LANE DETECTION USING CNN

This project document is being submitted in partial fulfilment of the course

CSI3006-Soft Computing Techniques

Under the able guidance of Professor Dr. Anuradha J

(School Of Computer Science Engineering)

Team Members

Brian E Shilo	21MIC0131
Vidhisha Muhta	21MIC0144
Mote Nandini Pramod	21MIC0190
Keshav Sharma	21MID0182
Poorvi Rai	21MID0243

DECLARATION BY THE CANDIDATES

We, the undersigned, hereby declare that the project report entitled “Lane Detection Using CNN” submitted by us to Vellore Institute of Technology University, Vellore in partial fulfillment of the requirement for the award of the degree of Integrated M. Tech in Computer Science is a record of bonafide project work carried out by us under the supervision of Professor Dr. Anuradha J. We declare that this report represents our collective concepts written in our own words and where others' ideas or words have been included, we have adequately cited and referenced the original sources. We further declare that we have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in our submission. We understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed. Further, we affirm that the contents of this report have not been submitted and will not be submitted either in part or in full, for the award of any other degree or diploma and the same is certified.

Place : Vellore　　　　　　　　Signature of the Candidate(s)

Date: 29/04/2024

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

School of Computer Science Engineering

BONAFIDE CERTIFICATE

This is to certify that the project report entitled "**Lane Detection using CNN**" submitted by us to Vellore Institute of Technology University, Vellore, in partial fulfillment of the requirement for the award of the degree of Integrated M. Tech in Computer Science is a record of bonafide work carried out by him/her under my guidance. The project fulfills the requirements as per the regulations of this institute and in my opinion meets the necessary standards for submission. The contents of this report have not been submitted and will not be submitted either in part or in full, for the award of any other degree or diploma and the same is certified.

Project Guide **Head of the Department**

Internal Examiner **External Examiner**

ACKNOWLEDGEMENT

We, the group of five members, would like to express our sincere gratitude to everyone who has contributed to the successful completion of our project entitled "Lane Detection using CNN".

First and foremost, we would like to express our heartfelt thanks to our guidance professor, Anuradha J, for providing us with the necessary guidance and motivation throughout the project. Her valuable suggestions, continuous support, and timely feedback were instrumental in shaping our project and enhancing its quality. We also express our gratitude to the management and faculty of SCOPE in Vellore Institute of Technology, Vellore, for providing us with an excellent platform to pursue our project and their constant encouragement and support.

We extend our thanks to all the individuals who have helped us in various ways during the project. Their cooperation, insights, and encouragement have been invaluable to us. Last but not least, we would like to express our sincere appreciation to our family and friends for their unwavering support, encouragement, and patience throughout the project.

Thank you all for your valuable contributions and support in making our project a success.

Place : Vellore <Signature of the Student>

(Name of the Student)

Date : 28/04/2024

Table of Contents:

ABSTRACT:

In today's fast-paced world, the ubiquitous use of vehicles is essential for fulfilling daily life needs. However, this reliance on vehicles also brings about inherent risks, including the potential for serious injuries and loss of life. This report proposes a novel approach to improving lane detection in autonomous vehicles by leveraging Convolutional Neural Networks (CNN).The proposed methodology involves extracting robust features from street images using CNNs and subsequently training a tree-like regression model based on the estimated lane line positions. By employing CNNs for lane detection, the technique is advanced through the implementation of example Segmentation, leading to a dual model approach. This dual model architecture enhances the accuracy and reliability of lane detection in complex urban environments. The primary technology utilized in this project is Convolutional Neural Networks (CNNs), which are powerful deep learning models capable of extracting intricate features from images. The implementation of this technique is facilitated through popular deep learning libraries such as Keras and TensorFlow, both of which are integrated with Python, providing a flexible and efficient platform for experimentation and deployment. Overall, this research presents a robust methodology for improving lane detection in autonomous vehicles, leveraging the capabilities of CNNs and demonstrating the potential of deep learning techniques to enhance safety and efficiency in transportation systems.

INTRODUCTION:

Driving is an integral part of our lives, relying heavily on our visual perception to navigate the roads safely. The markings on the road serve as our constant guide,helping us steer within designated lanes. However, the consequences of careless driving are severe and can lead to accidents, some even resulting in fatalities. This pervasive issue demands immediate attention and innovative solutions to enhance road safety.

Challenges on the Roads:

Accidents have become an unfortunate reality of our daily lives, with individuals losing their lives due to mistakes made on the road. The complexity of traffic and the risks associated with wrong lane selection contribute to the growing challenges on our roads. Addressing these issues is crucial to creating a safer and more efficient driving environment.

Technological Solution:

To tackle the rising concerns of road safety, the implementation of self-learning vehicles equipped with Automatic Lane Detection technology presents a promising solution. This technology not only predicts lane trajectories but also alerts drivers to make necessary lane changes. In critical situations, it can autonomously apply brakes to prevent potential collisions, marking a significant leap forward in enhancing overall safety.

Leveraging CNN for Lane Detection:

An essential aspect of developing self-driving vehicles is the integration of Convolutional Neural Networks (CNN) to automate the detection of lane lines. By employing CNN, we can extract robust features from road images, enabling the training of regression models. This technological advancement holds the key to creating vehicles that can autonomously navigate roads by interpreting and responding to lane markings, ensuring a safer and more secure driving experience for everyone.

HARDWARE AND SOFTWARE REQUIREMENTS:

Hardware requirements:

- CPU: Any modern multi-core CPU should suffice for running this code. Since the code does not explicitly mention GPU acceleration or deep learning training, a CPU with at least 2 cores should be adequate.
- Memory (RAM): A minimum of 4 GB of RAM is recommended, although having more RAM would be beneficial, especially if you plan to work with large images or videos.
- Storage: Sufficient storage space is needed to store the Python script, any required datasets or videos, and the output images or videos generated by the code.

Software requirements:

- Python: The code is written in Python, so you need to have Python installed on your system. Python 3.x is recommended.
- Python Libraries: The code relies on several Python libraries, including but not limited to:
  OpenCV (cv2):
  OpenCV is a popular library for computer vision tasks.
  It's used for reading images and videos, as well as for various image processing operations like color conversion, filtering, and edge detection.
  NumPy:

NumPy is a fundamental package for numerical computing with Python.

It's used for efficient array operations and mathematical computations.

Matplotlib:

Matplotlib is a plotting library for Python.

It's used for creating static, interactive, and animated visualizations in Python.

Pickle:

Pickle is a module used for serializing and deserializing Python objects.

It's used for saving and loading trained ML models or other Python objects to/from disk.

- Operating System: The code should be compatible with most major operating systems, including Windows, macOS, and Linux.

LITERATURE REVIEW:

a. Flexible Lane Detection using CNNs

There are approaches to lane line detection: one is primarily based on general machine learning techniques, while the alternative is based on a famous deep learning model in recent years. Traditional machine vision methods are heavily simulated through outside variables, and the cost is high. More and more researchers have utilized deep learning to detect lanes in recent years, with high-quality results. This method is distinguished by the fact that labeled datasets are often utilized to train models. This approach has a high level of precision, but because the label is set, the lane change scene is hard to alter and expand.

b. Dynamic Approach for Lane Detection using Google Street View and CNN

The use of a convolutional neural network (CNN) model based on SegNet encoder-decoder architecture is presented in this study as an original and pragmatic approach for lane detection. The encoder block creates low-resolution function maps from the input, while the decoder block classifies the function maps pixel-by-pixel. The suggested model was trained on a set of 2000 images after which compared to the ground reality provided in the dataset for evaluation. The discovered findings demonstrate that, due to the preprocessing required, the suggested approach is effective under intense occlusion conditions and offers better performance when compared to existing methods.

c. Gradient Map Based Lane Detection Using CNN and RNN

The majority of the research in the literature utilizes full-color images as the network's input. We provide an explanation for why an edge-based gradient map input may assist neural networks in increasing accuracy, processing time, and training time, particularly for any neural networks that could fit on low-power systems. We show that gradient map-based convolutional neural networks can achieve more accuracy at distinctive scales than RGB pixels, and that a compressed gradient map network can also achieve up to 3.6 times faster inference time while maintaining the same performance.

