SELF-DRIVING CARS ON INDIAN ROADS USING MACHINE LEARNING

Project work submitted in partial fulfillment of the requirements for the award of degree of

Bachelor of Technology

in

Electronics and Communication Engineering

by

A. GANESH LAKSHMAN (19131A0408)

A. JAYAKAR RAVITEJA (19131A0410)

B. SAI PRIYANKA (19131A0416)

A. MEGHANA (19131A0405)

Under the guidance of

MRS. A. M. CH. JYOTHI

ASST. PROFESSOR

COLLEGE OF ENGINEERING
(AUTONOMOUS)

Department of Electronics and Communication Engineering
GAYATRI VIDYA PARISHAD COLLEGE OF ENGINEERING (AUTONOMOUS)
(Affiliated to J.N.T University, Kakinada, A.P)
VISAKHAPATNAM- 530 048
June, 2023

Gayatri Vidya Parishad College of Engineering
(Autonomous)
Madhurawada, Visakhapatnam - 530048
Approved by AICTE, New Delhi and Affiliated to JNTU - Kakinada
Accredited by NAAC with "A" Grade with a CGPA of 3.47/4.00

ELECTRONICS AND COMMUNICATION ENGINEERING DEPARTMENT

CERTIFICATE

This is to certify that the project titled **SELF-DRIVING CARS ON INDIAN ROADS USING MACHINE LEARNING** a bonafide record of the work done by **A. GANESH LAKSHMAN (19131A0408), A.JAYAKAR RAVITEJA (19131A0410), B.SAI PRIYANKA (19131A0416), A.MEGHANA (19131A0405)** in partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology** in **Electronics and Communication Engineering** of the Gayatri Vidya Parishad College of Engineering (Autonomous) affiliated to Jawaharlal Nehru Technological University, Kakinada during the year **2022-2023.**

Under the guidance of:
Mrs. A.M.Ch.Jyothi
Assistant Professor
Dept. of ECE,
GVPCE (A)

Head of the Department:
Dr. N.Deepika Rani
Professor &HoD,
Dept. of ECE,
GVPCE (A)

Project Viva-voce held on _____

Signature of the Internal Examiner **Signature of the External Examiner**

DECLARATION

This is to certify that the project titled **SELF-DRIVING CARS MACHINE LEARNING ON INDIAN ROADS** a bonafide record of the work done by **A. GANESH LAKSHMAN (19131A0408), A.JAYAKAR RAVITEJA (19131A0410), B.SAI PRIYANKA 19131A0416), A.MEGHANA (19131A0405)** in partial fulfillment of the requirement for the award of degree of B. Tech. in Electronics and Communication Engineering to Gayatri Vidya Parishad College of Engineering (Autonomous), affiliated to J.N.T. University, Kakinada comprises only our original work and due acknowledgement has been made in the text to all other material used.

Date: **SIGNATURES:**

A. GANESH LAKSHMAN (19131A0408)

A. JAYAKAR RAVITEJA (19131A0410)

B. SAI PRIYANKA (19131A0416)

A. MEGHANA (19131A0405)

ACKNOWLEDGEMENT

I thank **Prof. A. B. K. RAO**, Principal, Gayatri Vidya Parishad College of Engineering (Autonomous) for extending his utmost support and cooperation in providing all the provisions for the successful completion of the project.

I consider it our privilege to express our deepest gratitude to **Prof. Dr. N. DEEPIKA RANI** , Head of Department, Electronics and Communication Engineering for her valuable suggestions and constant motivation that greatly helped the project work to get successfully completed.

I would like to thank our guide **Mrs. A.M.Ch. Jyothi**, Assistant Professor, Department of ECE for helping us throughout the development of project. I am thankful for their support, guidance and valuable suggestions. I also thank all the members of the staff in Electronics and Communication Engineering for their sustained help in our pursuits.

Finally, I thank all those who contributed directly or indirectly in successfully carrying out this work.

ABSTRACT

Nowadays automation is playing a vital role in the development of infrastructure. Automation in the automobile industry is advancing in the direction of driverless cars. The aim of this project is to develop a self-driving car using deep learning algorithms.The project uses computer vision, machine learning and convolutional neural networks to recognize the image and processing. The project starts with basic image processing techniques such as grayscale conversion, smoothening images and edge detection. Further, it uses the Hough transform algorithm for detection of lane lines in the road.The project implements the LeNet architecture for classifying road symbols and also uses the Nvidia model for behavioral cloning. The project also uses data collection and preprocessing techniques, such as data balancing and image augmentation. Finally, the project uses Flask and Socket.io for live testing of the self-driving car.

It starts with the basics of supervised learning, classification, linear models, and error functions. Then it covers the use of neural networks, including feedforward, backpropagation and non-linear boundaries.The project uses the Udacity simulation software to train a neural network to drive a car in a simulated environment.

.

CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Self-driving vehicles were introduced in the US and were later criticized for their errors and unreliability. But we see autonomous vehicles as the future. Sooner or later improved technology is going to change our lives.
Machines will imitate human behavior in unforeseen manner. But India due to its terrain, traffic and economy lacks in the autonomous vehicles segment. India needs to be ready for a revolution in the automobile industry. Infrastructure is going to remain the same at least for decades, our infrastructure like roads are well thought out for the Indian economy.
Roads in rural parts of India are designed economically efficient, they were built without disturbing the local ecosystem. Also, due to weather conditions in India, roads are frequently damaged. Hence resulting in twists and turns. So we cannot expect major infrastructural change, but the autonomous vehicle technology is already there and is improving day by day. India needs to adapt this technology smartly by tweaking algorithms but not infrastructure.

The Society of Automotive Engineers (SAE) has defined six levels of autonomous driving, from Level 0 to Level 5, based on the amount of human intervention required.

Level 0: No Automation: The driver is responsible for all aspects of driving.

Level 1: Driver Assistance: The system can control either the acceleration or the steering, but not both simultaneously. The driver is responsible for all other aspects of driving.

Level 2: Partial Automation: The system can control both the acceleration and the steering simultaneously, but the driver must remain alert and ready to take control at any time.

Level 3: Conditional Automation: The system can take full control of the vehicle in certain situations, but the driver must be prepared to intervene if the system encounters a scenario it cannot handle.

Level 4: High Automation: The system can take full control of the vehicle in most driving scenarios, and the driver is not required to be constantly alert.

Level 5: Full Automation: The system can handle all driving scenarios, and there is no need for a driver at all.

1.1. Problem Statement

Our project is to train, improve and modify these self-driving machine learning models to **Indian Roads** and driving psychology.

Ex: In the US they use traffic cones, in India we rarely see cones we use barriers. There is a huge chance that a machine might think it's just another car and make wrong decisions.

Fig 1.1: Traffic Cones

1.2. OBJECTIVES :

- Develop a safe and reliable autonomous driving system that can accurately navigate a vehicle on roads, without the need for human intervention.
- Use advanced machine learning algorithms to process and analyze visual data from cameras mounted on the car and use this information to make informed decisions about steering, acceleration and braking.
- The self-driving car system will be able to plan and execute safe and efficient driving maneuvers, including lane changing, merging and turning.
- The self-driving car system will be able to detect and recognize traffic signs, traffic signals on the road.
- Incorporate data from sensors and other sources to create a comprehensive understanding of the vehicle's environment, including other cars, pedestrians, traffic lights and road signs.
- Optimize the vehicle's performance by developing algorithms for efficient route planning, fuel consumption, and traffic flow management.
- Ensure safety by implementing fail-safe mechanisms, emergency stop procedures, and error handling protocols.
- Test the system thoroughly in a variety of real-world driving conditions, and collect data to improve its accuracy and performance over time.

CHAPTER 2

LITERATURE SURVEY

The literature survey for this project involved study of various topics related to self-driving cars and computer vision. The survey highlighted the importance of deep learning techniques such as convolutional neural networks (CNNs) for object recognition, lane detection, and behavior cloning. It also emphasized the use of real-time control systems such as Flask and Socket.io for communication between the car and server.

- **A study by Bojarski et al. (2016)** used a CNN-based model to train a self-driving car to navigate a variety of road conditions, achieving an accuracy of **98.9%** on a closed track.

- **A study by Schoettle and Sivak (2014)** analyzed data from real-world trials of self-driving cars, finding that autonomous vehicles had a lower crash rate than human-driven vehicles.

- **A study by the University of Michigan** found that self-driving cars could reduce thenumber of car accidents by up to 90%. The study also estimated that self-driving cars could save up to $190 billion in medical expenses and other costs associated with car accidents.

- Additionally, the survey covered the various levels of autonomous vehicles, including Level 0 (no automation) to Level 5 (full automation). It discussed the current state of the industry and the challenges that need to be addressed for the widespread adoption of self-driving cars.

- Furthermore, the survey explored the different software tools and libraries used in the project, such as Anaconda, NumPy, OpenCV, Keras, and TensorFlow. It also highlighted the importance of data preprocessing techniques such as image augmentation for improving the accuracy of deep learning models.

Overall, the literature survey provided a comprehensive understanding of the state-of-the-art techniques and technologies used in self-driving cars and their applications. It helped to identify the gaps and opportunities for further research in this field.

CHAPTER 3

3.1 Software used:

For this project, it is recommended to install Anaconda Distribution, which includes Jupyter Notebooks and necessary packages like NumPy for scientific computing. Anaconda provides a user-friendly environment for managing dependencies and packages, making it easier to set up and run the project. Also we are using some additional tools like numpy, keras etc.

