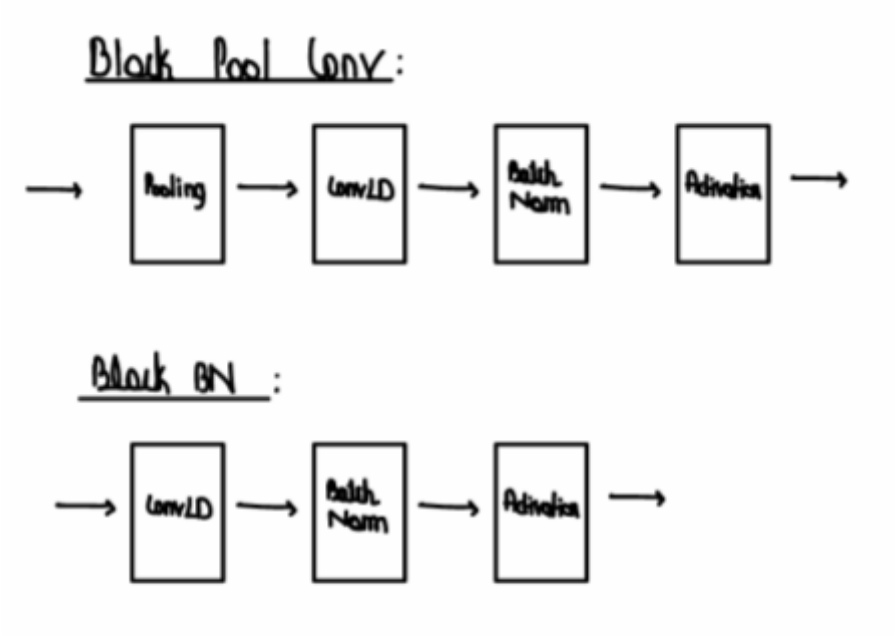

Figure 4. “Black Pool Conv” and “Block BN” blocks

Why the approach was chosen

The methodology was chosen because it is a light-model which can be trained from scratch. This is beneficial because we can easily collect a dataset consisting of Russian Plate numbers, alter the configs and quickly train the model from scratch. This allows us to easily adapt the model to our own use. At the same time the approach provides a wide variety of pre-trained models trained on diverse datasets, including a worldwide plate dataset which allows us to test our approach on a pre-trained model with good performance in terms of model accuracy. Taken both of these advantages together this lightweight and adjustable framework would allow quick deployment and testing of a minimal working product in a time-constrained environment.

Training strategy & Hyperparameters

To train the model a config file is loaded that contains information about the training data. The model was trained to predict 7 plate slots using the latin alphanumeric alphabet and input images of 70x140. This information was loaded through a .yaml file.

The model was trained with a reduce_lr_factor, early_stopping_patience and a batch_size of 128. The reduce_lr_factor is a parameter that reduces the learning rate by 0.85 when validation accuracy does not improve over an epoch. Contrastingly, the early_stopping_patience parameter stops the training when validation does not improve over 100 epochs. Additionally the ReLU activation function and the MaxPooling layers were used for Pooling and for non-linear transformation of layer output.

Implementation and Training

Code and Environment

The code can be cloned from the repository with the framework, available by the reference [1]. The requirements.txt specifying the modules and the environment used by the framework is available on the following google drive, by the following path [2].

Training Process

The model was trained for 750 epochs with a custom categorical cross-entropy loss function. The loss function calculates the mean loss over each license plate digit. The loss of a single digit is computed as the categorical_crossentropy loss with label smoothing and is given by the following equation.

$$categoricalCrossEntropy = \sum_{i=1}^{i=numClasses} y_{true,i} * log(y_{pred,i})$$

Additionally the model used the Adam optimizer, the accuracy and the top_3_k metric for model evaluation. The accuracy metric is computed as the general accuracy of license plate prediction whereas the top_3_k calculates how often the true license plate character is found in the 3 predictions with the highest probability.

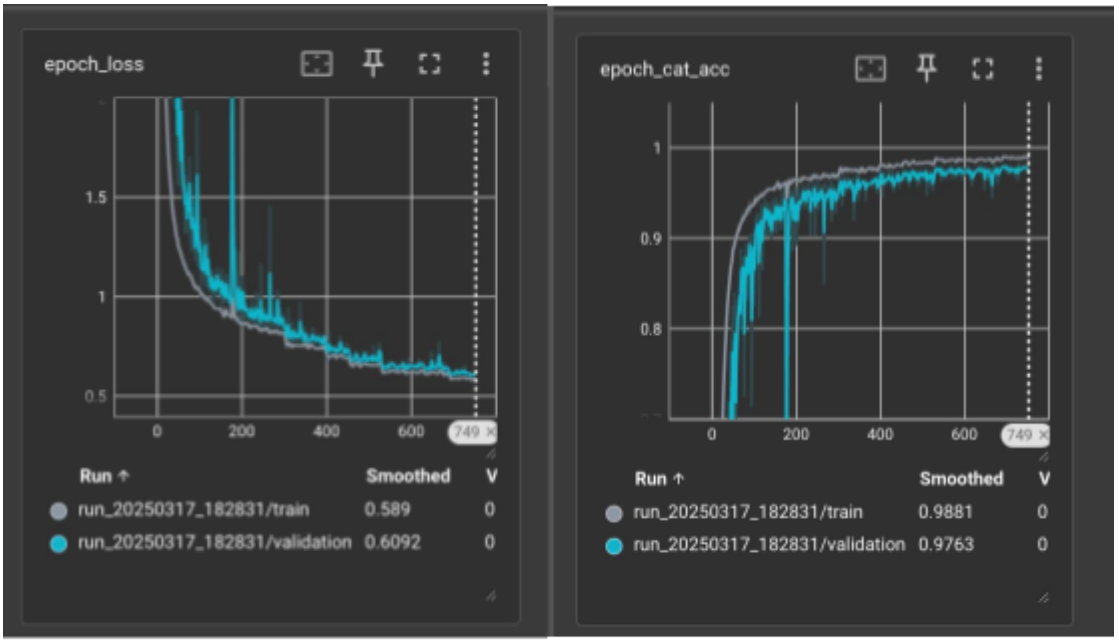

Figure 5. Graphs of loss (left hand side) and of accuracy (right hand side) over epochs