d. Multi-Lane Detection Using CNNs and A Novel Region-Grow Algorithm
They have cautioned a unique approach to robustly detect lanes in various circumstances in this study. HT (Hough Transform) can extract straight lines from images effectively in theory, but it cannot distinguish if the straight lines are lane markers. To address this challenge, they used a simple LeNet-modified convolutional neural network with geometric regulations to discriminate between different types of lines. In the subsequent phase, a region-grow technique is used to fit lanes, which is executed by progressively growing nearby ROI (Region-of-Interest).

e. Robust Lane Detection From Continuous Driving Scenes Using Deep Neural Networks.
Most methods depend on recognizing the lane from a single photo, which often results in poor performance when dealing with extreme conditions like strong shadow, severe mark degradation, severe car occlusion, and so on. In reality, lanes are road structures that run in a continuous line. As a result, the lane that cannot be effectively recognized in a single current frame can be inferred by combining information from prior frames. They explored lane detection using multiple frames from a continuous driving scenario and proposed a hybrid deep architecture that combines the convolutional neural network (CNN) and the recurrent neural network (RNN).

f. A review of lane detection methods based on deep learning
Lane detection is a software of environmental perception, which ambitions to detect lane areas or lane lines by using a camera. First, the paper covers the history of lane detection, consisting of traditional lane detection strategies as well as related deep learning strategies. Second, it divides existing lane detection strategies into two sorts: step and one-step. The creator introduces lane detection methods from perspectives that encompass network architectures, including type and object detection-based techniques, and end-to-end image segmentation.

g. CNN-based lane detection with instance segmentation in edge-cloud computing
In recent days, the use of cars is relatively increasing, and vehicles with self-driving capabilities are gaining popularity. While lane detection combined with cloud computing can effectively deal with the shortcomings of conventional lane detection based on function extraction and high definition. The traditional lane detection approach is advanced, and a dual model based on instance segmentation is built using the current popular convolutional neural network (CNN). The distributed computing architecture provided by edge-cloud computing is used to improve statistics processing efficiency in the image acquisition and processing processes. To achieve effective detection, using the lane fitting technique we can generate a variable matrix, which improves the real-time performance of lane detection. The method proposed in this paper has

produced accurate lane reputation results in a selection of scenarios, and the lane recognition performance is significantly better than that of other lane recognition models.

h. A machine learning approach for detecting and tracking road boundary lanes
Road boundary lanes are one of the most common causes of traffic injuries, and they endanger both the driver and the public. Both computer vision and machine learning approaches have difficulty detecting road boundary lanes. Many machine learning algorithms have been implemented in recent years, but they have failed to produce high efficiency and accuracy. This paper presents a novel technique to alert the driver when the car crosses the road boundary lanes using machine learning techniques in order to avoid road mishaps and ensure driver protection. The dataset's performance is measured by the generation of experimental results. The proposed technique produced high precision and high performance.

i. Traffic Lane Detection using Fully Convolutional Neural Network
Many research on traffic lane detection have been conducted by a selection of organizations. Most methods, however, detect lane areas using color functions or human-designed shape models. A traffic lane detection approach based on a fully convolutional neural network is proposed in this paper. A small neural network is built to implement function extraction from a large number of images in order to extract the best lane feature. Lane marking can be easily achieved with detected lane pixels using random pattern consensus rather than complicated post-processing. The class accuracy of the class network model for each class is more than 97.5%, according to experimental results. In 29 different street scenes, the detection accuracy of the model trained by the proposed detection loss characteristic can reach 82.24%.

j. Comparing of Some CNN Architectures for Lane Detection.
Advanced driver assistance capabilities are a valuable resource in the prevention of human-caused injuries, as well as the reduction of harm and costs. One of the most essential functions is lane-keeping assist, which prevents careless lane modifications and continues the car safely in its lane. Many studies used conventional computer vision strategies to detect lanes using an onboard camera on the vehicle as a cost-effective sensor solution. Some well-known CNN architectures have been used as the inspiration for developing a deep neural network in this study. The lane line coefficients for fitting a 2nd-order polynomial were outputs from this network.

k. Lane Detection and Classification Using Cascaded CNNs
Lane detection is extremely important for autonomous vehicles. For this reason, many methods use lane boundary information to detect the vehicle within the road or to integrate GPS-based localization. As with many other computer vision-based applications, convolutional neural networks (CNNs) constitute the state-of-the-art technology to identify lane obstacles. To build the system, 14336 lane barriers images of the TuSimple dataset for lane detection were labeled using 8 different instructions. Our dataset and the code for inference are available online.

COMPARATIVE STUDY ON VARIOUS PREVIOUSLY EXISTING APPROACHES:

Title	Year	Methodology	Advantages	Issues	Metrics Used	Author
Flexible lane detection using CNNs	2021	Convolutional Neural Network (CNN) Image Segmentation	Accurately performs lane boundary recognition	One limitation of CNNs is that they are not able to predict lanes obscured by obstacles or strong light	Accuracy, Precision	Li Haixia, Li Xizhou
Dynamic Approach for Lane Detection using Google Street View and CNN	2019	Machine-learning classifiers like Gaussian-mixture model (GMM), support-vector machine model (SVM), and multi-layered Perceptron classifier (MLPC) CNN Image Segmentation	Uses Google Street API, takes care of obstacles in the lane	Takes much time to train the model if the system specs are low	97% efficient if the training dataset is large	Rama Sai Mamidala, Uday Uthkota
Gradient Map Based Lane Detection Using CNN and RNN	2020	Convolutional Neural Network (CNN), Recurrent neural networks (RNN)	CNNs trained with gradient maps gain higher accuracy, reduces training and inference time	Training time is high	CNN with the gradient map achieved higher accuracy of 0.1% to 1.6%	Wu, Jiacheng Cui, Han
Multi-Lane Detection Using CNNs and A Novel Region-	2019	Hough Transform, Convolutional Neural Network (CNN),	Uses RANSAC to get the exact directions	-	Accuracy	Sun, Y, Li, J, Sun, Z.P.

grow Algorithm		Random sample consensus (RANSAC) to extract the main direction in local ROI (Region of interest)				
Robust Lane Detection From Continuous Driving Scenes Using Deep Neural Networks	2019	Convolutional Neural Network (CNN), Recurrent neural networks (RNN)	Higher performance, higher precision, robust	Detected lanes are not smooth	Performance, Precision, Accuracy	Qin Zou
A review of lane detection methods based on deep learning	2021	Image segmentation, Network Architectures, semi-supervised learning, meta-learning	-	Expensive computation and lack of generalization	Accuracy	JIGANG TANG
CNN based lane detection with instance segmentation in edge-cloud computing	2020	Cloud computing, CNN	Cloud data processing combined with edge computing can effectively reduce the computing load of the central nodes	Excessive calculation	Accuracy	Wei Wang, Hui Lin, Junshu Wang
A machine learning approach for detecting and tracking road boundary lanes	2021	Computer Vision, Machine learning	Edge detection using CNN, CNN combined with line detection (CNN-LD) offers accuracy and precision	-	Accuracy, Precision, Recall, F measure	Satish Kumar Satti, K Suganya Devi, Prasenjit Dhar, P Srinivasan