To install Anaconda on your computer

step1: Go to anaconda.com and click on download (anaconda automatically detects your computer's OS if you have turned on cookies).

step2: Complete the installation by following the instructions.

step3: Create a virtual environment by opening the conda terminal and running the below command.

```
conda create --name myenviron
```

step4: Activate this virtual environment by running the following.

```
conda activate myenviron
```

step5: Once you're in your environment run the following commands one after another.

```
conda install -c anaconda flask
conda install -c conda-forge python-socketio
conda install -c conda-forge eventlet
conda install -c conda-forge tensorflow
conda install -c conda-forge keras
conda install -c anaconda pillow
conda install -c anaconda numpy
conda install -c conda-forge keras
conda install -c conda-forge opencv
```

```
conda install -c conda-forge matplotlib

conda install -c conda-forge python-utils
```

3.2 DESCRIPTION OF LIBRARIES AND TOOLS

Anaconda : Anaconda is an open-source distribution of Python and R programming languages for scientific computing. It includes various libraries and packages that are commonly used in data science, such as NumPy, Pandas, Matplotlib, and Scikit-learn.

Fig 3.1: Logo of Anaconda

OpenCV : OpenCV (Open Source Computer Vision) is a popular computer vision library that provides various image processing and computer vision algorithms. In the self-driving car project, OpenCV was used for lane detection and object recognition.

Fig 3.2: Logo of OpenCV

TensorFlow : TensorFlow is a popular open-source machine learning library developed by Google. It provides various tools and APIs for building and training machine learning models. In the self-driving car project, TensorFlow was used for training the deep neural networks for behavior cloning.

Fig 3.3: Logo of Tensor flow

Keras : Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, Theano, or CNTK. In the self-driving car project, Keras was used for building the convolutional neural networks and training them for image recognition. Keras provides two modes to create the model, a simple and easy to use Sequential API as well as a more flexible and advanced Functional API.

Flask : Flask is a lightweight web application framework for Python. It is used for building web applications and RESTful APIs. In the self-driving car project, Flask was used for creating a server that communicated with the car in real-time.

Socket.IO : Socket.IO is a JavaScript library that enables real-time, bidirectional, and event-based communication between the browser and the server. In the self-driving car project, Socket.IO was used for real-time communication between the car and the server.

The Perceptron : It is a basic building block of neural networks that is used for binary classification. It takes inputs and weights, applies a step function to the sum of the inputs multiplied by their respective weights, and produces an output.

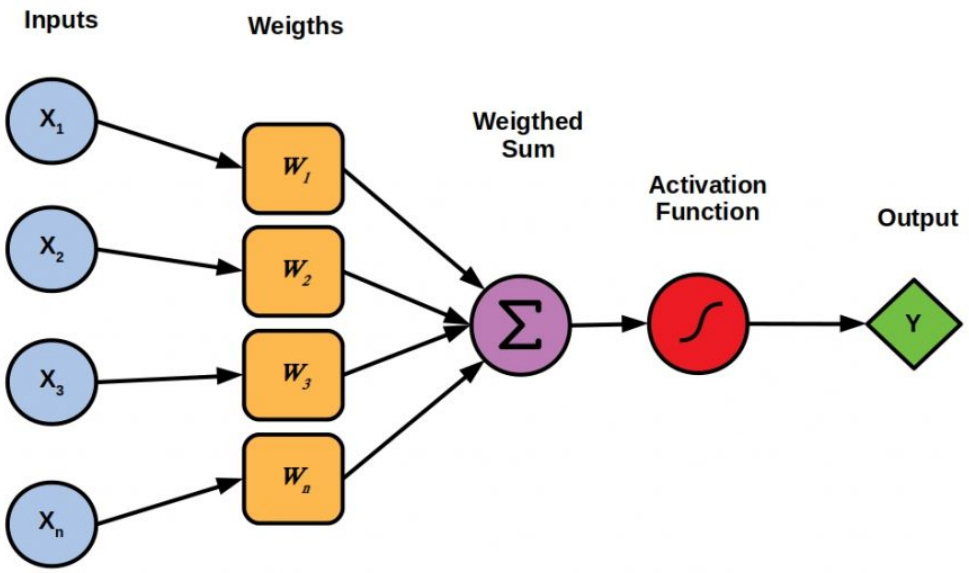


Fig 3.4: Perceptron

Machine Learning : It is the study of algorithms and statistical models that enable computer systems to learn from data without being explicitly programmed.

Supervised Learning : It is a type of machine learning where the algorithm learns from labeled data, i.e., data where the output is known, and tries to predict the output for new inputs.

