Fall 12-1-2015

Automatic License Plate Recognition Using Deep Learning Techniques

Naga Surya Sandeep Angara

AUTOMATIC LICENSE PLATE RECOGNITION USING DEEP LEARNING TECHNIQUES

by

NAGA SURYA SANDEEP ANGARA

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science in Electrical Engineering
Department of Electrical Engineering

Melvin Robinson, Ph.D., Committee Chair
College of Engineering

The University of Texas at Tyler
December 2015

The University of Texas at Tyler
Tyler, Texas

This is to Certify that the Master's Thesis of

NAGA SURYA SANDEEP ANGARA

has been approved for the thesis requirements on

NOVEMBER 12, 2015

for the Master of Science in Electrical Engineering

Approvals

Melvin Robinson

Thesis Chair: Melvin Robinson, Ph.D.

[illegible]

Member: Mukul V. Shirvaikar, Ph.D.

Ron J. Pieper

Member: Ron J. Pieper, Ph.D.

[illegible]

Chair and Graduate Coordinator: Hassan El-Kishky, Ph.D.

[illegible]

James K. Nelson, Jr., Ph.D., P.E.
Dean, College of Engineering

Acknowledgements

I am pleased to express my sincere and heartful thanks to everyone who has supported and encouraged me. Firstly, I would like to thank my family for supporting and constantly encouraging me in every phase of my life to achieve my goals. A special thanks for all the sacrifices that you have made for me. It means a lot to me.

I would like to express my profound gratitude to Dr. Melvin Robinson for his constant support, supervision and encouragement. I would also like to thank him for his patience and his timely suggestions throughout the research. I would like to express my sincere thanks to Dr. Ron Pieper and Dr. Mukul V. Shirvaikar for taking time to be part of my committee and reviewing my work. I would like to thank the entire Electrical Engineering department and The University of Texas at Tyler for supporting me throughout my Masters degree. I would also like to thank Dr. Michael Manry and Rohit Rawat of University of Texas at Arlington for their valuable suggestions during this research. Finally, I would like to thank my well-wishers and friends for their constant support and encouragement throughout the journey of the thesis.

Table of Contents

List of Tables

List of Figures

List of Algorithms

Abstract

AUTOMATIC LICENSE PLATE RECOGNITION USING DEEP LEARNING TECHNIQUES

NAGA SURYA SANDEEP ANGARA

Thesis Chair: Melvin Robinson, Ph.D.

The University of Texas at Tyler
December 2015

Automatic License Plate Recognition (ALPR) systems capture a vehicle's license plate and recognize the license number and other required information from the captured image. ALPR systems have number of significant applications: law enforcement, public safety agencies, toll gate systems, etc. The goal of these systems is to recognize the characters and state on the license plate with high accuracy.

ALPR has been implemented using various techniques. Traditional recognition methods use handcrafted features for obtaining features from the image. Unlike conventional methods, deep learning techniques automatically select features and are one of the game changing technologies in the field of computer vision, automatic recognition tasks and natural language processing. Some of the most successful deep learning methods involve Convolutional Neural Networks. This research applies deep learning techniques to the ALPR problem of recognizing the state and license number from the USA license plate.

Existing ALPR systems include three stages of processing: license plate localization, character segmentation and character recognition but do little for the state recognition problem. Our research not only extracts the license number, but also processes state information from the license plate. We also propose various techniques for further research in the field of ALPR using deep learning techniques.

Chapter One

Introduction

Automatic License Plate Recognition (ALPR) technology has seen rapid and widespread adoption by many agencies worldwide to enhance their enforcement, investigative and security capabilities. This technology helps in collecting vehicle information which reduces tedious and time consuming manual work. ALPR is especially useful to government agencies in tracking stolen vehicles, persons who violate traffic rules, vehicles in connection with a crime or other vehicles of interest. Other applications include private parking lot management, traffic monitoring, automatic traffic ticket issuing, automatic toll payment and surveillance.

1.1 Automatic license plate recognition system

ALPR systems capture an image of a vehicle's license plate and extract its license number, state and other information. The extracted license number portion of the plate is transformed into alphanumeric characters for recognition. These alphanumeric characters are compared to one or more databases to identify the license number of vehicle of interest of law enforcement agencies or other agencies. This whole process takes only a matter of seconds. A typical ALPR system employs the following:

1. Plate localization: finds and isolates the license plate from the captured image by the camera.
2. Plate orientation and resizing: deskews and resizes the image per requirements. Because the camera captures the image at an angle, the license plate images are quadrilateral, not rectangular. The deskew process helps in skew-slant–shear corrections to the captured image.

3. Image normalization: changes the intensity values of the image to match the characteristics of a reference image. Some of the approaches include scaling and histogram equalization.

4. Character segmentation: extracts the alphanumeric characters from the license plate.

5. Character recognition: recognizes the license number of the vehicle.

1.1.1 Difficulties

Successful ALPR systems must overcome a number of difficulties, namely:

1. Poor image resolution due to low quality cameras.

2. Poor lighting and low contrast due to overexposure, reflections and shadows.

3. Motion blur due to vehicle speed.

4. Various formats of plate designs, which differ from state to state.

Some of the difficulties can be overcome in some software, while some require hardware to overcome, others can be avoided by standardizing the license plate format.

1.2 Machine learning fundamentals

1.2.1 Supervised vs unsupervised learning

The learning process is categorized into two different types: supervised and unsupervised learning methods [1]. Supervised learning method is also known as learning with a teacher and the latter is known as learning without teacher.

1.2.1.1 Supervised learning

In supervised learning the classifier directly learns the relationship between the input pattern and the desired output. The training data consists of an input vector of data and corresponding labels.

The block diagram in Fig. 1.1 illustrates supervised learning. The teacher in the diagram provides a set of input patterns and the desired output (class label) to build the model. The system parameters are adjusted under the combined influence of the training set and the error signal using various optimization techniques. The difference between the desired response and the actual response serves as input to a cost function which helps to optimize the system parameters. The adjustment of network parameters occurs iteratively to make the system emulate the teacher. Optimal system parameters allow the system to correctly determine the system labels for unseen instances. Examples of supervised models neural networks, logistic regression perceptron, decision trees etc.

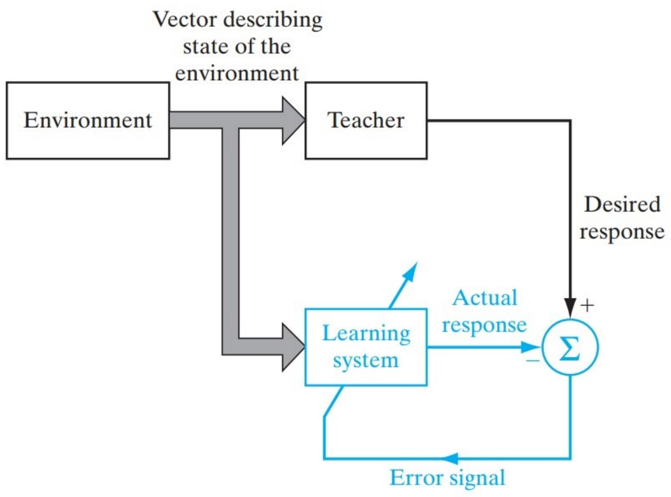


Figure 1.1: Block diagram of supervised learning (Haykin, Simon S., et al. [1])

In supervised learning the training set provided acts like a teacher. The training set is represented as $\left\{(\mathbf{x}^1, y^1), (\mathbf{x}^2, y^2), (\mathbf{x}^3, y^3), ..., (\mathbf{x}^i, y^i)\right\}$, where $\mathbf{x}^i$ is a feature vector of i^{th} pattern in the dataset and y^{i} is the desired output class for the respective feature vector and $\mathbf{w}$ is a vector of network parameters. The labelled examples are given as input for training system. The average loss $\text{E}_{\text{train}}\big(\mathbf{w}\big)$ is calculated while training the system using the training dataset. Finally the learning problem computes $\mathbf{w}$ parameters which minimizes $\text{E}_{\text{train}}\big(\mathbf{w}\big)$ [2,3]. The performance of the system is measured using test set which contains different data from the training set. The training and testing dataset consists of labeled examples, also disjoint sets. Some of the examples of trainable machines using supervised learning are convolution neural networks, support vector machines, multilayer perceptrons and decision trees.

1.2.1.2 Unsupervised learning

In unsupervised learning, there is no teacher to supervise the learning process. The machine learns the solution without a teacher. In this learning the data is provided and the desired output is not provided. The algorithm tries to find the similarities between the inputs and classify the inputs that have common properties into a respective classes. Some of the unsupervised models include clustering of data, Self-Organized Maps, k means algorithm etc.

1.3 Introduction to the deep learning

Deep learning is a new learning paradigm in deep structured learning or hierarchical learning [4, 5]. Deep learning is an active research area of neural networks, artificial intelligence, pattern recognition etc. Deep learning has different high-level definitions:

1. Deep learning has been characterized by learning or exploring automatic feature extraction from many layers of non-linear functional units. Each successive layer uses the output of one or more preceding layers as input.

2. High level features are learned from low level features which form a hierarchical representation for supervised or unsupervised feature extraction and transformation, and for pattern recognition, analysis and classification.

3. Learning where learning of data involves modeling the complex relationships among data.

Deep learning techniques are based on distributed representation of data. The underlying assumption behind distributed representations is that the data is generated by the interactions of many different factors on different levels of hierarchy. Deep learning assumes that these factors are organized into numerous levels, corresponding to different levels of abstraction or composition. The number of layers and sizes of layers can be varied and provide various levels of abstraction. Some of the reasons for increase in usage of deep learning in current research is due to the of high performance graphical processing units, advances in machine learning research and the availability of big data for training and testing.

Examples of recent developments in this research area include:

1. GoogLeNet [6] a 22 layer deep network which was presented in ImageNet Large-Scale Visual Recognition Challenge 2014 (ILSVRC14) stood first in the challenge with an error of 6.67%. The challenge is to classify the input image into one of the 1000 classes.

2. R-CNN is one of the state of art technology for object recognition by Girshick et al. [7]. R-CNN detects the object in an image and segments.

3. Neural network architecture provided by Google X lab [8] with one billion connections using $16,000$ computer processors. The network led to recognize cats from the youtube videos using deep learning algorithms. The system achieved an accuracy of 74.8% for identifying cats, 76.7% accuracy in identifying human body parts and 81.7% in detecting human faces.

Leading researchers in deep learning include Google, Facebook, UC-Berkley, UC-Irvine, University of Toronto and others, where ongoing research shows an empirical success in diverse applications of computer vision, voice recognition, pattern recognition, convolution neural networks.

1.3.1 Deep learning vs conventional learning

Deep learning techniques are powerful machine learning models that accomplish fabulous performance on the learning problems like object recognition [3, 9–11] and speech recognition [12].

Conventional learning extracts features from the available input and then classifies features from the previous step with a classifier. Feature extraction is a procedure of examining and getting valuable data from the given input data. Some feature extractors in the field of computer vision are scale-invariant feature transform(SIFT) [13] and histogram of oriented gradients(HOG) [14]. A typical learning methodology is a handcrafted feature extractor, where the features are selected manually and the selection of features is a difficult task which depends on the application. Designing good feature extractors is a painful job. Fig. 1.2 illustrates the block diagram of traditional model.

By contrast, deep learning neural networks can be trained using a labeled training set which have the potential to compute the network parameters. Deep learning mod-

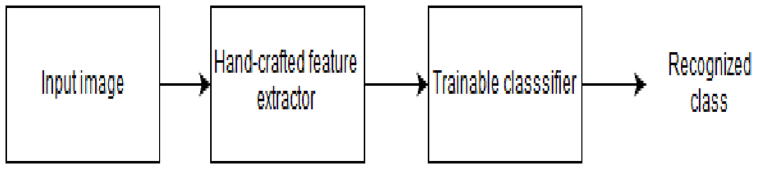


Figure 1.2: Block diagram of conventional learning mode (LeCun, Yann.,et al. [2])

els surpass the capabilities of conventional learning models. Deep learning techniques extract the relevant features from input data and learn from the training set itself. Fig. 1.3 illustrates a typical deep learning model.

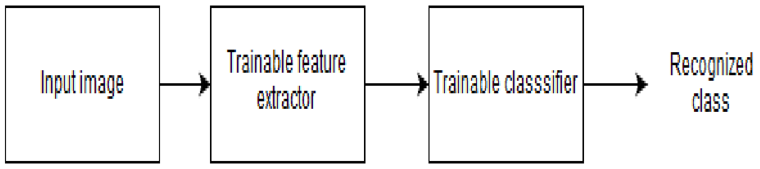


Figure 1.3: Block diagram of deep learning model (LeCun, Yann.,et al. [2])

1.3.2 Cost functions

In machine learning cost functions, also known as loss functions, are a measure of how well the output of a machine learning system agrees with the desired class of the training data. The goal in supervised learning is to minimize the loss incurred i.e. the optimal solution is the one which minimizes the loss function. A machine learning system computes a function $y_i = f(\mathbf{x}_i, \mathbf{w})$, where $\mathbf{x}_i$ is the i^{th} input vector and $\mathbf{w}$ represents the network parameters in the system. Here y_i represents the recognized class for the given input vector $\mathbf{x}_i$ or confidence score or probability associated with each class, let t_i be the desired class and N be the total number of patterns. The cost function is represented by $E(\mathbf{w})$ where $\mathbf{w}$ vector is the weight vector or network parameters.

A commonly used loss function for the regression problem is Mean Squared Error (MSE) [15], which is given as follows

$$E(\mathbf{w}) = \frac{1}{N}\sum_{i=1}^{N} ||t_i - y_i||^2 \tag{1.1}$$

A commonly used cost function for classification is cross entropy [15] given as:

$$E(\mathbf{w}) = \frac{1}{N}\sum_{i=1}^{N}[t_i \log y_i + (1 - t_i)\log(1 - y_i)] \tag{1.2}$$

1.3.3 Numerical optimization

Optimizing a cost function is the process of finding weight parameters $\mathbf{w}$, in order to minimize $E(\mathbf{w})$. A number of iterations takes place in order to compute the parameters which minimize the cost function. First note that if we make a small step in weight space from $\mathbf{w}$ to $\mathbf{w}+\Delta\mathbf{w}$ to then the change in the error function is $\Delta E \approx \Delta\mathbf{w}^T \nabla E(\mathbf{w})$, where the vector $\mathbf{w}$ points in the direction of greatest rate of the error function. The error is a smooth continuous function of $\mathbf{w}$, its smallest value will occur in weight space such that gradient of error function vanishes, so that point in weight space such that gradient of error function vanishes as otherwise we could make a small step in the direction of $\nabla E(\mathbf{w})$ and thereby further reduce the error to

$$\nabla E(\mathbf{w}) = 0 \tag{1.3}$$

The goal of optimization technique is to find a vector $\boldsymbol{w}$ or model parameters such that $E(\mathbf{w})$ is small. In order to find a solution for $\nabla E(\mathbf{w}) = 0$ we often make use of iterative numerical algorithms. In most techniques $\mathbf{w}^{(0)}$ is initialized with some initial value and later updated by moving in weight space in succession of steps of the form

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + \Delta\mathbf{w}^{(t)} \tag{1.4}$$

Where t represents the iteration step. The values of $\Delta\mathbf{w}^{(t)}$ depends on the algorithm selected. After each iteration the weight vector is updated until some stopping criteria is achieved.

1.4 Organization of thesis

This thesis is organized into six chapters. Chapter 2 briefly covers past ALPR research and present our methods. Chapter 3 provides information about segmentation techniques. Chapter 4 gives an overview of convolution neural networks, optimization of the CNN and our proposed architecture for the ALPR system. Chapter 5 presents the results acquired in the thesis work. Chapter 6 gives conclusions and out ideas for future work.

Chapter Two

Background

We briefly describe past ALPR research. Most of the research published in ALPR was conducted on license plates of countries which have strict design standards. As most of the previous systems concentrate only on license number, they are ineffective in detecting the state information from USA license plates.

As shown in Fig. 2.1, a typical ALPR system includes image capture, preprocessing, localization of license number and recognition of the license number. As our research concentrates on information extraction and recognition from the license plate, we review license plate segmentation and recognition.