Traffic Lane Detection using Fully Convolutional Neural Network	2018	CNN	Lane marking is easily realized by random sample consensus rather than complex post-processing	-	Accuracy	Jinji Zang, Wei Zhou, Gyanwen Zhang, Zhemin Duan
Comparing of Some Convolutional Neural Network (CNN) Architectures for Lane Detection	2020	CNN, Computer Vision	The developed model gives more crucial lane line coefficients for advanced driver assistance, keeping the car safely in the lane	Original resolution images can't be used, need to reduce the input image resolution	Accuracy	O. T. EKŞİ
Lane detection and classification using cascaded CNNs	-	Convolutional Neural Network (CNN), Recurrent neural networks (RNN), convolutional neural networks (CNNs)	-	-	Accuracy	-
Dynamic approach for Lane Detection using Google Street View and CNN	-	Lane detection algorithms have been the key enablers for fully-assistive and autonomous navigation systems. In this paper, a novel and pragmatic approach for lane detection	-	-	Accuracy, Precision	Rama Sai Mamidala, Uday Uthkota

		is proposed using a convolutional neural network (CNN) model based on SegNet encoder-decoder architecture. To enable real-time navigation, they extend their model's predictions interfacing it with the existing Google APIs, evaluating the metrics of the model tuning the hyper-parameters.				
A deep learning based fast lane detection	-	The main idea of the proposed method is to use one-dimensional pixel intensity distributions to detect lane markings. Since the lane markings have a special pattern, this one-dimensional distribution may provide sufficient discrimination for lane marking detection. Comparing the performance of	-	-	Accuracy	Erkan Oğuz, Ayhan Küçükmanisa, Ramazan Duvar, Oğuzhan Urhan

		the proposed method with image processing and deep learning based methods in comprehensive analysis. Providing high accuracy with very low computational load compared to high-performance deep learning models.				
Learning Lightweight Lane Detection CNNs by Self Attention Distillation	2019	We validate SAD on three popular lane detection benchmarks (TuSimple, CULane, and BDD100K) using lightweight models such as ENet, ResNet. The lightest model, ENet-SAD, performs comparatively or even surpasses existing algorithms. Notably, ENet-SAD has 20x fewer parameters and runs 10x faster compared to the state-of-the-art SCNN while still achieving compelling	-	-	Accuracy	Yuenan Hou, Zheng Ma, Chunxiao Liu, Chen Change Loy

		performance in all benchmarks.				
A Fast Learning Method for Accurate and Robust Lane Detection Using Two-Stage Feature Extraction with YOLO	2017	A CNN can be used to enhance the input images before lane detection by excluding noise and obstacles that are irrelevant to the edge detection result. Further, we modify the backpropagation algorithm to find the targets of hidden layers and effectively learn network weights while maintaining performance.	-	-	Accuracy, Precision	Jihun Kim, Jonghong Kim, Gil-Jin Jang, Minho Lee
Traffic Lane Detection using Fully Convolutional Neural Network	2019	The parameters of lane classification network model are utilized to initialize layers' parameters in lane detection network. In particular, a detection loss function is proposed to train the fully convolutional lane detection network whose output is pixel-wise detection of lane categories	-	-	Accuracy	Jinjuzang, Weizhou

		and location. The designed detection loss function consists of lane classification loss and regression loss. With detected lane pixels, lane marking can be easily realized by random sample consensus rather than complex post-processing.				

DRAWBACKS OF EXISTING SYSTEM:

Identified Gaps:

Lane Detection performance varies with Different Datasets:

Convolutional neural networks (CNNs) have demonstrated outstanding performance in vision-based lane detection. However, due to the dataset bias between the training and test datasets, sustaining the performance of the trained models under fresh test situations remains difficult. The dataset bias in lane detection algorithms may be divided into two types: lane position bias and lane pattern bias, with the former having a greater impact on lane detection performance.

Problems Faced:

Image and Video clarity might reduce in the future:

Lane detection warning is one important feature in Advanced Driver Assistance Systems (ADAS), which aims to improve overall safety on the road. However, challenges such as inconsistent shadows and fading lane markings often plague the road surface and cause the lane detection system to produce false warnings. Users are aggravated by the warning and tend to disable this safety feature.

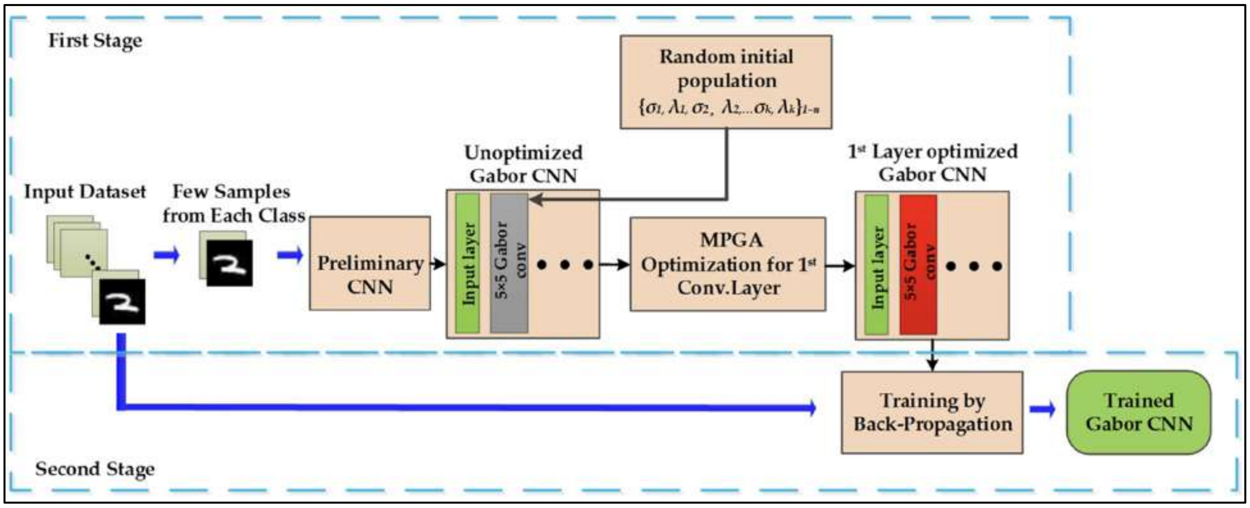
SOLUTION PROPOSED:

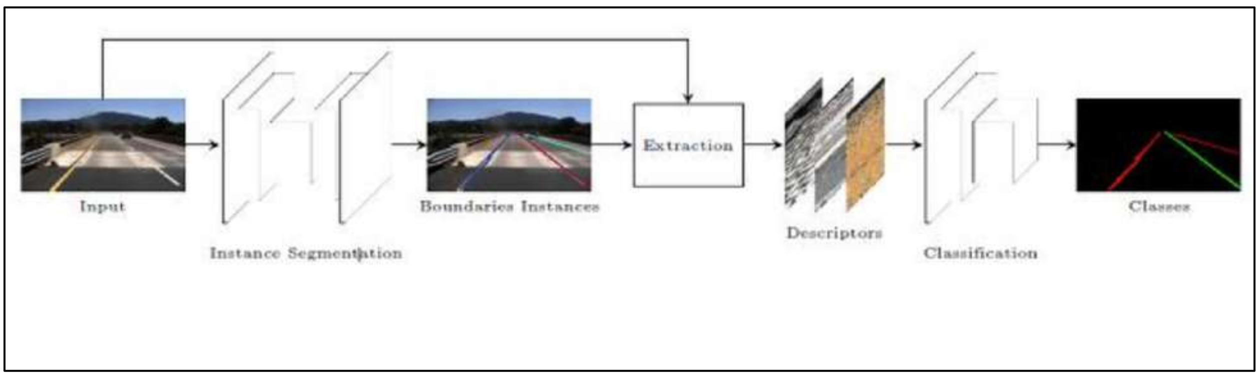
We propose an efficient Gabor filtering-based lane detection method to overcome the mentioned conditions and improve the accuracy of lane departure warning systems. Furthermore, it serves as a cost-effective solution to the lane departure warning problem, allowing for wide deployment. It is heuristically found that lane markings have a general directional property, which can be further enhanced by Gabor filters while suppressing inconsistent road shadows and markers. Enhanced lane markings are then subjected to adaptive Canny edge detection to extract distinct edge markings. Lastly, transformation is applied to label the correct lane candidates on the road surface. Additionally, we generated a dataset of Chinese roads with various driving conditions. As a proof of concept, a lane departure warning system is built based on the proposed lane detection method, which achieves an accuracy of 93.67% for lane detection and 95.24% for lane departure warning when tested on our challenging dataset.