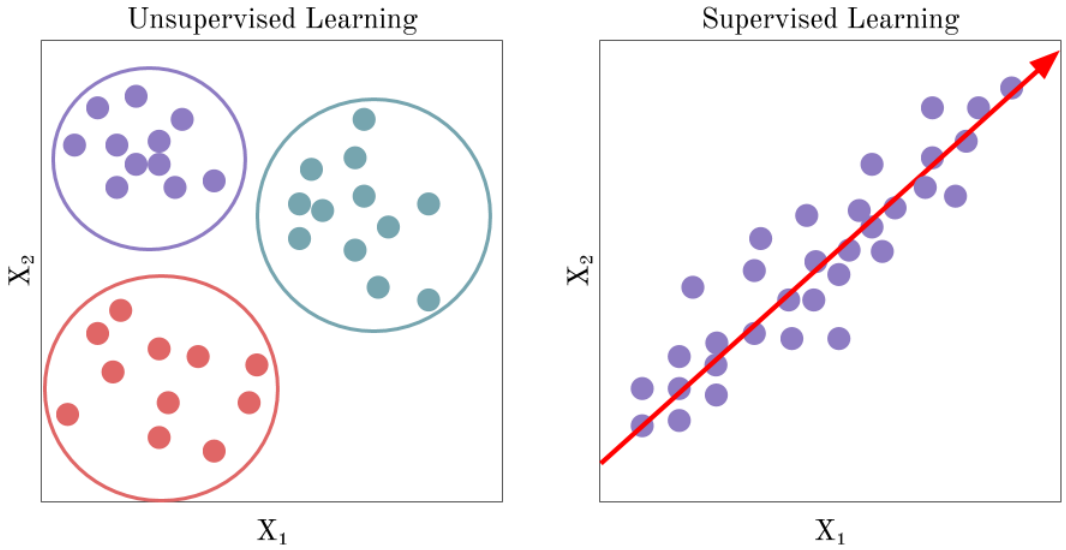
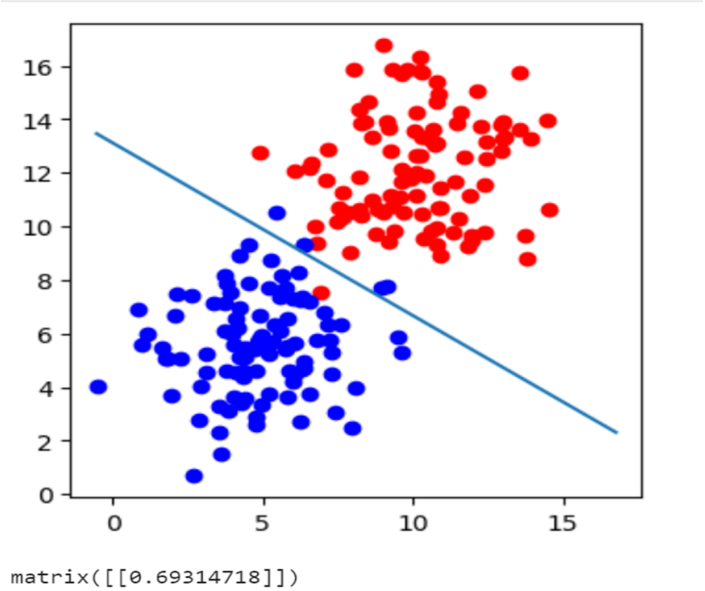


Fig 3.5: Unsupervised and Supervised Learning

Classification : It is a type of supervised learning problem where the goal is to classify data into predefined categories or classes.

Linear Model : It is a mathematical model that assumes a linear relationship between the input variables and the output variable.




```
'0]:  matrix([[0.69314718]])
```

Fig 3.6 : Classification of Perceptron

Cross Entropy : It is a measure of the difference between two probability distributions. In machine learning, it is commonly used as a loss function for classification problems.In binary classification, where there are only two possible outcomes (e.g., true or false) and in multi-class classification, where there are more than two possible outcomes.

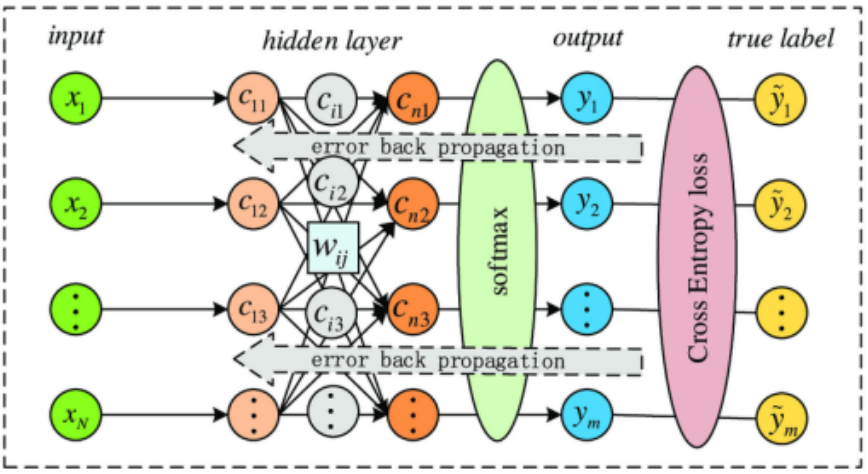


Fig3.7 : Cross Entropy

Gradient Descent : It is an optimization algorithm used to minimize a loss function by iteratively adjusting the model parameters in the direction of steepest descent.

