Automatic License Plate Detection and Recognition using Deep Learning and Image Processing

Pradyut Agrawal[1], Akshansh Jha[2], Ravneet Kaur[2], Anju Agrawal[2], Monika Bhattacharya[2,*]

[1]*Division of Electronics & Communication Engineering, Netaji Subhash Institute of Technology (University of Delhi), New Delhi-110078*
[2]*Device Modeling & Research Laboratory, Department of Electronics, Acharya Narendra Dev College, University of Delhi, New Delhi- 110019*

Abstract

From traffic management to license plate scanning, the field of traffic regulation is fraught with difficulties that need to be addressed with innovative solutions. Manual tracking infractions of traffic laws is conceivable, but it requires a substantial amount of manpower to monitor all vehicles and their license plates. When automobiles are travelling fast, the license photographs become blurry and this method becomes less efficient. In addition, it is difficult for toll collectors and traffic controllers to physically check license plate numbers at each and every toll gate or traffic post for stolen vehicles or vehicles that breach traffic laws. Maintaining records of several hundred vehicles becomes impractical and renders it nearly impossible to establish a coherent tracking system.

This paper discusses these problems and offers a novel system that dramatically streamlines and improves the efficiency with which traffic rule violations and license plate detection are recorded. The system uses deep learning and image processing to improve license plate detection. Websites having public databases of stolen cars were also scraped and was utilized to create a new database in the proposed system. Once a license plate is identified, a robust OCR (*Optical Character Recognition*) model is used to extract the text from the license plate, which is then compared with the newly created database values of stolen vehicles using cosine similarity of the letters and digits found in the identified license plate.

Keywords

License plate detection, OCR, Image processing, Deep Learning.[1]

Symposium on Computing & Intelligent Systems (SCI), *May 10, 2024, New Delhi, INDIA*
* Corresponding author.
† These authors contributed equally.
pradyut.agrawal18@gmail.com (P. Agrawal); akshansh.jha31@gmail.com (A. Jha); ravneetkaur@andc.du.ac.in (R. Kaur) ; anjuagrawal@andc.du.ac.in (A. Agrawal); monikabhattacharya@andc.du.ac.in* (M. Bhattacharya)
© 2024 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR-WS.org/Vol-3682/Paper1.pdf

CEUR Workshop Proceedings ceur-ws.org ISSN 1613-0073

1. Introduction

Vehicles, as the primary mode of transportation in today's industrialized society, are integral to virtually every facet of human endeavor. Infractions like speeding and running red lights will become more common as the number of vehicles on the road continues to rise rapidly. If drivers had to rely exclusively on numbered traffic officers to prevent the vast number of daily traffic violations, public transportation would grind to a standstill. Each vehicle has its own set of details represented by its License Plate (LP) [1]. Its likeness is a valuable tool for reaching its owner and exchanging data. As a result, pictures are frequently employed as the first means of determining a person's or vehicles identify. Furthermore, image analysis technology was already deeply established in almost every aspect of human activity. As a result, automatic collection and management of LP data from digital photographs has evolved as a useful tool for public transportation surveillance [2, 3].

The problem of manually identifying vehicles can be resolved through the implementation of Automatic License Plate Recognition (ALPR) systems. The ALPR is used for vehicle identification in a variety of traffic-related applications, such as toll management booths, airports, cargo areas, parking lot access validation, highways, and the detection of stolen vehicles. Recent developments in Parallel Processing and Deep Learning (DL) have aided many computer vision applications, including object detection/identification and optical character recognition (OCR)[4], and ALPR systems are no exception. By employing an ALPR system equipped with machine learning tools, it becomes possible to eliminate the need for manual work and labor-intensive tasks associated with tracking LPs, recording their numbers, and cross-referencing them with a database of stolen cars or vehicles violating traffic rules.

The proposed approach combines state-of-the-art machine learning techniques with various Image Processing (IP) techniques to achieve higher accuracy, reduced redundancy, and produce clear, sensible outputs for LP detection [5]. Once a LP is detected, a robust OCR model is employed to extract the text from the identified LP. This extracted text is then compared with values in the database using cosine similarity of the detected plate's letters and numbers. The system generates a list of all suspected plates based on the similarity ratio. In order to enhance user-friendliness, a Graphical User Interface (GUI) has been developed, enabling non-technical users to easily navigate the system. The GUI includes interactive buttons for loading and predicting images, as well as a button to check the current status of the database(s) [6].

2. Literature Review

The early development of automatic LP detection can be traced back to 1970 in the United Kingdom at the Police Development Branch. The initial design was established in 1979 by two companies, Computer Recognition System and EMI Electronics, located in Wokingham, UK. However, significant advancements in this field started emerging after the 1990s with the introduction of advanced and cost-effective technologies. LP detection and recognition have been extensively researched, resulting in numerous models and approaches.

