CAR LICENSE PLATE RECOGNITION BASED ON CNN

XIAOCHONG LIN
Department of Computer Science
ILLINOIS INSTITUTE OF TECHNOLOGY
10 West 35th Street Chicago, IL 60616
linxiaochong111@hotmail.com

Abstract

This system implements a car plate recognition system, which consists of car plate location, character recognition, and convolution neural network training and recognition module.

Experimental results show that compared with other single function methods like OpenCV template image recognition, the CNN method can reach a higher accuracy and lower loss rate.

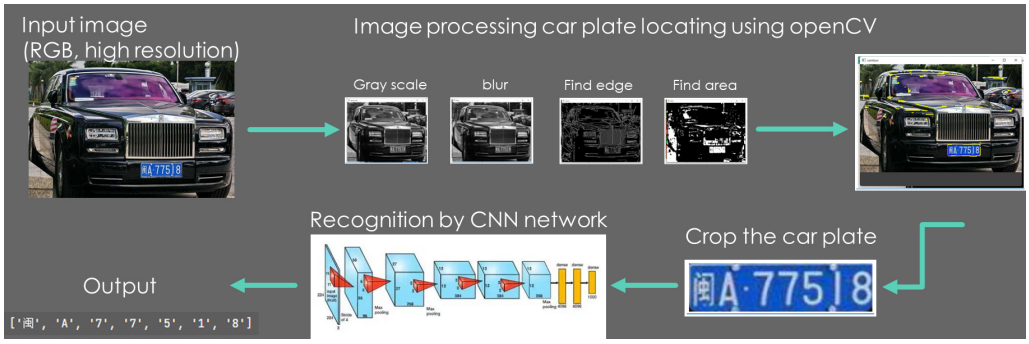
1 Introduction

1.1 Study background

- As a core component, Automatic License Plate Recognition (ALPR) plays an important role in modern Intelligent Transportation System (ITS). In the residential district vehicle access management, the expressway toll, the car license plate recognition has been widely used.
- Due to the complexity in real world, many existing license plate detection and recognition approaches are not robust and efficient enough for practical applications, therefore ALPR still a challenging task both for engineers and researchers.

1.2 Introduction of the system

Towards End-to-End car license plate detection and recognition using CNN



Submitted to CS548 SPRING 2022 project report. Do not distribute.

- This is a car plate recognition core system. It includes the core function of car plate recognition but does not include the application like user interface, package distribution and installation, and hardware utilization.
- This system succinctly takes license plate detection and recognition as two associated parts and is trained end-to-end.
- We need to train the model for car plate and character recognition separately. The training dataset can fetch from CCPD (Chinese City Parking Dataset, ECCV).
- Once the model is trained. We then can start to identify car plates. We just move the image to the specific directory, run the identification program and it will print out the result.

1.3 Installation

- For demonstration

  Move the image into the directory `./images/pictures/`

  run python carlPlateIdentity.py `{image name}`
- For training

  run python charNeuralNet.py 0

  run python plateNeuralNet.py 0
- For testing

  run python charNeuralNet.py 1

  run python plateNeuralNet.py 1

1.4 Environment

- OpenCV
- TensorFlow
- Python 3.8
- Numpy

2 Introduction of CNN

- In deep learning, a convolutional neural network (CNN, or ConvNet) is a class of artificial neural network (ANN), most applied to analyze visual imagery.
- It based on the shared-weight architecture of the convolution kernels or filters that slide along input features and provide translation-equivalent responses known as feature maps.
- CNNs are regularized versions of multilayer perceptions. Multilayer perceptions usually mean fully connected networks, that is, each neuron in one layer is connected to all neurons in the next layer.
- Convolutional networks were inspired by biological processes in that the connectivity pattern between neurons resembles the organization of the animal visual cortex.
- Individual cortical neurons respond to stimuli only in a restricted region of the visual field known as the receptive field. The receptive fields of different neurons partially overlap such that they cover the entire visual field.

