Autonomous Lane and Number plate detection using Classic Computer Vision and Deep Learning

CS 5330: Pattern Recognition and Computer Vision, Professor Bruce Maxwell

Dev Vaibhav
Dept. Electrical and Computer Eng.
Northeastern University
vaibhav.d@northeastern.edu

Siddharth Maheshwari
Dept. Electrical and Computer Eng.
Northeastern University
maheshwari.si@northeastern.edu

Skanda Akkihebbal Prasanna
Dept. Electrical and Computer Eng.
Northeastern University
akkihebbalprasanna.s@northeastern.edu

I. Abstract

This paper presents a system capable of autonomously detecting lanes and car number plates using a combination of classical computer vision techniques and deep learning models. It may be used to detect suspicious car number plates and trigger an alert system. Additionally, the system can also be used for keeping track of vehicles entering a building. Since the lane detection system uses classical CV, it is sensitive to the camera's pose with respect to the road, we have tried to make the code dynamic by adapting to different road conditions based on a selected polygon from the road video feed. This polygon governs the quality of the detected lanes. This project's potential application extends beyond the scope of the current study and can be used for various purposes, including road safety and security. Since C++ language is used to code, it can be used in real-time applications.

II. Related Work:

We did some literature review and found these papers to be interesting.

A. Robust Lane Detection and Tracking in Challenging Scenarios [12]

The paper proposes an approach for lane detection and tracking in challenging scenarios such as shadows, occlusion, and sharp curves. The proposed system in the paper used a traditional computer vision pipeline that consists of image pre-processing, lane detection, and lane tracking modules. Preprocessing includes Gaussian Blur, color space conversion, edge detection, and region of interest selection. The lane detection module uses a combination of Hough Transform and sliding window techniques to extract lane boundaries. The paper also discusses the handling of challenging scenarios that track the detected lanes over time and performs outlier rejection based on the lane width and distance constraints.

B. R-CNN (Region-based Convolutional Neural Networks) [2]

R-CNN is a region based Object Detection Algorithm developed by Girshick et al., from UC Berkeley in 2014. It is one of the first large and successful application of convolutional neural networks to the problem of object localization, detection, and segmentation. The approach was demonstrated on benchmark datasets, achieving then state-of-the-art results on the VOC-2012 dataset and the 200-class ILSVRC-2013 object detection dataset.

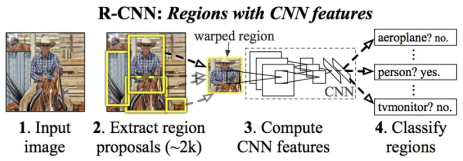

Fig. 1: R-CNN architecture

Region based CNN consists of three modules as shown in Fig. 1 — Region Proposal, Feature Extractor, and Classifier.

1) Region Proposal: When an input image is given, the region proposal tries to detect different regions (2000) in different sizes and aspect ratios. In other words, it draws multiple bounding boxes in the input image as shown in Fig. 1 (2. Extract region proposals (2k)).
2) Feature Extractor: Each proposed region will be trained by a CNN network and the last layer (4096 features) will be extracted as features so the final output from Feature extractor will be Number of proposed regions x 4096

3) Classifier: Once the features are extracted we need to classify the objects inside each regions. To do this a linear SVM model is trained for classification, Specifically one SVM model for each class.

Cons of R-CNN:

1) It takes a huge amount of time to train the network as you would have to classify 2000 region proposals per image.
2) It cannot be implemented real time as it takes around 47 seconds for each test image.
3) The selective search algorithm is a fixed algorithm. Therefore, no learning is happening at that stage. This could lead to the generation of bad candidate region proposals.

Some of these limitations like execution/ training time are overcome in YOLOv5s which is a major bottleneck in depolying R-CNN to real-time systems or whenever we have time constraint.