In the approach mentioned in [7], a robust technique utilizing deep neural networks is employed for LP detection in images. The detected LPs are then pre-processed and subjected to License Plate Recognition (LPR) using the LSTM Tesseract OCR Engine. In a paper by Hamidreza & Kasaei [8], a real-time LP detection and recognition model is developed based on morphology and template matching techniques. Another approach by Serkan O [9] involves utilizing edge detection algorithms and smearing algorithms for LP extraction. In a study conducted by R. Babu [10], a LPR model was created using the YOLOV3 object detection algorithm. However, YOLOV3 had limitations due to a training bias, as it could only detect objects in similar scales as it was trained on. YOLOV5, on the other hand, addressed this bias by utilizing CSP nets, representing a significant upgrade over YOLOV3 [11] in terms of performance.

In the projected work an advanced system for LP detection and recognition that leverages the power of DL and IP, resulting in a more accurate and intelligent approach is presented. The suggested system integrates state-of-the-art technologies to improve the speed and accuracy of LP detection and recognition in real time, which is useful for monitoring traffic, conducting surveys, and other similar applications.

3. Methodology & Modeling Approaches

The design of a LPR System involves three key stages that encompass the complete process flow:

i. Plate Localization and Resizing,
ii. Normalization
iii. Character Recognition

3.1 Plate Localization and Resizing

The key step in the license plate recognition system is to obtain localized regions of the plate. Numerous algorithms have been devised in the past to achieve optimal plate localization. All these techniques involve two main processes:

- **Vertical Edge Detection**- which is a technique to detect vertical edges. It is implemented through spatial filtering between a predefined mask and the image, which is obtained after binarization. A mask of size 3x3 used for edge detection is shown in Fig. 1

-1	0	1
-1	0	1
-1	0	1

Fig. 1. Edge Detector Mask

- **Adaptive Thresholding** - which calculates the threshold values of smaller regions. This method is useful where different regions might have different threshold values unlike

normal thresholding, where threshold value is global. To get a binary image b(x,y), a threshold of T(x,y) is applied [12] where

$$b(x,y) = f(x) = \begin{cases} 0, & \text{if } I(x,y) \leq T(x,y) \\ 1, & \text{Otherwise} \end{cases}$$

Two approaches have been developed in the past for achieving Adaptive Thresholding:

a) **The Chow and Kaneko Algorithm** which divides the image into a series of overlapping sub-images and calculates the optimal threshold value for each sub-image by analyzing its histogram. Interpolation of the sub-images enables obtaining threshold values at the pixel level

b) **Local Thresholding** method which operates by calculating the intensity values of local neighborhoods surrounding pixels. The mean of the local intensity distribution is commonly used for this calculation.

Both approaches are based on the assumption that smaller regions of the image exhibit uniform illumination.

- **Grab Cut Algorithm** -This algorithm is based on graph cuts and utilizes a Gaussian model to estimate the color distribution of the target object. By constructing Markov random fields over the pixels and employing energy functions, the algorithm prefers connected regions with the same tag or label. Graph optimization techniques are then employed to achieve efficient and effective object segmentation.

3.2 Normalization

Normalization is a process that involves modifying or adjusting the range of pixel values in an image. The primary objective of normalization is to rescale the image to a suitable state that meets the requirements of subsequent processing [13]. Normalization can be categorized into two types:

a) Linear Normalization which establishes a linear relationship between the original and transformed image. Mathematically, it is implemented as:

$$In = (I - Min)\frac{newMax - newMin}{Max - Min} + newMin$$

b) Nonlinear normalization which involves establishing a nonlinear relationship between the original and transformed image. It can be mathematically expressed as:

$$I_N = (newMax\text{-}newMin)\frac{1}{1+e^{-\frac{I-\beta}{\alpha}}} + newMin$$

where I_N is the normalized image, newMax and newMin are the maximum and minimum pixel intensities in the image, β is the range of pixel intensity and α is the width of the input image which is same as the total number of pixels in the image with noise reduction.

3.3 Character Recognition

OCR is the process of electronically converting images containing typed or handwritten text into digital or machine-readable code. OCR dates back to the 1920s, when physicist Emanuel Goldberg invented a new type of machine that could scan characters and convert them into telegraph code. He later greatly improved and developed an electronic document retrieval system. on the input image being isolated from the surrounding OCR relies on two primary algorithms:

i. **Matrix Matching:** This method performs a pixel-to-pixel comparison between the current image and a stored template. It heavily relies elements and the stored template being at a similar scale and font.

ii. **Feature Extraction:** This method involves decomposing the glyphs (individual characters) into distinct features such as closed loops, intersections, and lines. This approach enhances efficiency by reducing the dimensionality of character representation.

