Zaven and Sonia Akian
College of Science and Engineering (CSE)
BS in Computer Science

AUTOMATIC VEHICLE NUMBER PLATE DETECTION AND RECOGNITION

Authors:
Grigoryan Maro
Matinyan Arpen

Supervisor:
Yeghiazaryan Varduhi, Ph.D.

AUA American University of Armenia
MORE THAN AN EDUCATION- A COMMITMENT

Spring 2020

Acknowledgement

Foremost, we would like to express our sincere gratitude to our supervisor Yeghiazaryan Varduhi for continuous support during the project, for her patience and motivation. We want to thank her for providing guidance, support and feedback throughout this project.

Contents

Abstract

Automatic license plate detection and recognition is a big step towards intelligent transportation systems. The introduction of advanced data collection hardware, like the traffic monitoring cameras, assumes the development of corresponding software, which automates the processing and analysis of the generated data. This project explores the theoretical basis for the development of such software. The task of automatic license plate recognition is composed of two main stages: symbol extraction from the plate and symbol recognition. Our approach is an implementation of a novel hybrid system using the cutting-edge tools of image processing and machine learning. When applied on an image dataset, our method achieves accuracy above 97% for the extraction phase and above 93% for the recognition phase, thus showcasing promising results.

1 Introduction

In this research project, we propose the implementation of a new system for automated license plate detection and recognition. This has several useful applications, including effective detection of traffic law violations, parking management, authentication, and authorization control based on plate information, etc. The motivation behind the research idea is the expected increase in productivity of resource-consuming data analysis through the integration of automation technologies. The usage of automated license plate recognition systems by law enforcement agencies has recently gained momentum, thus raising the demand for such technologies.
Automatic number-plate recognition (ANPR) is a technology that uses optical character recognition on images to read vehicle registration plates without human intervention. The main algorithmic components of ANPR systems are the following:

1. Plate localization—Finding and isolating the plate from the image
2. Plate orientation and sizing—Rotation, skew correction, adjustment to the required size
3. Plate normalization—Brightness and contrast correction
4. Character segmentation
5. Character recognition

We address the second, fourth, and fifth components of the system in this project. After a thorough literature review and examination of already available similar schemes, we introduce a new hybrid system consisting of two major phases: plate character extraction and character recognition. The solution for the first stage is solely based on image processing tools and algorithms, whereas the second stage is implemented using a small Convolutional Neural Network and other techniques in Deep Learning.
Technical implementation details and technologies used are described in the **Tools and Technologies** section of the report. The solution scheme is constructed using observations of the available data. We define the necessary preprocessing steps of the scheme taking into account all possible limitations of the base dataset. A detailed description of the datasets is presented in the **Data** section. Figure 1 is a visualization of the proposed scheme, illustrating the main processing stages and the flow of the algorithm, which is thoroughly described and explained in the sections **Character Extraction** and **Character Recognition**. For each of the phases, we

conduct an assessment of the results with several evaluation metrics discussed in the **Result and Discussion** section. In the scope of **further research**, we consider implementing the remaining components of the general ANPR system and generalizing the results on a broader range of data sources, thus expanding the usability of the scheme.

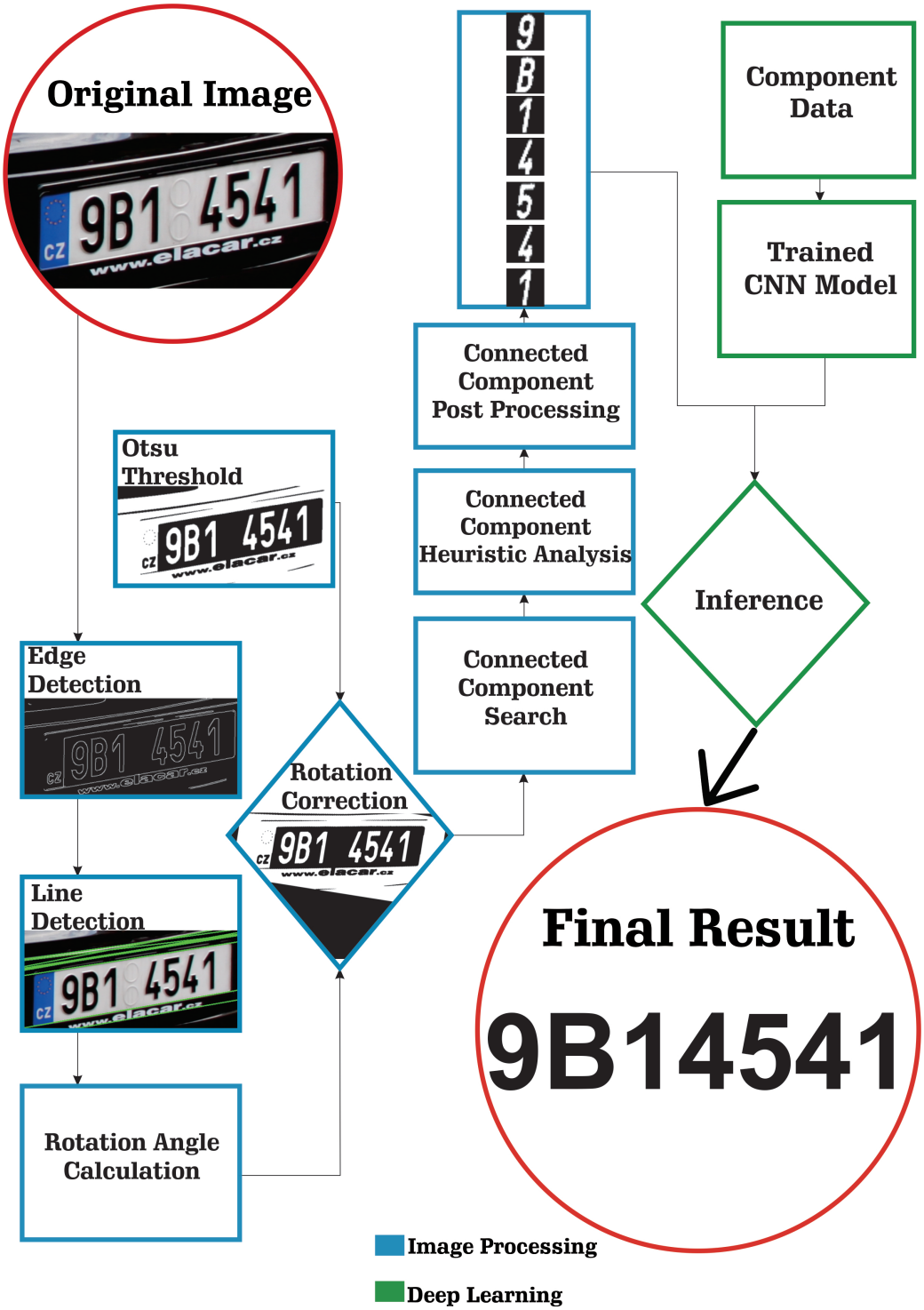


Figure 1: Schematic visualization of the algorithmic components of the system; blue Image Processing steps correspond to character extraction; and green Deep Learning steps correspond to character recognition

2 Tools and Technologies

For the technical implementation of the project, we used **Python 3** and **Anaconda** distribution for an optimal setup and package management. **NumPy** and **Matplotlib** were used for array processing and visualization purposes accordingly. We used **OpenCV-Python** to implement the main components of the image processing-related tasks in the first phase of the project (character extraction). For the second part of the project (character recognition), we used **Google Colab**, which provides a free space for code editing, CPU, and GPU for model training, and requires no manual setup. We used **Tensorflow** and **Keras** to construct and train the neural network. Finally, we used **Jupyter Notebook** as it provides an interactive development environment with a high rate of flexibility for testing, visualization, and evaluation.

3 Data

3.1 Character Extraction Data

Dataset choice plays a decisive role in the definition of the scope of the overall project, clarifies the required plate preprocessing stages, possible limitations, and expected results. Dataset sources may result in a specific bias based on several possible factors (country, region, vehicle type, laws, etc.), introducing respective processing requirements and limitations. In our project, we consider only cropped plates, omitting the initial stage of plate detection on vehicle images. We use the dataset introduced by Spanhel et al.[20], which is available for academic research. In total, it contains 8 folders with overall 642 images of **PNG** format. 4 of the folders contain images with low quality, and the other 4 contain the same set of images with comparably higher quality. We remove images with unnoticeable differences, low resolution (e.g., I00028.png—(44×127)), and other obvious defects. After such visual inspection, the final dataset contains 436 images. The background of the plates in the dataset is white, and the color of the characters is mainly black (green in some specific cases). The shape of the plates is rectangular. The dataset is composed of images with different sizes and brightness levels. We use the dataset to test the character extraction algorithm at the first stage and for the final recognition test at the second stage of the project. The majority of the plates are skewed, which introduces the need for a rotation correction step. Figure 2 illustrates 6 different samples from the extraction dataset.

Figure 2: 6 random samples from the extraction dataset with varying sizes, rotation angles, brightness and contrast levels, character colors, etc.

3.2 Character Recognition Data

The recognition dataset contains binary (0,255) images of uppercase letters and digits, each with dimensions 28 × 28. It contains 36 classes (A–Z, 0–9) with 1016 samples for each class (overall 36576 character images). We found the dataset in the GitHub repository `https://github.com/GuiltyNeuron/ANPR/tree/master/Licence_plate_recognition/USA_plates` . We use the dataset for training and validation with 80% and 20% proportions correspondingly. Figure 3 below illustrates 36 samples, one from each of the 36 classes of our recognition dataset.

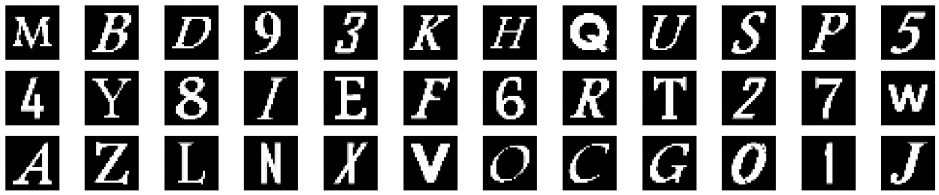


Figure 3: Samples of dimensions 28 × 28 from all 36 possible classes of the recognition dataset

As we have already mentioned in the Character Extraction Data subsection, we use the extracted character dataset from the first stage as testing set for the recognition phase. All 2979 valid characters from the previous character extraction stage are manually annotated, after which they are resized to 28 × 28 to comply with the trained model.