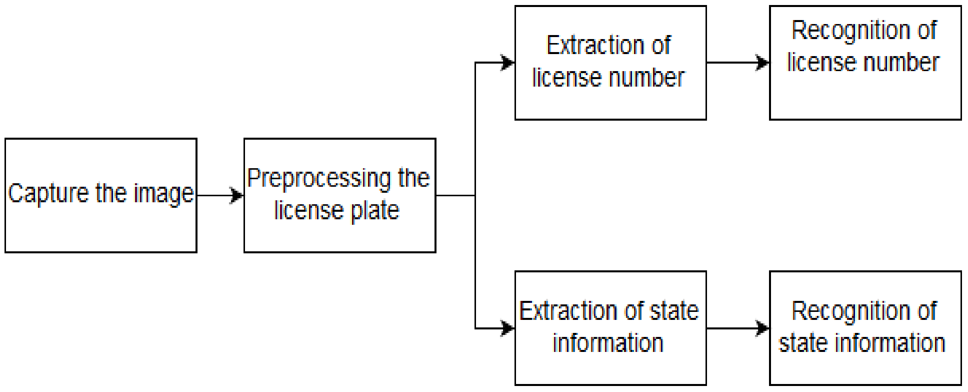

Capture the image
Preprocessing the license plate
Extraction of license number
Recognition of license number
Extraction of state information
Recognition of state information

Figure 2.1: Block diagram of typical ALPR system

Part of any ALPR system includes image segmentation which is a process that labels connected pixels. Image segmentation is a well studied problem. License plate segmentation is based on features used in the image. Some of the methods are discussed below:

1. Segmentation using pixel connectivity: segmentation is performed using connected pixels in binary image. Labeled pixels with same size and aspect ratio are considered characters. This method fails to extract joined characters [16].

2. Segmentation using projection profiles: the foreground and background pixels have opposite values in binary image. This helps to determine the starting and ending position of characters using vertical projection. The horizontal positon can be used to extract the license number information. The main advantage of this method is extraction of character is independent of position and simplest one. The disadvantage of this method is noise affects the projection values [16].

3. Segmentation using prior knowledge: this method uses the information like height, width, aspect ratio and color distribution. This method is simple but limited to prior knowledge. Any change in the license plate results in errors. Segmentation of binary image can be done using combined features i.e. two or more features. This method is more reliable but computationally quite complex [16].

4. Segmentation using contours: contour tracing technique [16] extracts the exact boundaries in an image. It also extracts information about the boundaries in the image. This main disadvantage of this technique is slow and generate incomplete or distoted image.

All ALPR systems have some sort of character recognition. Anagnostopoulos et al. [17] implemented the system using Probabilistic Neural Network (PNN) for character recognition. The PNN is trained to identify alphanumeric characters from license plate. The system has been tested on various license plates with different illumination conditions with segmentation accuracy of 96.5%, plate recognition accuracy of 89.1% and the overall accuracy of 86%.

An automatic license plate recognition system must recognize alphanumeric characters. Nikolaos [17] and Anagnostopoulos [17] implemented LPR system using sliding concentric window for segmentation of characters from license plate for faster detection of region of interest and probabilistic neural networks for recognition of alphanumeric characters. They tested a segmentation algorithm using 1334 images, where 1287 images were segmented properly which gives an accuracy of 96.5%. The authors reported a recognition rate of 89.1% with an overall success rate of 86.0%.

Much of the previous research in ALPR systems employs conventional recognition methods while we use deep learning techniques. Yann LeCun [3] performed some of the earliest work in the field of deep learning. He developed and applied Convolution Neural Networks to document analysis. He used a 7 layer CNN called LeNet-5 for the digits (0-9) recognition, takes 32×32 pixel image as input and resulted 0.35% error rate. These results were compared with other classifiers and are illustrated in the Fig. 2.2.

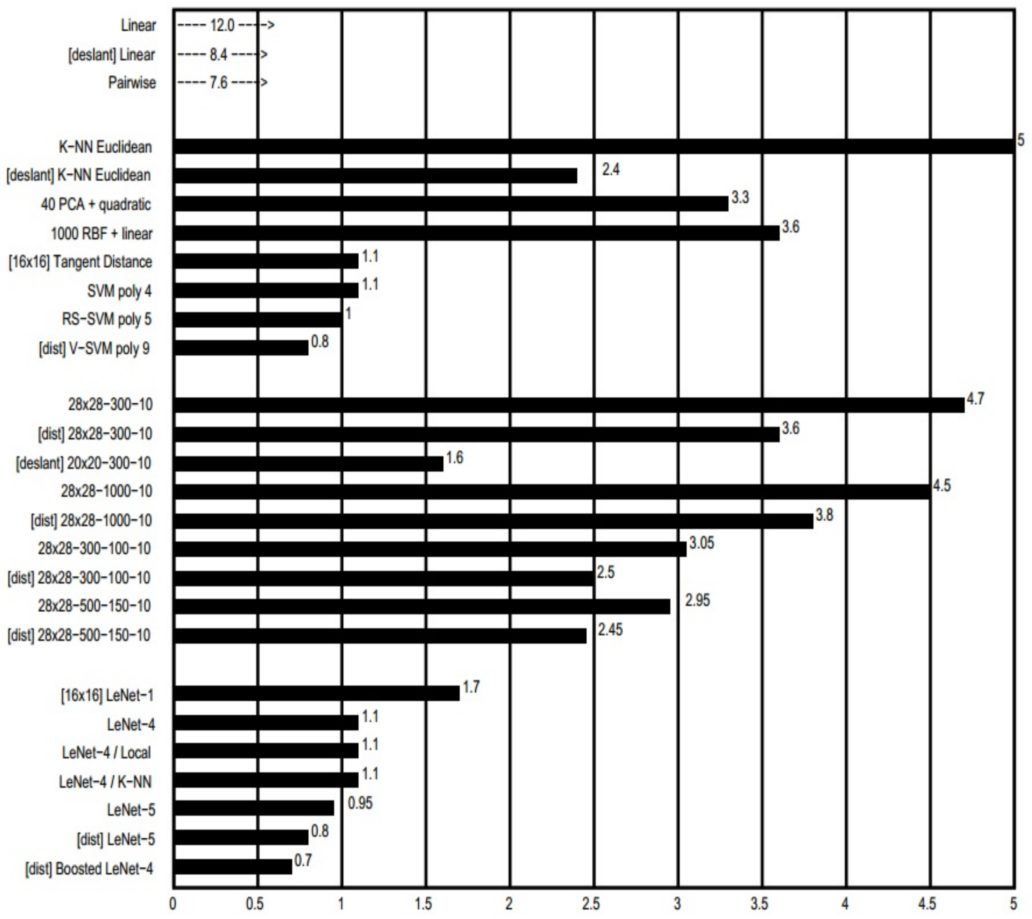


Figure 2.2: Error rate on the test set (%) for various classification methods (Anagnostopoulos et al. [3])

The study in Fig. 2.2 compares the test-set error obtained from linear classifier, Principal Component Analysis, nearest neighborhood classifier, radial basis function network, one-hidden layer fully connected Multilayer Neural Network, LeNet-5 etc.

The linear classifier obtains an error of 12% and on deslant data the test error rate is 8.4%. Deslant indicates that the classifier was trained and tested on deslanted version of the database. [dist] indicates that the training set was augmented with distorted example. The Principal Component Analysis and polynomial classifier obtains a 3.3% on regular testset. In the same way, regular test set error rate for Nearest Neighbor classifier 5%, Radial Basis Function Network- 3.6%, One-Hidden Layer Fully Connected Multilayer Neural Network- 4.7%, Two-Hidden Layer Fully Connected Multilayer Neural Network-3.05% for 28×28 pixel input image, Lenet-1 contribute 1.7% test-set error, Lenet-4 1.1%, Boosted LeNet-4 provide testset error rate of 0.7%. This study shows deep learning techniques provides great results in the field of patter classification.

Menoti [18] used a convolutional neural network for feature extraction and support vector machine for classification of characters. He achieved recognition rate of 98% for digits and 96% for characters. However the proposed work was limited to Brazilian license plates and alphanumeric character recognition.

Chen [19], used convolutional neural networks on Chinese license plates and achieved an accuracy of 98%. Razdi [20] used CNN for character recognition and with an accuracy of 98.79% on Malaysian license plates. He used 33 classes instead of 36 includes '$0-9$' numbers and 'A-Z' alphabets, with characters 'I', 'O' and 'Z' removed as they are very uncommon on Malaysian license plates.

Chapter Three

License Plate Segmentation

We now cover extracting the number and state from the license plate. Most of the work published on automatic license plate recognition initiated in countries where there are standards or strict rules over license plate design. As our research focuses on American license plates, most of this work is not applicable especially in state recognition. The main difficulty in extracting information from American license plates is that there are no standard design rules for the license plate and thus the appearance varies from state to state. Some other problems for the ALPR system include:

1. **Illumination:** due to environmental effects such as headlights lighting, rains etc, the illumination of the input image varies. Fig. 3.1 shows an image with poor illumination.

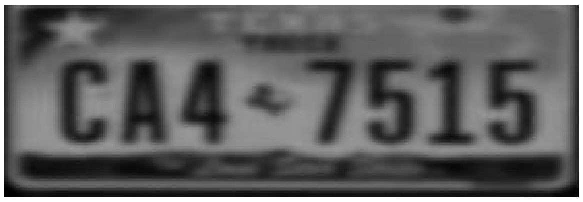


Figure 3.1: Degraded license plate with bad illumination

2. **Background** License plate backgrounds may contain designs or patterns which are difficult to separate from the foreground of the image as shown in Fig. 3.2.

3. **State location:** The location of the state ID in US license plates varies from state to state. This makes it difficult to generalize the system and requires more computation. Fig. 3.3 and Fig 3.4 illustrates the variety of state locations.

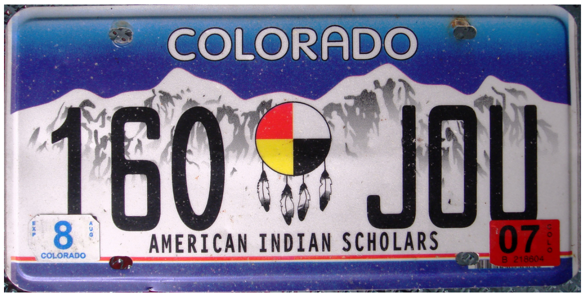


Figure 3.2: License plate with colorful background

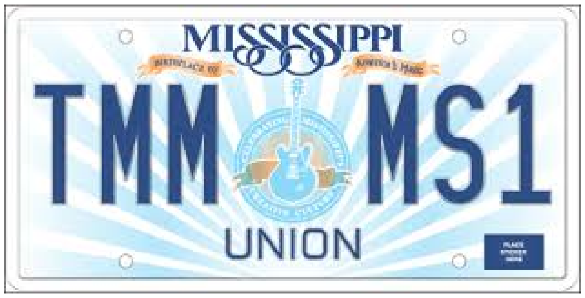


Figure 3.3: License plate where state is located at upper portion of image

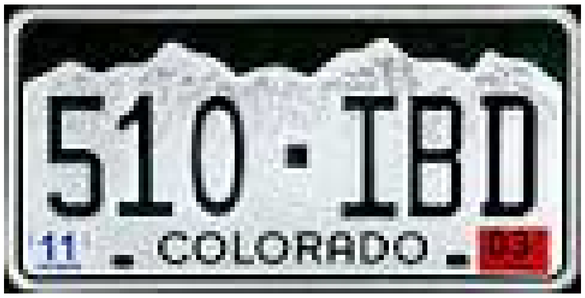


Figure 3.4: License plate where state is located at the lower portion of image

4. **Miscellaneous:** Presence of designs, shadows, unwanted symbols etc. Fig. 3.5 shows a license plate with shadow which makes segmentation difficult. Fig. 3.6 contains unwanted symbols in between license numbers which also affects segmentation accuracy.

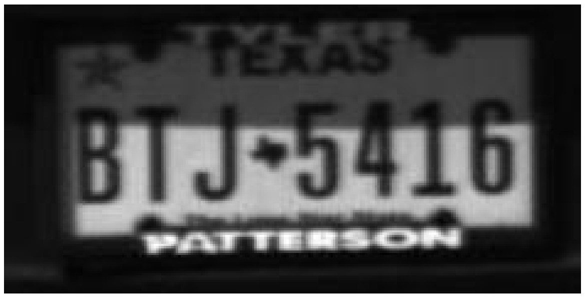


Figure 3.5: License plate image degraded by shadow

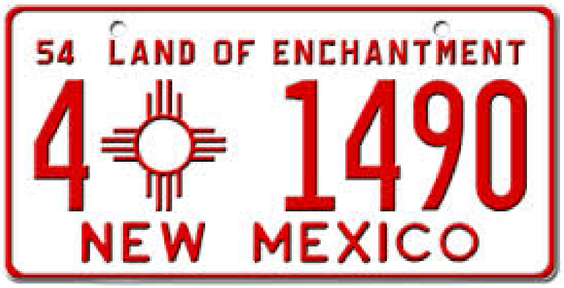


Figure 3.6: License plate containing unwanted symbols

Considering all the above problems and previous research done in the field of ALPR, our proposed system uses multiple binarization techniques. Multiple binarization techniques help to obtain efficient results in the extraction of state and license number in the license plate. The extraction of license plate information is divided into four stages:

1. Preprocessing the license plate
2. Binarization
3. Separation of state and license number
4. Extraction of license plate

3.1 Preprocessing of license plate

Preprocessing helps to enhance the visual appearance of the license plate and removes the noise or other unwanted distortion. Preprocessing is generally an important step in image processing systems, and it helps computationally in further stages of the system. We use anisotropic diffusion and histogram equalization to achieve a higher quality segmentation.

3.1.1 Anisotropic diffusion

Anisotropic diffusion (AD) suggested by Perona and Malik [21] aims at lowering the image noise without removing significant information from the image such as edges, lines or other details. Anisotropic diffusion is one of the pioneering works in reducing the noise using by partial derivatives. AD smooths the texture successfully without degrading the boundaries and small structures within the image. The gradient threshold parameter and stopping parameters behavior of anisotropic behavior. Unlike conventional smoothing filters, anisotropic diffusion is like an adaptive techniques which smooths the image inside a region, but leaves the boundaries or untouched edges. Let (x, y) be a pixel coordinate, if (x, y) is a part of edge pixel then AD will apply low smoothing at that coordinate, if not an edge pixel then AD applies high level of smoothing which exhibits the level of smoothing based on the pixel values here the level of smoothing of an image varies from pixel to pixel. Anisotropic diffusion has been used in many applications to enhance image for better results [22].

The basic diffusion equation provided in [21] is

$$\frac{\partial I(x,y,t)}{\partial t} = div[g(||\nabla I(x,y,t)||)]\nabla I(x,y,t) \tag{3.1}$$

Where t is time parameter, $I(x,y,t)$ is the input image, $\nabla I(x,y,t)$ is the gradient of image at time t, $g(x)$ is the conductance function. The function $g(x)$ is chosen to satisfy

$$\lim_{x \to 0} g(x) = 1 \tag{3.2}$$

where the diffusion is maximal within the region and

$$\lim_{x \to \infty} g(x) = 0 \tag{3.3}$$

which means the diffusion is minimal at edges. Two such functions proposed by Perona and Malik are

$$g_1(x) = e^{-(\frac{x}{K})^2} \quad \text{and} \quad g_2(x) = \frac{1}{1 + (\frac{x}{K})^2} \tag{3.4}$$

where K is the threshold parameter which controls the rate of diffusion and serves as a soft threshold between image gradients that are attributed to noise and those attributed to edges. The discrete function of anisotropic function is given as

$$I_{t+1}(s) = I_t(s) + \frac{\gamma}{\beta_s} \sum_{p \epsilon \beta_s} g_k(|\nabla I_{s,p}|)\nabla I_{s,p} \tag{3.5}$$

where I is the discretely sampled image, s denotes the position of pixel in 2D discrete grid, g is the conductance function and K is the gradient threshold parameter. $\gamma \in [0,1]$ and β_s represents the special 4-pixel neighbor pixels $s : \beta_s = N, S, W, E$, where N,S,W,E are North, South, West and East direction of pixel at position s.

Malik and Perona [22] gave two conductance functions, g_1 favors in high contrast edges over low-contrast one and g_2 favors wide regions over small ones. The gradient function is controlled by a parameter κ. Low value of κ gives small intensity gradients which block conduction and diffusion across step edges. κ values have been chosen as 5 [22] and after a few experiments. Larger values of κ lower the intensity gradients on conduction, gamma controls the diffusion maximum value is 0.25 [22] and step is used to scale the gradients in case the spacing between the adjacent pixels differ in x and y axes.

3.1.2 Histogram equalization

Histogram equalization improves the contrast of the input image. This is important to ALPR because it helps in enhancing the image when most of the pixels in the image are confined to region [23]. Basic histogram equalization uses global contrast which is not a good one for all the images.