3.4 Extraction and Vocalization of Text from Image

Pytesseract and pyttsx3 python libraries have been further used for extraction and vocalization of text from images. These tools have been used to incorporate another additional feature through which a license plate number can be vocalized (read out loud) after identification. Pytesseract is a highly proficient tool for optical character recognition (OCR) that utilizes the Tesseract engine. Pyttsx3 provides a powerful functionality for converting text into speech. It converts textual data into voice output (speech).

4. Process Flow

The primary steps for implementing the LP recognition system are given below and Fig. 2 presents the complete flow chart for the LP recognition system.

i. **Pre-Processing Steps- Resizing Image to match stride and LetterBox Based Padding-**The image of a vehicle is heavily preprocessed with a convolution filter and padding.

ii. **License Plate Detection and filtering out the cropped plate**-Pre-processed image is fed into the object detection machine learning models. The predicted result is then further processed and pruned to give the best cropped image.

iii. **Character extraction from cropped image-Image Re-scaling** - The cropped image is then fed into the OCR in order to detect the LP alphabets and numbers.

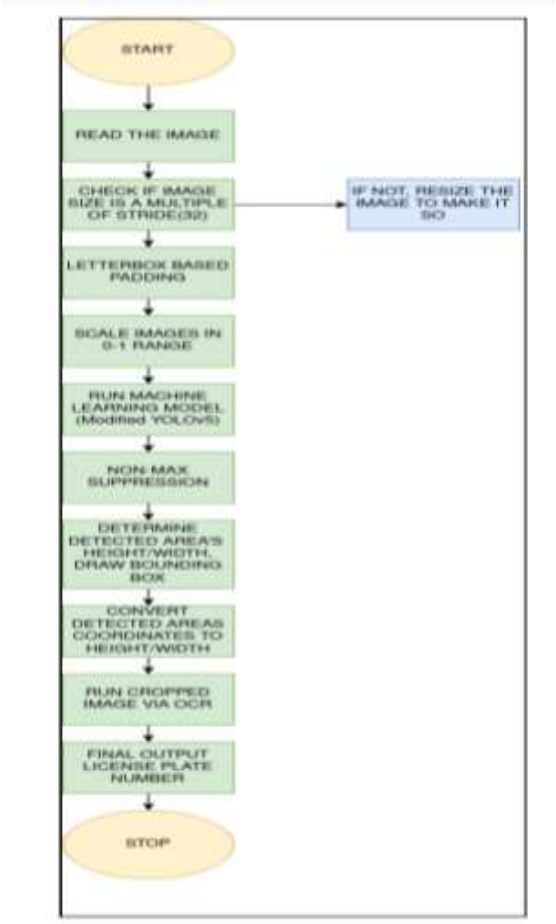


Fig. 2. Flowchart *for implementation of license plate recognition system*

The image is adjusted before processing so that its width and height are evenly divisible by the stride length of 32. Stride is a parameter used in Neural Networks as a measure to skip a given number of pixels while sliding the convolution filter based on the convolution formula given below:

$$F \circ I(x,y) = \sum_{j=N}^{N} \sum_{i=N}^{N} F(i,j) I(x+i, y+j)$$

The convolution "spreads" each pixel (i,j) in I, where I is the entire image and F is the second image which defines neighbour relationship. This is done as a means of image compression since it reduces convolution processing on highly correlated, unwanted pixels, thus increasing the efficiency of the overall network.

The image is reduced in size following the convolution technique so padding is done to keep the size of the original image the same. Padding is an additional layer applied to the edges of an image without changing its overall proportions. The original image's aspect ratio in the present work is preserved through scaling followed by letterbox-based padding. The

flow chart for letterbox-based paddings is shown in Fig. 3.

The resize ratio (r) was calculated to a 416 x 46 image, and the delta between new image's shape and r* original image's shape was then calculated. The image was padded based on the difference in height and width and then the image is fed to the machine learning model-YOLOv5 [14]. After modifying the hyper parameters, the trained model is loaded, and the processed image is sent forward to get the results. Fig. 4 shows the code snippets of the model architecture.

