**A**

**Project Report**

**On**

**VEHICLE NUMBER PLATE RECOGNITION**

**BTech-sem VII**

**Prepared By**

**Krunal Kantrodiya(IT-49)**

**Dharmesh Narola(IT-62)**

# 

## DEPARTMENT OF INFORMATION TECHNOLOGY

## FACULTY OF TECHNOLOGY,

## DHARMSINH DESAI UNIVERSITY

**COLLEGE ROAD, NADIAD- 387001**

December,2019

**A**

**Project Report**

**On**

**VEHICLE NUMBER PLATE RECOGNITION**

**BTech-sem VII**

## In partial fulfillment of requirements for

Bachelor of Technology

In

Information Technology

### **Submitted By:**

**1.Krunal Kantrodiya**

**2.Dharmesh Narola**

## Under the Guidance of

Prof. (Dr.) Mukesh M. Goswami



##### ***DEPARTMENT OF INFORMATION TECHNOLOGY***

**FACULTY OF TECHNOLOGY, DHARMSINH DESAI UNIVERSITY**

**COLLEGE ROAD, NADIAD- 387001**

**CANDIDATE’S DECLARATION**

We declare that pre-final semester report entitled “Vehicle Number Plate Recognition” is our own work conducted under the supervision of the guide Prof.(Dr.) Mukesh M. Goswami.

We further declare that to the best of our knowledge the report for B.Tech. VII semester does not contain part of the work which has been submitted either in this or any other university without proper citation

Krunal Kantrodiya

Student ID: 15ITUOF079

Dharmesh Narola

Student ID: 15ITUOS067

Submitted To:

Prof. (Dr.) Mukesh M. Goswami

Department of Information Technology,

Faculty of Technology,

Dharmsinh Desai University, Nadiad

Gujarat.

## DHARMSINH DESAI UNIVERSITY

## NADIAD-387001, GUJARAT



**CERTIFICATE**

**This is to certify that the project carried out in the subject of Software Design Project ,entitled “ Vehicle Number Plate Recognition ”and recorded in this report is a bonafide report of work of**

**1) Krunal Kantrodiya , Roll No. IT-49 , ID No. 15ITUOF079**

**2) Dharmesh Narola , Roll No. It-62 , ID No. 15ITUOS067**

**of Department of Information Technology, semester VII. They were involved in Project work during academic year 2018-2019.**

Prof. (Dr.) Mukesh M. Goswami

Department of Information Technology,

Faculty of Technology,

Dharmsinh Desai University, Nadiad

Date:

Prof. (Dr.) V. K. Dabhi

Head, Department of Information Technology,

Faculty of Technology,

Dharmsinh Desai University, Nadiad

Date:

**ACKNOWLEDGMENT**

First, We would like to thank our project advisor Prof.(Dr.) Mukesh M. Goswami for providing us an opportunity to work on this project under his guidance. He has provided valuable guidance, time and resources during different phases of this project. He had always provided motivation and shown utter dedication towards his work. By providing an opportunity to work on this project, he helped us to gain profound understanding of image processing and neural network fundamentals.

It is with great pride and pleasure that we submit this dissertation work as his student.

I would also like to thank Mr. Ronak for giving a second look to our

project and providing valuable suggestion to improve this project. His support during

completion phase of my project had been very instrumental.

Lastly, We would like to thank our friends for providing us constant inspiration and support during various aspects of this project.

**1. Krunal Kantrodiya**

**2. Dharmesh Narola**

**TABLE OF CONTENTS**

**Title Page No**

**ABSTRACT……………………………………………………………………i**

**TABLES………………………………………………………………………iii**

**1. List of Figures…………………………………………………………iv**

**2. List of Tables………………………………………………………….. v**

**CHAPTER**

1. **INTRODUCTION .................................................................................1**
   1. **Project Details ………………………………………………..........1**
   2. **Purpose…………………………………………………….........**
   3. **Scope…………………………………………….**
   4. **Challenges in ALPR………………………………………………**
2. **IMAGE PREPROCESSING…………………………………………**
   1. **Conversion to Gray Scale………………………………………..**
   2. **Blur the Image……………………………………………………**

**2.2.1 Gausian Blur……………………………………………….**

* 1. **Sobel Edge Detection Filter………………….…………………..**
  2. **Thresholding……………………………………………………..**

**2.4.1 Otsu’s Thresholding……………………………………….**

1. **NUMBER PLATE DETECTION…………………………………..**
   1. **getStructuringElement………………………………………….**
   2. **Morphology………………………………………………………**
   3. **Countour…………………………………………………………**
      1. **Selecting The Best Countour……………………………**
      2. **Cropping The Countour…………………………………**
   4. **Character Segmentation………………………………………...**
2. **OPTICAL CHARACTER RECOGNITION……………………...**
   1. **Feedforward Network……………………………………………**
   2. **Backpropagation Network…………………………………….**
3. **ACCURACY…………………………………………………………**
4. **CONCLUSION………………………………………………………**
   1. **Conclusion………………………………………………………..**
   2. **Future Enhancement…………………………………………….**

**APPENDICES ………………………………………………………………**

**REFERENCES………………………………………………………………**

**ABSTRACT**

Day by day we have been heard about the news of vehicle getting stolen from parking or from any other place in the city. So, to keep track of that stolen vehicle we should have to install the CCTV camera on every signal in the city. Also we have to install the number plate detection system which can detect the number plate of every vehicle on the traffic signal. For detecting the number plate from the moving vehicle there are many algorithm has been developed but still this area always remain evolving each every year, In number plate detection system image processing plays an important role, the system consist of basic operations preprocessing, image conversion from RGB to Gray, apply edge detection, apply morphological operators on same image then extract plate region from image and last process the plate region using OCR (optical character recognition).Every algorithm in this category always follows this basic steps, each algorithm has some pros and cons, because same algorithm cannot be useful for different environmental condition. The Algorithm’s efficiency is totally depends upon the quality of input image. E.g. resolution of camera, intensity of the image, illumination of image, shadow effect etc. In this paper we are focusing on which are different algorithm has been developed so far to improve the efficiency of the number plate detection system.