Let the image be of $N \times M$ pixels i.e. N rows and M columns, G be the number of gray levels in the image. The histogram equalization procedure is as follows:

1. Compute the histogram for the given input image. The histogram value or probability of occurrence intensity level k is given by

$$p_r(k) = \frac{n_k}{M \times N} \tag{3.6}$$

n_k is the number of pixels having intensity k, $k = 0, 1, 2, ..., G - 1$. The plot of $p_r(k)$ vs k gives the histogram of the input image.

2. The next step is to compute the transform function and the transfer function calculates the output pixel value for a given input pixel intensity , it is obtained as follows

$$o_k = T(k) = (G-1)\sum_{i=0}^{n} p_r(i) \tag{3.7}$$

   Where o_k is the output pixel value. The transformation function $T(k)$ input gray scale level to output gray scale level, and the function also called as histogram equalization.

3. The output image is obtained by transforming each pixel in the input image using the obtained transform function, where the pixels in the obtained image distributed uniformly.

Histogram equalization is done globally. This improves the contrast in the image but also lose some information. Unlike histogram equalization the adaptive method divides the image into small blocks where the size of the block is selected one, computes histogram and transformation function at each block and finally calculates the transformed pixel values therefore improves local contrast. In the homogeneous regions of the image the histogram will be strongly peaked and the obtained transformed function will distribute the narrow range pixels to the whole range of image, which results in amplifying small amounts of noise of the image [24]. To overcome the limitation of adaptive histogram equalization $\big($AHE$\big)$ the contrast limiting method has to been applied for each region and from which transformation function is obtained. This is known as Contrast Limited Adaptive Histogram Equalization $\big($CLAHE$\big)$ [24], which is an improved version of adaptive histogram equalization techniques. CLAHE has been developed for medical imaging especially for enhancement of low contrast images. Degraded license plate images can share the same qualities.

The contrast amplification for a pixel is proportional to slope of transformation function, which is proportional to cumulative distribution function (CDF) and therefore to the value of the histogram at that pixel value. In CLAHE, before computing CDF the predefined clipping value limits the amplification of histogram which further limits the slope of CDF and therefore the transformation function.

3.2 Binarization

Binarization is the process of converting a gray scale image to a binary image. The image consists of pixel values 0 or 1, where 0 indicates black and 1 or 255 indicates white. Binarization is an important step in improving the quality of extraction of license number and state portion from the number plate. Proper binarization method is necessary for separating the foreground and background pixels in the image. Primary step in binarization is selection of optimal threshold value. A pixel with value less than threshold is classified as background pixel and with value greater than as foreground pixel. For an image $I(x, y)$ where (x, y) is location of pixel, let t be threshold then the binary image is given as

$$O(x, y) = 0 \quad \text{if} \quad I(x, y) < t \tag{3.8}$$

$$O(x, y) = 1 \quad \text{or} \quad 255 \quad \text{if} \quad I(x, y) > t \tag{3.9}$$

A number of factors such as ambience light, gray level variance between foreground and background contrast make the thresholding scheme difficult. If the wrong threshold is selected foreground and background pixels can be misclassified.

The extraction of license number depends on the quality of binarized image. Selection of best binarized image provides overall accuracy of the system. In our system we consider a hybrid binarization method for better segmentation of characters or license numbers. As the images are captured in different environments, considering a single binarization technique is not effective, hence we consider multiple binarization techniques. In this hybrid technique, we apply five binarization methods to the input license plate and select the best method depending on the quality of results obtained using the selection criteria. The technique will be further explained in sections below. In every binarization method computation of threshold plays a key role and it will be explained in each procedure. Prior to binarization the license plate image in our system needs to be converted to 128 rows and 256 columns.

3.2.1 Global thresholding using percentage of foreground area

The global binarization technique is one of the simplest binarization methods. It selects a threshold value to classify a pixel of an image as background or foreground [23]. The threshold value is based on the required percentage of background pixels. It is calculated for the license number portion of the image rather than entire image. License number portion is approximately obtained by removing the margin from the top and bottom part license plate image. The margin is considered to be approximated 15% of the height of the image after several experiments. License number portion is only considered for binarization because selection of best techniques is based on number of charcaters objects available in the binarized image. Removing of margins also reduces the cost of computation. The main advantage of global thresholding is less computations. The pseudocode is as follows

Algorithm 1 Pseudocode for Global Thresholding

```
1: Import an input image $I$ of size $m \times n$ and the pixels range is $[0, L-1]$, where $L$
   is the maximum gray scale value.
2: Initialize the percentage number of total pixels in image required to be background
   area, let it be $background$ where $background \in [0, 1]$.
3: $pixels \leftarrow m \times n \times background.$
4: $sum \leftarrow 0$
5: for $i \leftarrow 0, L-1$ do
6:     $bin[i] \leftarrow 0$
7: end for
8: for $i \leftarrow 0, m-1$ do
9:     for $j \leftarrow 0, n-1$ do
10:        $temp \leftarrow I[i][j]$
11:                $bin[temp] = bin[temp] + 1$
12:    end for
13: end for
14: while $i \leq L-1$ and $sum \leq pixels$ do
15:     $sum \leftarrow sum + bin[i]$
16:         $threshold \leftarrow i$
17:         $i \leftarrow i + 1$
18: end while
19: for $i \leftarrow 0, m-1$ do
20:     for $j \leftarrow 0, n-1$ do
21:         if $I[i][j] \geq threshold$ then
22:             $I[i][j] \leftarrow 1$
23:         else
24:             $I[i][j] \leftarrow 0$
25:         end if
26:     end for
27: end for
```

3.2.2 Sauvola algorithm

The global thresholding techniques computes the threshold value for the entire image which works well for simple cases. The global techniques fail if the image is degraded one, image consists complex backgrounds or poorly illuminated. Unlike global thresholding, Sauvola [25] method of binarization is adaptive where the threshold value, $T(x, y)$ at pixel location (x, y) is computed using the mean and standard deviation at the same location within a window of size $w \times w$ as

$$T(x, y) = m(x, y)\left[1 + k\Big(\frac{\sigma(x, y)}{R - 1}\Big)\right] \tag{3.10}$$

Where $m(x, y)$ is the mean, $\sigma(x, y)$ is the standard deviation at location (x, y), R is the maximum value of standard deviation $\big(R = 128$ for a grayscale document$\big)$, k is a bias which takes positive values in the range $[0.2, 0.5]$.

As previously mentioned, the Sauvola algorithm is adaptive as it calculates the threshold according to the contrast in the neighborhood pixels. In high contrast locations $\sigma(x, y) \approx R$ and $T(x, y) = m(x, y)$. In low contrast images the threshold goes below the mean value which removes the relatively black regions in the background. The parameter k controls the threshold value in the local window. When the k value is higher, the threshold value in local window is lower than mean value. Experiments shows $k = 0.34$ gives best results. We have chosen the value of R to be equal to the maximum value of deviation obtained in the local window of size 40×10 and $k = 0.34$. The parameters are chosen by experiments and give optimal results for Savoula binarization.

3.2.3 Otsu algorithm

The Otsu algorithm [23] is a statistical decision theory problem whose objective is to minimize the error incurred in assigning pixels to foreground and background or classes. The method maximizes the variance between the foreground and background pixels in the image [23]. Otsu method computations entirely depends on the histogram of an image. Otsu's thresholding method involves iterating through all the possible threshold values and calculating a measure of spread for the pixel levels each side of the threshold, i.e. the pixels that either falls in foreground or background. The aim of Otsu method is to compute the threshold value where the sum of foreground and background spreads to its minimum. The pseudo code is as follows

Algorithm 2 Pseudocode for Otsu algorithm

```
1: Import the input image.
2: Compute the normalized histogram of the input image. Denote the components
   of the histogram by $p_i, i = 0, 1, 2, ..., L-1$, where $L$ is the maximum gray scale
   value in the image.
3: Compute the cumulative sum $P_i(k)$ for $k = 0, 1, 2, ..., L-1$ using equation $P(k) = \sum_{i=0}^{k} p_i$.
4: Compute the cumulative means $m(k)$ for $k = 0, 1, 2, ..., L-1$ using equation
   $m(k) = \sum_{i=0}^{k} i \times p_i$.
5: Compute the global intensity $m_G$, using the equation $m_G = \sum_{i=0}^{L-1} i \times p_i$.
6: Compute the variance $\sigma_b^2(k)$ where $k = 0, 1, 2, ..., L-1$ using equation
   $\sigma_b^2 = \frac{[m_G \times P_1(k) - m(k)]^2}{(P_1(k)[1 - P_1(k)])}$.
7: Obtain the Otsu threshold, $k^*$, as the value for which $\sigma_b^2(k)$ is maximum. If the
   maximum value is not unique, obtain $k^*$ by averaging the values of $k$ correspond-
   ing to the various maxima detected.
8: Obtain the separability measure, $\eta^*$, by evaluating equation $\eta(k) = \frac{\sigma_B^2(k)}{\sigma_G^2}$ at
   $k = k^*$ where $\sigma_G^2 = \sum_{i=0}^{L-1} (i - m_G)^2 \times p_i$
9: Binary image is obtained by assigning each pixel value in the input image less
   than $k^*$ value as background pixel and pixel value greater than threshold value
   as foreground pixel in the image.
```

3.2.4 Edge based binarization

Edge detection has been used in various vision applications, for instance the Canny edge detector has been used in order to find the edges for the input image and also reduces the amount of data to be processed. The Canny algorithm [23] satisfies the following criteria:

1. *Low error rate:* It gives a low error rate which means it detects all the edges in the image.
2. *Good localization:* The distance between the real edge pixels and detected edge pixels have to be minimized.

3. *Minimal response:* Only one detector response per edge.

```
Algorithm 3 Canny edge detection algorithm
1: Import the input image.
2: Apply Gaussian filter to remove noise from the image. Before locating the edges.
3: Find the intensity gradients of the image by applying the following convolution pairs at each pixel location in the image
       $G_x(i,j) = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$
       $G_y(i,j) = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$
   Where $(i,j)$ is the location in the image where the above operation takes place.
4: Compute the gradient value and angle using the following equations at each location in the image.
       $G(i,j) = \sqrt{G_x^2(i,j) + G_y^2(i,j)}$
       $\theta(i,j) = tan^{-1}\Big(\frac{G_y(i,j)}{G_x(i,j)}\Big)$
5: Non-Maximum suppression is applied to remove the pixels which are not part of an edge, results in thin lines at edges.
6: The final step in Canny edge detector is to use two thresholds i.e. upper and lower thresholds
   1. If the pixel gradient is higher than the upper threshold it is accepted as an edge.
   2. If the pixel gradient is below the lower threshold, then it is rejected.
   3. If the gradient is between the upper and lower threshold then it will accepted as an edge only if it is connected to the pixel whose gradient is above the threshold.
```

Then the obtained image and original image are divided into regions of size 10×5. The block size 10×5 is found to be best after testing various images for different block sizes. This block size helps in existence of partial characters in the given block, which confirms the existence of edges in most of the block. The blocks or regions are used to calculate the local threshold. The threshold for a region or block is obtained using gray scale values from the original image and neighboring pixels of all the edges. The threshold value for each block is calculated using algorithm 4. The obtained binary blocks are then combined to create complete binary image.

3.2.5 Median of images

The median is the center value of a sorted list of elements. If there are even number of elements in the list, the median will be the average of the two middle values. This helps us avoid considering the values that are spurious or outside the range. In image processing stacking considers all the pixel values to be the same location in a stack of images. Then choosing the median value from the stacked images at each location gives the resultant image pixel values. The implementation is provided in algorithm 5.

Algorithm 5 Pseudocode for median of images

```
1: Import the binary images obtained from global threshold, Sauvola algorithm,
   Otsu algorithm and edge based binarization, let them be $I_1$, $I_2$, $I_3$ and $I_4$.
2: let size of the image be $m \times n$.
3: let the output image be $O$ of size $m \times n$
4: for $i \leftarrow 0, m-1$ do
5:     for $j \leftarrow 0, n-1$ do
6:         $O[i,j] = median\{I_1[i,j], I_2[i,j], I_3[i,j], I_4[i,j]\}$
7:     end for
8: end for
```

In this research the binary images obtained by above algorithms are considered as stack of images i.e. Otsu, Global thresholding, Sauvola algorithm and Edge based binarization techniques. Median stacking is a powerful method in producing clean and noise free images. It works well because noise is random and also by combing multiple images, it can improve signal to noise ratio also. Median of stack images also helps in removing unwanted or accidental objects from a series of stationary photos or video frames.

3.2.6 Selection of best binarization technique

Because one binarization technique does not provide efficient results, we consider multiple binarization techniques. Considering multiple binarization techniques helps in segmenting the plate regardless of the environment in which plate was captured. After successfully binarizing the plate using various techniques, the next step is to

Algorithm 4 Pseudocode for edge based binarization

```
1: Import the input image $I_1$.
2: Apply edge based segmentation using canny edge detector on the Input image $I_1$,
   let the segmented image be $I_2$.
3: Divide $I_1$ and $I_2$ into blocks of size $10 \times 5$. Blocks obtained from $I_1$ are represented
   as $I_i[r, c]$ and blocks form $I_2$ are represented by $E_i[r, c]$ where $i$ is the block
   number, $[r, c]$ is the location in the respective block.
4: let $n$ be the total number of blocks.
5: $i \leftarrow 1$
6: repeat
7:     Creating an empty array for holding local threshold values $t_1 = \varnothing$
8:     $rows$ = number of rows in $I[i]_i$
9:     $cols$ = number of columns in $I[i]_i$
10:    for $r \leftarrow 3, rows - 2$ do
11:        for $c \leftarrow 3, cols - 2$ do
12:            if $E_i[r, c] == 255$ and $E_i[r, c-1] == 0$ and $E_i[r, c+1] == 0$ then
13:                $m = minimum(I_i[r, c-2], I_i[r, c+2])$
14:                if $m < 128$ then
15:                    $t_2 = mean(I_i[r, c-2], I_i[r, c+2])$, append $t_2$ to $t_1$
16:                end if
17:            end if
18:            if $E_i[r, c] == 255$ and $E_i[r-1, c] == 0$ and $E_i[r+1, c] == 0$ then
19:                $m = minimum(I_i[r-2, c], I_i[r+2, c])$
20:                if $m < 128$ then
21:                    $t_2 = mean(I_i[r-2, c], I_i[r+2, c])$, append $t_2$ to $t_1$
22:                end if
23:            end if
24:            if $t_1$ is empty then
25:                $localthreshold = 0$
26:            else
27:                $localthreshold == mean(t_1) - variance(t_1)$
28:            end if
29:            for $r \leftarrow 0, rows - 1$ do
30:                for $c \leftarrow 0, cols - 1$ do
31:                    if $I_i[r, c] < localthreshold$ then
32:                        $BinaryImage_i[r, c] = 0$
33:                    else
34:                        $BinaryImage_i[r, c] = 1$
35:                    end if
36:                end for
37:            end for
38:        end for
39:    end for
40: until i==n
41: The final binary image is obtained by combining all the individual $BinaryImage_i$.
```

choose the best binarized output plate.

To choose the best technique, prior knowledge of the characters is required, including height, width and aspect ratio of characters. After examination of several images, we found that the height of the characters should be approximately be 20% of the image and maximum height of the image should be 70% of the image height dimensions and aspect ratio equals to 0.5.

To measure the properties of characters in the binarized images, we use contouring techniques. The contours of a region $\mathcal{R}$ are the set of pixels that are adjacent to the pixels in the complement of $\mathcal{R}$ [23]. The contours technique extract characters from the binary image and measures the properties of each character found on the license plate portion. Using contours the height, width, aspect ratio and center coordinates of the objects in the binarized image can be computed. Features obtained from the contours helps in computing the number of characters in the binarized image that satisfy the height, width, aspect ratio and also determine whether the characters are near to the center. Initially the image having most characters satisfying the assumptions is considered as best binarization technique. If more than one image has same number of characters then the best technique will be selected using the number of foreground to background pixel ratio. The image which has more foreground to background pixel count ratio from the images which has same number of characters is considered as the best method. The foreground to background pixel ratio helps in selecting an image with fewer artifacts. Fig. 3.7 and Algorithm 6 clearly illustrates the procedure. Images 1-5 in Fig. 3.9 show the resultant images obtained from 5 binarization techniques for the input image shown in Fig. 3.8. If we apply contours technique and find the number of objects that satisfy the assumptions, we will get equal number of objects in all the binarized images, as the characters are not overlapping. However if we observe the edge based binarized image contains fewer artifacts, considering the foreground to background pixel count ratio helps in selecting edge based binarized images as better one among the available.