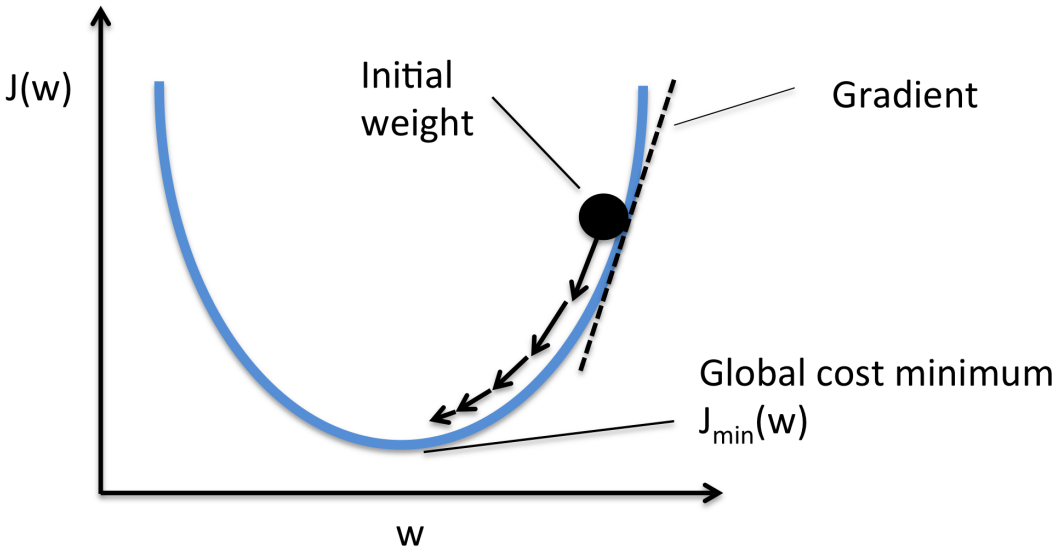


Fig 3.8: Gradient Descent

Keras – Predictions : Keras is a high-level neural networks API that is used for rapid prototyping and development of deep learning models. Predictions in Keras refer to the output generated by a trained model for new inputs.

Deep Neural Networks : They are neural networks that have multiple layers of neurons, enabling them to learn complex representations of data.In self-driving cars, deep learning networks are often used for perception tasks such as object detection, segmentation, and tracking. For example, convolutional neural networks (CNNs) can be used to detect and classify objects in the car's camera feeds, while recurrent neural networks (RNNs) can be used to track those objects over time.

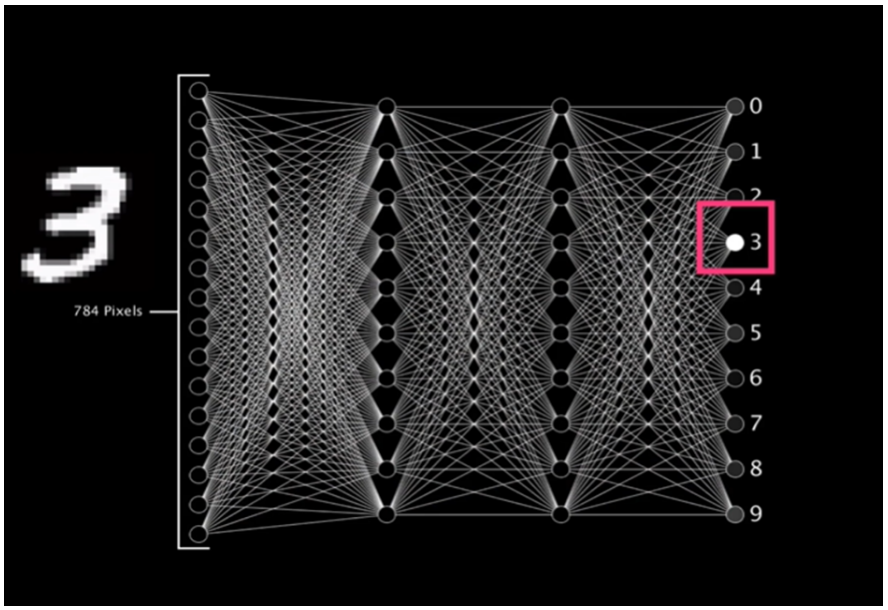


Fig 3.9 : Image recognition using CNN

Non-Linear Boundaries : They are decision boundaries that are not linear, meaning they cannot be described by a linear equation.

```
x = 0.1
y = 0.75
point = np.array([[x,y]])
prediction = model.predict(point)
plt.plot([x],[y],marker='o',markersize=10,color="red")
print("prediction is", prediction)
```

```
79/79 [==============================] - 0s 4ms/step
1/1 [==============================] - 0s 66ms/step
prediction is [[0.07375509]]
```