C. Number Plate Detection Using YOLOV4 and Tesseract OCR

A recent research paper by G. Poorani et al. from SKCT, TN, India (2022) [4] proposed a method for car number plate detection. The proposed approach employs the state-of-the-art YOLOv4 object detection algorithm for detecting the number plate region. Further, a series of pre-processing steps are applied to the detected region, including resizing the image to grayscale, applying Gaussian smoothing, Otsu binarization, and dilation, to improve the quality of the image. The paper also utilizes Tesseract OCR for recognizing the characters in the detected number plate region. Looking at this paper we found that shading can impact a lot while recognition of text therefore we decided to add Lumination correction in our project.

III. METHODOLOGY:

The code gives the user the flexibility to provide an image/ video/ live video feed as input based on the command line arguments. It can automatically identify if the input is image or a recorded video by counting number of frames in the source and can process the frames accordingly.

A. Lane detection using Classical CV

In the field of autonomous driving and advanced driver assistance systems(ADAS), lane detection and deviation from the lane plays an important role. The lane detection system uses sensors such as cameras or lidar to detect lane markings on the road and determine the vehicle's position within the lane. Road accidents are a major problem around the world and lane departure is one of the leading causes of accidents on highways and other high-speed roads. By providing warnings to the driver when the vehicle is drifting out of the lane or adjusting the vehicle's position automatically within the lane, a lane detection system can help prevent accidents caused by lane drift or unintended lane changes. As autonomous driving systems continue to evolve, the lane detection system will play a major role to ensure the safe and efficient operation of autonomous vehicles. As defined by the Society of Automotive Driving (SAE) in the current Level 3 and Level 4 autonomy, these systems are crucial for ensuring the safety of the driver and making sure that the vehicle stays within its designated lane and operates safely and effectively

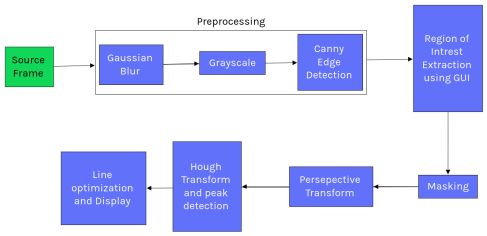

Fig. 2: Flow Chart of Lane Detection

1) Implementation: 1. Read Input Frame: The First Step is to read the input frame, which can be either live or recorded video frame

2. Preprocessing: The second step is to apply Gaussian blur to the input frame to remove noise and smoother the image refer 3 3. Grayscale Conversion:

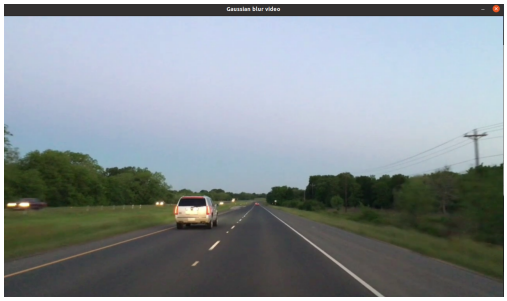

Fig. 3: Gaussian Blur on Video / live Frame

We use the denoise image and convert it to grayscale to simplify the frame 4 4. Canny Edge Detection: Canny Edge Detection is applied to detect the edges from the grayscale frame refer 5

5. Region of Interest Extraction using GUI: We have created a GUI for the user to select the polygon using mouse clicks which extracts part of the frame that contains lane lines refer 6

6. Perspective Transform: Next Step to detect lanes is to align our visual systems to visualize the road ahead in a manner that they seem to be looking at it from a bird's eye perspective, this will help in calculating