Algorithm 6 Pseudocode for selection of best binarization technique

```
1: Import the binary images obtained from global threshold, Sauvola algorithm,
   Otsu algorithm, edge based binarization and median of stack of images. let them
   be $I[0]$, $I[1]$, $I[2]$, $I[3]$, $I[4]$.
2: $max \leftarrow 0$
3: $max\_ratio \leftarrow 0$
4: $best \leftarrow 0$
5: $aspect\_ratio \leftarrow 0.5$
6: $letter\_height\_minimum = plateheight * 0.3$
7: $letter\_height\_maximum = plateheight * 0.7$
8: $i \leftarrow 0$
9: for $i \leftarrow 0, 4$ do
10:     $image \leftarrow I[i]$
11:     Find the contours in $image$ and a list of their bounding rectangles.
12:     for $contour$ in $contours$ do
13:         Compute the coordinates, height and width of each contour.
14:         if $height > letter\_height\_maximum$ then
15:             if $height < letter\_height\_minimum$ then
16:                 if $height \times aspect\_ratio - width > height \times 0.25$ then
17:                     Compute the center of each contours using parameters ob-
   tained in step 14.
18:                 end if
19:             end if
20:         end if
21:         Remove the objects which does not satisfy the above conditions.
22:         Remove the objects that are not nearly available on a horizontal line.
23:         $num\_letters[i] \leftarrow$ count of objects after removing unwanted objects
24:     end for
25: end for
26: $count \leftarrow$ number of images having maximum number of characters and also save
   their $i$ value.
27: if $count == 1$ then
28:     Best binarized image is $i$
29: else if $count > 1$ then
30:     Calculate foreground pixel to background pixel ratio, $r(i)$
31:     Best binarized image is $i$, where $i$ gives the images having maximum fore-
   ground to background pixel ratio.
32: end if
```

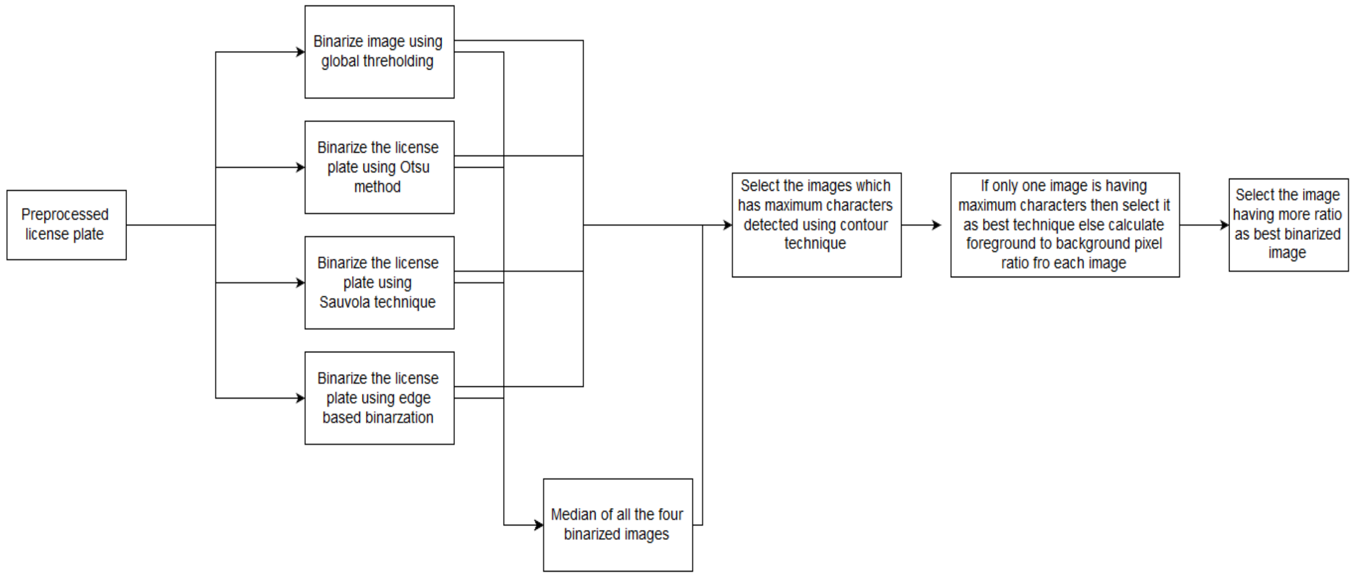

Preprocessed license plate
Binarize image using global threholding
Binarize the license plate using Otsu method
Binarize the license plate using Sauvola technique
Binarize the license plate using edge based binarzation
Median of all the four binarized images
Select the images which has maximum characters detected using contour technique
If only one image is having maximum characters then select it as best technique else calculate foreground to background pixel ratio fro each image
Select the image having more ratio as best binarized image

Figure 3.7: Block diagram of hybrid binarization technique

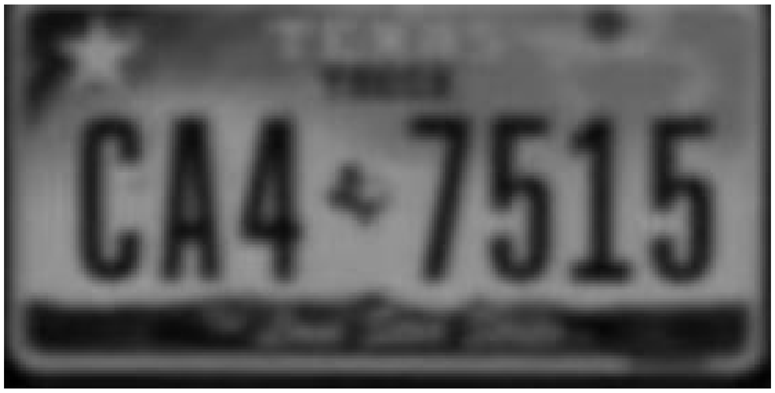


Figure 3.8: Input image

3.3 Extraction of state and license number

For simplicity we assume that the state is available in the upper portion of the license plate. So the first step in our system is separation of license number and state portion of the plate. To locate and extract the state and license number, we use horizontal histogram projection. The horizontal projection calculates the sum of pixels along the horizontal direction. The summation of all the pixels along each row gives horizontal projection values. We also use vertical projection to extract characters from the license plate candidate, vertical projection obtains the coordinates of the characters. The horizontal $H(y)$ and vertical projection $V(x)$ for an image I of dimension $M \times N$ is given as:

$$H(y) = \sum_{x} I[x, y] \tag{3.11}$$

$$V(x) = \sum_{y} I[x, y] \tag{3.12}$$

The horizontal projection profile helps in computing the row numbers in which the license plate portion exists. Vertical projection helps to extract characters independent of characters positions and able to deal with some rotations [16]. In order to achieve fast computation and less memory usage during horizontal and vertical projection procedure the image is converted to more compact version called the Skeleton. The skeleton of an image preserves the structure of object but removes all redundant pixels as shown in Fig. 3.10. Fig. 3.11 displays the extracted state information from the license plate and Fig. 3.12 displays the extracted license number from the license

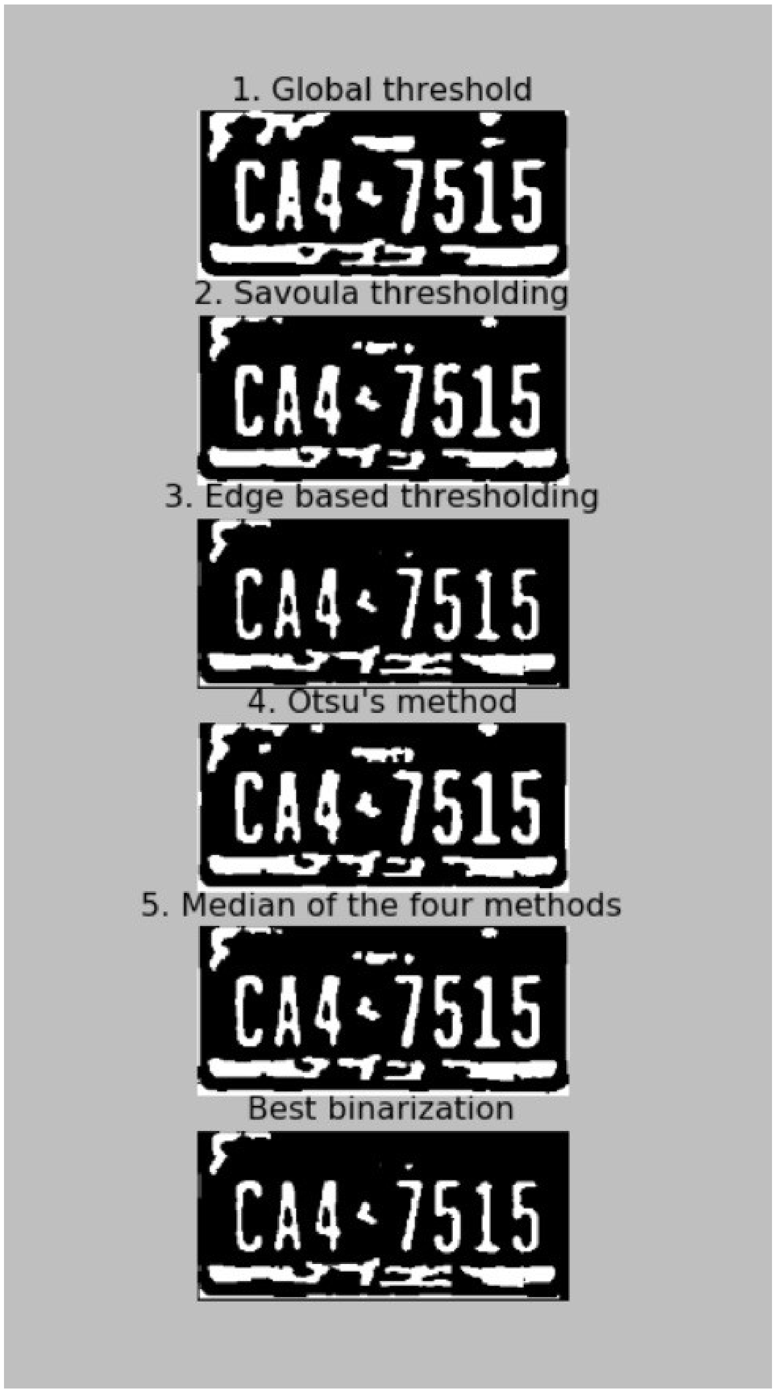


Figure 3.9: Plots 1-5 display the binarized images for various techniques. The last plot displays the best chosen technique

plate using horizontal projection. The pseudocode for horizontal projection method is described in Algorithm 7.

Algorithm 7 Pseudo code for extraction of state and license number using horizontal projection

```
1: Let the binary image be $I$
2: Skeletonize the image using morphological operations.
3: $r \leftarrow$ number of rows in the image.
4: $c \leftarrow$ number of columns in the image.
5: for $i \leftarrow 0, r-1$ do
6:     for $j \leftarrow 0, c-1$ do
7:         $Aray[i] = I[i][j] + Array[i]$
8:     end for
9: end for
10: $window \leftarrow [0.33, 0.33, 0.33]$
11: Convolution of sum with window, $Array \leftarrow Array * window$
12: $3 \times 1$ column matrix with all the values of matrix equals to 0.33.
13: $Z_1 \leftarrow Array[0 : r/2]$
14: $Z_2 \leftarrow Array[r/2 : r-1]$
15: $R_{start} \leftarrow$ index of array $Z_1$ having minimum value from the end. $R_{start}$ is the row
    number where license number starts.
16: $R_{end} \leftarrow$ index of array $Z_2$ having minimum value from index 0. $R_{end}$ is the row
    number where license number ends.
17: $license\_number \leftarrow I[R_{start} : R_{end}]$
18: $State \leftarrow I[0 : R_{start}]$
```

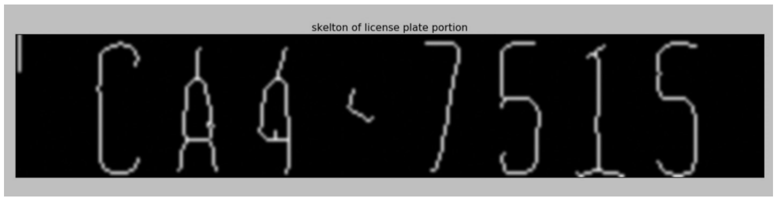


Figure 3.10: Skeletonized result of license number portion

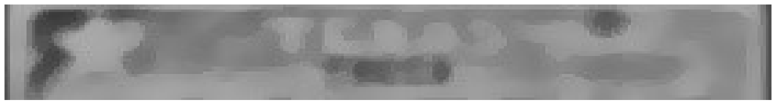


Figure 3.11: Extracted state information from the license plate which shows 'Texas'

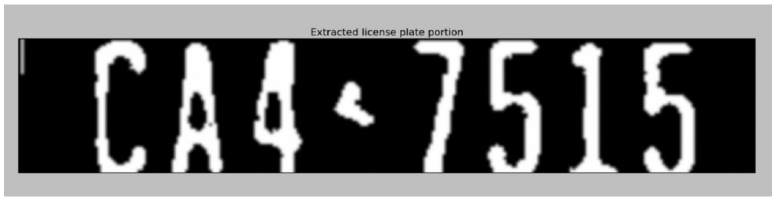


Figure 3.12: Extracted license number from the license plate

The license plate information is available between the rows R_{start} and R_{end} whose pixel values are selected from the binary image. As per the assumptions the state information exists from the first row to R_{start} from the input image i.e. gray scale image. The characters are in binary format and the state information is in gray scale format. In order to extract characters the license plate region obtained from the above operation need to be input for the character extraction operation. The pseudo code for vertical projection is described in algorithm 8 and the results are shown in Fig. 3.13

Algorithm 8 Pseudocode for extraction of characters from the license number portion using vertical projection

```
 1: Let the license number portion be $I$, where $I \leftarrow binary_image[R_{Start}, Rend]$.
 2: Normalize the image, i.e. convert the pixel value range to $[0, 1]$.
 3: $r \leftarrow$ number of rows in image $I$
 4: $c \leftarrow$ number of cols in image $I$
 5: Skeletonize the image using morphological operations.
 6: for $i \leftarrow 0, c - 1$ do
 7:     for $j \leftarrow 0, r - 1$ do
 8:         $Array[i] = I[i][j] + Array[i]$
 9:     end for
10: end for
11: Find the minimum values in Array and save their index values into array $cols$.
12: $size \leftarrow lengthcols$
13: for $i \leftarrow 0, size - 2$ do
14:     if $cols[i + 1] - cols[i] \geq 9$ then
15:         $Binary\_Object = I[:, cols[i] : cols[i + 1]]$
16:     end if
17: end for
```

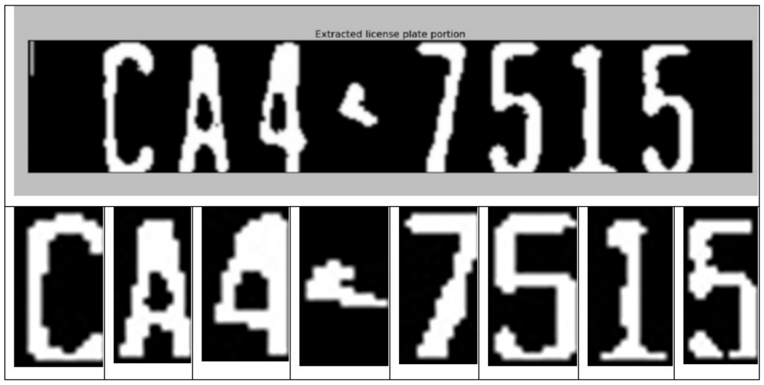


Figure 3.13: Segmented characters from license number portion

Chapter Four

Convolution Neural Networks

The Convolutional Neural Network (CNN), first proposed by LeCun [3] in 1988, is a neutral network model incorporating the following ideas: receptive fields, weight sharing and subsampling. It is a special type of multilayer perceptron trained in supervised mode using backpropagation. It is one of the most successful machine learning architectures in computer vision and has achieve state-of-the-art results in tasks as character recognition [3], object recognition, face detection [26] and pose estimation, speech recognition, license plate recognition [27–29], image preprocessing and segmentation tasks.

A CNN learns complex, high dimensional features from a large number of examples which makes it an obvious candidate for pattern recognition tasks. CNN architectures [3] ensure some degree of shift, scale and distortion invariance using some of the features such as local receptive fields, subsampling, shared weights etc. Unlike conventional pattern recognition tasks one of the benefits of CNN's lie in extraction features itself and which uses images as input for training, testing the network. The CNN builds complex features from large collection of examples and the complexity of learning features with more layers. This is done by successively convolving the input image with filters to build up a hierarchy of feature maps. The hierarchy results in complex features and learning, as well as translational and distortion invariant. The whole CNN can be expressed as a score function where raw image pixels are given as input on one end and determine the category or class score at the other end.