**LIST OF FIGURES**

**LIST OF TABLE**

**Chapter-1**

**INTRODUCTION**

With increasing number of vehicles on roads, it is getting difficult to manually enforce laws and traffic rules for smooth traffic flow. Toll-booths are constructed on freeways and parking structures, where the car has to stop to pay the toll or parking fees. Also, Traffic Management systems are installed on freeways to check for vehicles moving at speeds not permitted by law. All these processes have a scope of improvement. In the center of all these systems lies a vehicle. In order to automate these processes and make them more effective, a system is required to easily identify a vehicle. The important question here is how to identify a particular vehicle? The obvious answer to this question is by using the vehicle’s number plate.

Vehicles in each country have a unique license number, which is written on its license plate. This number distinguishes one vehicle from the other, which is useful especially when both are of same make and model. An automated system can be implemented to identify the license plate of a vehicle and extract the characters from the region containing a license plate. The license plate number can be used to retrieve more information about the vehicle and its owner, which can be used for further processing. Such an automated system should be small in size, portable and be able to process data at sufficient rate.

Various license plate detection algorithms have been developed in past few years. Each of these algorithms has their own advantages and disadvantages. As multiple detections are available for single license plate, post –processing methods are applied to merge all detected regions. In addition, trackers are used to limit the search region to certain areas in an image. In this approach, initial image processing and binarization of an image is carried out based on the contrast between characters and background in license plate. After binarizing the image, it is divided into different black and white regions. These regions are passed through elimination stage to get the final region having most probability of containing a number plate.

* 1. **PROJECT DETAIL**

This project is implemented on Python environment using concept of image processing and machine learning algorithm. Image processing is done using OpenCV

Python library. In image processing section number plate detection from input image of car and character segmentation from detected number plate is done and segmented characters are given as a input to the neural network for predict the character. Neural Network is implemented based on SGD and Back propogation algorithm for predict character. Input of the Neural Network for predicting character is in form of 20\*20 size and output will be predicted character. For Neural Network training dataset is created with each image size 20\*20. And training data is 70% of total data and remaining 30% of total data is used as testing data.

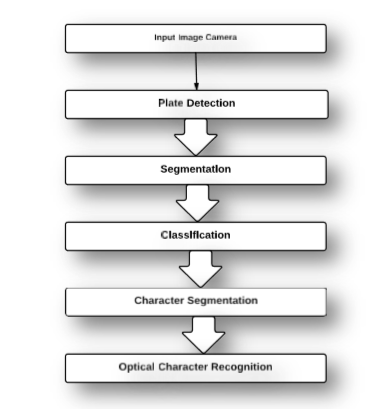


Fig1 Licence Plate Recognition Precess

**1.2 PURPOSE OF THE PROJECT**

The main purpose of this project is to detect a license plate from an image and recognition of number plate. An efficient algorithm is developed to detect a license plate in various luminance conditions. This algorithm extracts the license plate data from an image and provides it as an input to the stage of Car License Plate Recognition. The image of a vehicle is given as an input from the storage memory. Extracted image of the number plate can be seen on computer screen for verification purpose.

* 1. **SCOPE**

The scope of this project is to recognise the license plate from the given

Image of car not for image of bike and observe the output on computer screen. This project can work as a base for future improvements in the field of image processing, especially in license plate extraction and plate number recognition.

* 1. **CHALLANGES IN ALPR**

In the created nations the qualities of the tags are entirely kept up. For instance, the measure of the plate, shade of the plate, text style face/size/shade of every character, dispersing between ensuing characters ,the quantity of lines in the tag, script and so on are kept up particularly. A portion of the pictures of the standard tags utilized as a part of created nations.

Automatic license plate recognition has two major technological requirements

1. The quality of the license plate recognition algorithms.

2. The quality of the image acquisition(camera and the illumination conditions)

The better algorithms are:

1. Higher is the recognition accuracy.

2. Faster is the processing speed.

3. Wider is the range of picture quality it can be used on.

By and large, one LPR programming can read plates from one specific nation just .This is on the grounds that the geometrical structure of the plate and introduction, text styles, and grammar were imperative parts of the system LPR system. Without the earlier information of the plate geometry (character distribution, character spacing, plate color, dimension ratios etc.), the algorithm may out not even find the plate in the captured image.

Furthermore, there are wide variety of plate types:

4

1. Black characters on the white/light background.

2. White characters on the black/dark background.

3. Single row plates.

4. Multi row plates.

If the LPR system cannot utilize such useful information like the plate structure, it loses a helpful aid derived from its input data. This could result in the reduction of the license plate recognition system accuracy. Without using prior information of the plate, the remaining part of the recognition system should be significantly more robust and this leads to more challenges. The image acquisition technique determines the captures image quality of the license plate with which the detection algorithms have to work. Better the quality of the acquired images, higher is the accuracy one can achieve.

A well captured image has the following properties:

1. Good spatial resolution,

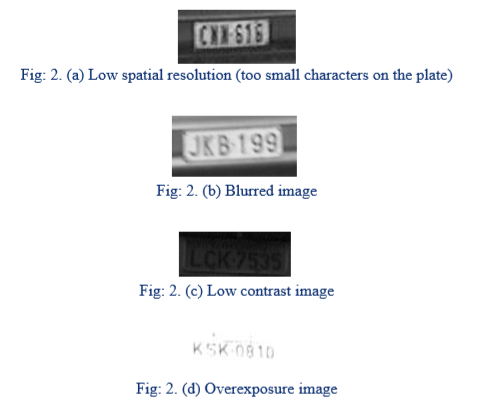
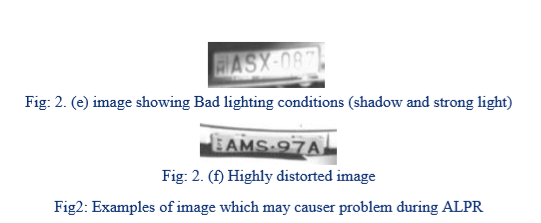
2. Good sharpness,

3. High contrast,

4. Adequate lighting conditions,

5. The Decent angle of view.

Example of some images which may create problem are shown in Fig 2.

**CHAPTER-2**

**IMAGE PREPROCESSING**

**2.1 CONVERSION TO GREY SCALE**

The algorithm described here is independent of the type of colors in image and relies mainly on the gray level of an image for processing and extracting the required information. Color components like Red, Green and Blue value are not used throughout this algorithm. So, if the input image is a colored image represented by 3-dimensional array in Python, it is converted to a 2-dimensional gray image before further processing.