4 Literature Review

According to Du et al.[8] **ANPR** is a widely used term to describe a technology that can detect and "read" vehicle registration plate numbers. The motivations for such technologies are most commonly road-rule enforcement, vehicle location identification, and others. The term was first used in 1976 at the Police Scientific Development Branch in Britain. The first stolen car detection with the help of ANPR technology was accomplished in 1981 [1]. However, it did not become widely used until the latest advances in technology. The cheaper and easier way of producing software resulted in a new wave of the ANPR evolution.

Another widely used term for ANPR is **License Plate Recognition (LPR)**.

The first ANPR solutions were based on image processing and **optical character recognition (OCR)**. A series of image manipulation techniques were used to detect the plate region, normalize and enhance the plate image and then OCR was utilized to extract the alphanumeric values. These solutions were identified to face several difficulties, the solution of which was essential for achieving the desired results.

According to Permaloff and Grafton [16] OCR is the mechanical conversion of images of typed, handwritten or printed text into machine-encoded text. It is one of the first addressed computer vision tasks, that dates back to 1970s. There are various challenges associated with OCR-based solutions.

Challenges associated with OCR:

- Poor file resolution: the car is too far, low-quality camera use,
- Motion blur,
- Poor lighting conditions,
- Extra object (sometimes dirt) on the plate,
- Use of different fonts,
- Text localization on the image.

According to Shperber [19] the basic OCR algorithm produces a ranked list of candidate characters. Matrix matching is used to compare a character image to a stored sample on a pixel-by-pixel basis. However this approach has a main drawback: despite high probabilistic results on a specific font type, it fails on a new one. Some of these problems can be solved within the software. On the contrary, it is primarily left to the hardware side of the system to work out solutions to other difficulties. Proper operations need constant hardware maintenance and improvements.

Contemporary text recognition tasks include deep learning. Ciregan et al. [5] suggest deep neural network architectures as an alternative to classical computer vision for text recognition.

More recent solutions suggest convolutional neural network-based approaches for license plate recognition. Draghici [7] introduces an artificial vision system, with an alternative to the OCR, based on a constraint decomposition training architecture. The system has shown the following performance (on average) on real-world data: successful character recognition of about 98% and successful recognition of complete registration plates about of 80%.

5 Character Extraction

5.1 Plate Binarization

Image binarization is one of the most important steps in segmentation tasks. Because of the structure simplicity, it is easy to work with binary images. Conversion of the image from grayscale to a black-white version is performed using various thresholding algorithms. Thresholding generally involves two steps: determination of threshold value and assigning all pixels to foreground or background classes accordingly. If the intensity of the pixel is higher than the determined threshold, then it belongs to the foreground, otherwise, to the background [22]. For each pixel with (x, y) coordinates in the image:

$$Binary(x,y) = \begin{cases} 255 & \text{if } Image(x,y) > \text{threshold value} \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

The right choice of a threshold value plays a crucial role in the performance of the overall algorithm. Manually chosen threshold values work for datasets with similar images. As we were working with images from various possible sources, we decided to use a more generalized technique for the threshold value determination.

5.1.1 Otsu Thresholding

In an ideal case, for the grayscale image, we expect to observe a bimodal histogram with two peaks representing both classes, as illustrated in Figure 4. Although one cannot observe ideal peaksin most images, Otsu's algorithm suggests a threshold value that lies between two comparably visible peaks such that variances of both classes are minimal. After the histogram is normalized, the probabilities of both classes are calculated in the following way:

L	number of the histogram bins: blocks used to combine values of separate frequencies
n_i	number of pixels in the i^{th} bin
N	total number of pixels (N = n_1 + n_2 + ... + n_L)
t	optimal threshold value
w_0	probability of the background class
w_1	probability of the foreground class

$$p_i = \frac{n_i}{N}, \;\; p_i \geq 0 \;\; \sum_{i=1}^{L} p_i = 1 \tag{2}$$

$$\begin{aligned} w_o = w(t) &= \sum_{i=1}^{t} p_i \\ w_1 = 1 - w(t) &= \sum_{i=t+1}^{L} p_i \\ w_0 + w_1 &= 1 \end{aligned} \tag{3}$$

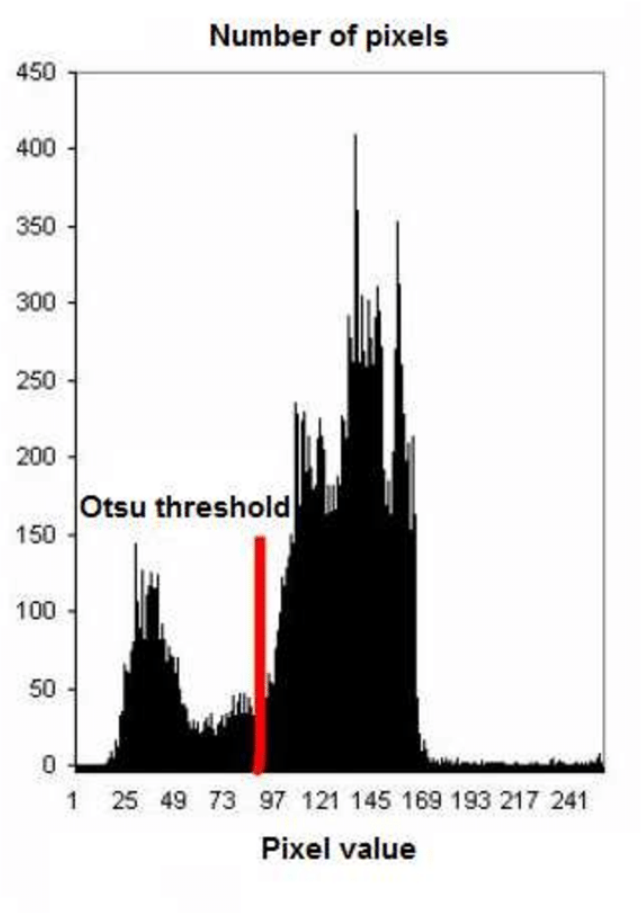


Figure 4: Visualization of comparably visible peaks in a bimodal (foreground and background) histogram and the optimal threshold value found with Otsu algorithm

(a) Original image from the cropped dataset

(b) Inverted image after Otsu Thresholding

Figure 5: Original image and the image after Otsu Thresholding

The background and foreground class mean values are calculated using w_0 and w_1:

μ_0 background class mean

μ_1 foreground class mean

μ_L total mean of the original picture

$$
\begin{aligned}
\mu_0 &= \sum_{i=1}^{t} \frac{i * p_i}{w_0} \\
\mu_1 &= \sum_{i=t+1}^{L} \frac{i * p_i}{w_1} \\
\mu_L &= \sum_{i=1}^{L} i * p_i \\
w_0 * \mu_0 &+ w_1 * \mu_1 = \mu_L
\end{aligned}
\tag{4}
$$

We proceed to class variance calculation and introduce σ_w^2 and σ_L^2:

σ_0^2 background class variance

σ_1^2 foreground class variance

$$
\begin{aligned}
\sigma_0^2 &= \sum_{i=1}^{t} \frac{(i-\mu_0)^2 * p_i}{w_0} \\
\sigma_1^2 &= \sum_{i=t+1}^{L} \frac{(i-\mu_1)^2 * p_i}{w_1}
\end{aligned}
\tag{5}
$$

$$
\begin{aligned}
\sigma_L^2 &= \sum_{i=1}^{L} (i-\mu_L)^2 * p_i \\
\sigma_w^2 &= w_0 * \sigma_0^2 + w_1 * \sigma_1^2
\end{aligned}
\tag{6}
$$

Then our problem is reduced to an optimization problem to search for a threshold t that maximizes the following objective function:

$$
J = \frac{\sigma_L^2}{\sigma_w^2} \tag{7}
$$

The results of Otsu thresholding are illustrated in Figure 5. For more details see [15].

5.2 Plate Rotation Correction

Deskewing is an essential stage of plate preprossing. Proper alignment of the plate assumes correction of two different factors: shear and rotation. For our purposes, rotation correction will significantly improve the processing state. We compute the rotation angle using Hough Line Transform [13].

5.2.1 Edge Detection

We do edge detection with the Canny algorithm as an initial stage of deskewing-related preprocessing. Canny is an edge detection algorithm composed of 4 main stages: noise reduction, image gradient intensity search, non-maximum suppression, and hysteresis thresholding. For details, please see [3].

- **Noise Reduction**
  Gaussian Filter is used as a noise reduction and smoothing operator. One dimensional Gaussian distribution function ($\mu = 0$) has the following form:

$$
G(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2}{2\sigma^2}} \tag{8}
$$

Accordingly, the two dimensional version of the function is

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{9}$$

Figure 6 displays the Gaussian kernel (in this case 5×5), which is the discrete approximation of the Gaussian function (9) when $\sigma = 1$.

$$\frac{1}{273}\begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix}$$

Figure 6: 5×5 Gaussian kernel

Once a suitable kernel has been calculated, the Gaussian smoothing is performed using standard convolution. For details please see [9].

- **Image gradient intensity search**
  The smoothed image is then filtered with both vertical and horizontal Sobel kernels displayed in Figure 7. The Sobel operator is a discrete differential operator. It utilizes two 3×3 kernels: one estimates the gradient in the x-direction, while the other one estimates the gradient in the y-direction.

$$\begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} \qquad \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Gx Gy

Figure 7: Vertical and horizontal 3×3 Sobel kernels

At each given point, magnitude and direction of the gradient can be approximated with

$$G = \sqrt{G_x^2 + G_y^2} \tag{10}$$

$$\theta = \arctan \frac{G_y}{G_x} \tag{11}$$

- **Non-maximum suppression**
  Non-maximum suppression discards all the gradient values, except the local maxima, which

indicate locations with the sharpest change of intensity value. It compares the edge magnitude of the current pixel with the edge magnitudes of the pixels in the positive and negative gradient directions. If the edge magnitude of the current pixel is the largest compared to the other pixels in the mask in the same direction, the value will be preserved, otherwise, suppressed.