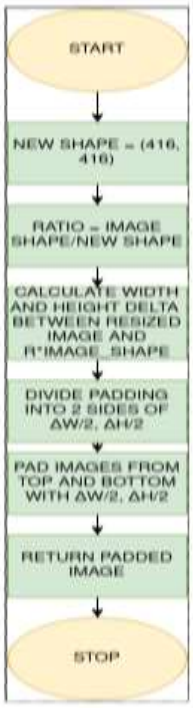


Fig. 3. Flowchart for Letter Box based padding

```
class Model(nn.Module):
    def __init__(self, model_cfg='yolov5s.yaml', ch=3, nc=None):
        super(Model, self).__init__()
        if type(model_cfg) is dict:
            self.md = model_cfg
        else:
            with open(model_cfg) as f:
                self.md = yaml.load(f, Loader=yaml.FullLoader)

        # Define model
        if nc:
            self.md['nc'] = nc
        self.model, self.save = parse_model(self.md, ch=[ch])


        # Build strides, anchors
        m = self.model[-1]
        m.stride = torch.tensor([64 / x.shape[-2] for x in self.forward(torch.zeros(1, ch, 64, 64))])
        m.anchors /= m.stride.view(-1, 1, 1)
        self.stride = m.stride

        # Init weights, biases
        torch_utils.initialize_weights(self)
        self._initialize_biases()
        torch_utils.model_info(self)

    def forward(self, x, augment=False, profile=False):
        if augment:
            img_size = x.shape[-2:]
            s = [0.83, 0.67]
            y = []
            for i, xi in enumerate((x,
                                    torch_utils.scale_img(x.flip(3), s[0]),
                                    torch_utils.scale_img(x, s[1])
                                    )):
                y.append(self.forward_once(xi)[0])
```

Fig. 4. Code snippet for model architecture

The last step was to resize the image to the final dimensions before continuing. Re-scaling the image keeps the range of weights small and is given by the formula, $x^1 = \frac{x}{255}$, where x is the original size of the image. This keeps the weights from exploding into very high numbers, which would make the convolution numbers reach very high values. If the scaling isn't done right, it can lead to high bias values, which lowers the confidence. Poor scalability results in poor accuracy and a lot of noise.

Computer vision requires identifying objects in an image. Object detection is harder than classification since classification doesn't locate objects in images. Models like YOLO can accurately locate targets in images. Convolution Neural Networks (CNN) helps YOLO detect objects. Applying a single Neural Network to an image, the algorithm then segments the image, locates bounding boxes, and makes probability predictions for each box. The bounding box with the highest probability is chosen to represent the object in the image.

YOLO v5 is a single-stage object finder and its great precision and instantaneous object detection have made it a favourite among researchers and developers. YOLO also uses many class masks to hide the items it finds at once. The various parameters and benchmarks for the YOLOv5 machine learning model are given in Table 1.

Table 1 YOLOv5 parameters and benchmarks

Activation function	The YOLOv5 model uses Leaky ReLU for some of the hidden layers and sigmoid activation function for the rest of the layers.
Optimization function	The default for YOLOv5s is chosen to be Stochastic Gradient Descent (SGD)
Cost function	In order to determine the loss of class likelihood and object scores, YOLOv5 uses "binary_cross_entropy_with_logits" function from PyTorch python library.

```
class Detect(nn.Module):
    def __init__(self, nc=80, anchors=()):
        super(Detect, self).__init__()
        self.stride = None
        self.nc = nc
        self.no = nc + 5
        self.nl = len(anchors)
        self.na = len(anchors[0]) // 2
        self.grid = [torch.zeros(1)] * self.nl
        a = torch.tensor(anchors).float().view(self.nl, -1, 2)
        self.register_buffer('anchors', a)
        self.register_buffer('anchor_grid', a.clone().view(self.nl, 1, -1, 1, 1, 2))
        self.export = False

    def forward(self, x):
        z = []
        self.training |= self.export
        for i in range(self.nl):
            bs, _, ny, nx = x[i].shape
            x[i] = x[i].view(bs, self.na, self.no, ny, nx).permute(0, 1, 3, 4, 2).contiguous()

            if not self.training:
                if self.grid[i].shape[2:4] != x[i].shape[2:4]:
                    self.grid[i] = self._make_grid(nx, ny).to(x[i].device)

                y = x[i].sigmoid()
                y[..., 0:2] = (y[..., 0:2] * 2. - 0.5 + self.grid[i].to(x[i].device)) * self.stride[i]
                y[..., 2:4] = (y[..., 2:4] * 2) ** 2 * self.anchor_grid[i]
                z.append(y.view(bs, -1, self.no))

        return x if self.training else (torch.cat(z, 1), x)

    @staticmethod
    def _make_grid(nx=20, ny=20):
        yv, xv = torch.meshgrid([torch.arange(ny), torch.arange(nx)])
        return torch.stack((xv, yv), 2).view((1, 1, ny, nx, 2)).float()
```