METHODOLOGY:

- Input Preprocessing: The code begins by preprocessing the input image. This might involve converting the image to grayscale to simplify processing and reduce computational load. Additionally, any necessary resizing or normalization of the image is performed to ensure consistency and compatibility with subsequent processing steps.
- Gabor Filtering: Gabor filtering is applied to the preprocessed image. This step involves convolving the image with a set of Gabor filters, which are complex sinusoidal functions modulated by a Gaussian envelope. The result is a series of filtered images, each capturing texture and edge information at various orientations and scales.
- Canny Edge Detection: Next, Canny edge detection is performed on the filtered images to identify prominent edges. This technique involves several steps including noise reduction through Gaussian smoothing, computation of gradients to detect intensity changes, non-maximum suppression to refine edges, and hysteresis thresholding to retain only strong edges. The output is a binary image highlighting the detected edges.
- Hough Transformation: The binary image generated from Canny edge detection is then processed using the Hough transformation technique. This involves representing lines within the image as points in a parameter space, where each point corresponds to a line. By analyzing the parameter space, the algorithm identifies peaks corresponding to lines, effectively detecting straight lines within the image. These lines are likely to represent lane markings.
- Lane Detection and Localization: Once the lines representing lane markings are detected, post-processing techniques are applied to refine and localize the detected lanes. This may involve filtering out noise, averaging and extrapolating lines to determine the boundaries of each lane, and calculating lane curvature or other relevant metrics.
- Visualization and Output: Finally, the detected lanes are visualized on the original input image or presented in a separate output image. This visualization provides a clear indication of where the algorithm has detected lane markings, facilitating further analysis or decision-making in applications such as autonomous driving or lane departure warning systems.

SYSTEM DIAGRAM:





SYSTEM DIAGRAM EXPLANATION:

Input: This refers to the data fed into the system for lane tracking. Inputs typically consist of images or videos captured by cameras mounted on vehicles or drones.

Interface Segmentation: Interface segmentation involves segmenting the input images or videos to isolate the regions containing lanes.

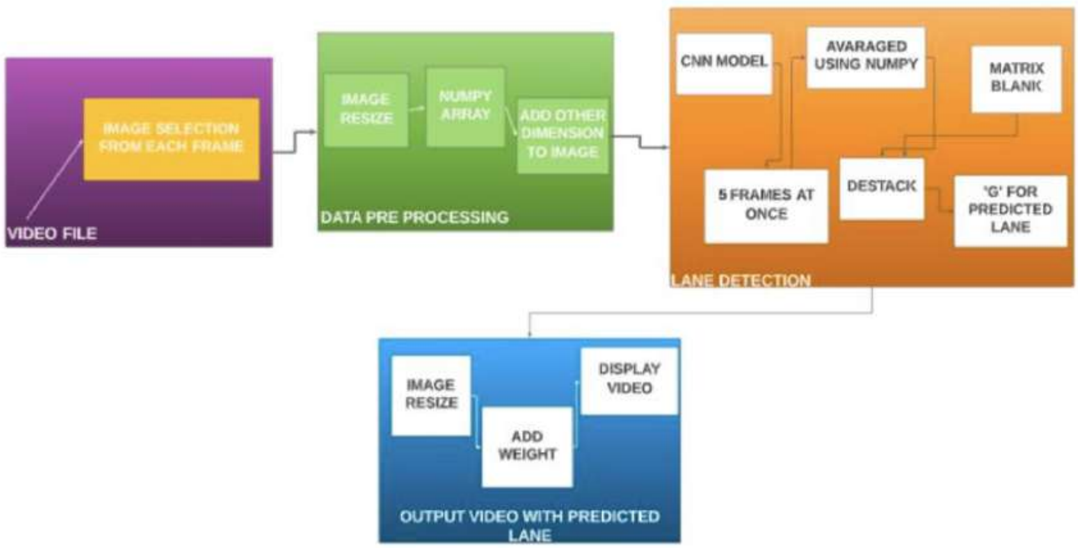
Boundaries Instance Extraction: Once the lane regions are identified through segmentation, the system extracts the boundaries of individual lane instances.

Descriptors: Descriptors are features extracted from the lane boundaries that encode essential characteristics such as curvature, slope, texture, and any other relevant information.

Classification: Classification involves categorizing the lanes based on their descriptors and other extracted features. This step typically assigns labels or classes to the lanes etc.

Classes: Classes refer to the different categories or types of lanes that the system can recognize and track. Common lane classes include straight lanes etc.

MODULES:



DETAILED DESCRIPTION OF MODULES:

VIDEO FILE :

Image selection from each file refers to the process of choosing representative frames from a video input for further analysis or processing. In the context of a video input module, this process involves selecting key frames that capture important moments or changes in the scene, rather than analyzing every single frame. The purpose of this module is to reduce computational load and streamline processing by focusing on relevant frames, particularly in scenarios where continuous analysis of every frame is unnecessary.

DATA PREPROCESSING :

In preparing video data for the "Precision Lane Tracking: A CNN-Centric Approach" project, several essential preprocessing steps are undertaken to optimize input for Convolutional Neural Network (CNN) analysis. Initially, the video is broken down into individual frames to enable frame-by-frame lane tracking. Image enhancement techniques are then applied to improve lane visibility, including contrast adjustments, brightness normalization, and noise reduction.

LANE DETECTION :

In the "Precision Lane Tracking: A CNN-Centric Approach" project, the lane detection module is a fundamental component aimed at accurately identifying and tracking lane markings on roads to enhance vehicle navigation and safety. Beginning with input data, typically images or video frames captured by onboard cameras, the module undergoes preprocessing to enhance image quality and reduce noise. Convolutional Neural Networks (CNNs) are then employed to extract relevant features from the images, including lane markings and visual cues indicative of lane boundaries.