- **Hysteresis thresholding**
  Two threshold values, minVal, and maxVal are introduced during this stage. Any edges with intensity gradient magnitude more than maxVal are kept and classified as "sure-edges." Similarly, those edges below minVal are discarded and classified as "sure-non-edges." Those between these two thresholds are classified as edges or non-edges based on their connectivity. If they are connected to "sure-edges" pixels, they are considered to be part of edges. Otherwise, they are also discarded.

(a) Original image from the cropped dataset

(b) Detected edges on the original image

Figure 8: Original image and detected edges

After successfully applying the Canny algorithm on the original image (Figure 8a) we observe quite compelling edges illustrated in Figure 8

5.2.2 Hough Space and Line Detection

Having strong edges on the image, we proceed to the next stage and try to find lines using the above-stated Hough Line Transform. To understand the algorithm, one needs a basic understanding of line representation in the polar coordinate system.
A point in the polar coordinate system can be described in the following way:

$$Point = (\rho, \theta) \tag{12}$$

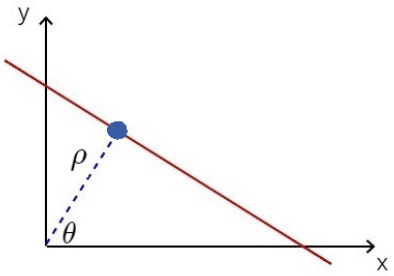


Figure 9: Visualization of (ρ, θ) parameters of the point in the Cartesian coordinate system (slightly modified version of figure in [13])

(ρ, θ) parameters are displayed in Figure 9. The parameter ρ is the distance of the point from the origin, and θ is the angle between x-axis and the line passing through the point and the origin.

We can observe that the combination of (ρ, θ) parameters also describes the line passing through the point and perpendicular to the distance line.

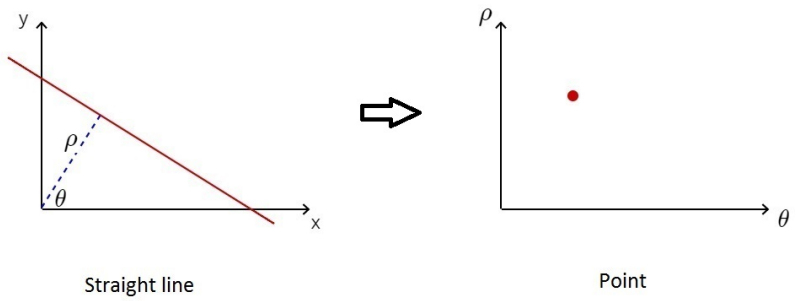


Figure 10: Mapping of a line in the Cartesian system to a point in the polar system [13]

The next step is the mapping from the image space to the so-called "Hough space". We simply take (ρ, θ) parameters of the above-mentioned line and use them as the coordinates for a point representation in the new space. Figure 10 illustrates the mapping of a line from the Cartesian coordinate system to a point in the polar coordinate system, using the pair of (ρ, θ) parameters. There are infinitely many lines passing through a single point in the Cartesian coordinate system. If we try to draw the corresponding points in the "Hough space", we will observe what is displayed in Figure 11:

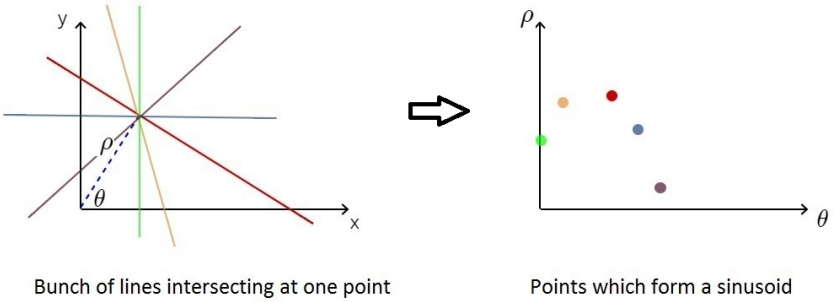


Figure 11: Mapping of several lines passing through a single point to corresponding points in the polar system [13]

It turns out that these points in (ρ, θ) space form a sinusoid. Drawing an infinite number of additional lines intersecting at this one point in the Cartesian coordinate system would result in a continuous sinusoid in the Hough space. One can say that a point in the Cartesian coordinate system results in a sinusoid in the Hough space, as illustrated in Figure 12.

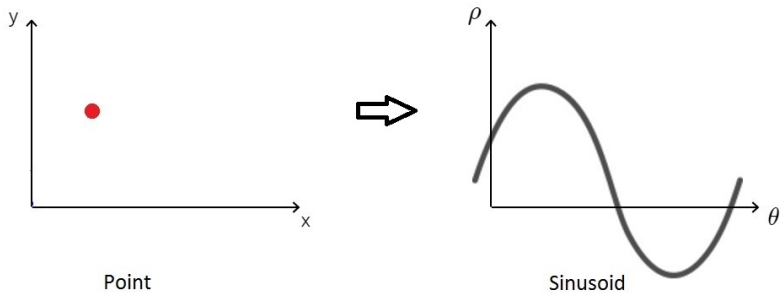


Figure 12: Mapping of the intersection point of infinitely many lines in the Cartesian space to the corresponding sinusoid in the Hough space [13]

Finally, if we draw points which form a line in the image space, we will obtain a bunch of sinusoids in the "Hough space". We can see in Figure 13 that the sinusoids intersect at exactly one point as expected.

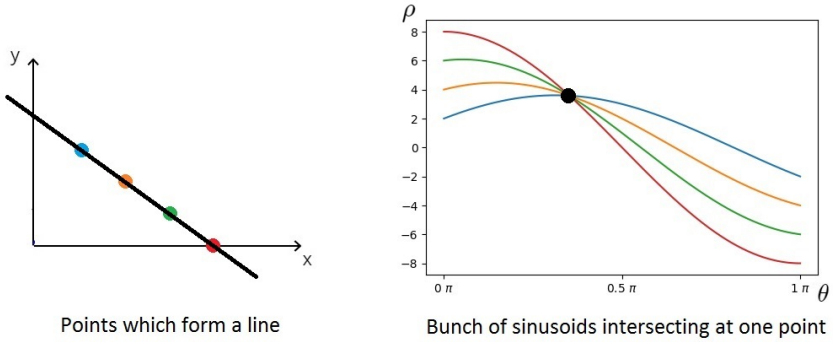
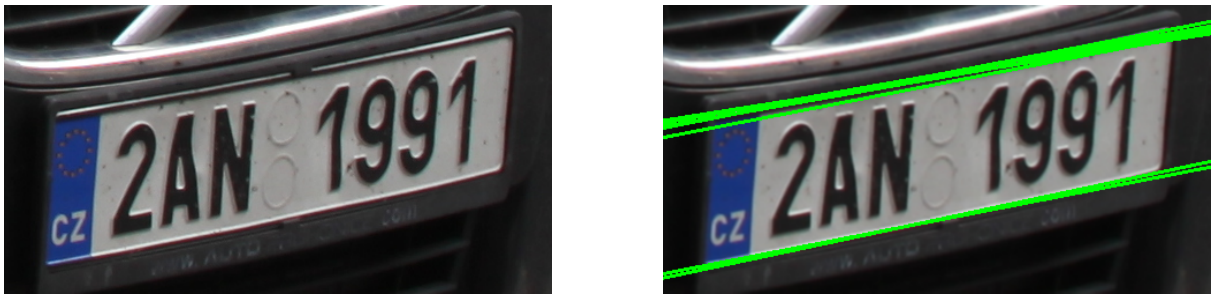


Figure 13: Mapping of several points on the same line to the corresponding sinusoids [13]

Using the above-described property, we find the lines on the original images. Figure 14 illustrates an example of the line detection stage implementation results.



(a) Original image from the cropped dataset (b) Original image with detected lines

Figure 14: Original image and detected major horizontal lines

5.2.3 Rotation Angle Calculation

Line detection is the most important part of skew angle evaluation. After having the polar representation of the line, it is easy to get the points on the line in the Cartesian coordinate system using formulas (13) and (14).

$$x = \rho * \cos\theta \tag{13}$$

$$y = \rho * \sin\theta \tag{14}$$

We identify two different points on the line: (x_1, y_1) and (x_2, y_2) in the following way

$$\begin{aligned} x_1 &= x + 1000 * (-\sin\theta) \\ y_1 &= y + 1000 * (\cos\theta) \\ x_2 &= x - 1000 * (-\sin\theta) \\ y_2 &= y - 1000 * (\cos\theta) \end{aligned} \tag{15}$$

Afterwards, we compute the angle between the line passing these two points and the horizontal x-axis in the following way:

$$\tan\theta = \frac{y_2 - y_1}{x_2 - x_1} \tag{16}$$

$$\theta = \arctan\frac{y_2 - y_1}{x_2 - x_1} \tag{17}$$

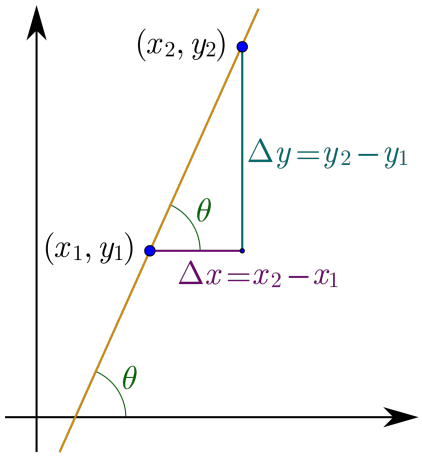


Figure 15: Geometric explanation of rotation angle calculation

Figure 15 precisely demonstrates the idea behind the angle calculation formulas (16) and (17). To get a more precise result, we average the angle over all the lines found in the image. We return θ * 180 / π (convert from radians to degrees) also taking into account the type of the angle (acute, obtuse, etc.) and making respective modifications to the return value if needed.