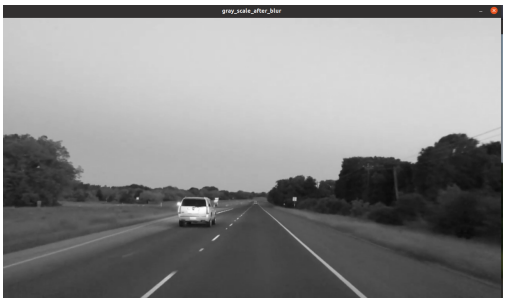

Fig. 4: Grayscale Conversion of Video/live frame

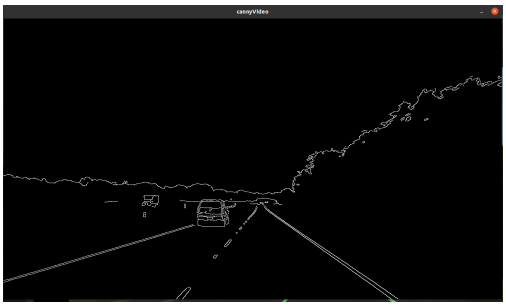

Fig. 5: Canny Edge Detection on Live Frame

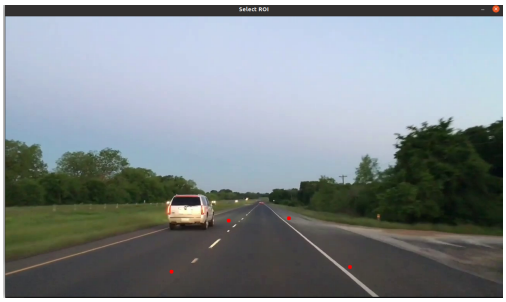

Fig. 6: ROI Extraction on Video/ live Frame

the curvature of the road. The bird's eye view can be achieved by applying the perspective transformation essentially mapping a set of four points bounding the lane in our input frame to the desired set of points and thus generating the desired top-down view. Refer 7

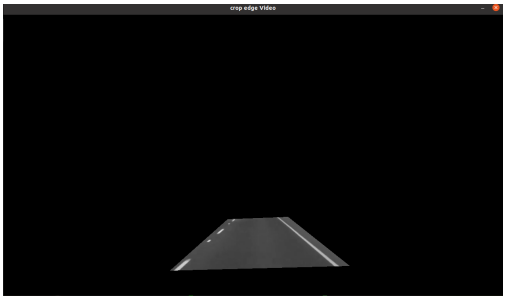

Fig. 7: Perspective Transformation on Video/ live frame

7. Hough Transformation and Peak Detection: After Computing Perspective Transformation, we compute the histogram of the bottom half of the transformed image to find the x-coordinates of the left and right lane lines. The histogram represents the frequency of occurrences of each pixel intensity in the frame and in our case, it shows where the lane lines are most likely to be.

After applying a perspective transform to the region of interest, we use the Hough line Transform to detect the lines in the transformed image. The Hough Line transform is a technique to detect straight lines in an image by converting the pixel coordinate to parameter space, where each line is represented by a point in the parameter space. The Hough Line transform can detect lines even if they are broken or partially visible in the image. After detecting the lines we optimize them to get the left and right lines

B. Object detection using YOLOV5s

YOLOv5 is available in four models, namely `n`, `s`, `m`, `l`, and `x`, standing for nano, small, medium, large and extra large respectively, each one of them offering different detection accuracy and performance as shown below in Figs 8, 9, and 10.

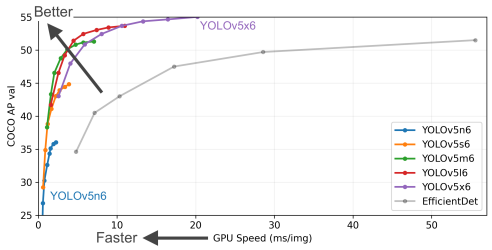

Fig. 8: YOLOv5 results provided by Ultralytics

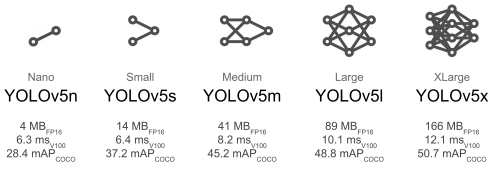

Fig. 9: YOLOv5 model comparison (from PyTorch)

Model Name	Params (Million)	Accuracy (mAP 0.5)	CPU Time (ms)	GPU Time (ms)
YOLOv5n	1.9	45.7	45	6.3
YOLOv5s	7.2	56.8	98	6.4
YOLOv5m	21.2	64.1	224	8.2
YOLOv5l	46.5	67.3	430	10.1
YOLOv5x	86.7	68.9	766	12.1

Fig. 10: YOLOv5 Model information

From Figs 8, 9, and 10, it can be seen that the `YOLOv5n` is the fastest model but is the least accurate. The accuracy increases as we move towards more complex models but simultaneously the speed decreases because number of parameters in the model increase.