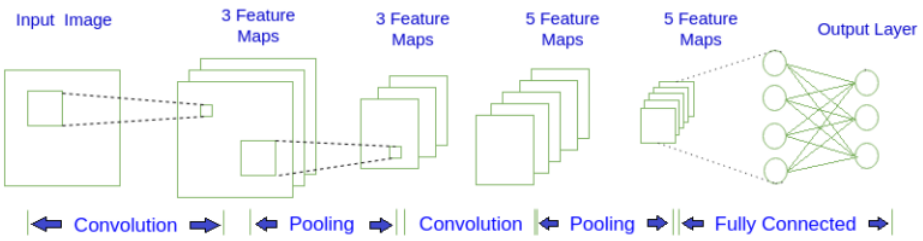
2.1 Architecture OF CNN

- A typical CNN consists of 3 parts: Convolution layer, Pooling layer and Fully connected layer

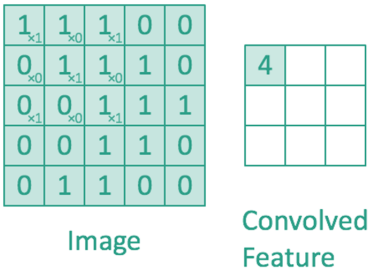
  The convolutional layer is responsible for extracting local features in the image;

  the pooling layer is used to significantly reduce the parameter magnitude;

  the fully connected layer is similar to the traditional neural network portion and is used to output the desired result.

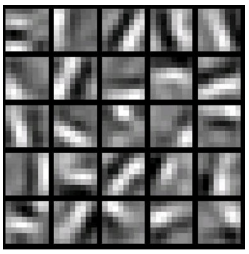


2.1.1 Convolution layer - extraction features



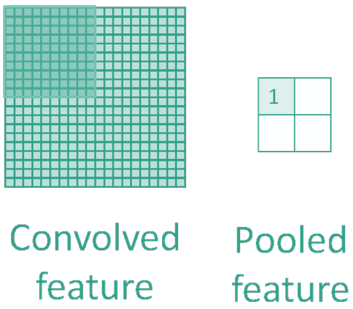
- The operation of the convolutional layer is as shown in the figure, using a convolution kernel to scan the entire picture:
- This process we can understand is that we use a filter (convolution kernel) to filter the small areas of the image to get the eigenvalues of these small areas.

- After passing through a convolutional layer, the image becomes abstracted to a feature map, also called an activation map, with shape: (number of inputs) x (feature map height) x (feature map width) x (feature map channels).



- In a specific application, there are often multiple convolution kernels. It can be considered that each convolution kernel represents an image mode. If an image block is convolved with the convolution kernel, the image block is considered to be Very close to this convolution kernel. The following are examples of 25 different convolution kernels:

2.1.2 Pooling layer - dimensionality reduction



- The pooling layer is simply a down sampling, which can greatly reduce the dimensions of the data.
- The reason for this is that even after the convolution is done, the image is still large (because the convolution kernel is small), so in order to reduce the data dimension, the downsampling is performed.

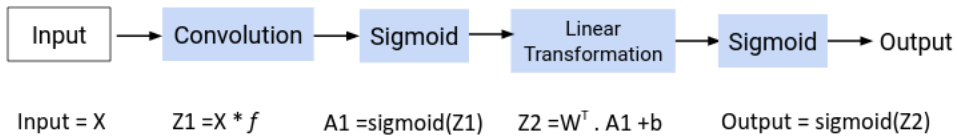
2.1.3 Fully connected layer - Classification

- Fully connected layers connect every neuron in one layer to every neuron in another layer. It is the same as a traditional multilayer perceptron neural network (MLP). The flattened matrix goes through a fully connected layer to classify the images.