Fig. 5. Code snippet for Plate Detection

The code snippet for LP detection is given in Fig. 5. Most object detection models, after sliding windows over the image, have more than one candidates/proposal for detected objects[15]. Proposals are simply highlighted areas in the search image where the sought-after object is most likely to be located. The adjoining windows of the candidate window share similar features with the candidate areas, yielding hundreds of candidate regions for the target image. As approaches for producing proposals must have a high recall rate, the

stage's restrictions must be loosened. However, processing hundreds of candidate windows incurs enormous compute costs for obvious reasons; Non-Max Suppression (NMS) comes to the rescue in this situation. The algorithm of NMS works as follows [16]:

i. Consider the proposal list "B" with the confidence score "S" and the overlap threshold "N."
ii. Select the greatest S score of confidence, remove it from B, and then add to D.
iii. Compare IoU (Intersection over Union) of the proposals, and if IoU is greater than N, remove it from B since this is likely to be a redundant one.
iv. Keep repeating the process till there are no more proposals left in B.

The technique relies on a single threshold value "N"; therefore, this parameter is crucial to the model's overall performance.

```
def non_max_suppression(prediction, conf_thres=0.1, iou_thres=0.6, fast=False, classes=None, agnostic=False):
    if prediction.dtype is torch.float16:
        prediction = prediction.float()  # to FP32
    nc = prediction[0].shape[1] - 5  # number of classes
    xc = prediction[..., 4] > conf_thres  # candidates

    min_wh, max_wh = 2, 4096  # (pixels) minimum and maximum box width and height
    max_det = 300  # maximum number of detections per image
    redundant = True  # require redundant detections
    fast |= conf_thres > 0.001  # fast mode
    multi_label = nc > 1  # multiple labels per box (adds 0.5ms/img)
    if fast:
        merge = False
    else:
        merge = True  # merge for best mAP (adds 0.5ms/img)

    output = [None] * prediction.shape[0]
    for xi, x in enumerate(prediction):
        x = x[xc[xi]]  # confidence

        # If none remain process next image
        if not x.shape[0]:
            continue
        # Compute conf
        x[:, 5:] *= x[:, 4:5]
        box = xywh2xyxy(x[:, :4])
```

Fig. 6. Code snippet for NMS application

Fig. 7. Bounding box around the License Plate using Non-Max Suppression (NMS)

The Predicted image is then applied to an NMS technique to eliminate hundreds of proposals and select the one with the highest degree of confidence. The code snippet for application of NMS is shown in Fig. 6. As illustrated in Fig. 7, NMS seeks the optimal bounding box around the License Plate and suppresses all others [17,18]. The basic concept is to repeatedly select the entity with the highest probability, output it as the prediction, and then eliminate any remaining boxes with an IoU ≥0.5 with the box output in the previous step.

The detected area is then used to figure out the height and width of the area of interest, which is then adjusted to make the final cropped image for LP detection. The OCR used for extraction of LP is Tesseract OCR. Tesseract is an open-source software OCR engine developed by Hewlett-Packard in 1980. Once a cropped image is captured, tesseract is applied on the cropped image. The flow chart for operation of Tesseract is given in Fig. 8.

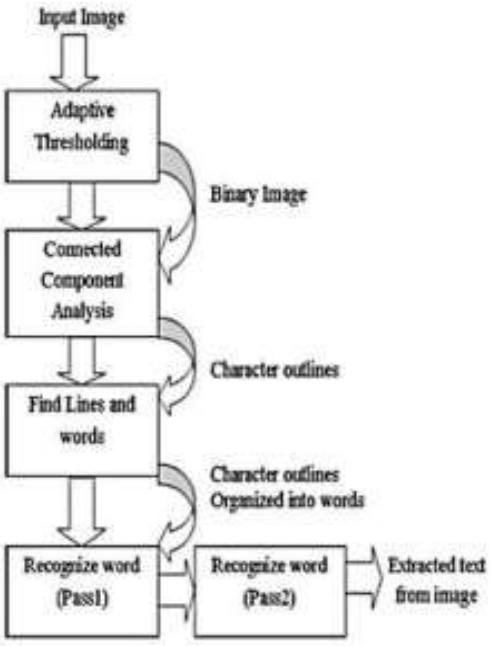


Fig. 8. Tesseract Flow Chart

The inputs for tesseract are binary images with polygon-based alphabets/numeric areas elucidated using adaptive thresholding.
The steps to extract text from the image is given below:

1. Linked component analysis saves outlines and merges them into alphanumeric blobs.
2. Blobs are organised into text-based lines, and proportional alphanumeric text is split into words by definite-ordered or fuzzy white spaces.
3. Recognition of text is a two-stage process:
   Stage 1: Every given word was acknowledged. The words that are recognized/classified are used as training data using adaptive filters.
   Stage 2: Words with low recognition accuracy ratings are re-recognised.
4. The final stage included napped white spaces and alternative hypotheses

The algorithms to extract lines and words from the image is given in Fig. 9 and Fig. 10 respectively.