We used YOLOv5s pretrained model from Ultralytics [5] as it provides a good balance between accuracy and speed required for our application to detect objects in the video which include mostly cars, pedestrians, laptop, chair, monitor, traffic signal. We also wanted to detect number plates but it was not present as one of the 80 classes in the pretrained model. For this, we found a dataset [6] on Roboflow and trained the model with three epochs. The dataset contains approx. 19K training, 1.8K validation, and 882 test images. Training took around 23 minutes to complete on Google CoLab and mAP of 98.2% was achieved.

The two models were converted to ONNX first 11 and then used in the C++ code with openCV DNN functionality 11. We took help from `https://netron.app/` to visualize the onnx model which helped a lot in understanding the model and debugging the code.

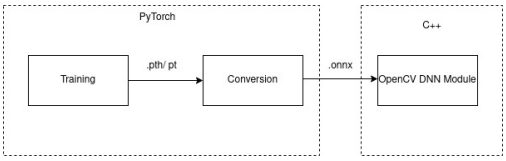

Fig. 11: PyTorch Model to ONNX conversion

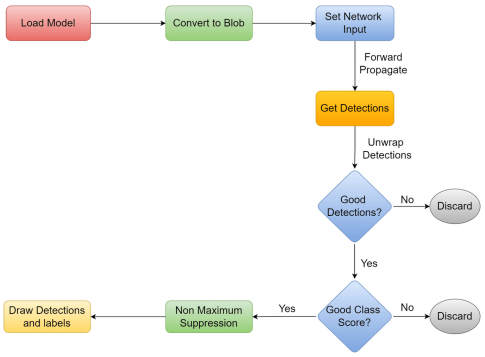

Fig. 12: YOLOv5s with OpenCV DNN workflow

Fig. 12 shows the workflow. Below parameters are used for different thresholds.

`SCORE THRESHOLD = 0.5` // To filter low probability class scores

`NMS THRESHOLD = 0.45` // To remove overlapping bounding boxes

`CONFIDENCE THRESHOLD = 0.45` // Filters low probability detections

After performing a forward pass to the network, we get a 2D array 13 of size 25200×85 (rows and columns) (in case of number plate, it is 25200x6). The rows represent the number of detections. So each time the network runs, it predicts 25200 bounding boxes. Every bounding box has a 1-D array of 85 entries that tells the quality of the detection. This information is enough to filter out the desired detections.

X	Y	W	H	Confidence	Class scores of 80 classes

Fig. 13: YOLOv5s output for 80 classes

The first two places are normalized center coordinates of the detected bounding box. Then comes the normalized width and height. Index 4 has the confidence score that tells the probability of the detection being an object. The following 80 entries tell the class scores of 80 objects of the COCO dataset 2017, on which the model has been trained.

The following post-processing is done to get good detections.

1) Loop through detections.
2) Filter out good detections.
3) Get the index of the best class score.
4) Discard detections with class scores lower than the threshold value.

After filtering good detections, we are left with the desired bounding boxes. However, there can be multiple overlapping bounding boxes which is solved by performing Non-Maximum Suppression. The function `cv::dnn::NMSBoxes` is used to do this.

C. Number plate recognition using Tesseract OCR

Tesseract OCR is a widely used open-source optical character recognition engine that has proven to be effective in recognizing characters in various languages. It provides accurate and reliable character recognition, even in the presence of noise and other distortions. Tesseract OCR is also flexible and can be adapted to various applications, including license plate recognition.