Converts an image from one color space to another.

The function can do the following transformations: CV\_BGR2GRAY, CV\_RGB2GRAY, CV\_GRAY2BGR, CV\_GRAY2RGB

Ex: gray = cv2.cvtColor(imgBlurred, cv2.COLOR\_BGR2GRAY)

The sample of original input image and a gray image is shown below:



Fig2.1: Input Image

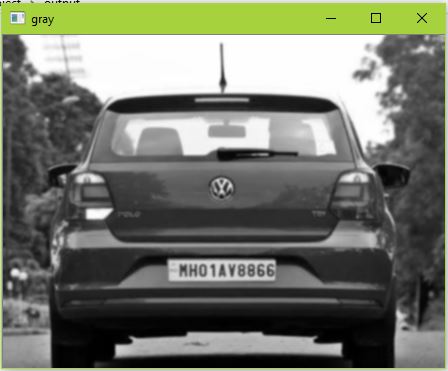


Fig2.2: Gray Image

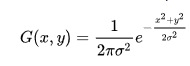
**2.2 BLUR THE IMAGE**

images can be filtered with various low-pass filters(LPF), high-pass filters(HPF) etc. LPF helps in removing noises, blurring the images etc. HPF filters helps in finding edges in the images.

Image blurring is achieved by convolving the image with a low-pass filter kernel. It is useful for removing noises. It actually removes high frequency content (eg: noise, edges) from the image. So edges are blurred a little bit in this operation. (Well, there are blurring techniques which doesn’t blur the edges too).

**2.2.1 Gausian Blur**

In [image processing](https://en.wikipedia.org/wiki/Image_processing), a Gaussian blur (also known as Gaussian smoothing) is the result of blurring an image by a [Gaussian function](https://en.wikipedia.org/wiki/Gaussian_function) . It is a widely used effect in graphics software, typically to reduce [image noise](https://en.wikipedia.org/wiki/Image_noise) and reduce detail. The visual effect of this blurring technique is a smooth blur resembling that of viewing the [image](https://en.wikipedia.org/wiki/Image) through a translucent screen, distinctly different from the [bokeh](https://en.wikipedia.org/wiki/Bokeh) effect produced by an out-of-focus lens or the shadow of an object under usual illumination. Gaussian smoothing is also used as a pre-processing stage in [computer vision](https://en.wikipedia.org/wiki/Computer_vision) algorithms in order to enhance image structures at different scales



where *x* is the distance from the origin in the horizontal axis, *y* is the distance from the origin in the vertical axis, and *σ* is the [standard deviation](https://en.wikipedia.org/wiki/Standard_deviation) of the Gaussian distribution. When applied in two dimensions, this formula produces a surface whose [contours](https://en.wiktionary.org/wiki/contour) are [concentric circles](https://en.wikipedia.org/wiki/Concentric_circles) with a Gaussian distribution from the center point. Values from this distribution are used to build a [convolution](https://en.wikipedia.org/wiki/Convolution) matrix which is applied to the original image. This convolution process is illustrated visually in the figure on the right. Each pixel's new value is set to a [weighted average](https://en.wikipedia.org/wiki/Weighted_average) of that pixel's neighborhood. The original pixel's value receives the heaviest weight (having the highest Gaussian value) and neighboring pixels receive smaller weights as their distance to the original pixel increases. This results in a blur that preserves boundaries and edges better than other.

In this, instead of box filter, gaussian kernel is used. It is done with the function, **cv2.GaussianBlur()**. We should specify the width and height of kernel which should be positive and odd. Gaussian blurring is highly effective in removing gaussian noise from the image.

blur = cv2.GaussianBlur (img , ( 5 , 5 ) , 0 )

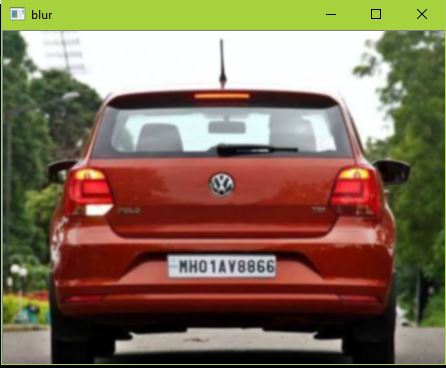
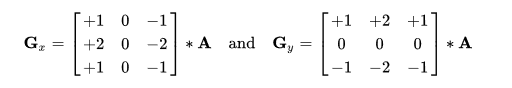


Fig2.3: Blured image

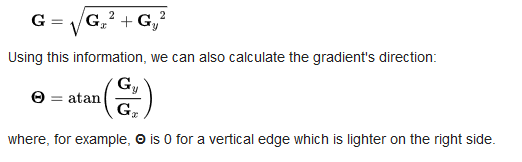
**2.3 SOBEL EDGE DETECTION FILTER**

The Sobel operator, sometimes called the Sobel–Feldman operator or Sobel filter, is used in [image processing](https://en.wikipedia.org/wiki/Image_processing) and [computer vision](https://en.wikipedia.org/wiki/Computer_vision), particularly within [edge detection](https://en.wikipedia.org/wiki/Edge_detection) algorithms where it creates an image emphasising edges.

The operator uses two 3×3 kernels which are [convolved](https://en.wikipedia.org/wiki/Kernel_(image_processing)#Convolution) with the original image to calculate approximations of the [derivatives](https://en.wikipedia.org/wiki/Image_Derivatives) – one for horizontal changes, and one for vertical. If we define A as the source image, and G*x* and G*y* are two images which at each point contain the horizontal and vertical derivative approximations respectively, the computations are as follows:



The *x*-coordinate is defined here as increasing in the "right"-direction, and the *y*-coordinate is defined as increasing in the "down"-direction. At each point in the image, the resulting gradient approximations can be combined to give the gradient magnitude, using:



Convolution is the process of adding each element of the image to its local neighbors, weighted by the kernel.

### Sobel operators is a joint Gausssian smoothing plus differentiation operation, so it is more resistant to noise. You can specify the direction of derivatives to be taken, vertical or horizontal (by the arguments, yorder and xorder respectively). You can also specify the size of kernel by the argument ksize. If ksize = -1, a 3x3 Scharr filter is used which gives better results than 3x3 Sobel filter.