4.1 Convolution

Convolution is an operation on two signals or two 2D images and a critical concept in many areas of mathematics and engineering. Convolution helps in extracting information from the image. Two properties of convolution are that it is shift invariant and it is a linear operation.

To illustrate 2D convolution, let the image be $x[n_1, n_2]$ of $N \times N$ points and kernel be $h[n_1, n_2]$ of $M \times M$ points where

$$\begin{aligned} x[n_1, n_2] = 0 \quad &\forall \quad n_1 < 0 \quad \text{and} \quad n_1 > N-1 && (4.1) \\ &\forall \quad n_2 < 0 \quad \text{and} \quad n_2 > N-1 \\ h[n_1, n_2] = 0 \quad &\forall \quad n_1 < 0 \quad \text{and} \quad n_1 > M-1 && (4.2) \\ &\forall \quad n_2 < 0 \quad \text{and} \quad n_2 > M-1 \end{aligned}$$

The convolution output $y[n_1, n_2]$ can be expressed as

$$\begin{aligned} y[n_1, n_2] &= x[n_1, n_2] * h[n_1, n_2] \\ &= \sum_{k_1=-\infty}^{\infty} \sum_{k_2=-\infty}^{\infty} x[k_1, k_2] h[n_1 - k_1, n_2, n_2 - k_2] && (4.3) \end{aligned}$$

4.2 Layers

Convolutional Neural Networks (CNNs) are often used to process 2D images to learn high dimensional, nonlinear mappings from the large set of examples. This makes them a good solution for many computer vision tasks. The CNN takes input an image and convolves it with a 2D kernel of adjustable weights. The same kernel is convolved with the input image at different points in the image, which is known as weight sharing technique. Weight sharing reduces the number of free parameters. The results of convolution are added together with an adjustable scalar called a bias. The output is then fed into an activation function which produces a 2D plane called a feature map. Convolution produces multiple feature maps whose number depends on the number of kernels which is a function of architecture. Each feature map is then connected to a subsampling layer which reduces the size of feature maps. These subsampled feature maps are passed through ReLU, local normalization layer which helps in retaining nonlinear properties. The number of 2D planes obtained after subsampling, ReLU and local normalization layer is same as number of feature maps obtained after convolution layer. The convolutional, subsampling, ReLU and normalization layers are stacked together, can be considered as block as shown in Fig. 4.1. The elements in the block can be changed depending on the application. The number of blocks depends on the depth of learning which will be discussed in further sections. In the next subsections we cover different types of layers in more detail

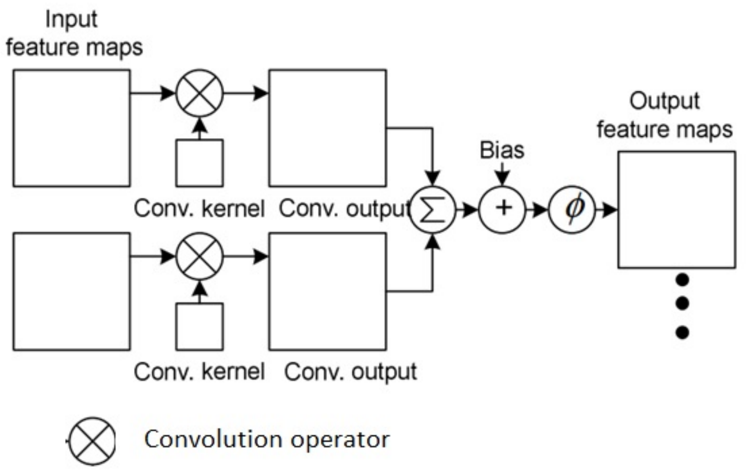


Figure 4.1: Example of Convolutional layer.

4.2.1 Convolutional layer

As previously mentioned, a convolutional layer takes an input image and convolves it with kernels to produce several two dimensional planes of neurons called feature maps. Each element of a feature map is obtained by convolving the respective kernel with units in the neighborhood in the previous layer. These outputs obtained after each convolutional layer are then summed up together with a trainable bias term which is then passed to an activation function to obtain each unit of a feature maps [30]. The input image could consist of hundreds or thousands of pixels but convolutional layers act as feature extractor to extract features such as corners, edges, endpoints or nonvisual features by convolving the input with kernels consisting of weights [3, 26]. As the weights are shared, the number of parameters to train the neural network are reduced. This also reduces the memory necessary to store these parameters during execution. The convolution operation in each convolutional layer makes a CNN translational and distortion invariant i.e. when the input image is shifted the output feature map will be shifted in the same amount as input. The number of kernels in each convolution layer depends upon the number of feature maps and varies from architecture to architecture. Fig. 4.1 illustrates the concepts in a convolutional layer

Let l be a convolution layer, $l-1$ be the previous layer or input layer which consists of feature map $m_1^{(l-1)}$. The size of the feature map is $m_2^{(l-1)} \times m_3^{(l-1)}$. If $l-1$ is an input layer then $l = 1$ and input will be an image I consisting of one or more channels and size of the image is $m_2^{(l-1)}$ rows and $m_3^{(l-1)}$ columns. The kernel used for computing a particular feature map $Y_i^{(l)}$ is the same that is $K_{(i,j)}^{(l)} = K_{(i,j)}^{(l)}$ for $j \neq k$ which explains the concept of weight sharing. The i^{th} feature map in layer l, denoted by $Y_i^{(l)}$, is computed as

$$Y_i^{(l)} = B_i^{(l)} + \sum_{j=1}^{m_1^{(l-1)}} K_{i,j}^{(l)} * Y_j^{(l-1)} \tag{4.4}$$

Where $B_i^{(l)}$ is a bias matrix and $K_{(i,j)}^{(l)}$ is the filter of size $2h_1^{(l)} + 1 \times 2h_2^{(l)} + 1$ connecting the j^{th} feature map in layer $l-1$ with i^{th} feature map in layer l [31]. When applying discrete convolution only in the so called valid region of the input feature maps, that is only for pixels where the sum of the product is defined properly, the output feature map has size

$$m_2^{(l)} = m_2^{(l-1)} - 2h_1^{(l)} \tag{4.5}$$

$$m_3^{(l)} = m_3^{(l-1)} - 2h_2^{(l)} \tag{4.6}$$

In practice the filters used for computing a fixed feature map $Y_i^{(l)}$ are the same that is $K_{(i,j)}^{(l)} = K_{(i,k)}^{(l)}$ for $j \neq k$, which demonstrates the concept of weight sharing.

To represent the above equations for convolution neural networks, each feature map $Y_i^{(l)}$ in layer l consists of $m_2^{(l)} \times m_3^{(l)}$ units arranged in a two dimensional array. The unit at position (r, s) computes the output

$$\begin{aligned} Y_i^{(l)} &= (B_i^{(l)})_{r,s} + \sum_{j=1}^{m_1^{(l-1)}} (K_{i,j}^{(l)} * Y_j^{(l-1)})_{r,s} \\ &= (B_i^{(l)})_{r,s} \sum_{j=1}^{m_1^{(l-1)}} \sum_{u=-h_1(l)}^{h_1^{(l)}} \sum_{v=-h_2(l)}^{h_2^{(l)}} (K_{i,j}^{(l)})_{u,v} (Y_j^{(l-1)})_{r+u,s+v} \end{aligned} \tag{4.7}$$

The trainable weights of the network can be found in the filters $K_{(i,j)}^{(l)}$ and the bias $B_i^{(l)}$. Each feature map in layer l is connected to one or more feature maps of the preceding layer as customized. A connection is associated with a convolution mask, with is a 2D matrix of adjustable entries called weights. The output feature maps are connected to ReLU layer individually which is discussed clearly in the next section.

4.2.2 ReLU layer

ReLU [32] and is used to model a neuron's output as a function of input, where f is the activation function used in layer l and do unit wise computation. The Rectified Linear network $\big($ReLU$\big)$ compute the function $f(x) = max(0, x)$, which increase the nonlinear properties of the overall network. The feature maps obtained from the convolutional layers enters to the ReLU layer as input. ReLU plays a key role in obtaining good performance and also trains the network faster [32]. Some of the activation functions used to increase nonlinearity in CNNs are saturating hyperbolic tangent $(tan(x), |tanh(x)|)$, sigmoid function $f(x) = \dfrac{1}{1+e^{-x}}$. The activation functions that are symmetric around origin are preferred as they results in zero-mean inputs to the following layer. Fig. 4.2 illustrates ReLU activation function

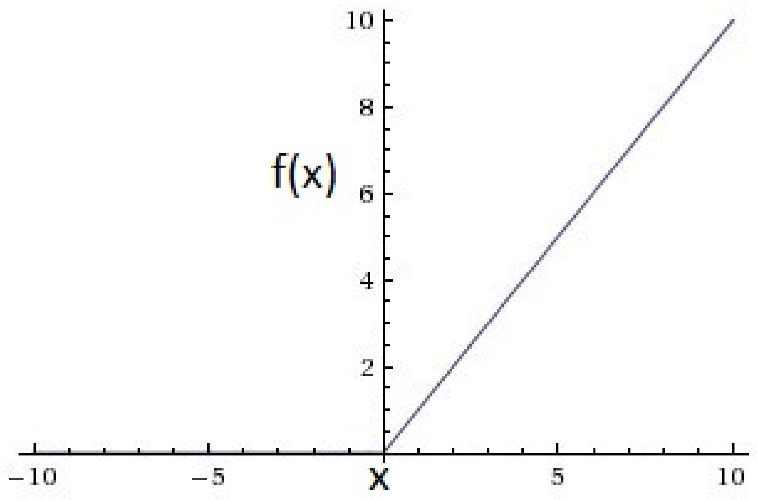


Figure 4.2: Rectified Linear Unit (ReLU) activation function, which is zero when $x < 0$ and then linear with slope 1 when $x > 0$

4.2.3 Subsampling or pooling layer

Once the features have been detected, the position or exact location of the feature is less important. Only the features relative to the neighborhood features become important. Subsampling provides a further translational invariance reduces distortions and noise. The subsampling layer reduces the spatial resolution and sensitivity to shifts and distortions. Pooling operates by placing the window at each position in the each feature map and computes one value per window for the output feature map, which results in subsampling and the window size is defined as required. Two pooling operations are defined below

Average pooling: computes the average of the values available in the window at each location in the input feature map.

Max pooling: selects the maximum values available in the window.

Let the n^{th} feature map in layer l be

2	**2**	2	2
4	**5**	4	5
2	2	2	2
4	5	4	5

Let the operation be average pooling, the feature map is divided into non-overlapping blocks of 2×2 size. The values at which the operation takes place are highlighted. The average pooling gives the output value as $\frac{2+2+3+5}{4} = 3$. The output feature map after average pooling operation is follows:

3	3
3	3

Let the operation be max pooling, then max-pooling selects the maximum value available in operating window and the ouput feature map for max pooling operation is as follows:

5	5
5	5

As discussed, the reduction in resolution reduces the number of computation and is done by averaging or selecting maximum or using any other operation applied in the neighborhood $\big($normally 2×2 $\big)$ on the available features from the previous layer. The idea is to skip the pixel position where the above discussed operation is applied, the skipping can be done vertical and horizontal direction which depends on the architecture and this skipping factor is called stride. After applying stride the output feature maps size is given by

$$m_2^{(l)} = \frac{m_2^{(l-1)} - 2h_2^{(l)}}{s_1^{(l)} + 1} \quad \text{and} \quad m_3^{(l)} = \frac{m_3^{(l-1)} - 2h_2^{(l)}}{s_2^{(l)} + 1} \tag{4.8}$$

Where $s_1^{(l)}$ and $s_2^{(l)}$ are the horizontal and vertical stride values.

4.2.4 Normalization layer

Response normalization helps in generalization of convolution neural network [32]. The local contrast normalization layer performs enforces local competitiveness between the adjacent units within a feature maps and also among units or neurons at same location in different feature maps. When a unit in a feature map fires at high activation level, normalization layer suppresses the activation of surrounding units. The activity of neuron $a^i_{(x,y)}$ is computed by applying kernel i at position (x, y), the normalized activity $b^i_{(x,y)}$, is given by

$$b^i_{x,y} = \frac{a^i_{x,y}}{\left(k + \alpha \sum_{j=max(0,i-\frac{n}{2})}^{min(N-1,i+\frac{n}{2})} (a^j_{x,y})^2\right)^\beta} \tag{4.9}$$

Where n is the number of adjacent kernel maps at the same special location, N is the total number of kernels in the layer. The values of constants k, n, α and β are determined using validation set. The normalization layer reduces the error rate, a 4-layer CNN on CIFAR-10 [32] dataset achieved 13% test error rate without normalization and 11% with normalization.

4.2.5 Fully connected layer

The final layer in CNN architecture is fully connected layer. In this layer, the units of all feature maps from the previous layer are fully connected to each unit in the current layer. The fully connected layer acts as a classifier and gives the confidence or class score to which the input image belongs. Let l be a fully connected layer, where layer l expects $m_1^{(l-1)}$ features maps and then i^{th} unit in the fully connected layer is given by

$$y_i^{(l)} = f(z_i^{(l)}) \quad \text{where} \quad z_i^{(l)} = \sum_{j=1}^{m_1^{(l-1)}} \sum_{r=1}^{m_2^{(l-1)}} \sum_{s=1}^{m_3^{(l-1)}} (W^{(l)}_{(i,j,r,s)})_{u,v} (Y_j^{(l-1)})_{r,s} \tag{4.10}$$

Where $W^{(l)}_{(i,j,r,s)}$ are the weights connecting the neuron at position (r, s) in j^{th} feature map in layer $(l - 1)$ and the i^{th} unit in layer l. If the previous layer is also a fully connected layer then

$$z_i^{(l)} = \sum_{k=1}^{m^{(l-1)}} w^{(l)}_{i,k} * Y_k^{(l-1)} \quad \text{or} \quad z^{(l)} = w^{(l)} y^{(l-1)} \tag{4.11}$$

Where $w^{(l)}_{(i,k)}$ denotes the weight connecting between the i^{th} neuron in layer $l - 1$ to the k^{th} unit in layer l.

The output is a vector where each units gives the class score, if there are n classes in the output layer then it is given as

$$y = [y_1, y_2, ..., y_n]^T \tag{4.12}$$

4.2.6 Softmax layer

Softmax layer is used at the end of the architecture after fully connected layer. Softmax layer [33, 34] is used for classification problems which normalizes the output and also provides probabilistic interpretation. Softmax function is a vector to vector transformation which given as

$$\phi(a_i) = \text{softmax}(a_i) = \frac{\mathrm{e}^{a_i}}{\sum\limits_j \mathrm{e}^{a_j}} \tag{4.13}$$

where $\sum\limits_i \phi_i(a) = 1$ and $\phi_i(a) > 0$. This output can be considered a probability distribution over a set of outcomes.

4.3 Performance optimization of the Convolution Neural Network

There are several techniques that can improve the performance of a CNN. The optimization techniques help in following ways

- Faster CNN convergence
- Avoiding the local minima
- Prevent overfitting

- Improving accuracy or reducing error rate

There are several factors that affect the Convolution Neural Networks performance as discussed below.

4.3.1 Data sets

Before going further two terms need to be understood: the training and testing set. Generally the accuracy of a neural network depends on the size of dataset [35, 36], which should be as large as possible. The dataset can be expanded by adding several forms of distorted data. This data helps in learning using different patterns to generalize the neural network. The available examples are divided into two sets, training and testing set in the ratio of 80 : 20. Each set consists of examples and the class information to which the respective example belongs, also called labeled data for supervised learning.