5.2.4 Affine Transform with Rotation Matrix

To obtain the necessary rotation matrix to perform the affine transform, we take the simplest rotation matrix where θ is the rotation angle in degrees:

$$R = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \tag{18}$$

It is slightly modified in the algorithm to rotate the matrix around the given center coordinate:

$$R = \begin{bmatrix} \cos\theta & \sin\theta & (1-\cos\theta) * center.x - \sin\theta * center.y \\ -\sin\theta & \cos\theta & \sin\theta * center.x - (1-\cos\theta) * center.y \end{bmatrix} \tag{19}$$

It can be easily checked that applying the matrix R on ($center.x$, $center.y$) will not affect it. The center coordinates are obtained using the minimum-area bounding rectangle around the non-zero pixels of the image (center expectedly being the intersection point of the diagonals of

the rectangle). After the affine transform by the rotation matrix obtained, the image is deskewed and ready for the next processing stage.

(a) Original image from the cropped dataset

(b) Original image with corrected rotation

Figure 16: Original image and the image after rotation

We observe the result illustrated in Figure 16b by applying the affine transform on the original image (Figure 16a).

5.3 Connected Component Search

5.3.1 Connected Component Labeling

After rotation angle identification, skew correction and binarization operations we need to identify the characters on the plate image. For that purpose, we find the connected components on the image and choose the ones that represent the plate characters. To find the connected components we use a Connected Component Labeling algorithm.

For identification of components on the image we differentiate between foreground($\mathcal{F}$) and background($\mathcal{B}$) pixels such that, $\mathcal{F} \cup \mathcal{B} = \mathcal{L}$ and $\mathcal{F} \cap \mathcal{B} = \emptyset$, where $\mathcal{L}$ is a 2-D rectangular lattice of an image I.

As the input is a binary image, all the black pixels will be considered as background and white pixels as foreground. Among the latter we need to find the connected components. At first, let's define the concepts of neighborhood and connectivity.

For an image defined over a 2-D rectangular lattice $\mathcal{L}$, $I(p)$ is the value at pixel $p \in \mathcal{L}$, with $p = (p_x, p_y)$. The four-neighborhood and the eight-neighborhood of a pixel p can be, respectively, defined as

$$\mathcal{N}_4(p) = \{q \in \mathcal{L} \mid |p_x - q_x| + |p_y - q_y| \leq 1\} \tag{20}$$

$$\mathcal{N}_8(p) = \{q \in \mathcal{L} \mid \max(|p_x - q_x|, |p_y - q_y|) \leq 1\} \tag{21}$$

The four- and eight-neighborhoods of a pixel are depicted in Figure 17.

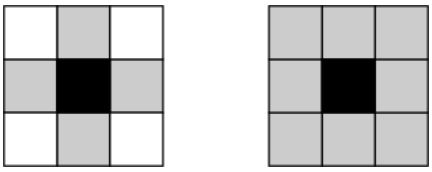


Figure 17: $\mathcal{N}_4(p)$ four-neighborhood and $\mathcal{N}_8(p)$ eight-neighborhood marked by gray pixels.

Eight-connectivity is a more natural way of human eye recognition of connectivity. So, for our problem we use the eight-neighborhood to define a connected component. The set $\mathcal{S}$ of $\mathcal{L}$ is defined to be a connected component in $\mathcal{L}$, if $\forall p, q \in \mathcal{S}$ the following statement (22) holds.

$$p \diamond q \Leftrightarrow \exists \{s_i \in \mathcal{S} | s_1 = p, s_{n+1} = q, s_{i+1} \in \mathcal{N}(s_i), i = 1, \ldots, n\} \tag{22}$$

The aim of the labeling algorithm is to assign a unique label to the connected components in the image. The label is a numeric value, that is assigned to all the pixels of the sets described in (22). The label 0 is reserved for the background pixels. The foreground pixels are assigned natural numbers starting from 1. Depending on the search order the regions may receive different labels. Also, different regions of the same component may end up being assigned different numeric labels. Thus, these labels need to be considered equivalent. Formally, given $p, q \in \mathcal{F}$ $p \diamond q \Leftrightarrow L(p) \equiv L(q)$. We can define the equivalence class of a label $L(p)$ as

$$[L(p)] = \{\lambda \in \mathbb{N} \mid \lambda \equiv L(p)\} \tag{23}$$

In general labeling algorithms perform the following 3 steps:

1. preliminary image scan (initial label assignment and label equivalence collection),
2. equivalence resolution,
3. final label assignment.

During the first step, $L(x) \equiv 0, \forall x \in \mathcal{B}$. For each pixel $x \in \mathcal{F}$, $L(x)$ is evaluated by only considering the labels of its already processed neighbors. When using eight-connectivity, these pixels belong to the scanning mask $\mathcal{M}(x) \subset \mathcal{N}_8(x)$ shown in Fig. 17 . Given x the pixel with coordinates (i, j) in the lattice identified as $x = l_{i,j}$, we can define

$$\mathcal{M}(x) = \{p = l_{i-1,j-1}, q = l_{i,j-1}, r = l_{i+1,j-1}, s = l_{i-1,j}\} \tag{24}$$

If conflicting values are yielded in $\mathcal{M}(x)$, then the pixel is assigned the smallest one, and the label equivalence is reserved. On the second stage, the initial labels are partitioned into disjoint sets. The equivalences found need to be resolved and the corresponding sets merged. For this reason disjoint-set data structures are used. A disjoint-set data structure maintains a collection $\mathcal{S} = \{S_1, S_2, \ldots, S_k\}$ of disjoint dynamic sets. Each set has a representative, which is a member of that set. Disjoint-set data structures support the following operations.

- UNION (S_x, S_y): unites the sets that contain x and y, into a new set that is the union of the two. Since we require the sets in the collection to be disjoint, conceptually we destroy sets S_x and S_y removing them from the collection $\mathcal{S}$ and add a new merged set. In practice, we often absorb the elements of one of the sets into the other set.
- FIND (x) returns the name of the representative of the (unique) set containing x.

More about disjoint sets can be found in [6].

Once the equivalences have been eventually solved, in the final step a second pass over the image is performed in order to assign to each foreground pixel the representative label of its equivalence class. For more details please see [12].

5.3.2 Connected Component Heuristic Analysis and Post-Processing

The result of connected component detection is the set of all the components on the image. Among those we decide which are valid plate characters. The proportion of the component surface area relative to the whole image surface area is calculated. Connected components that are too large or too small are removed. The surface area thresh-holding has chosen to be in the range of 0.1-9% of the whole image area.

We also observed that the connected components that correspond to characters, follow a certain shape pattern, that is the component height is always greater than the width. For an $n \times m$

component $\mathcal{A}$, where n is the height of the minimal bounding box of $\mathcal{A}$ and m is the width of the minimal bounding box of $\mathcal{A}$, the following holds:

$$\frac{n}{m} \geq 1 \tag{25}$$

The components which do not satisfy these conditions are removed from the set of all the identified components. After the components are recognized, we need to detect them and separate from the image. We detect the coordinates of the leftmost, rightmost, upmost and downmost pixels of the image and create the smallest bounding box. After that, the character component is placed on a black background box of a square shape and the size is reduced to a uniform shape to be 28 $\times$ 28. The result is illustrated in Figure 18.

Figure 18: An illustration of character extraction using Connected Component Labeling

6 Character Recognition

6.1 Model Architecture

6.1.1 Convolutional Neural Networks

Artificial neural networks are a biologically-inspired programming paradigm which enables a computer to learn from observational data. Neural networks in deep learning offer solutions to a wide range of challenges including speech recognition, natural language processing and image recognition. A Convolutional Neural Network (CNN) is a Deep Learning algorithm which can take in an input image, assign importance to various aspects/objects in the image and be able to differentiate one from another. The name "convolutional neural network" indicates that the network employs a mathematical operation called convolution. Convolution uses a convolution matrix, which is usually called a **kernel**.

$$\begin{array}{|c|c|c|c|c|c|} \hline \mathbf{I_{11}} & \mathbf{I_{12}} & \mathbf{I_{13}} & \mathbf{I_{14}} & \mathbf{I_{15}} & \mathbf{I_{16}} \\ \hline \mathbf{I_{21}} & \mathbf{I_{22}} & \mathbf{I_{23}} & \mathbf{I_{24}} & \mathbf{I_{25}} & \mathbf{I_{26}} \\ \hline \mathbf{I_{31}} & \mathbf{I_{32}} & \mathbf{I_{33}} & \mathbf{I_{34}} & \mathbf{I_{35}} & \mathbf{I_{36}} \\ \hline \mathbf{I_{41}} & \mathbf{I_{42}} & \mathbf{I_{43}} & \mathbf{I_{44}} & \mathbf{I_{45}} & \mathbf{I_{46}} \\ \hline \mathbf{I_{51}} & \mathbf{I_{52}} & \mathbf{I_{53}} & \mathbf{I_{54}} & \mathbf{I_{55}} & \mathbf{I_{56}} \\ \hline \mathbf{I_{61}} & \mathbf{I_{62}} & \mathbf{I_{63}} & \mathbf{I_{64}} & \mathbf{I_{65}} & \mathbf{I_{66}} \\ \hline \end{array} * \begin{array}{|c|c|c|} \hline \mathbf{K_{11}} & \mathbf{K_{12}} & \mathbf{K_{13}} \\ \hline \mathbf{K_{21}} & \mathbf{K_{22}} & \mathbf{K_{23}} \\ \hline \end{array} \tag{26}$$

The above given operation is the convolution of matrices. The matrix **I** is the image and the matrix **K** is the kernel. The resulting matrix **O** is obtained by the following equation (27)

$$O(i,j) = \sum_{k=1}^{m} \sum_{l=1}^{n} I(i+k-1, j+l-1) K(k,l) \tag{27}$$

Figure 19 illustrates how the resulting image matrix is obtained using the convolution operation.