In the context of license plate recognition, Tesseract OCR can help extract characters from the license plate image, which can then be used to identify the license plate number. Tesseract OCR can handle various font types and sizes, and can recognize both uppercase and lowercase characters, making it a suitable choice for license plate recognition. Overall, Tesseract OCR is a powerful tool for text recognition and has become a popular choice for license plate recognition applications.

1) Process Flow as per Fig 14: Lets look at how we are able detect the text from this ROI given by YOLOV5:

(a) Do Lumination Correction refer 18: The first step is to perform illumination correction on the input image. This is done to improve the contrast of the image and make the text more visible also get rid of shade. For doing this we defined a function called

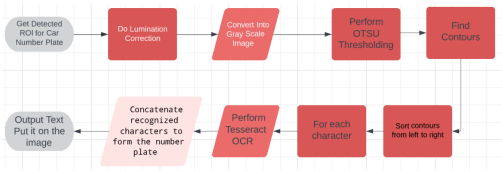

Fig. 14: Overall process of OCR

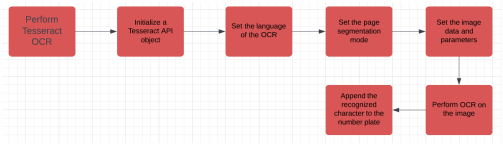

Fig. 15: Process of Converting image to text using Tesseract OCR

Fig. 16: Example Image Input

Fig. 17: YoloV5 result for Number Plate Detection

Fig. 18: After performing Lumination Correction followed by OTSU Thresholding

Fig. 19: Before performing Lumination Correction followed by OTSU Thresholding

Fig. 20: OTSU Thresholding on Number Plate after Lumination Correction

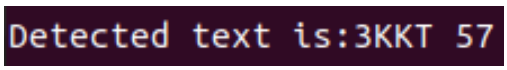

Fig. 21: Results of Detected Number Plate

"illuminationCorrection" which performs illumination correction on an input image. It converts the input image from the BGR color space to the LAB color space, splits the LAB image into its three channels, applies the **Contrast Limited Adaptive Histogram Equalization (CLAHE)** algorithm to the L channel to enhance the contrast of the image, merges the processed LAB channels, converts the LAB image back to the BGR color space, and replaces the original image with the corrected one. This function is useful in improving the image quality and reducing the effect of uneven lighting conditions on the object of interest.

(b) Convert Into Gray Scale Image: The next step is to convert the input image to grayscale. This is done to simplify the image processing tasks and reduce the computational overhead.

(c) Perform OTSU Thresholding refer 20: After converting to gray scale, the Otsu's method is applied to threshold the image using the `threshold` function. This is a common technique to separate foreground from the background in an image.

(d) Find Contours: Contours are found in the thresholded image to identify the individual characters. These contours are the boundaries of the connected components in the image. To find contours of each connected component using the `findContours` function with `RETR_EXTERNAL` and `CHAIN_APPROX_SIMPLE` flags.

(e) Sort contours from left to right: The identified contours are then sorted from left to right based on their x-coordinates using the `std::sort` function. This is done so that the recognized characters can be concatenated in the correct order to form the number plate.
(f) Perform OCR for each character refer 15: For each contour, a bounding box is extracted and used to extract the character image. Then, Tesseract OCR is applied to recognize the character.
(g) Concatenate recognized characters to form the number plate: The recognized characters are concatenated in the correct order to form the number plate. This is the final output of the system.
(h) Output Text refer 21: The recognized text is outputted by the OCR engine, and this text is superimposed on the input image to display the final output.

2) Process of Performing Tesseract OCR Fig 15: Lets see the steps of how we are performing Tesseract OCR:

(a) Initialize a Tesseract API object using the `tesseract::TessBaseAPI` class.
(b) Set the language of the OCR to English and the OCR engine mode to LSTM only using the `TessBaseAPI::Init` function.
(c) Set the page segmentation mode to `PSM_SINGLE_CHAR` using the `TessBaseAPI::SetPageSegMode` function.
(d) Set the image data and parameters using the `TessBaseAPI::SetImage` function to pass the character image as input to the OCR engine.
(e) Perform OCR on the image using the `TessBaseAPI::Recognize` function to extract the character information.
(f) Append the recognized character to the number plate.