4.3.2 Backpropagation

The main idea of backpropagation is to compute gradient for a network consisting of more than two processing layers. The layers between the input and output layers are known as hidden layers. If there are only two layers i.e. input and output layer, the error computation is simply the difference between output value and the target value and depending upon the error the weights can be changed. However if a network has more than two layers the computation of error gradient is difficult for the hidden layers, but can be done using back propagation. In backpropagation the gradients can be computed by propagating the output to input through the hidden layers, adjusting the weights and biases along the way. The amount of adjustments of weights and biases is calculated through gradient descent algorithm, where gradient descent seeks to minimize the error function by changing the weights and biases. The general process to train a neural network with backpropagation when an input vector $\mathbf{x}$ and the target or output value $\hat{\mathbf{y}}$ is given in Algorithm 9

4.3.3 Momentum

Momentum improves the convergence speed, where the current weight change depends on the previous weight change [3, 27]. A momentum is added to improve the

Algorithm 9 Pseudocode for backpropagation

```
1: The input vector $\mathbf{x}$ is applied to the neural network and propagated through all the layers.
2: Calculate the error using any error function. Let us assume the error function be mean square error and the error is given as $E_{MSE}(i,j) = \frac{|\hat{\mathbf{y}}(i,j)-x|^2}{n}$, where $\hat{\mathbf{y}}(i,j)$ be the expected result, $x$ be the desired output and $n$ be the total number of patterns provided for training.
3: Calculate the error gradient $\frac{\partial E}{\partial \mathbf{w}}$ for each layer.
4: Update weights
```

convergence speed. It controls the previous weight change on the current weight change from oscillating. The following shows the weight update formula. The weight update as follows

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \alpha \triangle \mathbf{w}(n) + \eta \triangle \mathbf{w}(n-1) \tag{4.14}$$

Where α is the momentum which lies between $0 \leq \alpha < 1$ and η is the learning rate [32, 36]. Learning rate is a crucial parameter which determines the rate of convergence and generalization of the neural network. It influences the speed at which the network attains minimum value of cost function. It is used in updating the hyper-parameters or weights during each iteration. The learning rate affects the quality of network, so optimal selection of learning rate is a crucial design constant. A small learning rate takes more time to converge and can lead to divergence. The optimal rate is given by

$$\eta_{opt} = \left(\frac{\partial^2 J}{\partial \mathbf{w}^2}\right)^{-1} \tag{4.15}$$

Where η_{opt} is optimal learning rate, J is the cost function and $\frac{\partial^2 J}{\partial \mathbf{w}^2}$ represents the double differentiation of cost function with the hyper parameters or weights.

4.3.4 Cross validation

Cross validation is a technique which separates data into disjoint sets, training set and test set. The available labeled examples are randomly divided into training and test set [1]. The motivation here is to train the CNN with the training data, which results in a model. The obtained model is tested with the test set i.e. to assess the performance and chose the best model. The training set and test sets are disjoint, so that new examples are given as input to the model and accuracy is obtained. This helps in generalization of the model.

4.3.5 Weight initialization

Initialization of CNN weights is a critical task to achieve good performance. Generally initializing all weights to 0 is ineffective. The initial weights to be initialized for fast and uniform learning, which helps in weights to reach equilibrium at same time. In practice the weights should are chosen randomly from same distribution function to ensure uniform learning [36].

4.3.6 Dropout method

Convolutional neural networks contain a large number of parameters due to multiple nonlinear hidden layers, these parameters model the complicated relationship between the inputs and outputs and also makes it a very powerful machine learning system. A model with large number of parameters is prone to overfitting. Over fitting is a problem in neural networks where the error rate on new test set is much higher than the error rate on training data.

Dropout helps to prevent overfitting and stops training a neural network when the error rate decreases in the test data set. Overfitting can be reduced by randomly omitting half of the kernels on each training set or omitting with a predefined ratio. Each kernel unit is retained with a fixed probability p which is independent of other kernel units, where p can be set using a validation set or simple 0.5, which seems to be optimal for a wide range of architectures [37]. The probability for the input layer feature maps must be higher than 0.5 because the lower probability makes to drop the input feature maps which further reduces the information in the input.

After reducing the number of nodes on the training data, dropout reduces overfitting and also improves the speed of training. The training of neural network with dropout layer shows a better error rate when compared to the network without dropout layer [38].

4.4 Number of convolution layers and features maps

When using CNNs for computer vision applications, the input unit is an image and number of output units depends on number of classes or categories. The number of convolution layers depends on the complexity of the dataset. In general, convolutional layers helps to learn more features deeply. Unfortunately often the accuracy gain becomes small after two or three layers, so it can be decided based on training time or generalization accuracy. The more the convolution layer more the training time. The number of convolution layers depends upon the application or task, so the best method is to increase the number of layers untill obtaining satisfying results.

There is no analytical way to determine the number of feature maps in each convolutional layer. A rough estimate is to double the number of feature maps than the previous layer and also depends on the dataset. A commonly used strategy is to use excessive number of feature maps and then removing them depending upon the training time, memory usage etc. In the same way sub sampling, the window size depends on the application as it throws most of the data away. For larger window sizes more data will be thrown away, so an appropriate window size should be selected. The best way to find a suitable network architecture is to perform trial and error tests, by applying pruning and constructive techniques as there is no standard analytical methods. Pruning techniques include selection of large network initially and then reducing the number of hyper parameters successively, whereas constructive technique includes selection of small network and then adding layers successfully [39].

4.5 Methodology

In this research a convolutional neural network (CNN) is used as feature extractor and classifier to recognize the alphabets (A-Z), numbers (0-9) and state information. The state and license plate numbers are recognized by separate CNN models. Training a CNN requires a lot of data and obtaining license plate samples of different states is a painful task. Our CNNs are implemented using Caffe, Scipy, Numpy, OpenCv libraries in Python and Linux environment. Caffe is a deep learning framework, developed by the Berkeley Vision and Learning Center (BVLC) [40]. The reasons behind implementation of CNN on Caffe is because of expressive architecture, computation speed which makes it suitable for research and industrial development, availability of resources, online community.

Graphics Processing Units (GPUs) are playing a key role in the field of high performance computing especially in the field of computer vision, machine learning problems [41]. CNNs include lots of computation which includes storage of model variables on memory and have to be updated for every iteration. To speed up executions, high speed memory access is required in order to simulate neural networks as normal CPU takes a lot of time for training the network. So CNNs have been implemented on GPU's using NVIDIA's GeForce GTX 980. GTX 980 is the world's most advanced GPU powered by NVIDIA® Maxwell^{TM} architecture. GTX 980 specifications include 2048 CUDA cores, base clock 1126 MHz, boost clock 1216 MHz, 4 GB memory with 7.0 Gbps memory clock. The hardware used for training the CNN is installed with Fedora 21 which runs on 12 core i7 5860 processor. The processor clock speed is 3 GHz and consists 64GB RAM, two 4GB GTX 980 graphic cards. In general training of CNN takes 1-6 days, which depends on the neural network architecture. The architecture of CNN are discussed in the next section.

4.5.1 Recognition of license number

This section describes the architecture of the convolution neural network for the recognition of characters A-Z and numbers $0-9$. The characters obtained from the segmentation process are fed as an input to the CNN for recognition. In this architecture number '0', 'Q' and alphabet 'O' are assumed as same class, hence the output categories is equal to 34. The dataset consists of images of alphabets and numbers and is divided in 13000 for training set and 3000 images for testing. All the images have been size-normalized and centered in a fixed size image 32×32 pixels. All the images are in binary format. Table 4.1 briefly provides information about each layer in the CNN. In Table 4.1 Conv + ReLU means the feature maps obtained from Conv layer are passed through ReLU layer, in the same way Pool+Norm layer. The result obtained from fully connected layer (ip1) is forwarded to ReLU4 and passed through Dropout layer (Dropout4). The fully connected layer provides the confident score of the image to be classified.

The number and alphabets are trained on two different architectures. The second architecture is trained using gray scale images of size 24×14. The training set consists of 9324 images testing set consists of 2318 images. This architecture consists of 36 output classes includes A-Z and $0-9$. The complete details of the architecture

Layer	Input size (*height* × *width*)	Filter size (*height* × *width*)	Number of filters	Pad (*height*× *width*)	Stride
Conv1	32 × 32	3 × 5	60	1 × 1	1
Conv2 + ReLU2	32 × 30	5 × 7	120	2 × 3	1
Pool2 + Norm2	32 × 30	8 × 8	120	-	2
Conv3 + ReLU3	32 × 30	5 × 1	384	2 × 0	1
Pool3 + Norm3	32 × 30	2 × 2	384	-	1
Fully connected(ip1) + ReLU4 + Dropout4	32 × 30	-	584	-	-
Fully connected(ip2)	584	-	34	-	-
Softmax layer	34	-	34	-	-

Table 4.1: Architectural layout of CNN for license number recognition using binary images

are provided in Table 4.2

4.5.2 Recognition of state

This section describes the CNN architecture for recognition of state. The extracted state portion of the license plate is fed as an input to the convolution neural network. The input image is a gray scale image of size 128 × 256. The architecture consists of 20 layers excluding the input layer. The architecture consists of 4 convolution layers. Each convolution is followed by subsampling layer, ReLU layer and local normalization layer, and can be expressed as single block [conv - pool - ReLU - LRN] Four blocks are arranged and at the end there are two fully connected layers with a dropout layer after the first fully connected layer. As there are 50 states in USA, the output layer finally consists of 50 classes. The proposed architecture for recognizing

Layer	Input size (*height* × *width*)	Filter size (*height* × *width*)	Number of filters	Pad (*height*× *width*)	Stride
Conv1	24×14	5×5	74	2×2	1
Conv2+ReLU2	24×14	5×3	150	2×1	1
Pool2+Norm2	24×14	4×4	150	-	1
Conv3	24×14	3×1	384	2×0	1
Conv4+RelU4	24×14	3×3	680	1×1	1
Pool4+Norm4	24×14	4×4	680	-	1
Conv5	24×14	3×3	784	1×1	1
Conv6+ReLU6	24×14	3×3	784	1×1	1
Fully connected(ip1) + ReLU7 + Dropout7	24×14	-	500	-	-
Fully connected(ip2)	500	-	36	-	-
Softmax layer	36	-	36	-	-

Table 4.2: Architectural layout of CNN for license number recognition using gray scale images.

state is described in Table 4.3.

Layer	Input size (*height* × *width*)	Filter size (*height* × *width*)	Number of filters	Pad	Stride
Conv1+ReLU1	128 × 256	5 × 11	20	0	1
Pool1+Norm1	124 × 246	2 × 2	20	0	2
Conv2+ReLU2	62 × 62	7 × 7	80	0	2
Pool2+Norm2	28 × 28	2 × 2	80	0	2
Conv3+ReLU3	14 × 14	3 × 3	180	0	2
Pool3+Norm3	6 × 6	1 × 1	384	0	1
Conv4+ReLU4	1 × 1	1 × 1	384	0	2
Pool4+Norm4	14 × 14	3 × 3	180	0	1
Fully connected(ip1) + ReLU5 + Dropout5	14 × 14	-	384	-	-
Fully connected(ip2)	384	-	50	-	-
Softmax layer	50	-	50	-	-

Table 4.3: Architectural layout of CNN for state recognition using gray scale images

Chapter Five

Results and Discussions

We now present the results of our ALPR system on various convolution neural networks architectures and compute the accuracy at various stages. A successful automatic license plate recognition system extracts information and recognizes it correctly. License plate data used to train the convolution neural network are obtained through Google images. We expanded dataset by generating additional images using distortions such as rotations, translations, skewing generated by affine transforming, adding contrast etc [42]. The complete results of our research are presented in the following sections.

5.1 Character and state segmentation

As discussed in Chapter 3, segmentation is one of the important steps of our system. The overall accuracy of the system is heavily influenced by this stage. Difficulties faced in this stage are

1. Unwanted objects such as handicapped symbols, state symbols
2. Illumination problems
3. Blurry images

Problems arose while dealing with the degraded images. When binarized the characters are overlap and extraction of characters fails most of the time.

The segmentation is evaluated by testing the segmentation algorithm on a database of 395 images. We manually evaluate the segmentation results by comparing the license number and the extracted information. The database of images also includes degraded images. To compute the accuracy, 394 are considered. Out of 394 images 372 images are segmented satisfactorily and 22 images failed. Some of the failed images segmented partially. This gives an accuracy of 94.4% for license number segmentation. The state portion has been extracted correctly for most of these images.

Successfully segmented images include unwanted objects such as state symbols, handicapped symbols as shown in Fig. 5.1.

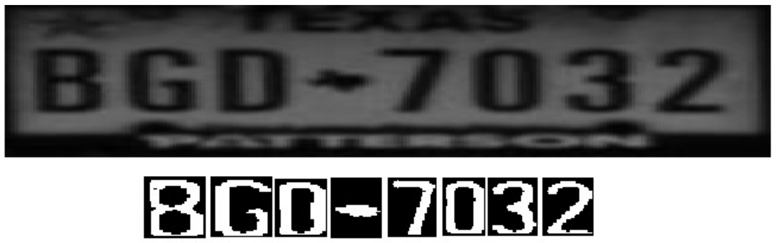


Figure 5.1: Extracted license number from the given input license plate

The state symbol is included because of inclusion of vertical projection technique as discussed in section 3.3 for character extraction.

5.2 Recognition of extracted information

The recognition part of our ALPR system features convolutional neural networks. The recognition of license number and state information is done separately using two separate architectures and are trained separately using different datasets.

5.2.1 License number recognition

The license number CNN has been trained using the architecture shown in Table 4.1 using binary images of size 32×32. The training set consists 6290 images and test dataset consists of 2804 images. The learned model has been validated with a set of 680 images where each class consists of 20 images. We consider characters 'Q','O'and the number 0 as one class because of geometrical similarities. Thus our system has 34 classes which include digits $0 - 9$ and characters A-Z. Table 5.1 illustrates the accuracy in each class.

Class	Number of images classified correctly	Number of images misclassified	Accuracy (%)
0	20	0	100
1	19	1	95
2	19	1	95
3	18	2	90
4	9	11	45
5	19	1	95
6	20	0	100
7	4	16	20
8	20	0	100
9	20	0	100
A	20	0	100
B	15	5	75
C	18	2	90
D	9	11	45
E	19	1	95
F	20	0	100
G	18	2	90
H	20	0	100
I	20	0	100
J	19	1	95
K	4	16	20
L	20	0	100
M	16	4	80
N	20	0	100
P	19	1	95
R	20	0	100
S	18	2	90
T	20	0	100
U	20	0	100
V	20	0	100
W	20	0	100
X	20	0	100
Y	20	0	100
Z	1	19	5
Overall accuracy	584	96	85.88

Table 5.1: Classification rate of each character and number using CNN trained with binary images

Misclassification for some of the characters is caused by similar geometric properties. Table 5.2 illustrates the misclassification among some classes. The first column in Table 5.2 shows the character class and the second column shows the estimated class by the neural network.

Desired class	Estimated class
I	1
D	0
J	L
M	W or H
Z	2
K	X
7	T

Table 5.2: Misclassified characters for binary image

Fig. 5.2 displays different rotations of templates '5'. When convolution is applied to all the images using same kernel, results in similar properties as convolution operation invariant to rotation [33]. Hence the main problem for misclassification is similar geometrical properties.

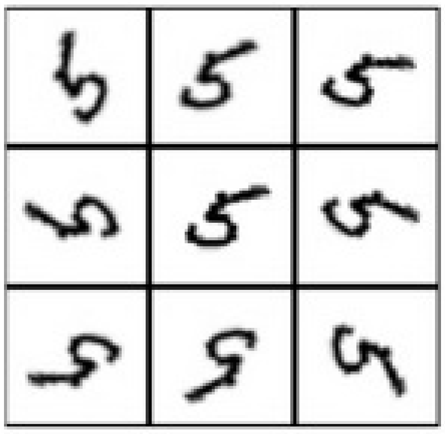


Figure 5.2: Example of learned invariance

Fig. 5.3 displays the extracted binarized characters from a license plate. We can observe ‘W’ is almost similar to ‘H’, ‘D’ is similar to ‘O’ or ‘0’. The other problems includes multiple formats of representation such as ‘I’ and ‘4’ which is displayed in Fig. 5.4. Some of the badly or partially segmented characters are also included i.e. like ‘F’ in Fig. 5.3.

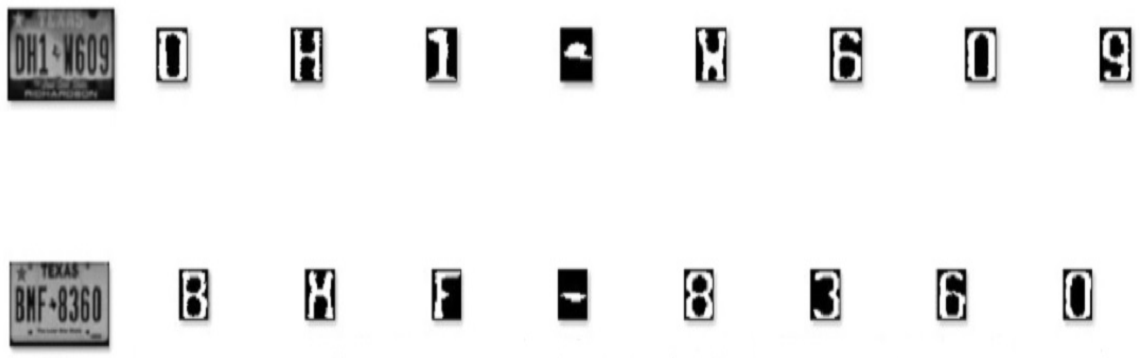


Figure 5.3: Characters with similar geometric properties

Fig. 5.4 shows representation of number 4 in two different ways. The character ‘I’ can also be represented in two different ways. The neural network is trained by including two different geometric structures into same category. This is one of the reasons for misclassification.