- A typical CNN is not just the 3-layer structure mentioned above, but a multi-layer structure, such as the structure of LeNet-5

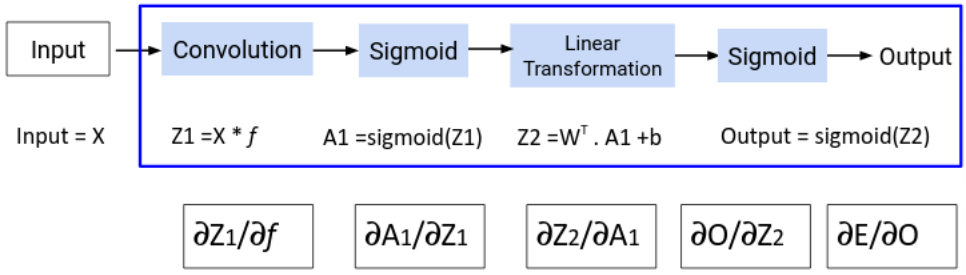
2.2 Principal of CNN

2.2.1 Forward Propagation to get the activation



- Load the input images in a variable (say X)
- Define (randomly initialize) a filter matrix. Images are convolved with the filter. $Z_1 = X \times f$
- Apply the Sigmoid activation function on the result $A = (Z_1)$
- Define (randomly initialize) weight and bias matrix. Apply linear transformation on the values $Z_2 = W^T \times A + b$
- Apply the Sigmoid function on the data. This will be the final output $O = (Z_2)$

2.2.2 Backward Propagation to get the error term



Backward Propagation (Convolution layer)

- from the above figure, we have $\frac{\partial E}{\partial f} = \frac{\partial E}{\partial O} \times \frac{\partial O}{\partial Z_2} \times \frac{\partial Z_2}{\partial A_1} \times \frac{\partial A_1}{\partial Z_1} \times \frac{\partial Z_1}{\partial f}$
- Once we have the value for $\frac{\partial E}{\partial f}$, we will use this value to update the original filter value: $f = f - learning\_rate \times \frac{\partial E}{\partial f}$

3 Code analyze

3.1 Repository address

`https://github.com/XIAOCHONG-LIN/CarPlateRecognizationCNN`

- *carPlateIdentity.py* the main module for identify the car plate
- *charNeuralNet.py* the training module for identify the character of car plate
- *plateNeuralNet.py* the training module for identify the car plate image
- `carIdentityData` the training and testing image dataset directory
- `./carIdentityData/model` - the directory where the model stored
- `./Images` the input directory for identification image

3.2 Data set

- We use CCPD (Chinese City Parking Dataset, ECCV) as the training and testing dataset.
- Dataset includes four directories
  - Character training dataset
  - Character testing dataset
  - Car plate training dataset
  - Car plate testing dataset

3.3 CNN building

- build three convolution and pooling layers
  - convolution kernel size 3x3
- build three fully connection layers