IV. Experiments and Results:

We performed these experiments to test the system and here are their results.

A. Lane detection using Classical CV

The system was tested with different videos under different lighting conditions. If there is traffic on the road, the system might fail if it obstructs the view in the selected polygon. Usually, most of the references we referred to had manually hard-coded the polygon vertices to suit the video they use. We wanted to dynamically do the process of selecting the polygon and then performing masking. Hence we created a GUI to select the polygon for the Video frame / live video through which we don't have to hard code the vertices of the polygon, we can adjust it dynamically. Here are some results for lane detection tested in different lighting conditions

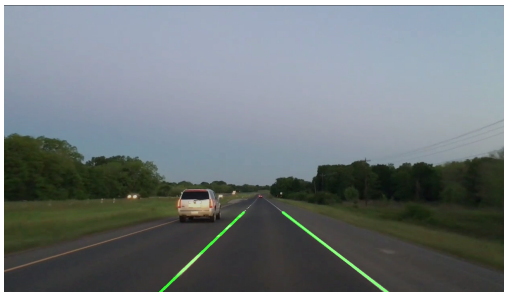

Fig. 22: Lane detection results with not-so-bright sunlight

Fig. 23: Lane detection results on a sunny day

Fig. 24: Lane detection results at dusk

B. Object and Number Plate detection using YOLOV5s

First, we tested the system on a bunch of images. The system worked pretty well in detecting the objects and the number plate. We then tested the system on a video feed. We observed that sometimes when a car is far away in the scene, it mistakenly classifies it as sheep in Fig. 26 but as soon as it comes closer, it correctly identifies it as a car in Fig. 27.

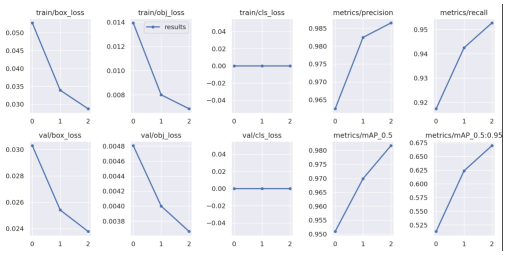

Fig. 25: Various losses for number plate detection model trained with 3 epochs

Fig. 26: Car detected as sheep when it is far

Fig. 27: Car detected correctly when it is near

Fig. 28 shows that the number plate is detected on the car and OCR has been tried which produces incorrect results. More explanation is explained in the next sub-section regarding OCR results. The first inference time shows the time taken for the original YOLOv5s model to detect the object (car/ traffic etc.) whereas the second time is for the second model which is used to detect just the number plate.

C. Number plate OCR using Tesseract

Here we tested out OCR system on live streaming, videos, and different images. While testing on different

Fig. 28: Number plate detected

images we were able to get 9/10 correct number plate recognition's which gives us accuracy of 90%. Here are some results referred in figs 31, 29, 30. For video and live streaming, it was observed that number plate detection was more accurate when the car was moving at a constant speed or was stationary. Number plates were successfully detected and recognized when the car was clearly visible and the number plate was not rotated. However, some number plates with different color backgrounds could not be recognized if the car was constantly accelerating or decelerating. This was because the thresholding varied constantly for each frame and Otsu thresholding was not effective when dealing with different color backgrounds, speeds, locations, lighting and orientations of the number plate.

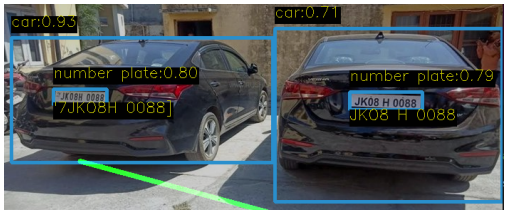

Fig. 29: 2 Cars plate recognition Simultaneously

Fig. 30: Example Recognition for a image

V. Challenges

While experimenting with lane detection, we found that our pipeline was able to work in different lighting conditions but it would fail if there are any vehicles that

Fig. 31: Example Recognition for a BMW

obstruct the field of view of the lanes as we can find the peaks through the histogram. Also, If the vehicle is moving at a high speed we were not able to detect the lanes continuously.