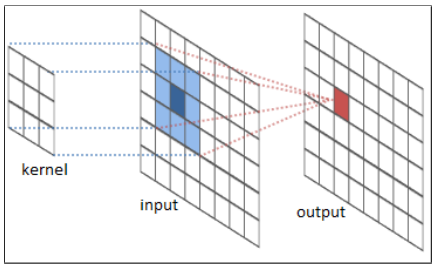


Figure 19: Convolution operation maps a region on the original image into an output value.

Convolutional networks are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers. These layers are illustrated in Figure 20.

CNN Layers: A convolutional neural network usually consists of several layers, which differ from each other and assist different purposes.

- **Convolutional layers** extracts different features from a source image. Convolutions conduct operations like blurring, sharpening, edge detection, noise reduction, or other operations that can help the model to learn specific characteristics of an image.
- **Pooling layers** reduce the dimensions of the data by combining the outputs of neuron clusters at one layer into a single neuron in the next layer. Local pooling combines small clusters, typically 2 x 2. Global pooling acts on all the neurons of the convolutional layer. In addition, pooling may compute a max or an average. Max pooling uses the maximum value from each of a cluster of neurons at the prior layer. Average pooling uses the average value from each of a cluster of neurons at the prior layer.

- **A fully connected layer**, also known as a **dense layer**, is used to receive the results of the CNN and generate a prediction.
- **A flattening layer**, transforms a two-dimensional matrix of features into a vector that can be fed into a fully connected neural network classifier.

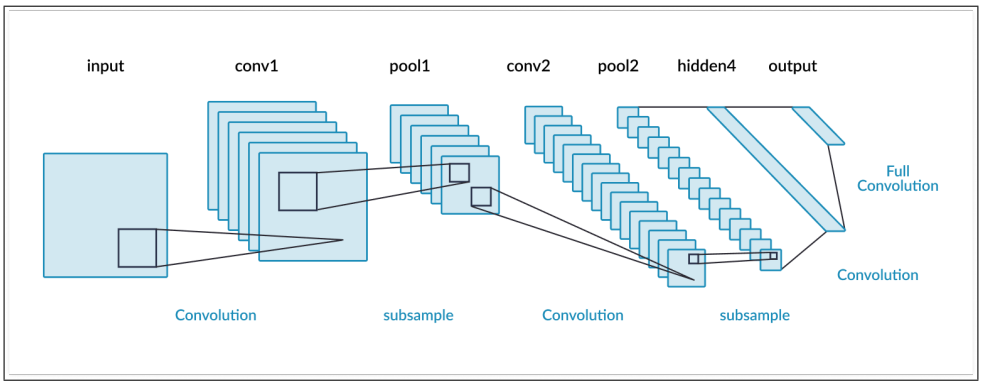


Figure 20: A CNN that contains convolutional, pooling, dense and flattening layers.

For more information about Convolutional Networks please see [11].

6.1.2 Activation Functions

The Rectified Linear Unit (ReLU) is the most commonly used activation function in deep learning models. The function returns 0 if it receives any negative input, but for any positive value x, it returns that value back. So, it can be written as $R(z) = \max(0, z)$. ReLU is illustrated in Figure 21.

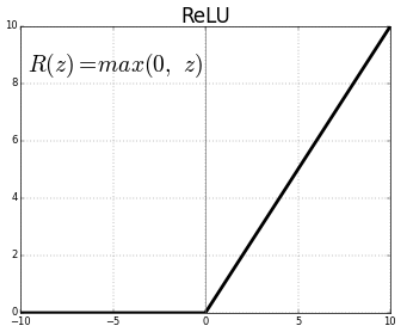


Figure 21: ReLU activation function maps negative values to 0 and returns the non-negative values.

ReLU is used in gradient-based methods for back-propagation of errors to train a neural network. This activation function is almost like a linear function, in most cases it preserves the useful properties of linear models, but is in fact non-linear and capable of learning complex relationships in the data.

Another activation function that is used in the model is **Softmax**. In mathematics, the Softmax function is a function that takes as input a vector of K real numbers, and normalizes it into a probability distribution consisting of K probabilities proportional to the exponentials of the input numbers.The standard (unit) Softmax function $\sigma : \mathcal{R}^K \to \mathcal{R}^K$ is defined by the formula (28)

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{i=1}^{K} e^{z_j}} \text{ for } i = 1, \ldots, K \text{ and } \mathbf{z} = (z_1, \ldots, z_K) \in \mathcal{R}^K \tag{28}$$

For more details please see [10]
In cases when we deal with feed-forward non-linear networks Softmax is used to achieve multiple outputs. We want to treat the outputs of the network as probabilities of pattern classes, conditioned on the inputs. Softmax can be considered a multi-input generalisation of the logistic, operating on the whole output layer with two main modifications:

- probability scoring, which is an alternative to squared error minimisation,
- normalised exponential multi-input generalisation of the logistic non-linearity

An illustration of how the Softmax activation function works is depicted in (29). More details can be found in [11]

$$z \begin{bmatrix} 2.0 \longrightarrow \\ 1.0 \longrightarrow \\ 0.1 \longrightarrow \end{bmatrix} S(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}} \begin{matrix} \longrightarrow p = 0.7 \\ \longrightarrow p = 0.2 \\ \longrightarrow p = 0.1 \end{matrix} \tag{29}$$

Softmax is used to perform multiple classification.

6.1.3 Architecture Overview

Figure 22 illustrates the structure and the main components of the CNN architecture we constructed for the component recognition stage. The Figure 22 was constructed using `model.summary()` (available in Keras), which gives the summary representation of the model, including layer information with respective shapes and the numbers of both trainable and non-trainable parameters. The model is composed of two consecutive **Convolutional layers**, each followed by a **Max-**