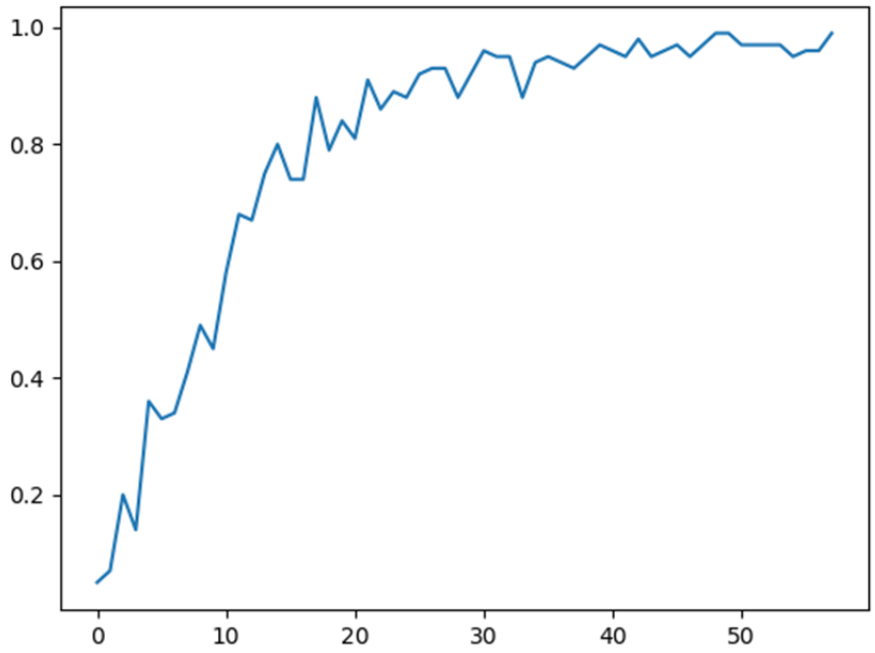
3.4 image processing

- Convert to grayscale
- Gaussian blur
- Canny edge finding
- Convert to hsv image
- Find color area
- Find the color edge
- Binarization
- Find contours
- Flood fill the area
- Get the mask area
- Crop the car license plate for recognition

4 Experiment result

4.1 Training the character

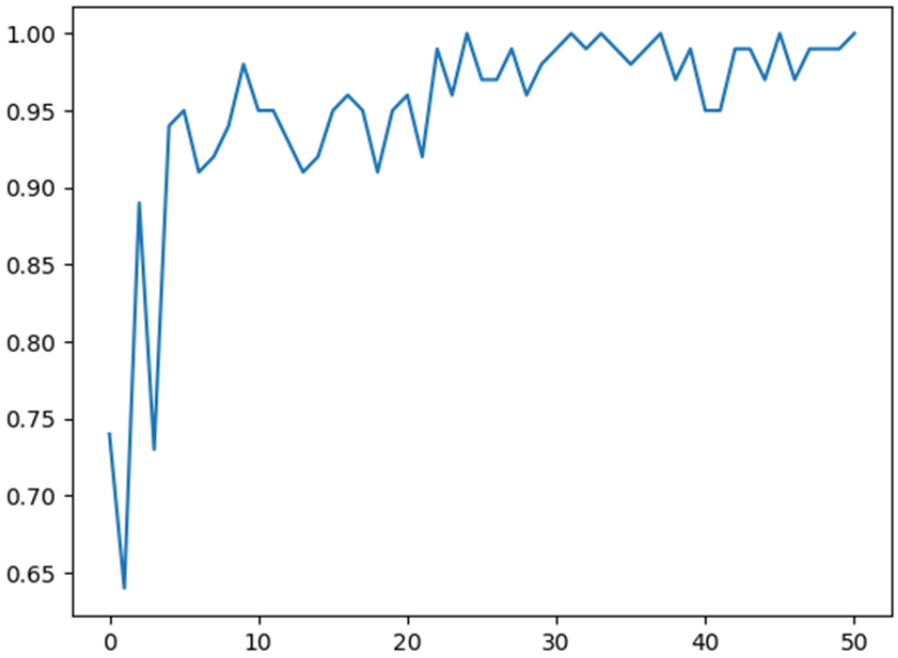
- in around 500 steps, the accuracy converges to higher than 99%



- testing result: *total: 549 images, incorrect: 26, accuracy: 0.9526411657559198*