### Sobel = cv2.Sobel (gray, cv2.CV\_8U ,1 ,0 , ksize=3)

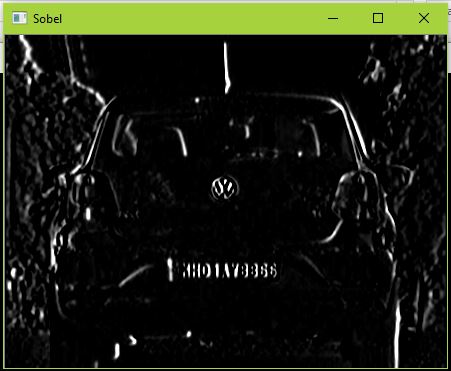


Fig2.4: Sobel Filter On Image

**2.4 THRESHOLDING**

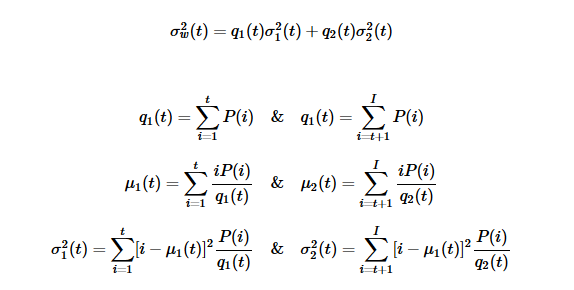
Here, the matter is straight forward. If pixel value is greater than a threshold value, it is assigned one value (may be white), else it is assigned another value (may be black). The function used is **cv2.threshold**. First argument is the source image, which **should be a grayscale image**. Second argument is the threshold value which is used to classify the pixel values. Third argument is the maxVal which represents the value to be given if pixel value is more than (sometimes less than) the threshold value. OpenCV provides different styles of thresholding and it is decided by the fourth parameter of the function. Different types are:

* cv2.THRESH\_BINARY
* cv2.THRESH\_BINARY\_INV
* cv2.THRESH\_TRUNC
* cv2.THRESH\_TOZERO
* cv2.THRESH\_TOZERO\_INV

**2.4.1 Otsu’s Thresolding**

The algorithm assumes that the image contains two classes of pixels following bi-modal histogram (foreground pixels and background pixels), it then calculates the optimum threshold separating the two classes so that their combined spread (intra-class [variance](https://en.wikipedia.org/wiki/Variance)) is minimal, or equivalently (because the sum of pairwise squared distances is constant), so that their inter-class variance is maximal.

Since we are working with bimodal images, Otsu's algorithm tries to find a threshold value (t) which minimizes the weighted within-class variance given by the relation :



It actually finds a value of t which lies in between two peaks such that variances to both classes are minimum.

For this, our cv2.threshold() function is used, but pass an extra flag, cv2.THRESH\_OTSU. **For threshold value, simply pass zero**. Then the algorithm finds the optimal threshold value and returns you as the second output, retVal. If Otsu thresholding is not used, retVal is same as the threshold value you used.

Ex: ret2,threshold\_img = cv2.threshold(sobelx,0,255,cv2.THRESH\_BINARY+cv2.THRESH\_OTSU)



Fig2.5: Otsu’s Threshold on image

**CHAPTER 3**

**NUMBER PLATE DETECTION**

The regions having the highest probability of containing the license plate are segmented out and the coordinates of such regions are stored in an array.

**3.1 STRUCTURING ELEMENT**

The structuring element is a small binary image, i.e. a small matrix of pixels, each with a value of zero or one:

* The matrix dimensions specify the *size* of the structuring element.
* The pattern of ones and zeros specifies the *shape* of the structuring element.
* An *origin* of the structuring element is usually one of its pixels, although generally the origin can be outside the structuring element.

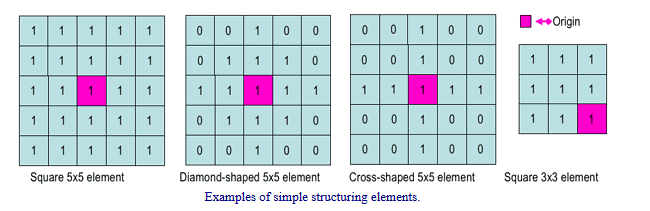


Fig3.1 Example of simple structuring elements

A common practice is to have odd dimensions of the structuring matrix and the origin defined as the centre of the matrix. Stucturing elements play in moprphological image processing the same role as convolution kernels in linear image filtering.

When a structuring element is placed in a binary image, each of its pixels is associated with the corresponding pixel of the neighbourhood under the structuring element. The structuring element is said to fit the image if, for each of its pixels set to 1, the corresponding image pixel is also 1. Similarly, a structuring element is said to hit, or intersect, an image if, at least for one of its pixels set to 1 the corresponding image pixel is also 1.

## getStructuringElement

Returns a structuring element of the specified size and shape for morphological operations.

cv2.getStructuringElement(shape, ksize[, anchwor]) → retval

 **shape** –

Element shape that could be one of the following:

* MORPH\_RECT - a rectangular structuring element:
* MORPH\_ELLIPSE - an elliptic structuring element, that is, a filled ellipse inscribed into the rectangle Rect(0, 0, esize.width, 0.esize.height)
* MORPH\_CROSS - a cross-shaped structuring element.
* CV\_SHAPE\_CUSTOM - custom structuring element (OpenCV 1.x API)

 **ksize** – Size of the structuring element.

# Ex: element = cv2.getStructuringElement(shape=cv2.MORPH\_RECT, ksize=(17, 3))

The function constructs and returns the structuring element that can be further passed to [createMorphologyFilter()](https://docs.opencv.org/2.4/modules/imgproc/doc/filtering.html#Ptr<FilterEngine> createMorphologyFilter(int op, int type, InputArray kernel, Point anchor, int rowBorderType, int columnBorderType, const Scalar& borderValue)), [erode()](https://docs.opencv.org/2.4/modules/imgproc/doc/filtering.html#void erode(InputArray src, OutputArray dst, InputArray kernel, Point anchor, int iterations, int borderType, const Scalar& borderValue)), [dilate()](https://docs.opencv.org/2.4/modules/imgproc/doc/filtering.html#void dilate(InputArray src, OutputArray dst, InputArray kernel, Point anchor, int iterations, int borderType, const Scalar& borderValue)) or [morphologyEx()](https://docs.opencv.org/2.4/modules/imgproc/doc/filtering.html#void morphologyEx(InputArray src, OutputArray dst, int op, InputArray kernel, Point anchor, int iterations, int borderType, const Scalar& borderValue)) . But you can also construct an arbitrary binary mask yourself and use it as the structuring element.