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 28, 28, 64)        320
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 14, 14, 64)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 14, 14, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 14, 14, 32)        8224
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 7, 7, 32)          0
_________________________________________________________________
dropout_2 (Dropout)          (None, 7, 7, 32)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 1568)              0
_________________________________________________________________
dense_1 (Dense)              (None, 256)               401664
_________________________________________________________________
dropout_3 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_2 (Dense)              (None, 36)                9252
=================================================================
Total params: 419,460
Trainable params: 419,460
Non-trainable params: 0
```

Figure 22: Symbol Recognition model architecture overview. The model is composed of 2 convolutional layers, each followed by a maximum pooling layer, one flatten and two dense layers

imum Pooling layer for downsampling. **Flatten layer** serves as a connection between the second convolutional layer and the Dense layer. **Dense layer** is the final output layer. One

can notice that the second component of the Dense layer shape is 36, which is the number of classification categories: 10 digits and 26 uppercase letters. Figure 23 is a basic visualization of the architecture.
We use **ReLU** activation for all the main layers, except the last layer, where **Softmax** is used, as it is the usual choice of activation in case of multi-class classification tasks. The input shape of the images is (28,28,1), which means that we feed single channel (height, width) = (28, 28) binary images into the model. The total number of parameters is a measurement of the model complexity. One can also notice three **Dropout layers**, which will be discussed in the "Regularization" section.

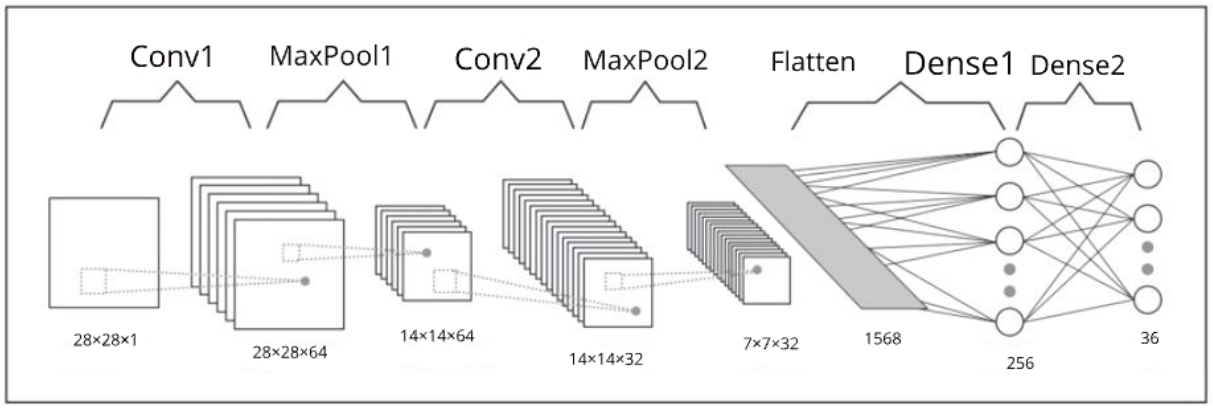


Figure 23: Character recognition CNN architecture visualization.

6.2 Loss Optimization

6.2.1 Cost Function Construction

The right choice for a loss function depends solely on the problem nature and understanding. the symbol recognition task is a multi-class classification, which means that the default choice for a loss function is **Cross entropy**. **Categorical cross entropy** is a loss function that is used for single label categorization. In other words, a specific data sample can belong to one class only. The outputs of our network are probabilities, as we use softmax activation in the last layer, so the cross entropy loss represents the negative log-likelihood of the observed data, which means that minimizing that loss is the same as maximizing the log-likelihood of the training data. The formula below computes the cross entropy loss for the binary case, where y = 0, 1 and $\hat{y} \in [0,1]$.

$$Cost(y, \hat{y}) = -\, y \log(\hat{y}) + (1-y) \log(1-\hat{y}); \tag{30}$$

We slightly modify the formula in case of multi-class classification.

N number of classes

$y_{o,c}$ true class (c) of observation o

$\hat{y}_{o,c}$ predicted probability of observation o being a class c

$$Cost(y, \hat{y}) = -\sum_{c=1}^{N} y_{o,c} \log(\hat{y}_{o,c}); \quad N > 2 \tag{31}$$

As we are dealing with integer targets instead of categorical vectors as targets, we use **Sparse categorical cross entropy**. It's an integer-based version of the categorical cross entropy loss function.

6.2.2 General Gradient Descent

Gradient Descent is an optimization algorithm widely used in Deep Learning to perform cost function minimization. The general idea is to move towards the opposite direction of the gradient: the direction of decrease.

$$w_n = w_{n-1} - \alpha \cdot \Delta Cost(w_{n-1}); \;\; n >= 0 \tag{32}$$

We update the trainable parameters: weights and biases. α coefficient is added to control the step size towards the opposite direction of the gradient. The right choice of alpha parameter and overall step size is crucial. Figure 24 illustrates the effects of too small and too large step size choices.

Small step size:

- Pros: iterations will more likely converge
- Cons: need more iterations and thus evaluations of ΔCost(w)

Large step size:

- Pros: better use of each ΔCost(w_n) may reduce the total iterations
- Cons: can cause overshooting, sometimes even diverged iterations

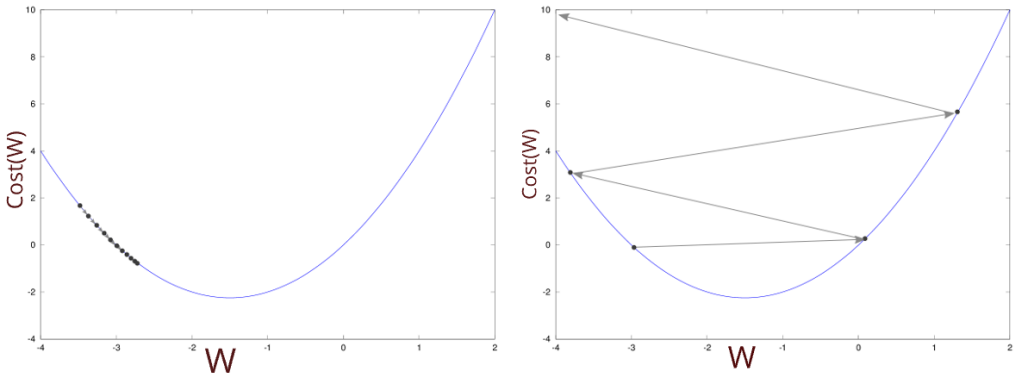


Figure 24: Visualization of effects of small and large step size choices: slow updates in case of small step size and overshooting in case of large step size

Gradient Descent methods can be classified based on step size (e.g., Steepest Descent, Newton's method, Conjugate Gradient Methods) and training batch size (Batch Gradient Descent, Mini-Batch GD and Stochastic GD).

6.2.3 Mini Batch Gradient Descent

Batch Gradient Descent, or vanilla gradient descent, computes the gradient of the function with respect to the whole set of data points. The batch gradient descent surely converges to a minimizer (be it local or, in case of a convex function/surface, even global). However, it's difficult to use. For one iteration, we need to go over all the data points, which creates different problems - memory insufficiency, slow update speed, and being unable to update the model in case of the availability of new data points. Loading an enormous dataset into memory and computing partial derivatives of the huge cost function is computationally intensive and resource-consuming.

- Pros:
  - Guaranteed to converge to the global minimum for convex error surfaces
  - Guaranteed to converge to the local minimum for non-convex error surfaces
- Cons:
  - Low speed
  - High memory usage
  - No possibility of online learning

Stochastic Gradient Descent updates parameters for every single data point one by one. It solves some of the problems of Batch GD: higher speed of learning, low memory usage, and possibility of online learning. However, SGD does not guarantee convergence mainly because of the high rate of oscillation.

- Pros:
  - Higher speed
  - Low memory usage
  - Possibility of online learning
- Cons:
  - High variance updates and high rate of fluctuation
  - Convergence issue

To address the above-described challenges, **Mini-Batch Gradient Descent** has been introduced. Mini-batch GD takes a small random subset from the whole set of data points (the subset size in practice usually varies from 64 to 256) and performs an update using this small batch. On the one hand, the choice of a small batch decreases memory usage. On the other hand, it decreases the variance solving the divergence issue in SGD. The main challenge of the mini-batch GD is the choice of batch size, which is a training hyperparameter. The other problem is that Mini-Batch GD tends to get stuck in the proximity of a saddle point as the gradient close to the saddle point is usually very small (close to zero vector). Generally, the mini-batch GD is the golden middle of Batch GD and SGD, which is why it is widely used in production.

- Pros:
  - Reduced variance of updates compared to SGD
  - Possibility of online learning
  - More efficient memory usage compared to BGD
  - Higher speed compared to BGD
- Cons:
  - Challenging choice of Batch Size as a training hyperparameter

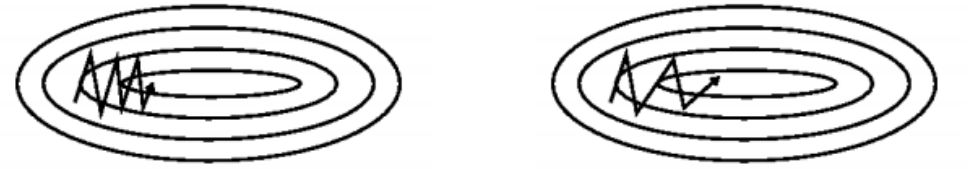
6.2.4 Adam Optimizer

Researchers have introduced several modifications to the general gradient descent algorithm, making it more efficient and practical (RMSProp, Momentum, Adam, etc.).
General GD has trouble navigating ravines, i.e. areas where the surface curves much more steeply in one dimension than in another, which are common around local optima. In these scenarios, GD oscillates across the slopes of the ravine while only making hesitant progress along the bottom towards the local optimum. To address the challenge, we take an **exponentially weighted averages** of the gradient steps.

$$\begin{aligned} v_n &= \beta \cdot v_{n-1} + (1-\beta) \cdot \Delta Cost(w_n) \\ v_{n-1} &= \beta \cdot v_{n-2} + (1-\beta) \cdot \Delta Cost(w_{n-1}) \\ v_{n-2} &= \beta \cdot v_{n-3} + (1-\beta) \cdot \Delta Cost(w_{n-2}) \end{aligned} \tag{33}$$