Algorithm to find lines in the extracted text

1. **Line Finding:**
   i. Line filtering and blob generation remove vertically contacting character drop-covers in a simple percentile stature.
   ii. The text size is approached to middle-height (median)
   iii. Filtered blobs are placed atop non-covering, equal, but inclining lines.
   iv. Lines are relegated using the least median of squares to determine baselines.
   v. Last advance consolidates blobs with half-horizontal overlaps and aligns diacritical imprints with the right base.
2. **Fitting over the baseline:**
   i. The quadratic spline function fits baselines using text lines, allowing tesseract to handle pages with bended lines.
   ii. Quadratic spline fits the most crowded partition with least square fit.
3. **Chopping and Pitch detection:**
   i. Fixed-pitch lines are examined.
   ii. When fixed pitch text is found, tesseract slices the words into pitch-based characters and disables the chopper and associator.
4. **Proportional Word Finding:**
   i. Misinterpreting word spacing can lead to the emission of undetected or erroneous words. Tesseract estimates gaps between a base line and a mean line in a confined vertical range.

Fig. 9. Algorithm for finding line from the extracted text

Algorithm to find words in the text

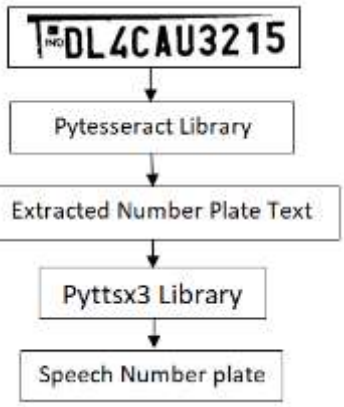
1. **Chopping Joined Characters:**
   a. Chops the blob with worst confidence
   b. Curved vertices or line sections determine chop-points, which may take up to 3 sets to cleave an ASCII set.
2. **Joining and linking Broken characters:**
   a. If the word has low accuracy/precision after potential cleaves, the associator tries to get the best first pursuit of the segmented graph.
   b. New candidate states are selected from a need line and assessed by grouping unclassified mixes of sections.
   c. The associating method is done after chopping and is inefficient, and it has the advantage of requiring fewer complex data structures than would be required to maintain the whole graph of segmentation.
   d. Tesseract's ability to successfully organise fragmented characters offers it an advantage over other/current OCR algorithms.

Fig. 10. Algorithm for finding words from the extracted text

Finally, pre-processing on the detected string is done in order to prune out unnecessary detections.

After detecting the LP, the Zonal Integrated Police Network (ZIPNET) website was scraped to extract license plates from the FIRs registered within North Delhi. A database is created using the scraped license plates. In order to identify suspected stolen vehicles, a matching function is implemented that utilizes a similarity score to identify potential matches when the model output is not completely accurate. Finally, a Graphical User Interface (GUI) was built for the front end using the Kivy Python module, as shown in Fig. 11. The user can input the image from which they wish to identify and recognise the license plate number via the graphical user interface.

Fig. 11. GUI of Application



5.

Fig. 12. Conversion of License Plate number into speech

Conversion of License Plate number into audio (voice) output is illustrated in the flow chart shown in Fig. 12. [19]

5. Results and Discussion

A robust Deep Learning Model has been trained specifically **for plate detection, designed** to meet the requirements of the Indian System. The model was evaluated using a dataset of 50 images. The extracted text from an image is determined by the average confidence of the model used to recognize the LP. As indicated in Table 2, the model's confidence that a LP is present in the image shown in Fig. 11 is 75.75 %.

Considering that Indian Number plates have ten letters/numbers, various metrics were evaluated for the input image shown in Fig. 13 and is given in Table 3, for text matches.

Fig. 13. Input image for the detection system

Table 3 Performance metrics of proposed LP detection system

S. No.	Performance Metrics	% values
1	Accuracy of perfect matches with the original license plate	80%
2	% Images with 9 letters matching	4%
3	% Images with 8 letters matching	6%
4	% Images with < 8 letters matching	10%
5	% Images where the predicted number contains the original number as suffix	6%
6	Average cosine similarity in the test data	95.2%
7	Average % of letter match	88.5%

From the input image, vehicle LP was cropped using ML algorithms as shown in Fig. 14 (a) which was preprocessed to obtain the LP as shown in Fig 14 (b).