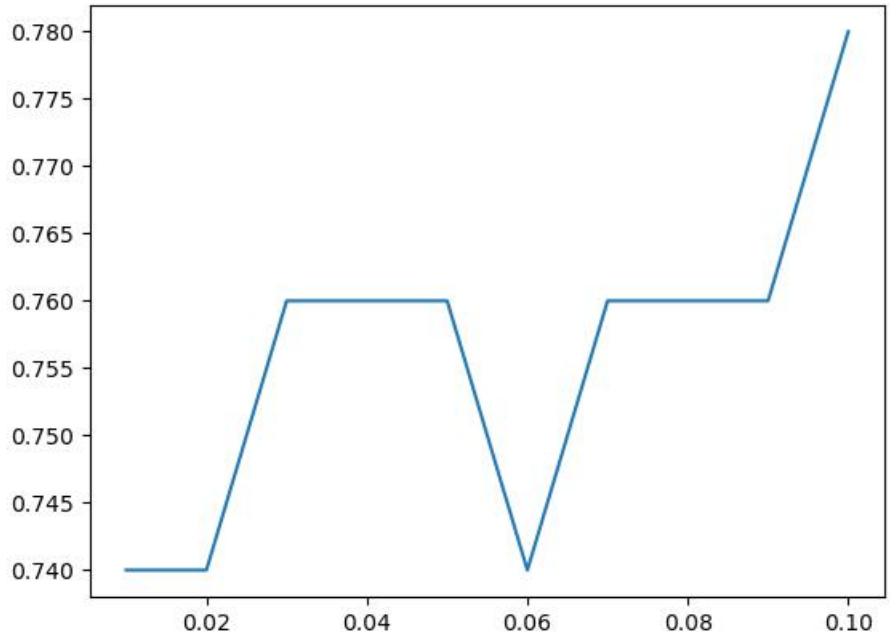
4.2 Training the car license plate

- in around 250 steps, the accuracy converges to higher than 99%



- testing result: *total: 200 images, incorrect: 3, accuracy: 0.985*

4.3 Overall testing result End-To-End



- *total: 50 images, the best accuracy: 0.78*

5 Summary and prospect

- This system is still trivial comparing the current car license plate recognition system. It just runs a basic functional module for recognize the car plate. But it covers the main idea of the recognition system.
- With more complicate environment such as the image is distorted by light, environment color, angle of car license plate, etc. this system will reduce the accuracy.
- Although at the training and testing stage, CNN recognition can reach a very high accuracy, when combine it in the whole system, the low accuracy of car license plate location recognition by openCV reduce the overall accuracy greatly.
- There is a technology named lightweight CCN DSNET exist which could be the next study direction to improve the accuracy in complicate environment.

6 Reference

[1] `https://blog.csdn.net/jmh1996/article/details/88951797`

[2] `https://en.wikipedia.org/wiki/Convolutional_neural_network`

[3] `https://easyai.tech/en/ai-definition/cnn/`

[4] `http://ufldl.stanford.edu/tutorial/supervised/FeatureExtractionUsingConvolution/`

[5] `https://www.analyticsvidhya.com/blog/2020/02/mathematics-behind-convolutional-neural-network/`

[6] SLPNet: Towards End-to-End Car License Plate Detection and Recognition Using Lightweight CNN `https://link.springer.com/chapter/10.1007/978-3-030-60639-8_25`