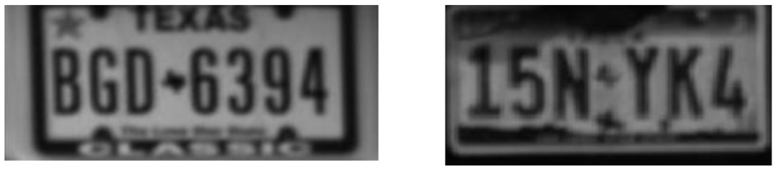


Figure 5.4: Different representation of same character

The main reason for misclassification of characters is due to binarization of degraded images. Binarization of degraded images leads to misclassification of pixels into foreground and background pixels, which changes the geometrical properties of characters. As an attempt to increase the accuracy, CNN described in Table 4.2 has been trained using gray scale images. The usage of gray scale images reduces loss of information during binarization. At the same time, a larger range of values for gray scale image pixels $[0, 255]$ should help classification. This CNN architecture consists of 36 classes, whereas the CNN trained with binary images consists of 34 classes, as the loss of information is minimized. The architecture is trained using gray scale images of size 24×14. The CNN for license number recognition has been trained using 9324 images and test dataset consists of 2318 images. The learned model has been validated with a set of 720 images where each class consists of 20 images. Table 5.3 displays the accuracy results obtained for CNN trained with gray scale images of size 24×14.

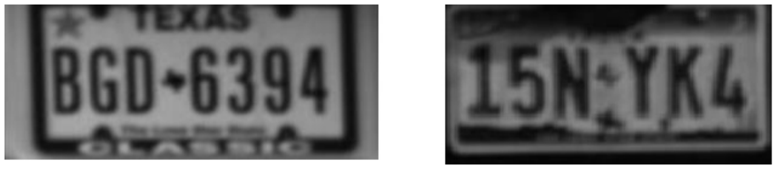


Figure 5.5: Different representation of same character

As shown in Table 5.3 the accuracy obtained for the gray scale images is better than the binary images. Most of the characters show an accuracy of more than 80% and the least is 60%. The misclassification is due to degraded images and due to similar geometrical properties as discussed before. Table 5.4 illustrates the misclassification among some classes. The accuracy is low for only two characters 'M' and 'X'. The usage of gray scale images helps CNN to differentiate 'Q', 'O' and '0' which is difficult with binary images.

5.2.2 State recognition

For state recognition we use the architecture described in Table 4.3. We train using 59469 images and with 19857 test images in gray scale format. The major problem in extraction of license of state from the license plate is localization of state information on the license plate. We assume the state portion lies above to the license number. The other problem includes the degraded license plates, presence of shadows results in loss during binarization as shown in Fig. 5.5. The loss of information during binarization of state portion effects the overall accuracy of the system. Hence for the state portion gray scale images has been considered for training, where a loss of information can be avoided. The following images in Fig. 5.5 shows binarized state portion of different plates of Alabama state, where we can clearly observe the loss of information after binarization. Loss of information effects the accuracy of the system and this is the major reason for training the system using gray scale images.

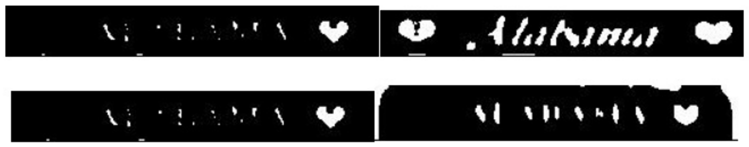


Figure 5.6: Alabama state license plates showing loss of information after binarization

State training has been done in various stages. The CNN has been trained using 3 architectures, where the number of convolution layers varies. The first architecture consists of 2 convolution layers, the second one consists of 3 convolution layer and the last one consists of 4 convolution layer. The three CNN has been trained with the same datasets where the training set consists of 58000 images and the testing dataset consists of around 19000 images. The image size is considered as 128 × 256. The CNN's has been trained for 100000 iterations where the accuracy is calculated using a cross validation set. The cross validation set consists of 100 images for each state, it is used to select the final model for each architecture. Table 5.5 displays the accuracy at every 10000 iterations for all the three CNN architectures.

As shown in Table 5.5, the best accuracy given by a CNN consisting of 2 convolution layers is 72.32%, whereas the CNN consisting of 4 convolution layers gives an accuracy of 80.08%. The primary reason for getting good accuracy on 4-convolutional layer is due to ability to learn the state better. For example at 50000 iterations the study shows the 4 convolutional layer architectures give an accuracy of 80.08%, whereas the other two architectures recorded low accuracy. The accuracy of each architecture increases up to some iteration but after that the accuracy is getting decreases, this shows that CNN gets overfits. Overfitting means the CNN is becoming adaptive to the training set and this makes to increase the error on new set or cross validation images.

The training of the CNN has been done using different sizes of the images like 40 × 256 and 128 × 256. The best results are obtained by using 128 × 256. The main reason behind this nature is due to availability of information. As the size the increase the information will also increase which helps to extract best features on convolution. The architecture described in the Fig. 5.2 consists of 4 convolution layers, which helps to learn more complex features. The study was also done on various architectures. The CNN has been trained for 100000 iterations and took a snapshot of the model for every 10000 iterations. The final model has been selected based on the best accuracy obtained on a cross-validation set. A cross validation dataset is a set of images which are not included in the training or testing dataset. The set consists of 5000 images, 100 for each class. The final model of an architecture has been selected based on the performance on cross validation set. The performance of various image sizes are given in Table 6.2

5.3 Overall performance

The recognition rate of license number using binary characters is 15.3% where 57 license plates are recognized correctly out of 372 satisfactorily segemented images. The state recognition accuracy for successfully segmented images is 88.97% where 331 images are correctly recognized out of 372 images. The overall accuracy considering all the stages is 13%. The overall includes all the correctly segmented images and incorrectly segmented images.

The accuracy for license number recognition using CNN trained with gray scale images is 91.12% where 339 license plate numbers are correctly recognized and 33 images recognized incorrectly. Though the architecture has ability to distinguish '0' and 'O', they are considered as same class while manually validating the results which is due to similar geometrical properties. The state recognition accuracy is 88.97% where 331 images are correctly recognized out of 372 images. The overall accuracy of the system considering all the stages is 81.1%, where 302 out of 394 successfully segmented and recognized correctly. The results are embedded in Table 5.7.

The overall accuracy of binary images is very low when compared to gray scale images. The reason behind this is due to character recognition stage, where all the characters in the license number should be recognized correctly. The classification rate for binary images according to Table 5.1 is 85.58%, when it comes to overall recognition the system fails when compared to gray scale images because of the misclassified characters. Table 5.2 shows the misclassification of characters, where the classifier confuses between most of the characters. This is avoided in gray scale images as binarization technique plays a minimum role in usage of gray scale images.

Class	Number of images classified correctly	Number of images misclassified	Accuracy (%)
0	20	0	100
1	20	0	100
2	20	0	100
3	20	0	100
4	20	0	100
5	19	1	95
6	19	1	95
7	20	0	100
8	20	0	100
9	20	0	100
A	18	2	90
B	20	0	100
C	17	3	85
D	20	0	100
E	16	4	80
F	19	1	95
G	19	1	95
H	19	1	95
I	18	2	90
J	18	2	90
K	17	3	85
L	19	1	95
M	12	8	60
N	20	0	100
O	20	0	100
P	20	0	100
Q	19	1	95
R	16	4	80
S	18	2	90
T	17	3	85
U	20	0	100
V	18	2	90
W	20	0	100
X	12	8	60
Y	19	1	95
Z	15	5	95
Overall accuracy	665	55	92.36

Table 5.3: Classification rate of each character and number using CNN trained with gray scale images

Desired class	Estimated class
M	W
X	K

Table 5.4: Misclassified characters for gray scale images

Iteration number(×1000)	2-convolution layer CNN accuracy (%)	3-convolution layer CNN accuracy (%)	4-convolution layer CNN accuracy (%)
10	2	4.94	2
20	2	47.58	66.88
30	4.7	66.88	69
40	20.88	70.82	76.58
50	35.38	71.98	**80.88**
60	57.42	72.18	79.38
70	69.8	72.14	78.62
80	**72.32**	71.86	78.34
90	72.18	**72.32**	78.98
100	72.14	71.72	77.92

Table 5.5: Comparison of performance of various architectures

Image size(rows × columns)	Accuracy (%)
40 × 256	66.32
128 × 256	80.88

Table 5.6: Comparison of performance of various architectures

Stage	Binary license number accuracy(%)	Gray scale license number accuracy (%)
Character and state segmentation stage	94.41	94.41
Character recognition stage	15.3	91.12
State recognition stage	88.97	88.97
Overall accuracy	13	81.1

Table 5.7: Independent success rate of each stage for binary and gray scale images

Chapter Six

Conclusion and Future work

6.1 Conclusion

This thesis has investigated the use of deep learning techniques in the field of automatic license plate detection system. We proposed a system which consists of two stages: character segmentation and character recognition. An important aspect of this system is to incorporate a hybrid binarization technique which helps to improve quality of segmentation. This technique helps in the segmentation of degraded images.

Another important aspect incorporated in this research is using convolutional neural networks for feature extraction and license plate recognition. The recognition task includes recognition of license number and state information from the license plate. Our study also includes how the data format impacts the recognition system and different techniques to optimize convolution neural networks. Our experiments show an overall accuracy of 81.1% using gray scale images. Our CNNs are implemented using Caffe, Scipy, Numpy, OpenCv libraries in Python and Linux environment. The hardware used for training the CNN is installed with Fedora 21 which runs on 12 core i7 5860 processor. The processor clock speed is 3 GHz and consists 64GB RAM, two 4GB GTX 980 graphic cards. In general training of CNN takes 1-6 days, which depends on the neural network architecture. Our results show the usage of deep learning techniques in the field of ALPR system. This system can be used for practical use by the following improvements

1. Usage of real time data to train the convolution neural networks. Real time data includes images collected from the toll gates, parking lots and other agencies etc.
2. Inclusion of localization of license plate, de-skewing stage.

3. Localizing and segmenting the state information automatically.

6.2 Limitations

The system shows satisfactory results in segmentation stage and recognition stage. There are certain limitation on the system as it is developed with certain assumption. The limitations are as follows:

1. The state information is assumed to be at upper portion of license plate. The license plate with state information at lower portion of license plate fails, but recognizes license number successfully.

2. For binary images '0', 'O' and 'Q' are considered as same class.

3. Recognition of unwanted symbols are ignored in this research and are included in future study.

4. '0' and 'O' are considered as same class for gray scale images because of similar geometric structure.

5. Need for more data.

6.3 Future work

The automatic license plate recognition system proposed in this research has several limitations. Most major being that the state information position is assumed to be at top part of license plate. Though most of the plates consists of state information at the upper part of license plate, the proposed system will not be able to recognize the state information if the position of state information is changed. Inclusion of recognizing unwanted symbols improves the accuracy of the system. Future work includes localization and detection of state information [43]. It also might locate and detect the license number from the license plate. Future work also includes implementation of license plate segmentation using deep learning techniques, expected to have good accuracy. Implementation of an ALPR system completely using deep learning techniques might consume lot of execution time and is challenging due to limited availability of datasets. Deep learning techniques are providing efficient solutions in the field of Artificial intelligence, pattern recognition etc with the incorporation of graphic processing units.

Bibliography

[1] S. Haykin, *Neural Networks and Learning Machines.* Pearson Education, 2011. [Online]. Available: https://books.google.com/books?id=faouAAAAQBAJ

[2] Y. LeCun and M. Ranzato, "Deep learning tutorial," in *Tutorials in Int. Conf. on Mach. Learning (ICML'13)*, 2013.

[3] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[4] Y. Bengio, "Learning deep architectures for ai," *Found. Trends Mach. Learning*, vol. 2, no. 1, pp. 1–127, Jan. 2009. [Online]. Available: http://dx.doi.org/10.1561/2200000006

[5] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural Computation*, vol. 18, no. 7, pp. 1527–1554, 2006.

[6] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," *arXiv preprint arXiv:1409.4842*, 2014.

[7] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Comput. Vision and Pattern Recognition (CVPR), 2014 IEEE Conf.*, pp. 580–587.

[8] Q. V. Le, "Building high-level features using large scale unsupervised learning," in *Acoust., Speech and Signal Processing (ICASSP), 2013 IEEE Int. Conf.*, pp. 8595–8598.

[9] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[10] R. Socher, A. Karpathy, Q. V. Le, C. D. Manning, and A. Y. Ng, "Grounded compositional semantics for finding and describing images with sentences," *Trans. Assoc. Computational Linguistics*, vol. 2, pp. 207–218, 2014.

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

[12] G. E. Dahl, D. Yu, L. Deng, and A. Acero, "Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition," *Audio, Speech, and Language Processing, IEEE Trans. on*, vol. 20, no. 1, pp. 30–42, 2012.

[13] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *Int. Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, 2004.

[14] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *Comput. Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Comput. Soc. Conf.*, vol. 1, pp. 886–893.

[15] G. E. Nasr, E. Badr, and C. Joun, "Cross entropy error function in neural networks: Forecasting gasoline demand." in *FLAIRS Conference*, 2002, pp. 381–384.

[16] S. Du, M. Ibrahim, M. Shehata, and W. Badawy, "Automatic license plate recognition (alpr): A state-of-the-art review," *IEEE Trans. Circuits and Syst. for Video Technol*, vol. 23, no. 2, pp. 311–325, 2013.

[17] C.-N. E. Anagnostopoulos, I. E. Anagnostopoulos, I. D. Psoroulas, V. Loumos, and E. Kayafas, "License plate recognition from still images and video sequences: A survey," *IEEE Trans. Intell. Transp. Syst.*, vol. 9, no. 3, pp. 377–391, 2008.

[18] R. Prates, G. Cámara-Chávez, W. R. Schwartz, and D. Menotti, "Brazilian license plate detection using histogram of oriented gradients and sliding windows," *arXiv preprint arXiv:1401.1990*, 2014.

[19] Y.-N. Chen, C.-C. Han, C.-T. Wang, B.-S. Jeng, and K.-C. Fan, "The application of a convolution neural network on face and license plate detection," in *Pattern Recognition, 2006. ICPR 2006. 18th Int. Conf.*, vol. 3. IEEE, 2006, pp. 552–555.

[20] S. A. Radzi and M. Khalil-Hani, "Character recognition of license plate number using convolutional neural network," in *Visual Informatics: Sustaining Research and Innovations*. Berlin: Springer, 2011, pp. 45–55.

[21] P. Perona and J. Malik, "Scale-space and edge detection using anisotropic diffusion," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 12, no. 7, pp. 629–639, 1990.

[22] J. Weickert, "A review of nonlinear diffusion filtering," in *Scale-space Theory in Computer Vision.* Berlin: Springer, 1997, pp. 1–28.

[23] R. Gonzalez and R. Woods, *Digital Image Processing.* Pearson Education, 2011. [Online]. Available: https://books.google.com/books?id=MaYuAAAAQBAJ

[24] K. Zuiderveld, "Contrast limited adaptive histogram equalization," in *Graphics gems IV.* San Diego: Academic Press Professional, Inc., 1994, pp. 474–485.

[25] J. Sauvola and M. Pietikäinen, "Adaptive document image binarization," *Pattern Recognition*, vol. 33, no. 2, pp. 225–236, 2000.

[26] S. Duffner, "Face image analysis with convolutional neural networks," Ph.D. dissertation, University of Freiburg Freiburg, Germany, 2008.

[27] Z. Zhao, S. Yang, and X. Ma, "Chinese license plate recognition using a convolutional neural network," in *Computational Intell. and Ind. Application, 2008. PACIIA'08. Pacific-Asia Workshop on*, vol. 1. IEEE, 2008, pp. 27–30.

[28] Y.-N. Chen, C.-C. Han, C.-T. Wang, B.-S. Jeng, and K.-C. Fan, "The application of a convolution neural network on face and license plate detection," in *Pattern Recognition, 2006. ICPR 2006. 18th Int. Conf.*, vol. 3. IEEE, 2006, pp. 552–555.

[29] C.-C. Han, C.-T. Hsieh, Y.-N. Chen, G.-F. Ho, K.-C. Fan, and C.-L. Tsai, "License plate detection and recognition using a dual-camera module in a large space," in *Security Technol., 2007 41st Annual IEEE Int. Carnahan Conf.*, pp. 307–312.

[30] D. Strigl, K. Kofler, and S. Podlipnig, "Performance and scalability of gpu-based convolutional neural networks," in *Parallel, Distributed and Network-Based Processing (PDP), 2010 18th Euromicro Int. Conf.* IEEE, 2010, pp. 317–324.