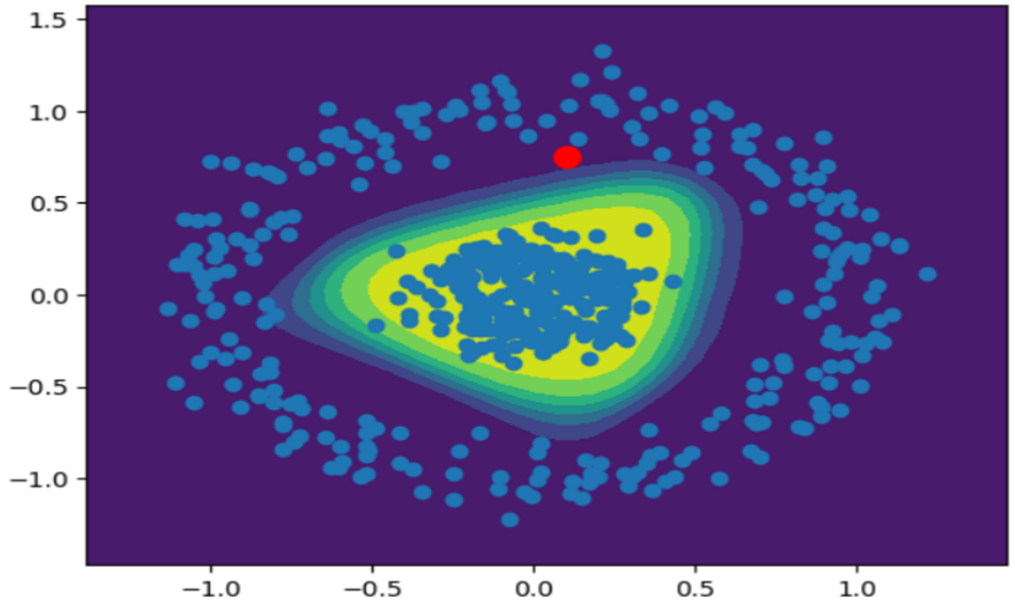


Fig 3.10 : Spot the given input value

Feedforward Process: It is the process by which the inputs are propagated through the neural network, layer by layer, to produce the output.

Backpropagation : It is an algorithm used to train neural networks by computing the gradient of the loss function with respect to the model parameters and updating the parameters in the opposite direction of the gradient.

Multiclass Classification : It is a type of classification problem where the goal is to classify data into more than two classes.

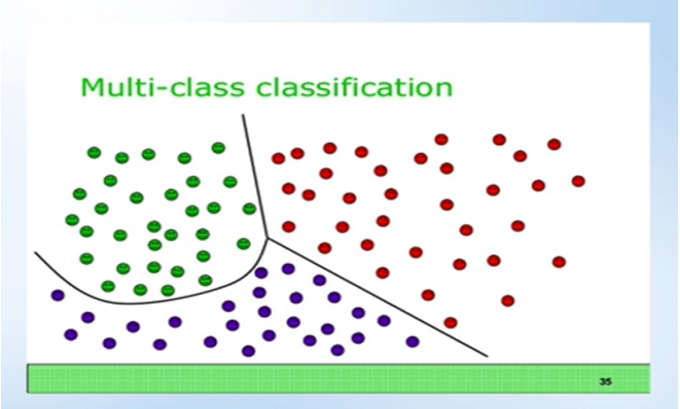


Fig 3.11: Multi-class classification

Softmax : It is a function that takes a vector of numbers as input and produces a probability distribution as output, where each element of the output vector represents the probability of the corresponding class.

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

Fig 3.12 : Softmax

Convolutional Neural Networks : They are neural networks that are specifically designed for image processing tasks. They use convolutional layers to automatically learn spatial features from the input images.