Model Evaluation

Metrics

To evaluate the model we have used the plate_accuracy, plate_precision and top_3_k metrics. The plate_accuracy metric measures how many plates were classified correctly (exact match of the plate number). For the plates that were not classified correctly we have used the plate_precision and the top_3_K metrics where the former measures how many characters within the plate were correctly classified whereas the latter evaluates whether the top3 (probability-wise) most likely characters are classified correctly.

Testing on new data

Accuracy	Plate Precision	Top 3-K
0.920	0.858	0.953

Table 1. Accuracy, Plate Precision and Top 3-K of the trained model

Error interpretation

With respect to plate accuracy the model has managed to achieve extremely high accuracy suggesting that the trained model can be readily used in production for license plate OCR. In terms of plate precision, the metric suggests that in many cases the model predicted a single incorrect character while the top 3 - K score suggests that in 95% of the cases the model still assigned a high probability to the correct character. Since the model learns the general distribution of the data and license plates follow a format (LETTER LETTER DIGIT DIGIT DIGIT LETTER LETTER) most errors were due to model mistaking similar letters (U and V; Q and O)). This error can be mitigated by checking whether a similar plate exists in our database (by replacing Q's with O's).

Additionally we conducted tests on a dataset with plate numbers from a different distribution (Russian plates), the results demonstrated poor performance, suggesting that our model can not generalize to a different character plate distribution. The issue can be resolved by training the OCR model on a dataset under a different character distribution.

Deployment and Demonstration

How can the model be used in production?

The model offers the possibility to implement the automatic opening of gates and barriers when the car arrives. The realisation consists in the possession of the camera located near the gate and the database in which license plate collection is stored. Such systems can be installed at various locations for personal or commercial use.

Development of an API

In order for a driver to have the possibility to add their plate for granted access the API with telegram bot was developed. This was realised with the library “pyTelegramBotAPI“. The bot was made asynchronous allowing multiple users to utilise it at once. The bot adds the license plate entered by the user with the prompt “Plate: …” to a “database.txt” file which is used by the model to verify the belongingness of the identified plate number from the webcam. If the user enters the license plate which is already stored in the database, the bot notifies them about this. All the codes are stored in uppercase. The code for the processing and addition to the database is presented in the figure (6).

```
@bot.message_handler(func=lambda message: True)
async def get_text_messages(message):

    if message.text == 'Add plate number':
        await bot.send_message(message.from_user.id, 'Please enter the plate number in the format: "Plate: ..."') #ответ бота

    elif message.text.split(" ")[0] == 'Plate:':
        number = ""
        for i in range(1, len(message.text.split(" "))):
            number += message.text.split(" ")[i]
        number = number.upper()
        reply = 'The plate number is added into the database!'
        with open("database.txt", "a") as myfile:
            if number in open('database.txt').read():
                reply = 'The plate number is already in the database'
            else:
                number += "\n"
                myfile.write(number)
        await bot.send_message(message.from_user.id, reply)

    else:
        await bot.send_message(message.from_user.id, 'Please follow the prompt "Plate: ..."')
```

Figure 6. Code of the telegram bot for processing and addition of the license plate to the database

Interactive prototype

The interactive prototype consists of two main parts:

1. Addition of the license plate to the database through the telegram bot
2. Usage of VS Code and live webcam to identify in real time whether or not the shown plate is registered in the database

Such a system can be deployed on various servers hosting the bot and the plate detection model parts.

Conclusions and Further Perspectives

Key project results

Among the projects results several key points can be mentioned that allow to grasp the overall work performed:

- The dataset for the project consists of a number of license plates with the data augmentation part that utilises techniques like shifting, scaling, etc.
- The light-weight Fast Plate OCR methodology was used and trained on the utilised dataset providing with good values in accuracy and loss
- The telegram bot was implemented for the possibility of a regular user to access the gate opening system

- The interactive prototype was implemented and tested using the real time plate recognition through the webcam

The achieved results allowed us to fully attain the goals of the project, providing information and experience on how basic CV techniques can be used and implemented to solve day-to-day tasks.

What problems remain unsolved?

Considering the unsolved parts of the initial project formulation, the dataset specificity can be mentioned. Due to the fact that license plates differ from country to country the described version focuses on the Argentinian regions recognition. The implementation of the system's versatility would be the next goal to excel in. In addition, even though the interactive prototype allows testing and verification of the correctness of the model's performance, the simulation and visualisation of the gate opening/closure process would be much easier for the regular user to evaluate.

Ways to improve the model

1. Train on datasets with license plates from different countries
2. Grant access for a car with a license plate for a set amount of time
3. Develop the API allowing to delete unused or irrelevant license plates
4. Use a license plate detection model to extract plates in the wild.
5. Implement an algorithm which will check similar plate combinations
6. Integrate the model into the API to add license plates into the database from photo

References

[1] - https://github.com/ankandrew/fast-plate-ocr
[2] - https://drive.google.com/file/d/1p0sDDkpPJjHkerbEo5BDa0c3PVdW_UaJ/view?usp=sharing
[3] - https://github.com/RomanGorelsky/CV_Project_Plates.git