[31] Y. LeCun, K. Kavukcuoglu, and C. Farabet, "Convolutional networks and applications in vision," in *Circuits and Syst. (ISCAS), Proc. of 2010 IEEE Int. Symp.*, pp. 253–256.

[32] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *Advances Neural Inform. Proc. Syst.*, 2012, pp. 1097–1105.

[33] Y. Bengio, I. J. Goodfellow, and A. Courville, “Deep learning,” 2015, book in preparation for MIT Press. [Online]. Available: http://www.iro.umontreal.ca/~bengioy/dlbook

[34] Y. Tang, “Deep learning using linear support vector machines,” *arXiv preprint arXiv:1306.0239*, 2013.

[35] C. Bishop, *Pattern Recognition and Machine Learning*, ser. Information Science and Statistics. New York: Springer, 2006. [Online]. Available: https://books.google.com/books?id=kTNoQgAACAAJ

[36] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern classification.* New York: John Wiley & Sons, 2012.

[37] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: A simple way to prevent neural networks from overfitting,” *The J. Mach. Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[38] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, “Improving neural networks by preventing co-adaptation of feature detectors,” *arXiv preprint arXiv:1207.0580*, 2012.

[39] S. Theodoridis and K. Koutroumbas, *Pattern Recognition.* New York: Elsevier Science, 2008. [Online]. Available: https://books.google.com/books?id=QgD-3Tcj8DkC

[40] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, “Caffe: Convolutional architecture for fast feature embedding,” *arXiv preprint arXiv:1408.5093*, 2014.

[41] E. László, P. Szolgay, and Z. Nagy, “Analysis of a gpu based cnn implementation,” in *Cellular Nanoscale Networks and Their Applications (CNNA), 2012 13th Int. Workshop.* IEEE, 2012, pp. 1–5.

[42] P. Y. Simard, D. Steinkraus, and J. C. Platt, “Best practices for convolutional neural networks applied to visual document analysis,” in *Advances in Neural Information Processing Systems 5 (NIPS’92).* IEEE, 1992, p. 958.

[43] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun, “Overfeat: Integrated recognition, localization and detection using convolutional networks,” *arXiv preprint arXiv:1312.6229*, 2013.

Appendix A: sample codes

A.1 Configuration file for CNN to classify characters using grayscale format

```
name: "lprStateNet"
layer {
  name: "lpr"
  type: "Data"
  top: "data"
  top: "label"
  include {
    phase: TRAIN
  }
  transform_param {
    scale: 0.00390625
  }
  data_param {
    source: "examples/lprNum/lprNumTrainGray2414C1"
    batch_size: 128
    backend: LMDB
  }
}
layer {
  name: "lpr"
  type: "Data"
  top: "data"
  top: "label"
  include {
    phase: TEST
  }
  transform_param {
    scale: 0.00390625
  }
  data_param {
    source: "examples/lprNum/lprNumTestGray2414C1"
    batch_size: 30
    backend: LMDB
  }
}
layer {
  name: "conv1"
  type: "Convolution"
  bottom: "data"
  top: "conv1"
  param {
    lr_mult: 1
  }
  param {
    lr_mult: 2
  }
  convolution_param {
    pad_h: 3
    pad_w: 2
    num_output: 74
    kernel_h: 5
    kernel_w: 5
    stride: 1
    weight_filler {
      type: "xavier"
    }
    bias_filler {
      type: "constant"
    }
  }
}
layer {
  name: "conv2"
  type: "Convolution"
  bottom: "conv1"
  top: "conv2"
  param {
    lr_mult: 1
  }
  param {
    lr_mult: 2
  }
  convolution_param {
    pad_h: 2
    pad_w: 1
    num_output: 150
    kernel_h: 5
    kernel_w: 3
    stride: 1
    weight_filler {
      type: "xavier"
    }
    bias_filler {
      type: "constant"
    }
  }
}
layer {
  name: "relu2"
  type: "ReLU"
  bottom: "conv2"
  top: "conv2"
}
layer {
  name: "pool2"
  type: "Pooling"
  bottom: "conv2"
  top: "pool2"
  pooling_param {
    pool: MAX
    kernel_size: 4
    stride: 1
  }
}
layer {
  name: "norm2"
  type: "LRN"
  bottom: "pool2"
  top: "norm2"
  lrn_param {
    local_size: 5
    alpha: 0.0001
    beta: 0.75
  }
}
layer {
  name: "conv3"
  type: "Convolution"
  bottom: "norm2"
  top: "conv3"
  param {
    lr_mult: 1
  }
  param {
    lr_mult: 2
  }
  convolution_param {
    num_output: 384
    pad_h: 2
    pad_w: 0
    stride: 1
    kernel_h: 3
    kernel_w: 1
    weight_filler {
      type: "xavier"
    }
    bias_filler {
      type: "constant"
    }
  }
}
layer {
  name: "conv4"
  type: "Convolution"
  bottom: "conv3"
  top: "conv4"
  param {
    lr_mult: 1
  }
  param {
    lr_mult: 2
  }
  convolution_param {
    pad: 1
    num_output: 680
    kernel_size: 3
    stride: 1
    weight_filler {
      type: "xavier"
    }
    bias_filler {
      type: "constant"
    }
  }
}
layer {
  name: "relu3"
  type: "ReLU"
  bottom: "conv4"
  top: "conv4"
}
layer {
  name: "pool3"
  type: "Pooling"
  bottom: "conv4"
  top: "pool3"
  pooling_param {
    pool: MAX
    kernel_size: 4
    stride: 1
  }
}
layer {
  name: "norm3"
  type: "LRN"
  bottom: "pool3"
  top: "norm3"
  lrn_param {
    local_size: 5
    alpha: 0.0001
    beta: 0.75
  }
}
layer {
  name: "conv5"
  type: "Convolution"
  bottom: "norm3"
  top: "conv5"
  param {
    lr_mult: 1
  }
  param {
    lr_mult: 2
  }
  convolution_param {
    num_output: 584
    stride: 1
    pad: 1
    group: 2
    kernel_size: 3
```

```
      weight_filler {
        type: "xavier"
      }
      bias_filler {
        type: "constant"
      }
    }
}
layer {
  name: "conv6"
  type: "Convolution"
  bottom: "conv5"
  top: "conv6"
  param {
    lr_mult: 1
  }
  param {
    lr_mult: 2
  }
  convolution_param {
    pad: 1
    num_output: 784
    kernel_size: 3
    stride: 1
    weight_filler {
      type: "xavier"
    }
    bias_filler {
      type: "constant"
    }
  }
}
layer {
  name: "relu6"
  type: "ReLU"
  bottom: "conv6"
  top: "conv6"
}
layer {
  name: "ip1"
  type: "InnerProduct"
  bottom: "conv6"
  top: "ip1"
  param {
    lr_mult: 1
  }
  param {
    lr_mult: 2
  }
  inner_product_param {
    num_output: 500
    weight_filler {
      type: "xavier"
    }
    bias_filler {
      type: "constant"
    }
  }
}
layer {
  name: "relu5"
  type: "ReLU"
  bottom: "ip1"
  top: "ip1"
}
layer {
  name: "Drop5"
  type: "Dropout"
  bottom: "ip1"
  top: "ip1"
  dropout_param {
  dropout_ratio: 0.5
}
}
layer {
  name: "ip2"
  type: "InnerProduct"
  bottom: "ip1"
  top: "ip2"
  param {
    lr_mult: 1
  }
  param {
    lr_mult: 2
  }
  inner_product_param {
    num_output: 36
    weight_filler {
      type: "xavier"
    }
    bias_filler {
      type: "constant"
    }
  }
}
layer {
  name: "accuracy"
  type: "Accuracy"
  bottom: "ip2"
  bottom: "label"
  top: "accuracy"
  include {
    phase: TEST
  }
}
layer {
  name: "loss"
  type: "SoftmaxWithLoss"
  bottom: "ip2"
  bottom: "label"
  top: "loss"
}
```

A.2 Sauvola binarization technique

```
import numpy as np

def averagefilter(image, window=[3, 3]):
    m = window[0]
    n = window[1]

    if m % 2 == 0:
        m = m - 1
    if n % 2 == 0:
        n = n - 1

    (rows, columns) = image.shape

    pad_width = (((m + 1) / 2, (m - 1) / 2), ((n + 1) / 2, (n - 1) / 2))
    imageP = np.pad(image, pad_width, 'edge')
    imageD = np.array(imageP, dtype=np.double)

    t = np.cumsum(np.cumsum(imageD, axis=0), axis=1);

    imageI = (t[m:rows + m, n: columns + n] +
              t[0:rows, 0:columns] -
              t[m:rows + m, 0:columns] -
              t[0:rows, n:columns + n])

    imageI = imageI / (m * n)

    return imageI

def sauvola(image, window=[3, 3], threshold=0.34):
    # Convert to double
    imageD = np.array(image, dtype=np.double)

    # Mean value
    mean = averagefilter(image, window)

    # Standard deviation
    meanSquare = averagefilter(np.square(imageD), window)
    deviation = (meanSquare - np.square(mean)) ** 0.5

    # Sauvola
    R = np.max(deviation)
    threshold = mean * (1 + threshold * (deviation / R - 1))
    output = imageD > threshold

    return output
```

A.3 Global thresholding technique

```
import numpy as np

def threshold(block, percentage_black):
    # Finds a threshold for binarizing an image based on a required perc
    # of foreground pixels

    bin_count = 256
    height = block.shape[0]
    width = block.shape[1]

    area = height * width
    percentage_area = np.floor(percentage_black * area);

    cumulative_intensity_sum, theshold, found = 0, 0, 0
    imhist, bins = np.histogram(block.flatten(), range=[0, bin_count], b
    cumulative_intensity_sum = np.cumsum(imhist)

    # Calculating binarization threshold
    for index in range(bin_count):
        if cumulative_intensity_sum[index] >= percentage_area:
            threshold = index
            break

    return threshold
```

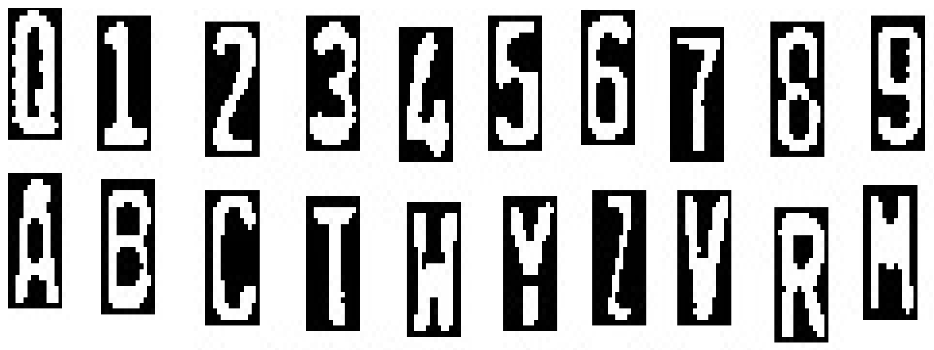
A.4 Edge based binarization

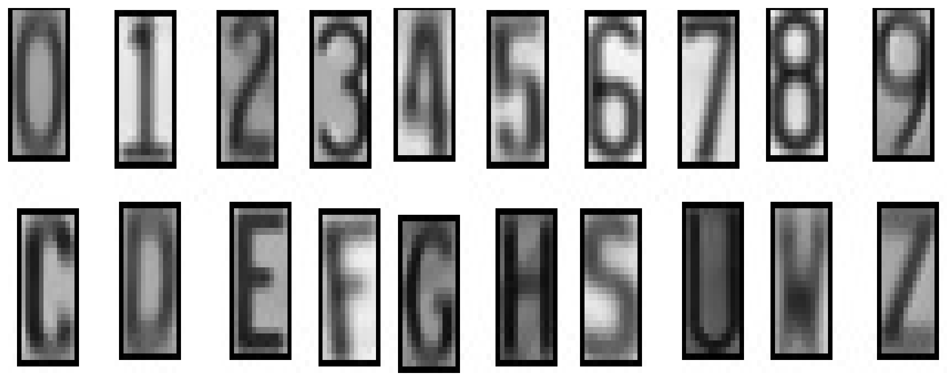
```
import numpy as np
import cv2

def edge_based_binarization(image):
    # binarizes the image based on threshold determined from edge region
    edge_image = cv2.Canny(image, 0, 255)

    binary_image = np.zeros(shape=image.shape)
    Hblocks = 10
    Vblocks = 5

    blockH = np.floor(image.shape[0] * 1.0 / Vblocks)
    blockW = np.floor(image.shape[1] * 1.0 / Hblocks)

    debug = False
    for r in range(Vblocks):
        for c in range(Hblocks):
            r0 = r * blockH + 1
            c0 = c * blockW + 1
            imgblock = image[r0:r0 + blockH, c0:c0 + blockW]
            edgeblock = edge_image[r0:r0 + blockH, c0:c0 + blockW]

            t = find_threshold(imgblock, edgeblock);
            binary_image[r0:r0 + blockH, c0:c0 + blockW] = imgblock < t;

    binary_image = np.array(binary_image, dtype=np.uint8)

    return binary_image
```

```
def find_threshold(imgblock, edgeblock):
    # finds the threshold for a block using grayscale values from the
    # neighbouring pixels of all edges
    t1 = []
    for r in range(3, imgblock.shape[0] - 2):
        for c in range(3, imgblock.shape[1] - 2):
            if (edgeblock[r, c] == 255 and
                edgeblock[r, c - 1] == 0 and
                edgeblock[r, c + 1] == 0):
                m = min(imgblock[r, c - 2], imgblock[r, c + 2])
                if m < 128:
                    t2 = np.mean(imgblock[r, c - 2: c + 2]) * 1.0
                    t1.append(t2)
            if (edgeblock[r, c] == 255 and
                edgeblock[r - 1, c] == 0 and
                edgeblock[r + 1, c] == 0):
                m = min(imgblock[r - 2, c], imgblock[r + 2, c])
                if m < 128:
                    t2 = np.mean(imgblock[r-2: r+2,c]) * 1.0
                    t1.append(t2)

    if len(t1) == 0:
        t = 0
    else:
        t = np.mean(t1) - np.std(t1)

    return t
```

Appendix B: Sample training images for state recognition

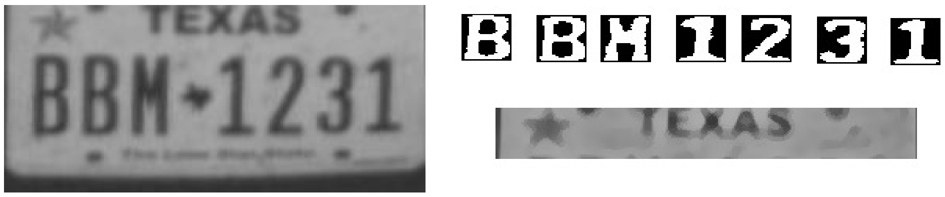
Appendix C: Sample training images for character recognition using binary and gray images

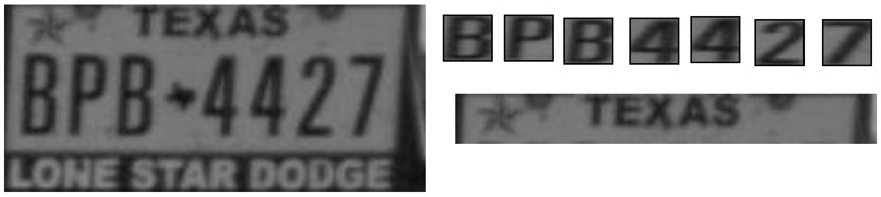




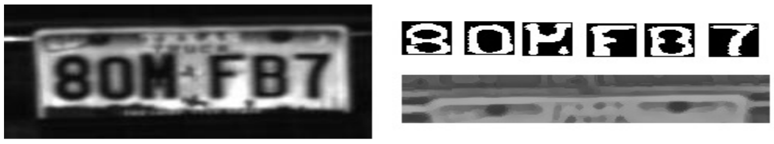
Appendix D: Sample results

D.1 Successfully recognized images

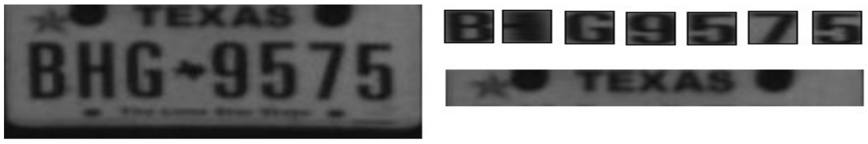




D.2 Failed images



Recognized license number: BOM FBT



Recognised license number: B4G 9575