**3.2 MORPHOLOGY**

Morphological operators often take a [binary image](http://homepages.inf.ed.ac.uk/rbf/HIPR2/binimage.htm) and a [structuring element](http://homepages.inf.ed.ac.uk/rbf/HIPR2/strctel.htm) as input and combine them using a set operator (intersection, union, inclusion, complement). They process objects in the input image based on characteristics of its shape, which are encoded in the structuring element.

Usually, the structuring element is sized 3×3 and has its origin at the center pixel. It is shifted over the image and at each pixel of the image its elements are compared with the set of the underlying pixels. If the two sets of elements match the condition defined by the set operator (e.g. if the set of pixels in the structuring element is a subset of the underlying image pixels), the pixel underneath the origin of the structuring element is set to a pre-defined value (0 or 1 for binary images). A morphological operator is therefore defined by its structuring element and the applied set operator.

For the basic morphological operators the [structuring element](http://homepages.inf.ed.ac.uk/rbf/HIPR2/strctel.htm) contains only foreground pixels (i.e. ones) and `don't care's'. These operators, which are all a combination of [erosion](http://homepages.inf.ed.ac.uk/rbf/HIPR2/erode.htm) and [dilation](http://homepages.inf.ed.ac.uk/rbf/HIPR2/dilate.htm), are often used to select or suppress features of a certain shape, e.g. removing noise from images or selecting objects with a particular direction.

The more sophisticated operators take zeros as well as ones and `don't care's' in the structuring element. The most general operator is the [hit and miss](http://homepages.inf.ed.ac.uk/rbf/HIPR2/hitmiss.htm), in fact, all the other morphological operators can be deduced from it. Its variations are often used to simplify the representation of objects in a (binary) image while preserving their structure, e.g. producing a skeleton of an object using [skeletonization](http://homepages.inf.ed.ac.uk/rbf/HIPR2/skeleton.htm) and tidying up the result using [thinning](http://homepages.inf.ed.ac.uk/rbf/HIPR2/thin.htm).

Morphological operators can also be applied to [graylevel images](http://homepages.inf.ed.ac.uk/rbf/HIPR2/gryimage.htm), e.g. to reduce [noise](http://homepages.inf.ed.ac.uk/rbf/HIPR2/noise.htm) or to brighten the image. However, for many applications, other methods like a more general [spatial filter](http://homepages.inf.ed.ac.uk/rbf/HIPR2/filtops.htm) produces better results.

Morphology is done using following function.

cv2.morphologyEx(src=threshold\_img, op=cv2.MORPH\_CLOSE, kernel=element, dst=morph\_img\_threshold)

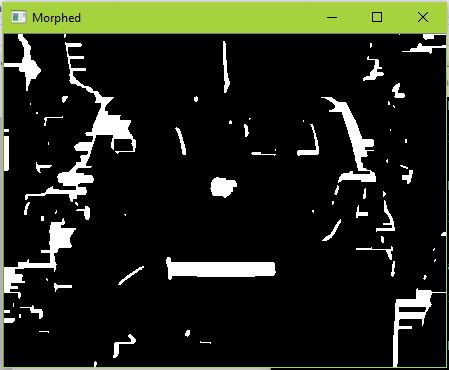


Fig3.2: Morphed Image

**3.3 COUNTOURS**

The minimum or the smallest bounding or the enclosing box for any point set in N dimensions is the box with the smallest measure within which all points lie. When the other kinds of measure are used, the minimum box is usually called accordingly depending on the measure used.eg. “Minimum-perimeter bounding box”. The minimum bounding box of any point set is same as the minimum bounding box of its convex hull, this is a fact which can be used heuristically to speed up computation. The term “box”/”hyper rectangle” has come from its usage in Cartesian coordinate system, where it can be indeed visualized as a rectangle, rectangular parallelepiped etc. In the case of two-dimensional it is called the minimum bounding rectangle. In other words it is a rectangle which has the minimum height and which that covers all the pixels present in a particular connected component or region.

Contours can be explained simply as a curve joining all the continuous points (along the boundary), having same color or intensity. The contours are a useful tool for shape analysis and object detection and recognition.

* For better accuracy, use binary images. So before finding contours, apply threshold or canny edge detection.
* findContours function modifies the source image. So if you want source image even after finding contours, already store it to some other variables.
* In OpenCV, finding contours is like finding white object from black background. So remember, object to be found should be white and background should be black.

**im2,contours, hierarchy= cv2.findContours(morph\_img\_threshold,mode=cv2.RETR\_EXTERNAL,method=cv2.CHAIN\_APPROX\_SIMPLE)**

An image with contouring of captured image is shown in Fig.3.3



Fig 3.3: detected countours in image

**3.3.1 Selecting Best Countour**

In a figure there can be many bounding boxes identified but to select the best possible candidate for license plate region requires to identify few properties of the license plates as discussed below:

1. Contrast present in the Contours: As the license plate consists of dark coloured numbers to the lighter background .The centre row of the box can be scanned and total number sudden contrast change can be recorded, if the image is binarized. The box having the maximum changes in contrast within the box is the possible candidate for the license plate.

2. Aspect Ratio: The aspect ratio of an image is the ratio of the width of the image to its height. The inverse of the aspect ratio should be less than 1 for any license plate. Hence, all the regions which don’t satisfy this property can be rejected.

3. Width of license plate: The width of the license plate region also has a threshold limit.it cannot be more than some threshold value. In this project after analysing various dataset we have used the threshold limit.

**3.3.2**  **Cropping the Contour**

After identifying the best possible Contour candidate for the license plate the coordinates of the contour are noted and the box is cropped from the image and sent to character segmentation module for the further processing as shown in figure 3.4:



Fig3.4: Cropped countour from image

**3.4 CHARACTER SEGMENTATION**

The character segmentation process acts as bridge between plate detection and optical character recognition modules. Its main function is to segment the characters in the selected candidate region (extracted license plate) such that each character can be sent to the optical character recognition module individually for recognition.