We faced some challenges in training the YOLOv5s model and using it in C++ with OpenCV but we overcome them by following some tutorials [9] [10], [8] and [11]. A lot of issues were faced in performing OCR using Tesseract but we gained a better understanding of the system through [13] and [14]. Although, the OCR pipeline is not perfect, we are much more confident and have direction to improve it

VI. Future Improvement Areas

As a future direction, we would like to explore more deep learning-based techniques for lane detection, as it is an essential component of our system. We believe that the current classical CV approach can be further improved to increase the accuracy of lane detection in challenging scenarios such as bad weather conditions or high-speed moving vehicles, or obstruction of lanes by vehicles. We found that when we try it with different video inputs we were finding it difficult to visualize the results and had to resize the frames.

In addition, we also plan to investigate other approaches for Optical Character Recognition (OCR) apart from Tesseract OCR, which is the current OCR algorithm used in our system. We aim to test and compare the performance of different OCR algorithms and select the one that offers the best results in terms of accuracy, speed, and efficiency.

Furthermore, we would like to experiment with different object detection algorithms other than YOLOv5. We believe that exploring other object detection algorithms can help us improve the system's performance, particularly in detecting and recognizing different types of vehicles and objects.

Overall, we are optimistic that exploring these future directions will enhance the system's capabilities and contribute to the development of a more robust and efficient system for lane detection and car number plate recognition.

VII. Conclusion

In conclusion, we have successfully developed a system capable of detecting lanes and car number plates using a combination of classical computer vision techniques and deep learning models. We have tested our system on live cameras and traffic, and it has demonstrated reliable performance in detecting lanes, identifying objects, and recognizing their number plates.

The system's dynamic behavior allows it to adapt to different road conditions, making it suitable for various scenarios, such as detecting suspicious cars' number plates and tracking vehicles entering buildings. We believe that this system's potential applications extend beyond the scope of this study and can be used in various fields such as road safety, traffic management, and security.

In summary, our system offers a promising solution for detecting lanes, object,s and car number plates autonomously, which could contribute significantly to improving road safety and security.

References

[1] https://medium.com/@SunEdition/lane-detection-and-turn-prediction-algorithm-for-autonomous-vehicles-6423f77dc841
[2] Ross Girshick and Jeff Donahue and Trevor Darrell and Jitendra Malik: Rich feature hierarchies for accurate object detection and semantic segmentation 2014 https://arxiv.org/pdf/1311.2524.pdf
[3] https://medium.com/analytics-vidhya/region-based-convolutional-neural-network-rcnn-b68ada0db871
[4] https://www.pnrjournal.com/index.php/home/article/view/426
[5] https://github.com/ultralytics/yolov5
[6] https://universe.roboflow.com/augmented-startups/vehicle-registration-plates-trudk/model/2
[7] https://pyimagesearch.com/2020/09/21/opencv-automatic-license-number-plate-recognition-anpr-with-python/
[8] https://medium.com/@freshtechyy/deploying-pytorch-model-into-a-c-application-using-onnx-runtime-f9967406564b
[9] https://learnopencv.com/object-detection-using-yolov5-and-opencv-dnn-in-c-and-pytho
[10] https://www.youtube.com/watch?v=GRtgLlwxpc4
[11] https://learnopencv.com/custom-object-detection-training-using-yolov5/
[12] https://ieeexplore.ieee.org/document/4459093
[13] https://pyimagesearch.com/2021/11/15/tesseract-page-segmentation-modes-psms-explained-how-to-improve-your-ocr-accuracy
[14] https://tesseract-ocr.github.io/tessdoc/ImproveQuality.html