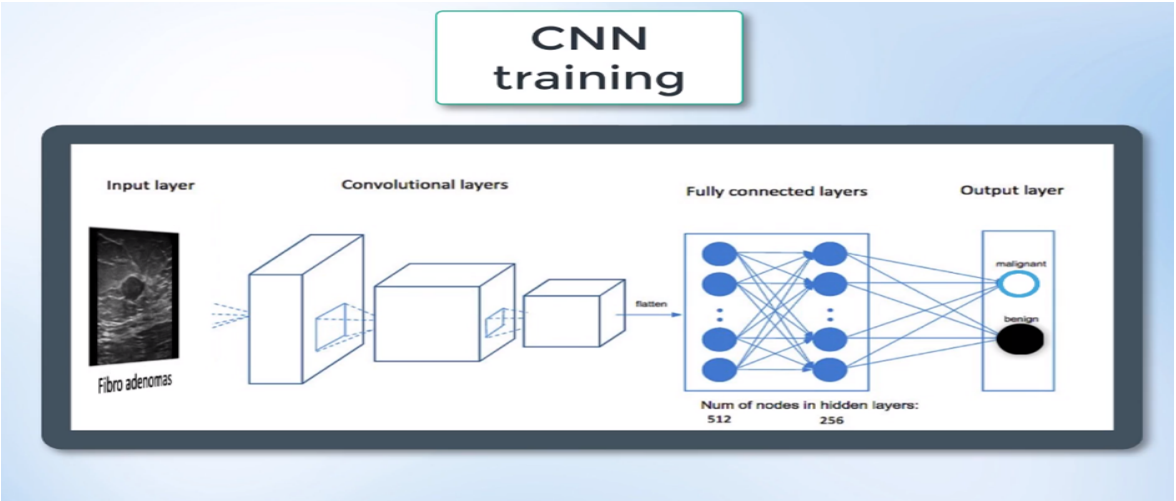


Fig 3.13 : Layers of Convolution Neural Network (CNN)

Convolutional Layer : It is a layer in a convolutional neural network that performs the convolution operation, i.e., it applies a set of filters to the input image to produce a set of feature maps.

Pooling : It is a downsampling operation used in convolutional neural networks to reduce the spatial dimensions of the feature maps and extract the most important features.

Fully Connected Layer : It is a layer in a neural network where all the neurons are connected to all the neurons in the previous layer, enabling the network to learn complex non-linear mappings between the input and output.

CHAPTER 4
LANE DETECTION

4.1 Algorithm:

- Convert the video frames to grayscale using cvtColor() function from OpenCV library.
- Apply Gaussian blur to smoothen the image using the GaussianBlur() function.
- Apply Canny edge detection on the image using the Canny() function to detect edges.
- Define a region of interest (ROI) in the image using a polygon. The ROI should include only the area where the lane lines are expected to be found.
- Use the bitwise_and() function to mask the image with the ROI, to obtain the edges detected only in the ROI.
- Apply Hough transform on the edges detected in the ROI using the HoughLinesP() function to detect lines.
- Separate the detected lines into left and right lane lines based on their slope.
- Compute the average slope and intercept of the left and right lane lines.
- Extrapolate the left and right lane lines to the bottom and top of the ROI by computing their x-intercepts using their slope and intercept.
- Draw the left and right lane lines on the original image using the line() function.

4.2 Block Diagram:

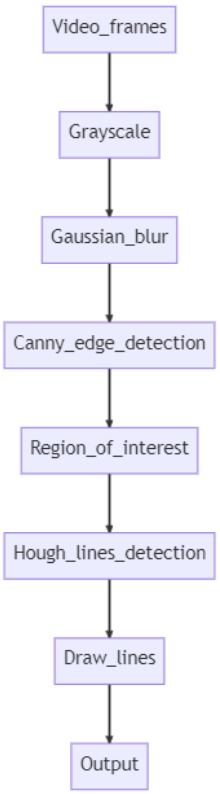


Fig 4.1 : Flowchart describing the procedure of Lane detection

4.3. Code and Output:

4.3.1. COMPUTER VISION / FINDING LANE LINE :

- Finding lane lines is a common task in computer vision and is an essential component of many autonomous driving systems. The goal of this task is to identify the lanes on a road from a video stream or an image and draw them on top of the image for visualization.

Fig4.2. : Lanes on a road from a video stream or an image are identified and drawn them on top of the image for visualization.

4.3.2. GRAYSCALE CONVERSION :

- Grayscale conversion is a process of converting an image from its original color representation to a gray or black and white representation. In grayscale images, each pixel has a single intensity value that represents the brightness of the pixel, ranging from 0 (black) to 255 (white).
- The most common method of grayscale conversion is to take a weighted average of the three color channels in the image. This is done because the human eye is more sensitive to green than to red or blue, so a weighted average of the three channels produces an image that looks more natural to the human eye. The formula for the weighted average is typically:

$$Y = 0.299*R + 0.587*G + 0.114*B$$

- Where R, G, and B are the red, green, and blue channels of the original image, and Y is the resulting grayscale pixel value.
- Python provides several libraries for working with images, including PIL (Python Imaging Library) and OpenCV. Both libraries provide functions for converting color images to grayscale, as well as for performing various image processing operations on the resulting grayscale image.

```
image = cv2.imread("test_image.jpg")

lane_image = np.copy(image)  # different from image

gray = cv2.cvtColor(lane_image, cv2.COLOR_RGB2GRAY)
```

4.3.3. SMOOTHENING IMAGE :

Smoothing or blurring an image is a common image processing operation used to reduce noise or remove high-frequency details from an image. This can improve the quality of the image and make it easier to process.