CODE:

```
import numpy as np
import cv2
import matplotlib.pyplot as plt

# Define a class to receive the characteristics of each line detection
class Line:
    def __init__(self):
        self.detected = False
        self.recent_xfitted = []
        self.bestx = None
        self.best_fit = None
        self.current_fit = [np.array([False])]
        self.radius_of_curvature = None
        self.line_base_pos = None
        self.diffs = np.array([0, 0, 0], dtype='float')
        self.allx = None
        self.ally = None

# Global variables
line_l = Line()
line_r = Line()
src = np.float32([[580, 460], [705, 460], [1130, 720], [190, 720]])
dst = np.float32([[300, 0], [980, 0], [980, 720], [300, 720]])
```

```
M = cv2.getPerspectiveTransform(src, dst)
Minv = cv2.getPerspectiveTransform(dst, src)
ym_per_pix = 30 / 720  # meters per pixel in y dimension
xm_per_pix = 3.7 / 700  # meters per pixel in x dimension

def abs_sobel_thresh(img, orient='x', sobel_kernel=3, thresh=(0, 255)):
    gray = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
    if orient == 'x':
        sobel = cv2.Sobel(gray, cv2.CV_64F, 1, 0, ksize=sobel_kernel)
    elif orient == 'y':
        sobel = cv2.Sobel(gray, cv2.CV_64F, 0, 1, ksize=sobel_kernel)
    abs_sobel = np.absolute(sobel)
    scaled_sobel = np.uint8(255 * abs_sobel / np.max(abs_sobel))
    grad_binary = np.zeros_like(scaled_sobel)
    grad_binary[(scaled_sobel >= thresh[0]) & (scaled_sobel <= thresh[1])] = 1
    return grad_binary

def mag_thresh(img, sobel_kernel=3, mag_thresh=(0, 255)):
    gray = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
    sobelx = cv2.Sobel(gray, cv2.CV_64F, 1, 0, ksize=sobel_kernel)
    sobely = cv2.Sobel(gray, cv2.CV_64F, 0, 1, ksize=sobel_kernel)
    abs_sobelxy = np.sqrt(sobelx ** 2 + sobely ** 2)
    scaled_sobel = np.uint8(255 * abs_sobelxy / np.max(abs_sobelxy))
    mag_binary = np.zeros_like(scaled_sobel)
    mag_binary[(scaled_sobel >= mag_thresh[0]) & (scaled_sobel <= mag_thresh[1])] = 1
    return mag_binary

def dir_thresh(img, sobel_kernel=3, thresh=(0, np.pi/2)):
    gray = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
    sobelx = cv2.Sobel(gray, cv2.CV_64F, 1, 0, ksize=sobel_kernel)
    sobely = cv2.Sobel(gray, cv2.CV_64F, 0, 1, ksize=sobel_kernel)
    abs_sobelx = np.absolute(sobelx)
    abs_sobely = np.absolute(sobely)
    gradient_direction = np.arctan2(abs_sobely, abs_sobelx)
    dir_binary = np.zeros_like(gradient_direction)
    dir_binary[(gradient_direction >= thresh[0]) & (gradient_direction <= thresh[1])] = 1
    return dir_binary

def color_thresh(img, s_thresh=(0, 255), v_thresh=(0, 255)):
    hls = cv2.cvtColor(img, cv2.COLOR_RGB2HLS)
```

```
    s_channel = hls[:, :, 2]
    s_binary = np.zeros_like(s_channel)
    s_binary[(s_channel >= s_thresh[0]) & (s_channel <= s_thresh[1])] = 1
    hsv = cv2.cvtColor(img, cv2.COLOR_RGB2HSV)
    v_channel = hsv[:, :, 2]
    v_binary = np.zeros_like(v_channel)
    v_binary[(v_channel >= v_thresh[0]) & (v_channel <= v_thresh[1])] = 1
    c_binary = np.zeros_like(s_channel)
    c_binary[(s_binary == 1) & (v_binary == 1)] = 1
    return c_binary

def thresh_pipeline(img, gradx_thresh=(0, 255), grady_thresh=(0, 255), mag_thresh=(0, 255),
dir_thresh=(0, np.pi/2),
                    s_thresh=(0, 255), v_thresh=(0, 255)):
    gradx = abs_sobel_thresh(img, orient='x', sobel_kernel=3, thresh=gradx_thresh)
    grady = abs_sobel_thresh(img, orient='y', sobel_kernel=3, thresh=grady_thresh)
    mag_binary = mag_thresh(img, sobel_kernel=3, mag_thresh=mag_thresh)
    dir_binary = dir_thresh(img, sobel_kernel=3, thresh=dir_thresh)
    color_binary = color_thresh(img, s_thresh=s_thresh, v_thresh=v_thresh)
    combined_binary = np.zeros_like(color_binary)
    combined_binary[((gradx == 1) & (grady == 1)) | ((mag_binary == 1) & (dir_binary == 1)) |
(color_binary == 1)] = 1
    return combined_binary

def find_lane_pixels(binary_warped):
    # Take a histogram of the bottom half of the image
    histogram = np.sum(binary_warped[binary_warped.shape[0]//2:, :], axis=0)
    # Create an output image to draw on and visualize the result
    out_img = np.dstack((binary_warped, binary_warped, binary_warped))
    # Find the peak of the left and right halves of the histogram
    midpoint = np.int(histogram.shape[0]//2)
    leftx_base = np.argmax(histogram[:midpoint])
    rightx_base = np.argmax(histogram[midpoint:]) + midpoint
    # Choose the number of sliding windows
    nwindows = 9
    # Set height of windows
    window_height = np.int(binary_warped.shape[0]//nwindows)
    # Identify the x and y positions of all nonzero pixels in the image
    nonzero = binary_warped.nonzero()
    nonzeroy = np.array(nonzero[0])
```

```
nonzerox = np.array(nonzero[1])
# Current positions to be updated later for each window in nwindows
leftx_current = leftx_base
rightx_current = rightx_base
# Set the width of the windows +/- margin
margin = 100
# Set minimum number of pixels found to recenter window
minpix = 50
# Create empty lists to receive left and right lane pixel indices
left_lane_inds = []
right_lane_inds = []

# Step through the windows one by one
for window in range(nwindows):
    # Identify window boundaries in x and y (and right and left)
    win_y_low = binary_warped.shape[0] - (window + 1) * window_height
    win_y_high = binary_warped.shape[0] - window * window_height
    win_xleft_low = leftx_current - margin
    win_xleft_high = leftx_current + margin
    win_xright_low = rightx_current - margin
    win_xright_high = rightx_current + margin
    # Draw the windows on the visualization image
    cv2.rectangle(out_img, (win_xleft_low, win_y_low),
                (win_xleft_high, win_y_high), (0, 255, 0), 2)
    cv2.rectangle(out_img, (win_xright_low, win_y_low),
                (win_xright_high, win_y_high), (0, 255, 0), 2)
    # Identify the nonzero pixels in x and y within the window #
    good_left_inds = ((nonzeroy >= win_y_low) & (nonzeroy < win_y_high) &
                  (nonzerox >= win_xleft_low) & (nonzerox < win_xleft_high)).nonzero()[0]
    good_right_inds = ((nonzeroy >= win_y_low) & (nonzeroy < win_y_high) &
                  (nonzerox >= win_xright_low) & (nonzerox < win_xright_high)).nonzero()[0]
    # Append these indices to the lists
    left_lane_inds.append(good_left_inds)
    right_lane_inds.append(good_right_inds)
    # If you found > minpix pixels, recenter next window on their mean position
    if len(good_left_inds) > minpix:
        leftx_current = np.int(np.mean(nonzerox[good_left_inds]))
    if len(good_right_inds) > minpix:
        rightx_current = np.int(np.mean(nonzerox[good_right_inds]))
```

```
    # Concatenate the arrays of indices (previously was a list of lists of pixels)
    try:
        left_lane_inds = np.concatenate(left_lane_inds)
        right_lane_inds = np.concatenate(right_lane_inds)
    except ValueError:
        # Avoids an error if the above is not implemented fully
        pass

    # Extract left and right line pixel positions
    leftx = nonzerox[left_lane_inds]
    lefty = nonzeroy[left_lane_inds]
    rightx = nonzerox[right_lane_inds]
    righty = nonzeroy[right_lane_inds]

    return leftx, lefty, rightx, righty, out_img

def fit_polynomial(binary_warped):
    # Find our lane pixels first
    leftx, lefty, rightx, righty, out_img = find_lane_pixels(binary_warped)

    # Fit a second order polynomial to each using `np.polyfit`
    left_fit = np.polyfit(lefty, leftx, 2)
    right_fit = np.polyfit(righty, rightx, 2)

    # Generate x and y values for plotting
    ploty = np.linspace(0, binary_warped.shape[0]-1, binary_warped.shape[0])
    try:
        left_fitx = left_fit[0] * ploty ** 2 + left_fit[1] * ploty + left_fit[2]
        right_fitx = right_fit[0] * ploty ** 2 + right_fit[1] * ploty + right_fit[2]
    except TypeError:
        # Avoids an error if `left` and `right_fit` are still none or incorrect
        print('The function failed to fit a line!')
        left_fitx = 1 * ploty ** 2 + 1 * ploty
        right_fitx = 1 * ploty ** 2 + 1 * ploty

    return left_fit, right_fit, left_fitx, right_fitx, ploty, out_img

def search_around_poly(binary_warped):
    # HYPERPARAMETER
    # Choose the width of the margin around the previous polynomial to search
```

```
    margin = 100

    # Grab activated pixels
    nonzero = binary_warped.nonzero()
    nonzeroy = np.array(nonzero[0])
    nonzerox = np.array(nonzero[1])
    left_fit = line_l.best_fit
    right_fit = line_r.best_fit

    left_lane_inds = ((nonzerox > (left_fit[0] * (nonzeroy ** 2) + left_fit[1] * nonzeroy +
                                   left_fit[2] - margin)) & (nonzerox < (left_fit[0] * (nonzeroy ** 2) +
                                                                         left_fit[1] * nonzeroy + left_fit[2] + margin)))
    right_lane_inds = ((nonzerox > (right_fit[0] * (nonzeroy ** 2) + right_fit[1] * nonzeroy +
                                    right_fit[2] - margin)) & (nonzerox < (right_fit[0] * (nonzeroy ** 2) +
                                                                           right_fit[1] * nonzeroy +
                                                                           right_fit[2] + margin)))

    # Extract left and right line pixel positions
    leftx = nonzerox[left_lane_inds]
    lefty = nonzeroy[left_lane_inds]
    rightx = nonzerox[right_lane_inds]
    righty = nonzeroy[right_lane_inds]

    # Fit new polynomials
    left_fit, right_fit, left_fitx, right_fitx, ploty, out_img = fit_polynomial(binary_warped)

    return left_fit, right_fit, left_fitx, right_fitx, ploty, out_img

def measure_curvature_real(ploty, left_fit_cr, right_fit_cr):
    # Define y-value where we want radius of curvature
    # We'll choose the maximum y-value, corresponding to the bottom of the image
    y_eval = np.max(ploty)

    # Calculation of R_curve (radius of curvature)
    left_curverad = ((1 + (2 * left_fit_cr[0] * y_eval * ym_per_pix + left_fit_cr[1]) ** 2) ** 1.5) /
np.absolute(
        2 * left_fit_cr[0])
    right_curverad = ((1 + (2 * right_fit_cr[0] * y_eval * ym_per_pix + right_fit_cr[1]) ** 2) **
1.5) / np.absolute(
        2 * right_fit_cr[0])
```

```
    return left_curverad, right_curverad

def measure_lane_offset(img_shape, left_fitx, right_fitx):
    # Calculate lane center
    lane_center = (left_fitx[-1] + right_fitx[-1]) / 2
    # Calculate image center in x dimension
    image_center = img_shape[1] / 2
    # Calculate lane offset from center and convert to meters
    lane_offset = (image_center - lane_center) * xm_per_pix
    return lane_offset

def draw_lanes(original_img, binary_img, left_fit, right_fit, Minv):
    # Create an image to draw the lines on
    warp_zero = np.zeros_like(binary_img).astype(np.uint8)
    color_warp = np.dstack((warp_zero, warp_zero, warp_zero))

    # Generate x and y values for plotting
    ploty = np.linspace(0, binary_img.shape[0] - 1, binary_img.shape[0])
    left_fitx = left_fit[0] * ploty ** 2 + left_fit[1] * ploty + left_fit[2]
    right_fitx = right_fit[0] * ploty ** 2 + right_fit[1] * ploty + right_fit[2]

    # Recast the x and y points into usable format for cv2.fillPoly()
    pts_left = np.array([np.transpose(np.vstack([left_fitx, ploty]))])
    pts_right = np.array([np.flipud(np.transpose(np.vstack([right_fitx, ploty])))])
    pts = np.hstack((pts_left, pts_right))

    # Draw the lane onto the warped blank image
    cv2.fillPoly(color_warp, np.int_([pts]), (0, 255, 0))

    # Warp the blank back to original image space using inverse perspective matrix (Minv)
    newwarp = cv2.warpPerspective(color_warp, Minv, (original_img.shape[1],
original_img.shape[0]))
    # Combine the result with the original image
    result = cv2.addWeighted(original_img, 1, newwarp, 0.3, 0)

    return result

def process_image(img):
    # Undistort the image
```

```
    undistorted = cv2.undistort(img, mtx, dist, None, mtx)
    # Apply thresholds
    thresholded = thresh_pipeline(undistorted, gradx_thresh=(20, 100), grady_thresh=(20, 100),
                                  mag_thresh=(30, 100), dir_thresh=(0.7, 1.3), s_thresh=(170, 255),
v_thresh=(20, 100))
    # Apply perspective transform
    warped = cv2.warpPerspective(thresholded, M, thresholded.shape[::-1],
flags=cv2.INTER_LINEAR)
    # Detect lanes
    if not line_l.detected or not line_r.detected:
        left_fit, right_fit, left_fitx, right_fitx, ploty, out_img = fit_polynomial(warped)
    else:
        left_fit, right_fit, left_fitx, right_fitx, ploty, out_img = search_around_poly(warped)

    # Calculate curvature and offset
    left_curverad, right_curverad = measure_curvature_real(ploty, left_fit, right_fit)
    curvature = (left_curverad + right_curverad) / 2
    lane_offset = measure_lane_offset(img.shape, left_fitx, right_fitx)

    # Draw the lane onto the original image
    result = draw_lanes(img, warped, left_fit, right_fit, Minv)

    # Add text about curvature and offset
    font = cv2.FONT_HERSHEY_SIMPLEX
    cv2.putText(result, 'Radius of Curvature = {:.2f} m'.format(curvature), (50, 50), font, 1, (255,
255, 255), 2,
                cv2.LINE_AA)
    cv2.putText(result, 'Vehicle is {:.2f} m left of center'.format(lane_offset), (50, 100), font, 1,
(255, 255, 255),
                2, cv2.LINE_AA)

    return result

# Load camera calibration data
dist_pickle = np.load('camera_cal/calibration_pickle.p')
mtx = dist_pickle['mtx']
dist = dist_pickle['dist']

# Test on a sample image
img = cv2.imread('test_images/test3.jpg')
```

```
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
result = process_image(img)
plt.imshow(result)
plt.show()
```