Normalized or standardized are of a fancy format the conditions of the license plates are important criteria for efficient segmentation because if numbers are of a fancy format the conditions of the license plate as described .Once the license plate is localized we proceed to obtain the individual characters .A license plate as described above has high intensity variation regions. This forms the basis for character segmentation. Sometimes it is observed that along with license numbers, various texts may be present, which have to be removed. By various observation we observed that for the license plate regions the amount of white on black is specific for the number regions and falls within a certain range.

Morphological technique are used to remove small white areas which escape range corrections. Finally individual characters are extracted to pass on through the optical character recognition system. Segmentation is one of the most important processes in the automatic number plate recognition, because all further steps rely on it. If the segmentation fails, a character can be improperly divided in two pieces, or two characters can be wronged merged together which would lead to the failure of following stages of recognition.

The second phase of the segmentation is an enhancement of segments. The segment of a plate contains besides the character also undesirable elements such as noise due to shadows or defects in camera equipment as well as redundant space on the sides of characters

The character segmentation process takes the extracted license plate region from the preceding module as the input .The input is a coloured JPEG image. For our process, we work only binary images and thus the first part of segmentation is binarisation of the image as shown in Fig 3.6. Fig 3.7 shows character segmented image.

Fig 3.5 localised plate Fig3.6 : Binarised plate



Fig 3.7 Segmented character from image

**CHAPTER 4**

**OPTICAL CHARACTER RECOGNITION**

The neural networks are typically made up of many artificial neurons. An artificial neurons. An artificial neuron is an analogy to biological neuron. It is simply electronically modelled to the biological neuron. The number of neurons that are used depends on the task at hand. The number of neurons used can be few as two or three or large as two or several thousand. There are many ways of connecting artificial neurons together to form a neural network. Some of the ways are discussed below.

**4.1 FEEDFORWARD NETWORK**

**Feedforward Neural Networks** are [artificial neural networks](https://brilliant.org/wiki/artificial-neural-network/) where the connections between units do not form a [cycle](https://brilliant.org/wiki/graphs/##graphs-basic). Feedforward neural networks were the first type of artificial neural network invented and are simpler than their counterpart, [recurrent neural networks](https://brilliant.org/wiki/recurrent-neural-network/). They are called feedforward because information only travels forward in the network (no loops), first through the input nodes, then through the [hidden nodes](https://brilliant.org/wiki/artificial-neural-network/#putting-it-all-together) (if present), and finally through the output nodes.

In feedforward Neural Network each input into the neuron has its own weight associated with it. A weight can simply be a floating point no. and it is these that we adjust when we come to train the network. The weights in most of the neural networks can be both negative and positive, therefore, it helps in providing excitory or else inhibitory influences to each input. As each of the input enters the nucleus it is then multiplied by its weight. The nucleus sums up all these new input values and gives us the activation which is again a floating point no. and can be negative or positive. The threshold value is decided and if the activation value is greater than a threshold value the neuron outputs 1(considering these are two outcomes 1 and 0 to the input) as a signal. If the activation is less than the threshold value the neuron then outputs zero.

A neuron can take any number of inputs from one to n, here n is the total number of inputs. The inputs, therefore, may be represented as x1,x2,x3…xn. The corresponding weights for the inputs can be represented as w1,w2,w3,w3..wn. The weighted sum of the links and its corresponding weights is called the activation value as discussed above.

a = x1w1+x2w2+x3w3…xnwn where, a is the activation value.

Each of the input is sent to every neuron of the hidden layer and each hidden layers neurons output is connected to every neuron of the next layer. There is no predefined number of the neurons to be present in a particular layer it can be arbitrary and it totally depends on the problem. An image with functionality of neural networking is shown in Fig 4.1.

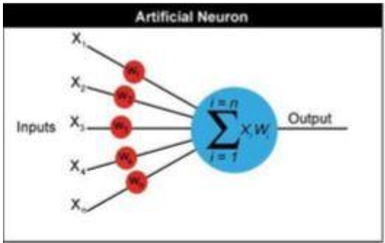


Fig 4.1: Functional diagram of Neural Networking

**4.2 BACKPROPAGATION NETWORK**

Backpropagation is a method used in [artificial neural networks](https://en.wikipedia.org/wiki/Artificial_neural_network) to calculate a [gradient](https://en.wikipedia.org/wiki/Gradient) that is needed in the calculation of the [weights](https://en.wikipedia.org/wiki/Artificial_neural_network#Components_of_an_artificial_neural_network) to be used in the network. It is commonly used to train [deep neural networks](https://en.wikipedia.org/wiki/Deep_neural_network), a term referring to neural networks with more than one hidden layer.

Backpropagation is a special case of a more general technique called [automatic differentiation](https://en.wikipedia.org/wiki/Automatic_differentiation). In the context of learning, backpropagation is commonly used by the [gradient descent](https://en.wikipedia.org/wiki/Gradient_descent) optimization algorithm to adjust the weight of neurons by calculating the [gradient](https://en.wikipedia.org/wiki/Gradient) of the [loss function](https://en.wikipedia.org/wiki/Loss_function). This technique is also sometimes called backward propagation of errors, because the error is calculated at the output and distributed back through the network layers.

A back propagation networks learns by example various sets of datasets are provided as input. The various inputs provided helps the network to calculate and recalculate the networks weight value so that when the network is trained it can give the required output. Fig 4.2 shows declaration of layers for neural networking. The network is initialized by first setting random weights which generally have very small value such as values between -1 and 1. There are two passes in the Back Propagation Algorithm. After the networks is setup with the random weights the output is calculated this is called the forward pass the result obtained in the forward pass may not be equal to the required result or the target and so the error is calculated for each neuron which is Target-Actual Output. The error calculated for each neuron is then mathematically used to change the weights so the next time the forward pass will have minimum the error. The character is recognized after training the network with various datasets of the particular character to get maximum accuracy and minimum error.