(a) (b)

Fig. 14. (a) Extracted LP image from the input image (b) pre-processed image for the detection system

Using the extracted image, the system could recognize the content of the LP using OCR and the result obtained is given in Fig. 15.

CHARCTER RECOGNITION : DL4CAU3215

Fig. 15. Detected LP

It is found that the proposed LP detection system for traffic management can accurately detect the LP with an accuracy of 80%. Since some of the predicted strings and the original strings were not always equal, nor did they differ by one or two characters, however, on taking a manual inspection at it, they looked extremely similar. Hence, it makes sense to extract out the information about the similarity index of the predicted string and the actual result. In order to do that, metric of cosine similarity was used. In mathematical terms, Cosine similarity calculates the similarity of two vectors in an inner product space. Hence, it is used as a measure with similarity in text analysis. The formula for calculation of cosine similarity is given as follows:

$$Cos\theta = \frac{\vec{a} \cdot \vec{b}}{\|\vec{a}\| \, \|\vec{b}\|} = \frac{\sum_1^n a_i b_i}{\sqrt{\sum_1^n a_i^2} \sqrt{\sum_1^n b_i^2}}$$

where, $\vec{a} \cdot \vec{b} = \sum_1^n a_i b_i = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n$ is the dot product of the two vectors.

The average cosine similarity of 95.2% in the test data was obtained using the proposed system. Graphical representation in the form of a bar plot of metrices 1,2,3 and 4 as given in the Table 3 is shown in Fig. 16.

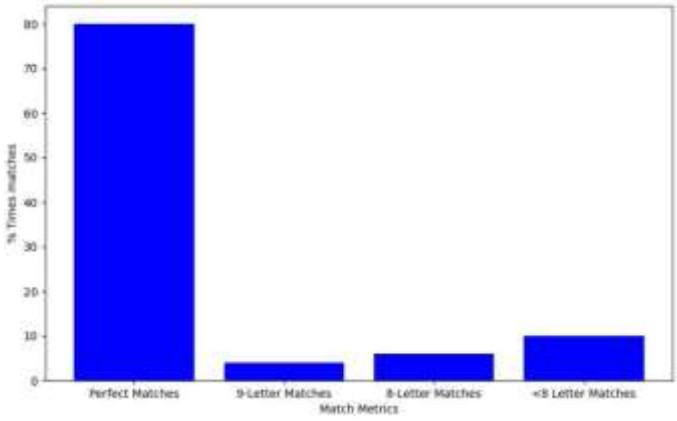


Fig. 16. Plot showing number of letters matched

The result for Average percentage of letters match is given in Table 4 and its bar chart is given in Fig. 17.

Table 4 % times each letter matched

Position of Letter	1	2	3	4	5	6	7	8	9	10
%match	94	92	90	86	86	88	88	94	92	90

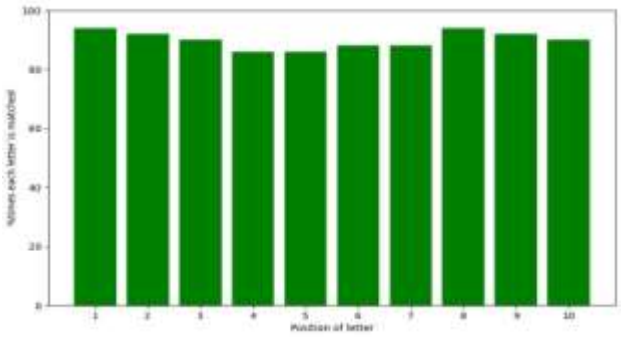


Fig. 17. Plot showing % times each number matched

Based on the various metric values of the suggested system, it can be concluded that the system is a reliable option for LP detection and can aid in the detection of stolen vehicles utilising the developed GUI application.

6. Conclusion

Utilizing a reliable DL Model, the proposed LP Detection system has been developed. The model has been trained exclusively for plate detection, in accordance with Indian System specifications. In addition, image processing techniques such Letter Box padding and NMS have been added to increase the model's accuracy. The extracted LP is subsequently forwarded to the Number Extraction Module for further processing. Tesseract, an OCR Engine responsible for extracting the numbers from the license plate and presenting the output in text format, is used to process the extracted LP image. The generated text is further processed to remove any irrelevant detections. Finally, the output from this module is transmitted to the Data Scraping module and shown on the Kivy Python module-built GUI.

The presented technique additionally determines whether the detected vehicle's LP matches any entries in the stolen vehicles database. The extracted string from the Number Extraction Module is searched in the database, and the status of stolen or not stolen is displayed on the user interface based on the results. The extracted string and the bounding box around the LP received from the LP detection module is also displayed. An additional useful feature has also been added in the system to convert the detected License Plate number into Voice output (speech).