The current value of v_n depends on all previous values of v, which got coefficients (weights) in the sum. This weight is β to power of i multiplied by $(1-\beta)$ for $(n-i)^{th}$ value of the gradient. As β is a number between 0 and 1 (0.9 recommended by the algorithms authors), older v_n-s get smaller and smaller coefficients. At some point it becomes so small, that those terms with such a small coefficient are neglected. To sum up, we construct momentum, which is a weighted average of the gradients. Momentum helps to keep the speed of the progress in fast directions and speed up the progress in slow directions (illustrated in Figure 25).

$$\begin{aligned} v_n &= \beta \cdot v_{n-1} + (1-\beta) \cdot \Delta Cost(w_{n-1}) \\ w_n &= w_{n-1} - \alpha \cdot v_n; \;\; n \geq 0 \end{aligned} \tag{34}$$



(a) SGD without momentum (b) SGD with momentum

Figure 25: Visualization of GD with and without momentum (Retrieved from [17])

As you can see the updates in horizontal direction without momentum are very slow while the updates of vertical direction are big. This causes that unnecessary fluctuation which slows the convergence to the minimum. Momentum here helps to reduce the step size in vertical direction, as we sum together many vectors that point towards opposite directions and increase the step size in horizontal direction, because we sum vectors that face the same direction. The described idea is implemented in the famous optimization algorithm called **Momentum**
We already saw how momentum can speed up gradient descent. There is also another algorithm called **RMSProp** that is widely used in deep learning applications. The name stands for Root Mean Square Prop. As we saw before, the main problem with Gradient Descent was the possibility of huge oscillation in vertical direction and slow progress in horizontal direction and this is what RMSProp is intended to solve. The intuition is similar to momentum, but RMSProp works on amplifying the slow directions and slowing the fast directions in a different manner. In slow

directions, s_n will be a small number. As it is in the denominator of the fraction, the update in that direction will be bigger. For fast directions, the RMSProp causes the opposite effect. RMSprop divides the learning rate by an exponentially decaying average of squared gradients. Hinton suggests β to be set to 0.9, while a tested good default value for the learning rate α is 0.001In practice, a very small positive parameter ε is added to s_n just to avoid 0 in the denominator.

$$s_n = \beta \cdot s_{n-1} + (1 - \beta) \cdot \Delta Cost(w_{n-1})^2$$

$$w_n = w_{n-1} - \alpha \cdot \frac{\Delta Cost(w_{n-1})}{\sqrt{s_n}} \tag{35}$$

$$w_n = w_{n-1} - \alpha \cdot \frac{\Delta Cost(w_{n-1})}{\sqrt{s_n} + \varepsilon}$$

Adaptive Moment Estimation (Adam) is another method that computes adaptive learning rates for each parameter. In addition to storing an exponentially decaying average of past squared gradients like RMSprop, Adam also keeps an exponentially decaying average of past gradients, similar to momentum. Taking into account this description one can imagine Adam as a combination of previous two: Momentum and RMSprop. The algorithm leverages the power of adaptive learning rates methods to find individual learning rates for each parameter. Adam uses the squared gradients to scale the learning rate like RMSprop and it takes advantage of momentum by using moving average of the gradient instead of gradient itself like SGD with momentum. Having both of these enables us to use Adam for broader range of tasks.

$$\begin{aligned} v_n &= \beta_1 \cdot v_{n-1} + (1 - \beta_1) \cdot \Delta Cost(w_{n-1}) \\ s_n &= \beta_2 \cdot s_{n-1} + (1 - \beta_2) \cdot \Delta Cost(w_{n-1})^2 \end{aligned} \tag{36}$$

$$v_n^{corr} = \frac{v_n}{1 - \beta_1^n}$$

$$s_n^{corr} = \frac{s_n}{1 - \beta_2^n} \tag{37}$$

$$w_n = w_{n-1} - \alpha \cdot \frac{v_n^{corr}}{\sqrt{s_n^{corr}} + \varepsilon}$$

When the v_n and s_n are small (especially, at the beginning), they are biased towards zero. This is why a special step of bias correction is applied as shown in the equations 37. In practice, the most common value for β_1 is 0.9, for β_2 is 0.999 and 10^{-8} for ε.

6.3 Training and Hyperparameter Tuning

Symbol Recognition task implementation is composed of three main stages: training, validation and testing. Training and validation are strongly interconnected. As we have already mentioned, the training process is predominantly cost function minimization. The main challenge of this stage is the hyperparameter tuning. Below is a comprehensive list of our model's most important parameters and training hyperparameters:

- Number of training images – 29233
- Number of validation images – 7345

- Number of test images – 2979
- Optimization algorithm – Adam
- Training batch size – 1024
- Initial learning rate – 0.001
- β_1 – 0.9
- β_2 – 0.999
- Training epochs – 100

6.3.1 Regularization

The central challenge in machine learning is that we must perform well on new, previously unseen inputs, not just those on which our model was trained. The ability to perform well on previously unobserved inputs is called **generalization**. The wrong choice of flexibility and complexity of the model may result in well known problems in machine learning: underfitting or overfiting (See Figure 26). An underfit model has low variance and high bias. On the other hand, an overfit model has high variance and low bias. To overcome underfitting, we may introduce higher capacity for the model, train longer. In case of overfitting the most obvious solution is increasing the amount of training data. However, in most of the cases, data is not available or it requires cosly annotation. **Dropout**, which was already mentioned in the "Architectur Overview" section, is a widely used technique to address the problem of overfitting.

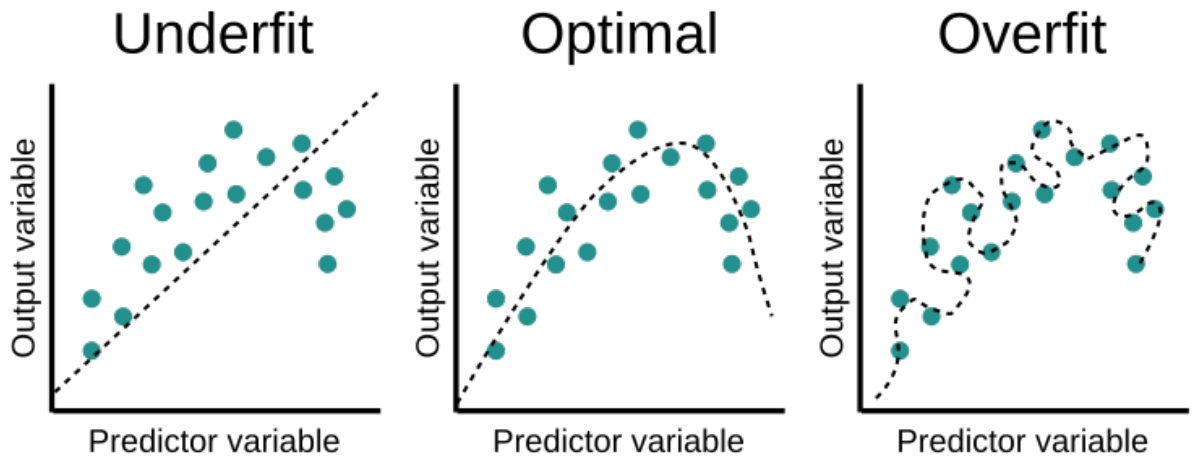


Figure 26: Visualization of Underfitting, Optimal fitting and Overfitting

For each training example within a mini-batch regularization implements the following steps:

- For each layer setup a value p — probability of keeping each node.
- Remove all the nodes (as well as ingoing and outgoing links) that were not selected to stay. As a result we have a new, “thinned” network of the layer.

There are three dropout layers in our model: dropout_1 (p = 0.3), dropout_2 (p = 0.3), dropout_3 (p = 0.5). For details please see [21]

7 Results And Discussion

7.1 Character Extraction Phase Results and Evaluation

Result Evaluation is an essential part of the project. At this stage, we assess the quality and effectiveness of the algorithm. Observation of the current state estimates the status of the outcome(not the process) and uncovers the necessity of further improvements.
Most of the time, evaluation of Computer Vision algorithms is subjective because of the significant visual inspection component. Symbol extraction task is not an exclusion. At this stage, we evaluate only the results of the extraction component and not the process(separate stages of the algorithm). Using visual evaluation, we construct a slightly modified version of a Confusion Matrix with the following components:

- **"True Positive"** – Extracted characters that identify as valid plate symbols
- **"False Positive"** – Extracted characters that do not identify as valid plate symbols
- **"False Negative"** – Non - extracted characters that identify as valid plate symbols

We omit the **"True Negative"** as it is both technically impossible and unnecessary to count the number of non-extracted characters that do not identify as valid plate symbols. We add to the constructed table the total amount of classified characters (**"Total_Classified"**), valid characters (**"Total_Valid"**) and the total amount of extracted characters (**"Total_Extracted"**).

Modified Confusion Matrix						
Total_Images	Total_Classified	Total_Valid	Total_Extracted	TP	FP	FN
436	3067	3040	3027	3000	27	40

To fully capture the nature of the resulted output, we introduce several evaluation metrics: accuracy, precision, recall, F1_Score. Using the data in the previous table, we construct a new table populating it with the computed values of the respective evaluation metrics:

- **Accuracy** – ratio of the number of correctly extracted characters to the total number of classified characters.

$$Accuracy = \frac{TP}{Total\_Classified} = \frac{TP}{TP + FP + FN} \tag{38}$$

- **Precision** – ratio of the number of correctly extracted characters to the total number of extracted characters

$$Precision = \frac{TP}{Total\_Extracted} = \frac{TP}{TP + FP} \tag{39}$$

- **Recall** – ratio of the number of correctly extracted characters to the total number of valid characters

$$Recall = \frac{TP}{Total\_Valid} = \frac{TP}{TP + FN} \tag{40}$$

- **F1_Score** – Harmonic mean of previously calculated Precision and Recall

$$F1\_Score = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{41}$$