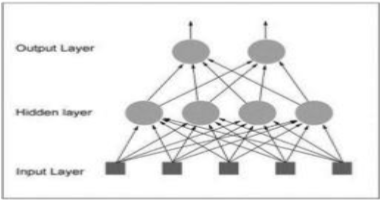


Fig 4.2: Layers Declaration for neural networking

**CHAPTER 5**

**ACCURACY**

This chapter explained about the results achieved using the developed algorithm for License Plate Recognition. Number plate detection algorithm is implemented using python image processing and character recognition is implemented on the concept of machine learning in python using feed forward and back propagation neural network algorithm

Table 5.1 Accuracy of Models

|  |  |
| --- | --- |
| Number Plate Detection | 93 % |
| Neural Network Testing | 88 % |

**CHAPTER 6**

**CONCLUSION**

**6.1 CONCLUSION**

The sole objective of this algorithm was to resolve the computational and mathematical complexities of ANPR. By addressing the logical sequences of the problem statement this paper has proposed an algorithm that easily bypasses the computational cost and complexities of machine learning and character recognition approach. By using pre-defined functions available for pre-processing in Python computational cost has been reduced. The algorithm presented in this paper is successful for most images, acquired under different conditions. This algorithm overcomes the drawbacks of the previous proposed algorithms with lesser complex and easily maintainable approach.

**6.2 FUTURE ENHANCEMENT**

In further we will have to find more accuracy. This algorithm fails to detect number plates under changing illuminations. Further research on this algorithm will be based on that. In future we want to add feature that is able for detecting 2 wheeler’s number plate also. We further want to extend the ambit of this work in real time taking video input and directly connecting it to the database for better policing and surveillance.

**APPENDICES**

**Code for image preprocessing**

**main.py**

import numpy as np

import cv2

import network

def preprocess(img):

cv2.imshow("Input",img)

imgBlurred = cv2.GaussianBlur(img, (5,5), 0)

cv2.imshow("blur",imgBlurred)

cv2.waitKey(0)

gray = cv2.cvtColor(imgBlurred, cv2.COLOR\_BGR2GRAY)

cv2.imshow("gray",gray)

cv2.waitKey(0)

sobelx = cv2.Sobel(gray,cv2.CV\_8U,1,0,ksize=3)

cv2.imshow("Sobel",sobelx)

cv2.waitKey(0)

ret2,threshold\_img = cv2.threshold(sobelx,0,255,cv2.THRESH\_BINARY+cv2.THRESH\_OTSU)

cv2.imshow("Threshold",threshold\_img)

cv2.waitKey(0)

return threshold\_img

def cleanPlate(plate):

print ("CLEANING PLATE. . .")

gray = cv2.cvtColor(plate, cv2.COLOR\_BGR2GRAY)

\_, thresh = cv2.threshold(gray, 0, 255, cv2.THRESH\_BINARY\_INV | cv2.THRESH\_OTSU)

temp=thresh.copy()

character\_dimensions = (0.30\*plate.shape[0], 0.80\*plate.shape[0], 0.02\*plate.shape[1], 0.15\*plate.shape[1])

min\_height, max\_height, min\_width, max\_width = character\_dimensions

im1,contours,hierarchy = cv2.findContours(temp,cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)

rects = [cv2.boundingRect(ctr) for ctr in contours]

characters = []

if contours:

for rect in reversed(rects):

if (rect[2] < rect[3] and rect[2]<max\_width and rect[3]>min\_height and rect[3]<max\_height):

cleaned\_final = plate[rect[1]:rect[1]+rect[3], rect[0]:rect[0]+rect[2]]

gray\_image = cv2.cvtColor(cleaned\_final, cv2.COLOR\_BGR2GRAY)

resized\_img=cv2.resize(gray\_image,(20,20))

cv2.imshow("resized",resized\_img)

cv2.waitKey(0)

cv2.rectangle(plate, (rect[0], rect[1]), (rect[0] + rect[2], rect[1] + rect[3]), (0, 255, 0),1)

flat\_bin\_image = np.reshape(resized\_img,(400,1))

flat\_bin\_image=flat\_bin\_image.astype(np.float32)/255

characters.append(flat\_bin\_image)

cv2.imshow("Function Test",resized\_img)

cv2.waitKey(0)

height, width = plate.shape[:2]

return thresh,[0,0,width,height],np.array(characters)

else:

return plate,None,None

def extract\_contours(threshold\_img):

element = cv2.getStructuringElement(shape=cv2.MORPH\_RECT, ksize=(17, 3))

morph\_img\_threshold = threshold\_img.copy()

cv2.morphologyEx(src=threshold\_img, op=cv2.MORPH\_CLOSE, kernel=element, dst=morph\_img\_threshold)

cv2.imshow("Morphed",morph\_img\_threshold)

cv2.waitKey(0)

im2,contours, hierarchy= cv2.findContours(morph\_img\_threshold,mode=cv2.RETR\_EXTERNAL,method=cv2.CHAIN\_APPROX\_SIMPLE)

return contours

def ratioCheck(area, width, height):

ratio = float(width) / float(height)

if ratio < 1:

ratio = 1 / ratio

aspect = 4.7272

min = 15\*aspect\*15

max = 125\*aspect\*125

rmin = 3

rmax = 6

if (area < min or area > max) or (ratio < rmin or ratio > rmax):

return False

return True

def isMaxWhite(plate):

avg = np.mean(plate)

if(avg>=115):

return True

else:

return False

def validateRotationAndRatio(rect):

(x, y), (width, height), rect\_angle = rect

if(width>height):

angle = -rect\_angle

else:

angle = 90 + rect\_angle

if angle>15:

return False

if height == 0 or width == 0:

return False

area = height\*width

if not ratioCheck(area,width,height):

return False

else:

return True

def cleanAndRead(img,contours,net):

for i,cnt in enumerate(contours):

min\_rect = cv2.minAreaRect(cnt)

if validateRotationAndRatio(min\_rect):

x,y,w,h = cv2.boundingRect(cnt)

plate\_img = img[y:y+h,x:x+w]

if(isMaxWhite(plate\_img)):

clean\_plate, rect, characters = cleanPlate(plate\_img) cv2.imshow("clean",clean\_plate)

cv2.waitKey(0)

if rect:

x1,y1,w1,h1 = rect

x,y,w,h = x+x1,y+y1,w1,h1

cv2.imshow("Cleaned Plate",clean\_plate)

cv2.waitKey(0)

print ("Detected Text : ")

predicted\_char=""

for x in characters:

predicted\_char = predicted\_char+ net.predict(x)

print(predicted\_char)

cv2.rectangle(img , (x,y), (x+w,y+h), (0,0,255),1)

cv2.imshow("ori",img)

cv2.waitKey(0)

if \_\_name\_\_ == '\_\_main\_\_':

print ("DETECTING PLATE . . .")