References

[1] S. Mehtab and J. Sen, "Stock Price Prediction Using CNN and LSTM-Based Deep Learning Models," *2020 International Conference on Decision Aid Sciences and Application, DASA 2020*, pp. 447–453, Nov. 2020, doYao L. Zhao Y. Fan J. Liu M. Jiang J. and Wan Y 2019 Research and Application of License Plate Recognition Technology Based on Deep Learning. *Journal of Physics: Conference Series,* IOP Publishing. 1237(2): 022155

[2] Wang H. Li Y. Dang L. M. and Moon H. 2021 Robust Korean license plate recognition based on deep neural networks. *Sensors*, 21(12), 4140.

[3] Nguyen H. 2022. An efficient license plate detection approach using lightweight deep convolutional neural networks. *Advances in Multimedia*, *2022*, 1-10.

[4] Villena Toro J., Wiberg A. and Tarkian M. 2023 Optical character recognition on engineering drawings to achieve automation in production quality control. *Frontiers in Manufacturing Technology*, *3*, 1154132.

[5] Selmi Z. Halima M. B. Pal, U. and Alimi M. A. 2020 DELP-DAR system for license plate detection and recognition. *Pattern Recognition Letters*, 129, 213-223..

[6] Omar N. Sengur A. & Al-Ali S. G. S. 2020 Cascaded deep learning-based efficient approach for license plate detection and recognition. *Expert Systems with Applications*, 149, 113280.

[7] Singh J. & Bhushan B. 2019 Real time Indian license plate detection using deep neural networks and optical character recognition using LSTM tesseract. In *2019 international conference on computing, communication, and intelligent systems (ICCCIS)* (pp. 347-352). IEEE.

[8] Kasaei S. H. Kasaei, S. M. & Kasaei, S. A. 2010 New morphology-based method for robustiranian car plate detection and recognition. *International Journal of Computer Theory and Engineering*, *2*(2), 264.

[9] Ozbay S. & Ercelebi E. 2007 Automatic vehicle identification by plate recognition. *International Journal of Computer and Information Engineering*, *1*(9), 1418-1421.

[10] Babu R. N. Sowmya V. & Soman K. P. 2019 Indian car number plate recognition using deep learning. In *2019 2nd international conference on intelligent computing, instrumentation and control technologies (ICICICT)* (Vol. 1, pp. 1269-1272). IEEE.

[11] J. Redmon A. Farhadi (2018) "YOLOv3: An Incremental Improvement," https://arxiv.org/abs/1804.02767,pp. 1-6

[12] Singh T. Romen, Roy Sudipta , Singh O. Imocha, Sinam Tejmani , Singh Kh.Manglem 2011. A New Local Adaptive Thresholding Technique in Binarization. *International Journal of Computer Science Issues*, 8: 271-277.

[13] Patro S. G. O. P. A. L., & Sahu, K. K. 2015 Normalization: A preprocessing stage. *arXiv preprint arXiv:1503.06462*.

[14] Sharanu S. T. Jadav R. 2023 Automatic Number Plate Recognition System for Vehicle Identification Using Yolov5". *European Chemical Bulletin*, 12, 8111-8115.

[15] Tuli S., Basumatary N., & Buyya R. 2019 Edgelens: Deep learning based object detection in integrated iot, fog and cloud computing environments. In *2019 4th International Conference on Information Systems and Computer Networks (ISCON)* (pp. 496-502). IEEE.

[16] Rothe R., Guillaumin M., & Van Gool L. 2015 Non-maximum suppression for object detection by passing messages between windows. In *Computer Vision–ACCV 2014: 12th Asian Conference on Computer Vision, Singapore, Singapore, November 1-5, 2014, Revised Selected Papers, Part I 12* (pp. 290-306). Springer International Publishing.

[17] Lü, Y. (Ed.). 2020 *Pattern Recognition and Artificial Intelligence: International Conference, ICPRAI 2020, Zhongshan, China, October 19-23, 2020: Proceedings*. Springer.

[18] Michel M., & Burnett N. 2019 Enabling GPU-enhanced computer vision and machine learning research using containers. In *High Performance Computing: ISC High Performance*

2019 International Workshops, Frankfurt, Germany, June 16-20, 2019, Revised Selected Papers 34 (pp. 80-87). Springer International Publishing.

[19] Kotwal N. Sheth A. Unnithan G. Kadaganchi N. 2021 Optical Character Recognition using Tesseract Engine. *International Journal of Engineering Research & TechnologY,* 10: 381-385.