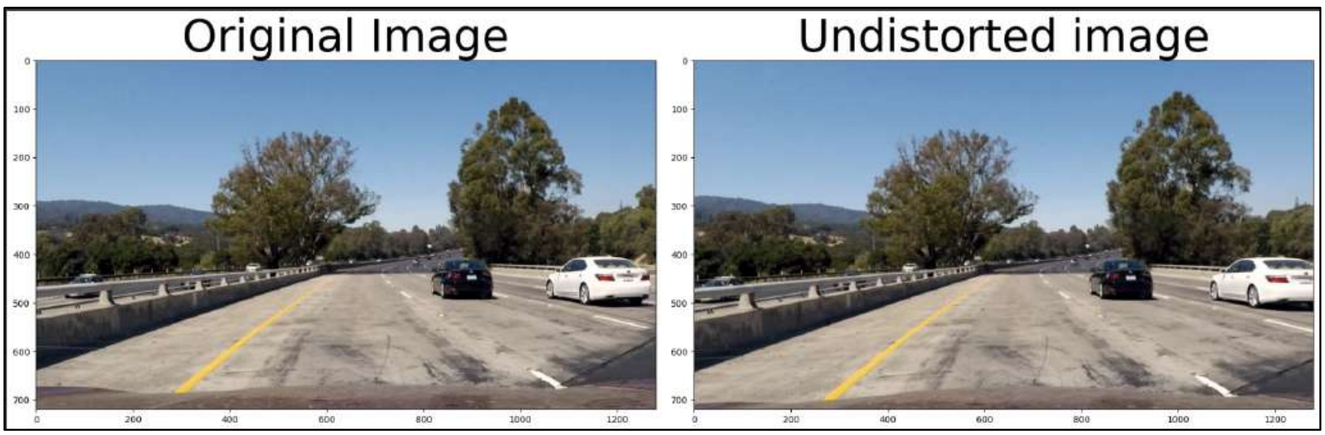
CODE IMPLEMENTATION:

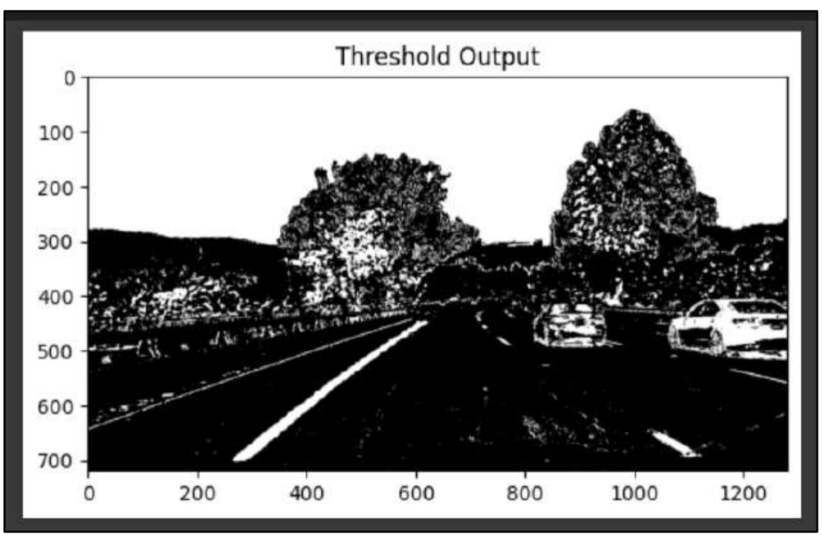
https://colab.research.google.com/drive/1qGnoJoDxV6WpQU1MMmxxwGqBBKtaERtX#scrollTo=rSdUK8nk0tYT