A Appendix

A.1 Image preprocessing

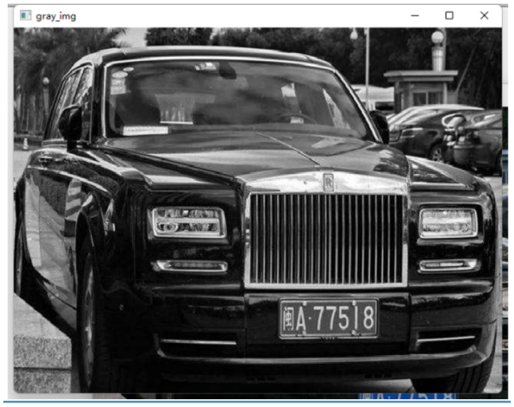


Figure 1: Convert to grayscale

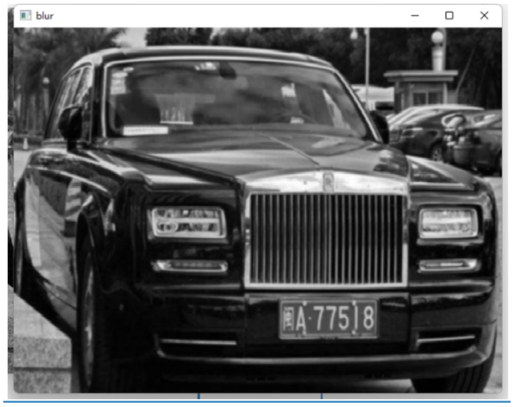


Figure 2: Gaussian blur

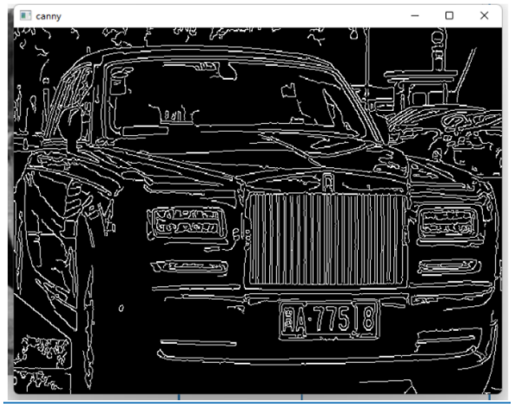


Figure 3: Canny edge finding

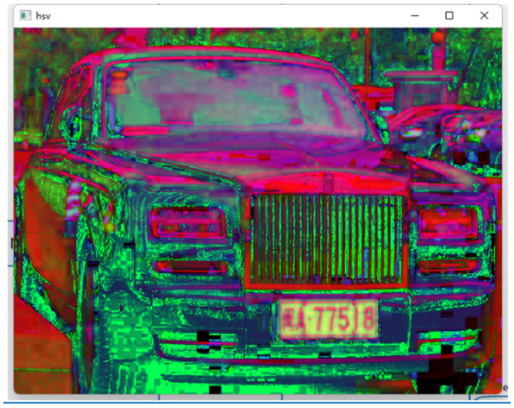


Figure 4: Convert to hsv image

Figure 5: Find color area

Figure 6: Find the color edge

Figure 7: Binarization

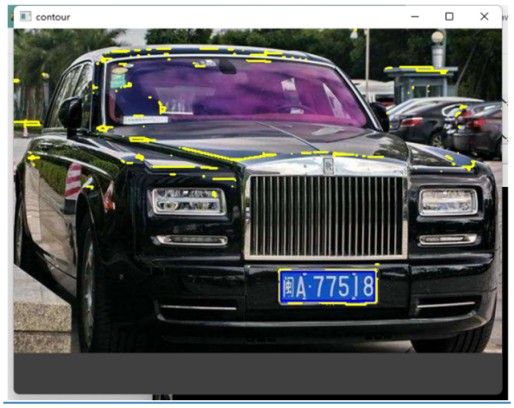


Figure 8: Find contours

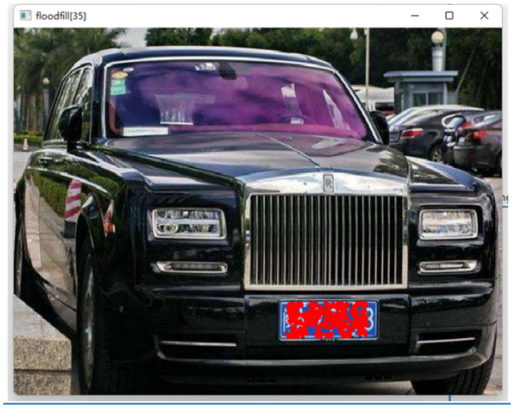


Figure 9: Flood fill the area

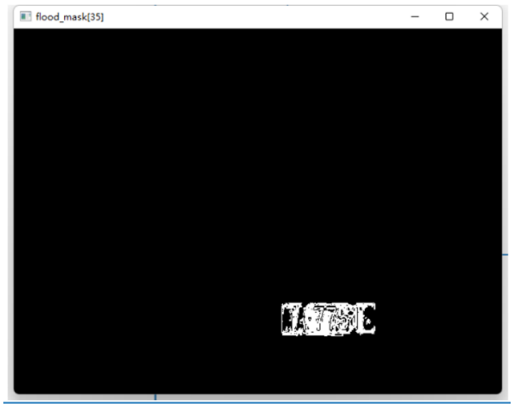


Figure 10: Get the mask area

Figure 11: Crop the car license plate