One can notice that the formulas of the metrics are slightly modified because of the unavailability of True Negative(TN) component.

Evaluation Metrics				
Total_Images	Accuracy	Precision	Recall	F1_Score
436	97.81%	99.10%	98.68%	98.88%

7.2 Character Recognition Phase Results and Evaluation

We use Python's Matplotlib library to illustrate the loss and accuracy curves for both training and validation sets. See Figure 27 and Figure 28 for details. After 100 epochs of training we observe model convergence and the following results:

Train/Validation/Test Results					
Training Loss	Training Acc	Validation Loss	Validation Acc	Test Loss	Test Acc
0.0966	96.57%	0.0845	97.36%	0.3295	93.28%

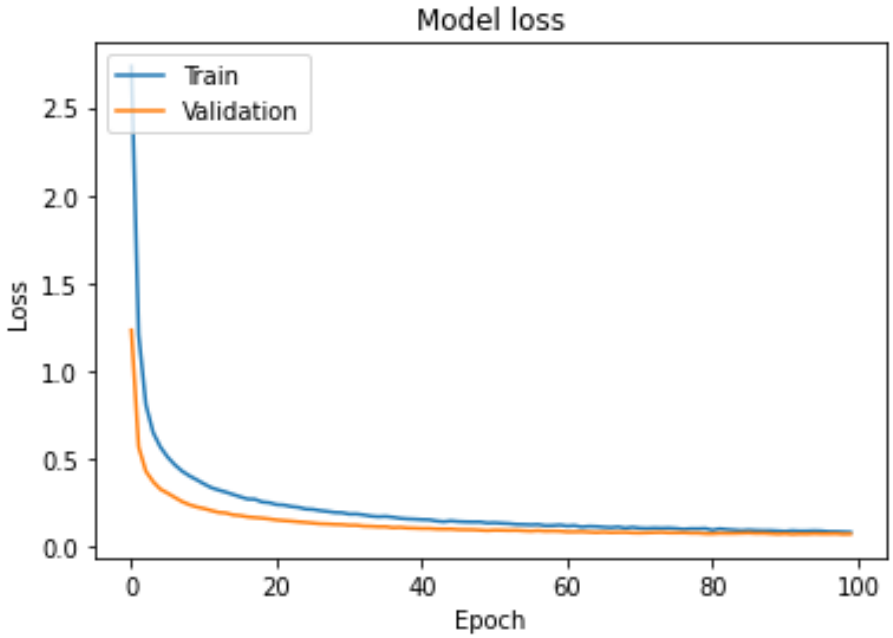


Figure 27: Training and validation loss curves for epochs[0,100]

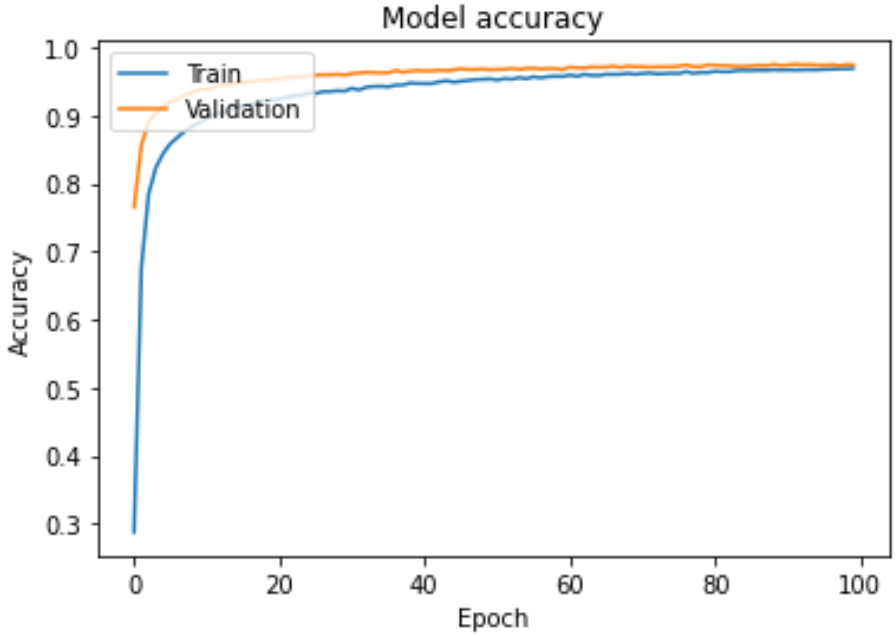


Figure 28: Training and validation accuracy curves for epochs[0,100]

8 Conclusions and Future Work

8.1 Future Work and Final Considerations

The logical continuation of the project is the expansion of the scope of the project and the improvement of already available results. **Plate localization** and **normalization** are the two major additions to a project.
Plate localization is an indivisible part of ANPR systems. Figure 29 illustrates the processing step. The collected data form the hardware equipment is composed of vehicle images and not the cropped plate, which makes this preprocessing stage unavoidable. The most challenging task of the plate localization is its definition in terms of computer vision. Several similar schemes use statistical analysis of the projections of vertical and horizontal edges to define the plate on the image, assuming that in the plate region, there is an increased amount of occurrence of horizontal and vertical edges. Another approach suggests a vehicle plate number localization using a modified GrabCut algorithm. For details, please see [18].

Figure 29: Illustration of plate localization phase

In the project, we partially address the **plate normalization** component of the ANPR algorithm by incorporating the Otsu thresholding, which finds an optimal value based on the image information. However, we consider the possibility of further improvement via integration of **histogram equalization** methods. Contrast enhancement is a common preprocessing step in digital image processing that increases the separation between the darkest and brightest areas of the image. The conventional approach of contrast adjustment is histogram equalization. There are several advanced approaches for histogram equalization like Contrastive Limited Adaptive Equalization and Dynamic Histogram Equalization; however, at this stage, we consider the traditional Global Histogram Equalization(GHE)[2] algorithm. Although it may result in some specific side effects like information loss and artifact appearances, this simple transformation significantly improves the state of the image. The new distribution is obtained using the Cumulative Distribution Function of the current distribution:

L number of the histogram bins

n_i number of pixels in the i^{th} bin

N total number of pixels (N = n_1 + n_2 + ... + n_L)

k 0, 1, ..., L - 1

$$new_k = \sum_{i=0}^{k} \frac{n_i}{N} = CDF(old_k) \tag{42}$$

Figure 30 illustrates the original image and the image after histogram equalization. Histograms of the corresponding images are added below to illustrate the effect of the transformation.
Character segmentation can possibly be improved by implementing additional layers of component post-processing, like morphological operations, skeletonization, character contour, and edge

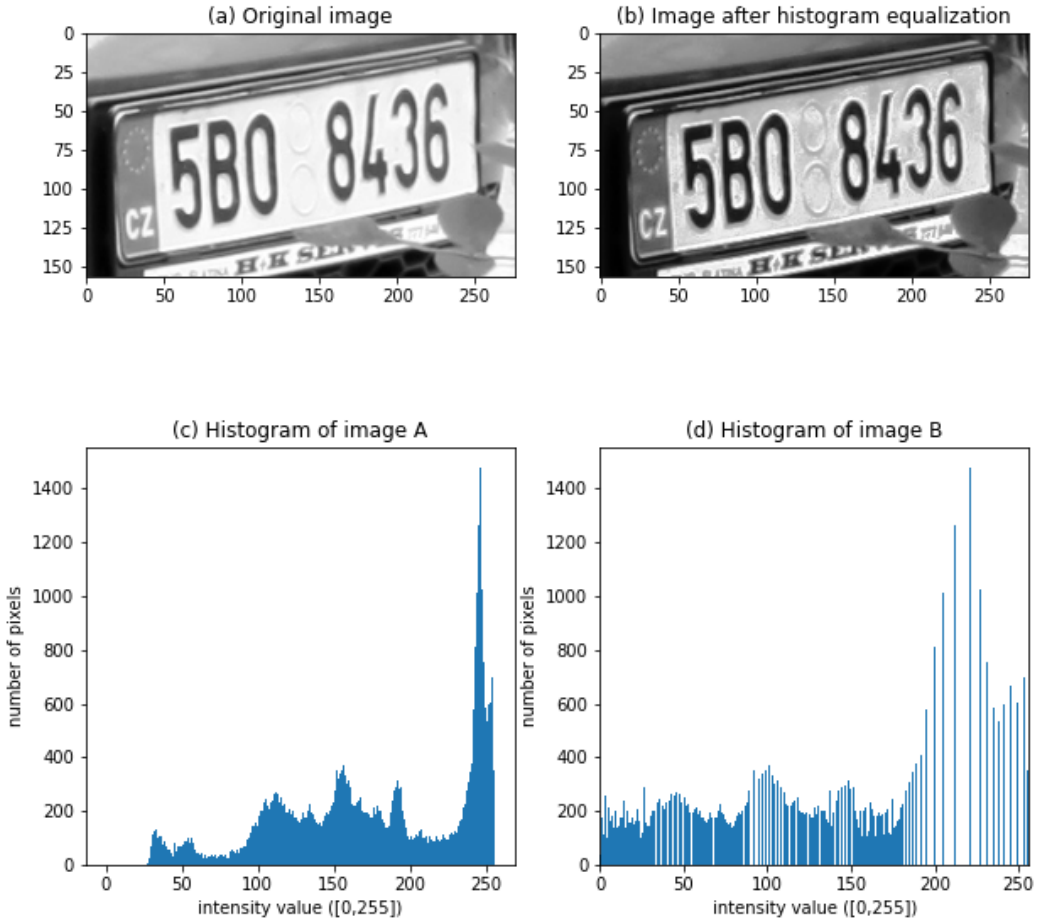


Figure 30: Original image and image after histogram equalization with corresponding histograms

analysis, etc.

The generalization of the projects is another aspect of improvement. Data sources introduce additional constraints on the processing stages. One can consider datasets from a broader range of sources and address the challenges of newly introduced limitations like various colors of the character and plate background, character fonts, plate shape, etc.

Further research and experimentation are needed to check the possibility of improvement and completion of the suggested scheme. After having all working components with the respective algorithmic and mathematical aspects starting from image source to expected output, implementation of the general and complete ANPR software is possible.

8.2 Conclusions

Intelligent traffic management systems are an indispensable part of Smart cities and digital ecosystems, making the exploration of its theoretical and implementation bases a subject of research interest. ANPR solutions are widely used in many countries, enhancing the productivity of transportation management systems. This project addressed the main components of ANPR systems by exploring its mathematical foundations. Despite resource limitations (datasets, time), we were able to construct a simple scheme implementing the two major components of ANPR systems: character extraction and recognition. Although the separate components of the scheme have not been fine-tuned, we were able to observe compelling results. Further improvements, completion, and generalization of the scheme form a solid base for effective ANPR system implementation.

References

[1] *History of ANPR.* http://www.anpr-international.com/history-of-anpr/.

[2] M. Abdullah-Al-Wadud, H. Kabir, A. Dewan, and O. Chae, *A dynamic histogram equalization for image contrast enhancement*, IEEE Transactions on Consumer Electronics, 53 (2007), pp. 593–600.

[3] G. Bradski, *Canny edge detection.* https://docs.opencv.org/trunk/da/d22/tutorial_py_canny.html/.

[4] ——, *The opencv library*, Dr. Dobb's Journal of Software Tools, (2000).

[5] D. Ciregan, U. Meier, and J. Schmidhuber, *Multi-column deep neural networks for image classification*, in 2012 IEEE Conference on Computer Vision and Pattern Recognition, 2012, pp. 3642–3649.

[6] T. Cormen, C. Leiserson, R. Rivest, and C. Stein, *Introduction to Algorithms*, The MIT Press, 3 ed., 2009.

[7] S. Draghici, *A neural network based artificial vision system for licence plate recognition*, International Journal of Neural Systems, 8 (1997), pp. 113–126.

[8] S. Du, M. Ibrahim, M. Shehata, and W. Badawy, *Automatic license plate recognition (alpr): A state-of-the-art review*, IEEE Transactions on Circuits and Systems for Video Technology, 23 (2013), pp. 311–325.

[9] R. Fisher, S. Perkins, A. Walker, and E. Wolfart, *Gaussian smoothing.* https://homepages.inf.ed.ac.uk/rbf/HIPR2/gsmooth.htm, 2003.

[10] X. Glorot, A. Bordes, and Y. Bengio, *Deep sparse rectifier neural networks*, 14th International Conference on Artificial Intelligence and Statistics(AISTATS), 15 (2011).

[11] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, MIT Press, 2016. http://www.deeplearningbook.org.

[12] C. Grana, R. Cucchiara, and D. Borghesani, *Optimized block-based connected components labeling with decision trees*, IEEE Transactions on image processing, 19 (2010).

[13] T. Kacmajor, *Hough lines transform explained.* https://medium.com/@tomasz.kacmajor/hough-lines-transform-explained-645feda072ab/.

[14] O. Martinsky, *Algorithmic and mathematical principles of automatic number plate recognition systems*, Brno University of technology, (2007), pp. 20–23.

[15] N. Otsu, *A threshold selection method from gray-level histograms*, IEEE transactions on systems, man, and cybernetics, 9 (1979), pp. 62–66.

[16] A. Permaloff and C. Grafton, *Optical character recognition*, PS: Political Science and Politics, 25 (1992), pp. 523–531.

[17] S. Ruder, *An overview of gradient descent optimization algorithms*, arXiv preprint arXiv:1609.04747, (2016).

[18] A. Salau, T. Yesufu, and B. Ogundare, *Vehicle plate number localization using a modified grabcut algorithm*, Journal of King Saud University-Computer and Information Sciences, (2019).

[19] G. Shperber, *A gentle introduction to ocr*, Apr 2020.

[20] J. Špaňhel, J. Sochor, R. Juránek, A. Herout, L. Maršík, and P. Zemčík, *Holistic recognition of low quality license plates by cnn using track annotated data*, in 2017 14th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS), IEEE, Aug 2017, pp. 1–6.

[21] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, *Dropout: a simple way to prevent neural networks from overfitting*, The journal of machine learning research, 15 (2014), pp. 1929–1958.

[22] J. Yousefi, *Image binarization using otsu thresholding algorithm*, University of Guelph, Ontario, Canada, (2011).