img = cv2.imread("C:/Users/Admin/Desktop/project/ml\_code/car\_image/45.JPEG")

cv2.imshow("Input",img)

net = network.Network([400, 100, 36])

threshold\_img = preprocess(img)

contours= extract\_contours(threshold\_img)

cleanAndRead(img,contours,net)

**Network.py**

import random

import pickle

import numpy as np

class Network(object):

def \_\_init\_\_(self, sizes):

self.num\_layers = len(sizes)

self.sizes = sizes

with open('C:/Users/Admin/Desktop/project/ml\_code/ml\_models/model1.pkl', 'rb') as input:

net=pickle.load(input)

self.biases= net.biases

self.weights= net.weights

def feedforward(self, a):

for b, w in zip(self.biases, self.weights):

a = sigmoid(np.dot(w, a)+b)

return a

def predict(self, a):

for b, w in zip(self.biases, self.weights):

a=sigmoid(np.dot(w ,a)+b)

ind=np.argmax(a)

if(ind>=0 and ind <=9):

text=ind+48

else:

text=ind+65-10

return chr(text)

def SGD(self, training\_data, epochs, mini\_batch\_size, eta,

test\_data=None):

if test\_data:

test\_data = list(test\_data)

n\_test = len(test\_data)

#n\_test = len(test\_data)

training\_data = list(training\_data)

n = len(training\_data)

for j in range(epochs):

random.shuffle(training\_data)

mini\_batches = [ training\_data[k:k+mini\_batch\_size] for k in range(0, n, mini\_batch\_size)]

for mini\_batch in mini\_batches:

self.update\_mini\_batch(mini\_batch, eta)

if test\_data:

print ("Epoch {0}: {1} / {2}".format(

j, self.evaluate(test\_data), n\_test))

else:

print ("Epoch {0} complete".format(j))

def update\_mini\_batch(self, mini\_batch, eta):

nabla\_b = [np.zeros(b.shape) for b in self.biases]

nabla\_w = [np.zeros(w.shape) for w in self.weights]

for x, y in mini\_batch:

delta\_nabla\_b, delta\_nabla\_w = self.backprop(x, y)

nabla\_b = [nb+dnb for nb, dnb in zip(nabla\_b, delta\_nabla\_b)]

nabla\_w = [nw+dnw for nw, dnw in zip(nabla\_w, delta\_nabla\_w)]

self.weights = [w-(eta/len(mini\_batch))\*nw

for w, nw in zip(self.weights, nabla\_w)]

self.biases = [b-(eta/len(mini\_batch))\*nb

for b, nb in zip(self.biases, nabla\_b)]

def backprop(self, x, y):

nabla\_b = [np.zeros(b.shape) for b in self.biases]

nabla\_w = [np.zeros(w.shape) for w in self.weights]

activation = x

activations = [x] # list to store all the activations, layer by layer

zs = [] # list to store all the z vectors, layer by layer

for b, w in zip(self.biases, self.weights):

z = np.dot(w, activation)+b

zs.append(z)

activation = sigmoid(z)

activations.append(activation)

# backward pass

delta = self.cost\_derivative(activations[-1], y) \* \

sigmoid\_prime(zs[-1])

nabla\_b[-1] = delta

nabla\_w[-1] = np.dot(delta, activations[-2].transpose())

for l in range(2, self.num\_layers):

z = zs[-l]

sp = sigmoid\_prime(z)

delta = np.dot(self.weights[-l+1].transpose(), delta) \* sp

nabla\_b[-l] = delta

nabla\_w[-l] = np.dot(delta, activations[-l-1].transpose())

return (nabla\_b, nabla\_w)

def evaluate(self, test\_data):

test\_results = [(np.argmax(self.feedforward(x)), y)

for (x, y) in test\_data]

cnt=0

for x,y in test\_results:

if(y>='A' and y<='Z'):

t=ord(y)-65+10

if(y>='0' and y<='9'):

t=ord(y)-48

if(x==t):

cnt+=1

return cnt

def cost\_derivative(self, output\_activations, y):

return (output\_activations-y)

def sigmoid(z):

return 1.0/(1.0+np.exp(-z))

def sigmoid\_prime(z):

return sigmoid(z)\*(1-sigmoid(z))

**Dataloader.py**

import \_pickle as cPickle

import gzip

import cv2

import os

import numpy as np

from skimage.filters import threshold\_otsu

def get\_root\_directory():

current\_dir = os.path.dirname(os.path.realpath(\_\_file\_\_))

dir\_split = os.path.split(current\_dir)

root\_directory = dir\_split[0]

return root\_directory

def read\_training\_data(training\_directory):

letters = [

'0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'A', 'B', 'C', 'D',

'E', 'F', 'G', 'H', 'J', 'K', 'L', 'M', 'N', 'P', 'Q', 'R', 'S', 'T',

'U', 'V', 'W', 'X', 'Y', 'Z'

]

image\_data = []

image\_testdata = []

target\_data = []

target\_testdata = []

for each\_letter in letters:

for each in range(10):

img\_details = cv2.imread(training\_directory+'/'+each\_letter+'/'+each\_letter+'\_'+str(each)+'.jpg', cv2.IMREAD\_GRAYSCALE)

flat\_bin\_image = np.reshape(img\_details,(400,1))

flat\_bin\_image=flat\_bin\_image.astype(np.float32)/255

if(each >= 7 and each <=9):

image\_testdata.append(flat\_bin\_image)

target\_testdata.append(each\_letter)

else:

image\_data.append(flat\_bin\_image)

target\_data.append(each\_letter)

return (np.array(image\_data), np.array(target\_data),np.array(image\_testdata), np.array(target\_testdata))

def load\_data():

root\_directory = get\_root\_directory()

training\_20X20\_dir = os.path.join(root\_directory, 'training\_data', ' train20X20')

tr\_d,tr\_dl,te\_d,te\_dl = read\_training\_data(training\_20X20\_dir)

training\_results = [vectorized\_result(y) for y in tr\_dl]

training\_data = zip(tr\_d, training\_results)

test\_data = zip(te\_d, te\_dl)

print(training\_data)

return (training\_data, test\_data)

def vectorized\_result(j):

if(j>='A' and j<='Z'):

j=ord(j)-65+10

e = np.zeros((36, 1))

e[int(j)] = 1

return e