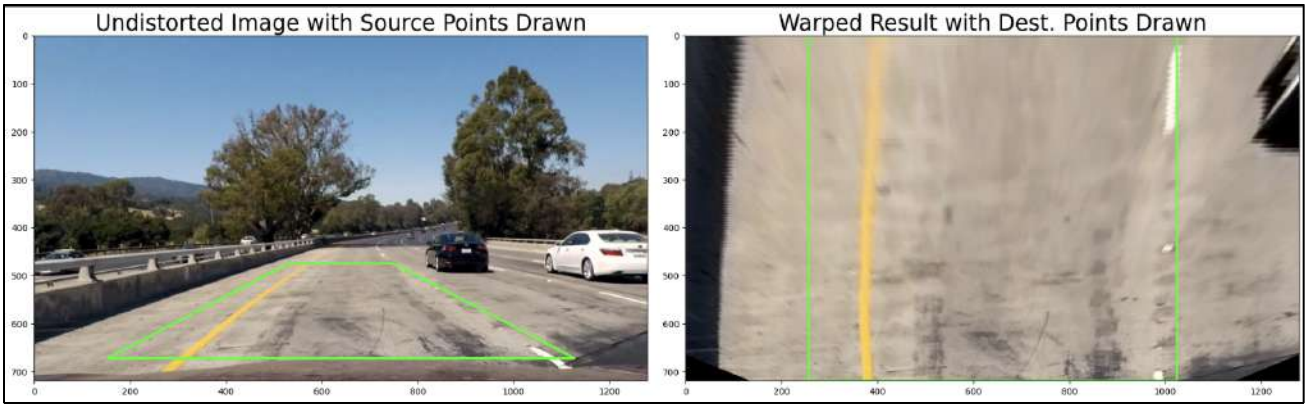
OUTPUT VIDEO LINK:

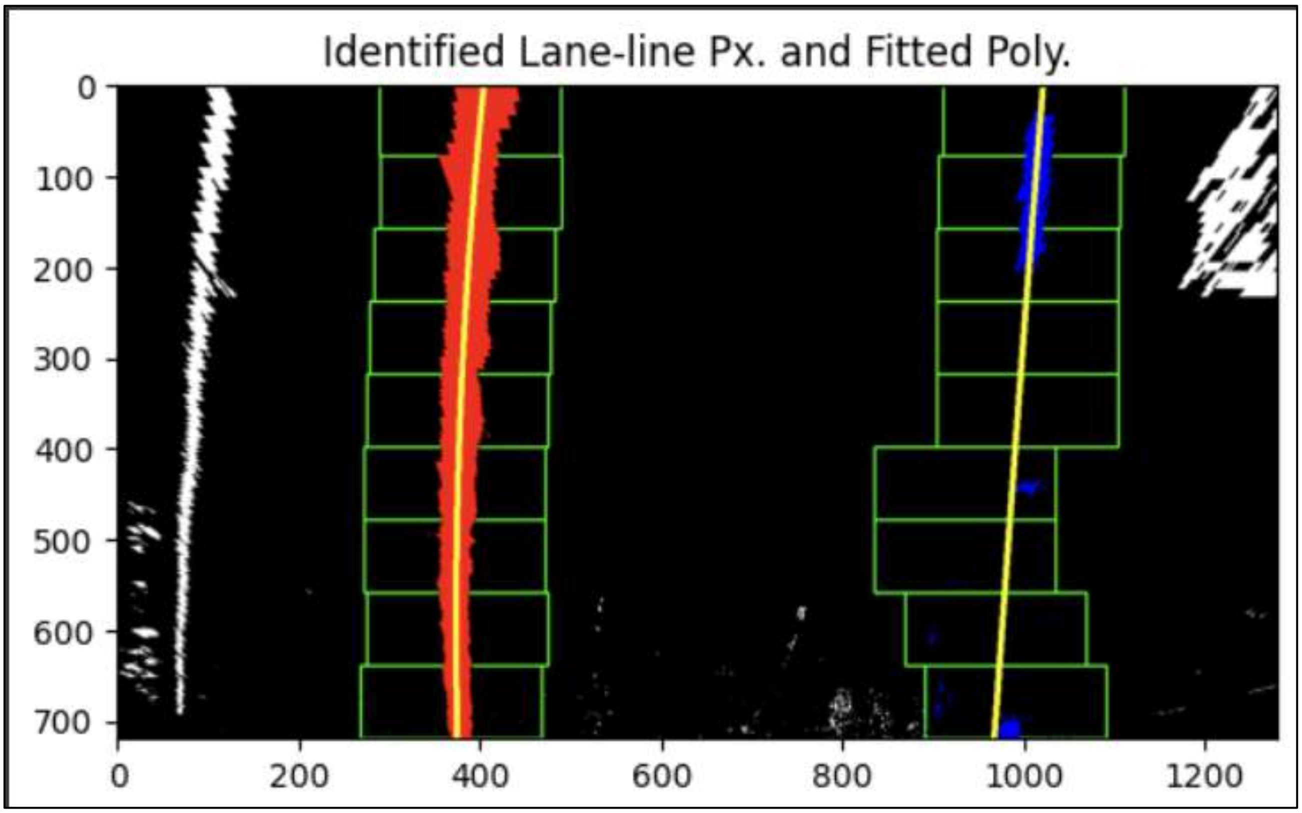
project_video_output_try17.mp4 - Google Drive

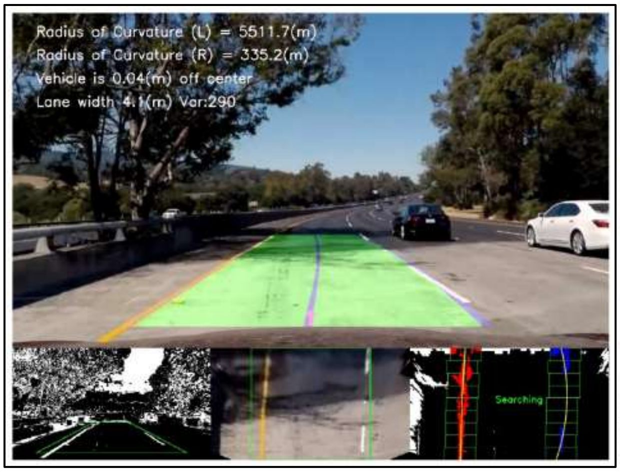
OUTPUT:

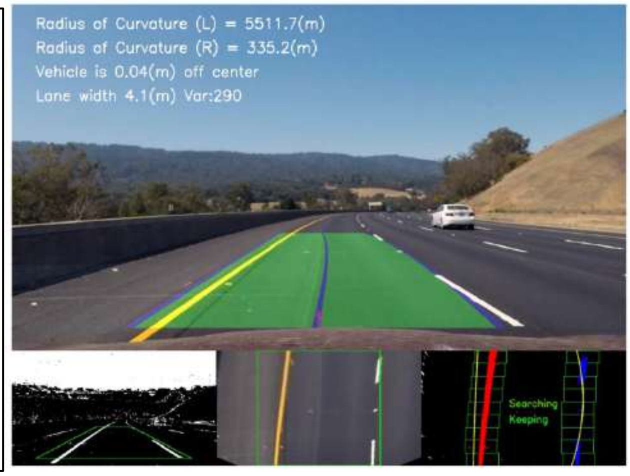


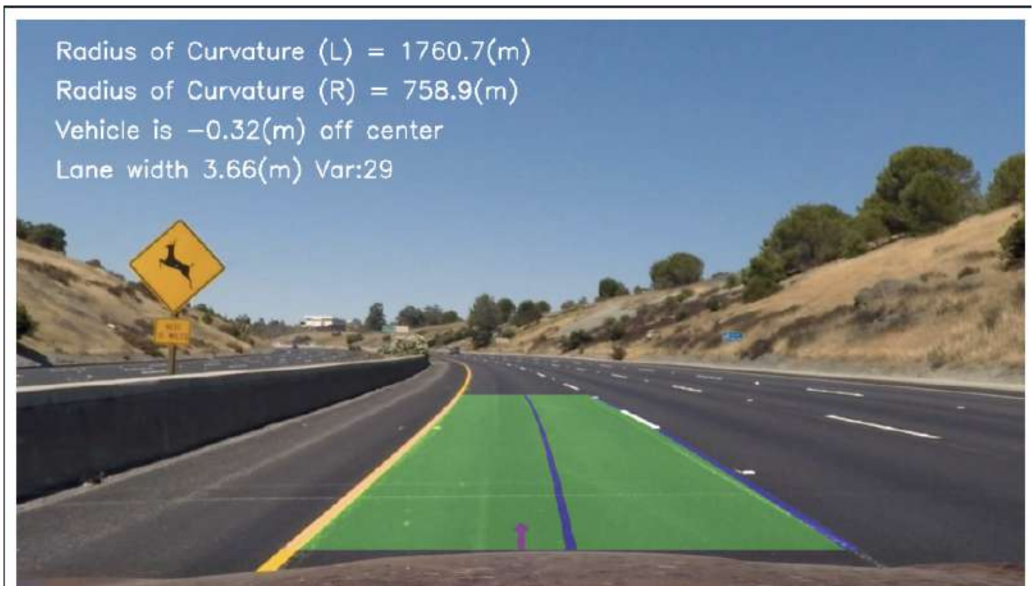



Undistorted Image with Source Points Drawn
Warped Result with Dest. Points Drawn


Identified Lane-line Px. and Fitted Poly.
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200


Radius of Curvature (L) = 5511.7(m)
Radius of Curvature (R) = 335.2(m)
Vehicle is 0.04(m) off center
Lane width 4.1(m) Var:290
Searching


Radius of Curvature (L) = 5511.7(m)
Radius of Curvature (R) = 335.2(m)
Vehicle is 0.04(m) off center
Lane width 4.1(m) Var:290
Searching
Keeping


Radius of Curvature (L) = 1760.7(m)
Radius of Curvature (R) = 758.9(m)
Vehicle is −0.32(m) off center
Lane width 3.66(m) Var:29


Radius of Curvature (L) = 5511.7(m)
Radius of Curvature (R) = 335.2(m)
Vehicle is 0.04(m) off center
Lane width 4.1(m) Var:290

CONCLUSION:

Lane detection, a crucial aspect of autonomous driving systems, benefits significantly from the integration of Convolutional Neural Networks (CNNs) and Gabor filtering. This research has illustrated the promising potential of this combined approach in achieving robust and accurate lane identification across diverse traffic situations.

CNNs offer a powerful tool for learning intricate features and patterns from images, enabling them to effectively recognize lanes in varying conditions. Their ability to generalize from training data facilitates adaptability to different environments, making them well-suited for real-world applications. Moreover, CNNs excel in capturing complex spatial relationships, which is essential for accurately delineating lane boundaries amidst challenging scenarios such as occlusions or varying road geometries.

Complementing CNNs with Gabor filtering further enhances lane detection performance by improving edge and texture information critical for lane recognition. Gabor filters effectively highlight lane markings and contours, enabling more precise lane boundary detection, especially in situations where lane markings are faint or obscured.

The integration of these strategies has demonstrated notable improvements in lane detection accuracy across a range of scenarios, including changing lighting conditions, diverse road textures, and varying lane markings. By leveraging the CNN's feature learning capabilities and the Gabor filter's ability to enhance lane characteristics, our system has shown promising results in reliably identifying lanes in challenging environments.

Continued research and development in this area hold great potential for further enhancing the robustness and real-time performance of lane detection systems. By refining algorithms, optimizing network architectures, and exploring additional data augmentation techniques, we can improve the system's ability to handle even more complex and dynamic traffic scenarios.

Ultimately, advancements in lane detection technology contribute significantly to the advancement of autonomous driving systems and overall road safety. By developing more dependable and adaptive lane recognition systems, we can accelerate the adoption of autonomous vehicles and mitigate the risks associated with human error, ultimately leading to safer and more efficient transportation systems for all road users.

REFERENCES:

[1] Aly, M. "Real time detection of lane markers in urban streets." In Proc. IEEE Intell Vehicles Symp., Jun 2008, pp. 7-12.

[2] Bar Hillel, A., R. Lerner, D. Levi, and G. Raz. "Recent progress in road and lane detection: A survey." Mach. Vis. Appl., vol. 25, no. 3, pp. 727-745, Apr. 2014.Borkar, A., M. Hayes, and M. T. Smith.

[3] "Robust lane detection and tracking with ransac and Kalman filter." In Proc. 16th IEEE Int. Conf. Image Process, (ICIP), Nov. 2009, pp. 3261-3264.

[4] EKŞİ, O. T., and G. GÖKMEN. "Comparing of Some Convolutional Neural Network (CNN) Architectures for Lane Detection." Balkan Journal of Electrical and Computer Engineering, 8(4), 314-319.

[5] Haixia, L., and L. Xizhou. "Flexible lane detection using CNNs." 2021 International Conference on Computer Technology and Media Convergence Design (CTMCD). IEEE, 2021.
[6] Huang, A. S., D. Moore, M. Antone, E. Olson, and Teller. "Finding multiple lanes in urban road networks with vision and lidar." Auto. Robots, vol. 26, nos. 2-3, pp. 103- 122, Apr. 2009.
[7] Kong, H., J.-Y. Audibert, and J. Ponce. "Vanishing point."
[8] Mamidala, R. S., et al. "Dynamic approach for lane detection using Google street view and CNN." TENCON 2019-2019 IEEE Region 10 Conference (TENCON). IEEE, 2019.
[9] Nan, Z., P. Wei, L. Xu, and N. Zheng. "Efficient Lane boundary detection with spatial-temporal knowledge filtering." Sensors, val. 16, no. 8, p. 1276, 2016.
[10] Neven, D., B. D. Brabandere, S. Georgoulis, M. Proesmans, and L. V. Gool. "Towards end-to-end lane detection: An instance segmentation approach." In Proc. IEEE Intell. Vehicles Symp. (IV), Jun. 2018, pp. 286- 291.
[11] Satti, S. K., et al. "A machine learning approach for detecting and tracking road boundary lanes." ICT Express, 7(1), 99-103, 2021.
[12] Sun, Y., J. Lu, and Z. P. Sun. "Multi-Lane Detection Using CNNs and A Novel Region-grow Algorithm." Journal of Physics: Conference Series, 1187(3), IOP Publishing, 2019.
[13] Tang, J., S. Li, and P. Liu. "A review of lane detection methods based on deep learning." Pattern Recognition, 111, 107623, 2021.
[14] Wang, W., H. Lin, and J. Wang. "CNN based lane detection with instance segmentation in edge-cloud computing." Journal of Cloud Computing, 9, 1-10, 2020.
[15] Wen, T., et al. "Bridging the Gap of Lane Detection Performance Between Different Datasets: Unified Viewpoint Transformation." IEEE Transactions on Intelligent Transportation Systems, 22(10), 6198-6207, Oct. 2021, doi: 10.1109/TITS.2020.2989349.
[16] Wu, J., H. Cui, and N. Dahnoun. "Gradient Map Based Lane Detection Using CNN and RNN." In 2020 22th International Conference on Digital Signal Processing and its Applications (DSPA). IEEE, 2020.
[17] Yoo, H., U. Yang, and K. Sohn. "Gradient-enhancing conversion for illumination-robust lane detection." IEEE Trans. Intell. Transp. Syst., 14(3), pp. 1083-1094, Sep. 2013.
[18] Yoo, H., S.-W. Lee, S.-K. Park, and D. H. Kim. "A robust lane detection method based on vanishing point estimation using the relevance of line segments." IEEE Trans. Intell. Transp. Syst., 18(12), pp. 3254-3266, Dec. 2017.
[19] Zang, J., et al. "Traffic lane detection using fully convolutional neural network." In 2018 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC). IEEE, 2018.
[20] Zou, Q., et al. "Robust lane detection from continuous driving scenes using deep neural networks." IEEE transactions on vehicular technology, 69(1), 41-54, 2019.