The most commonly used method for smoothing an image is to use a convolution kernel or filter, which is a small matrix of values that is applied to each pixel in the image. The kernel is typically a square or rectangular matrix of odd size, such as 3x3, 5x5, or 7x7.

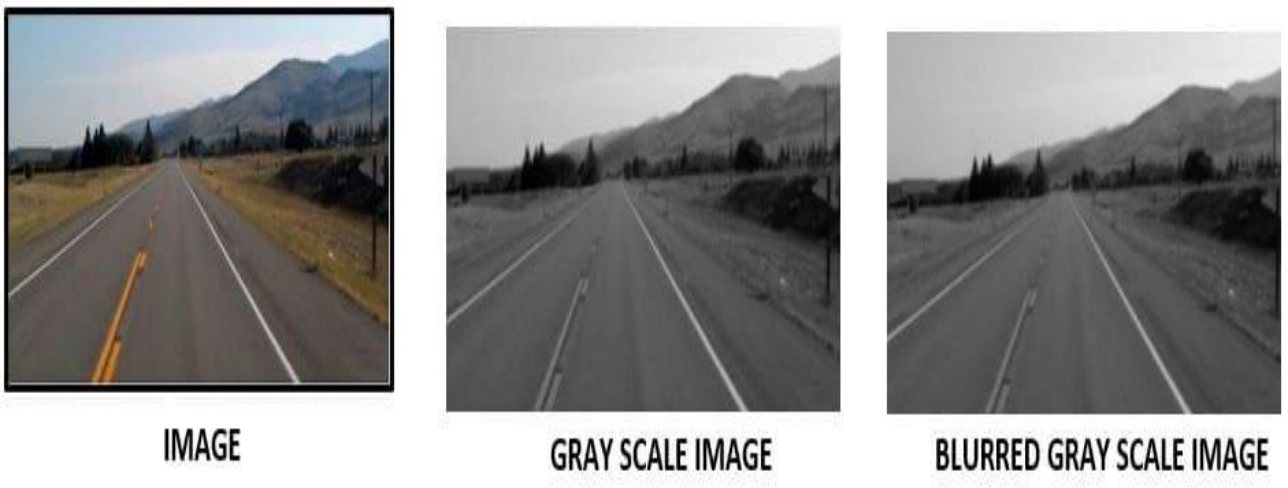


Fig 4.3.: Conversion of image to blurred gray scale image

4.3.4. SIMPLE EDGE DETECTION :

- Edge detection is a common image processing technique used to identify the boundaries of objects in an image. It is a fundamental operation in many computer vision applications and is used for tasks such as object recognition and tracking.

- The simplest edge detection method is the Sobel operator, which is a 3x3 convolution kernel that is used to compute the gradient of the image. The Sobel operator computes the horizontal and vertical derivatives of the image and combines them to produce an edge map.

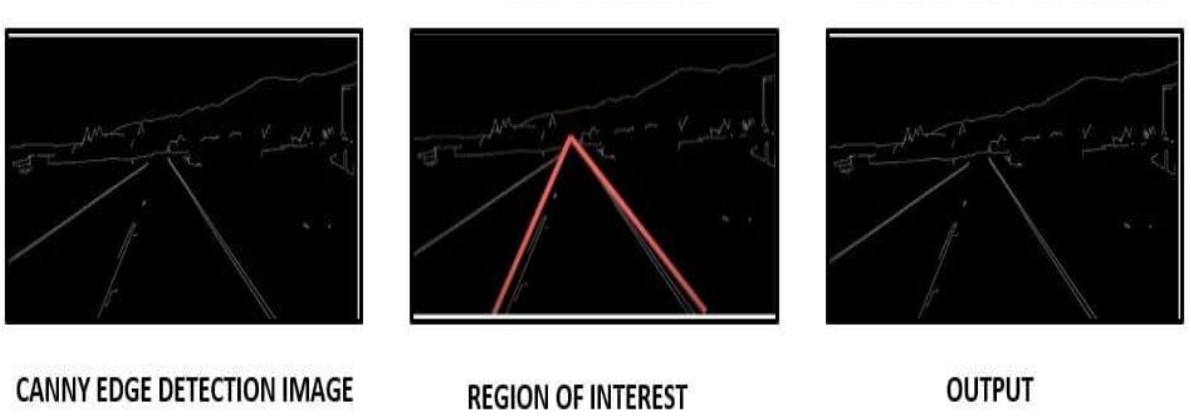


Fig4.4. : Edge detection

4.3.5. REGION OF INTEREST :

- In computer vision and image processing, a region of interest (ROI) is a specific area or subset of an image that contains the information or features that are of interest for a particular task.
- The process of defining an ROI involves selecting a subset of the image that contains the object or feature of interest and cropping the image to only include that subset.
- Once the ROI has been defined, it can be used for further analysis or processing. In tracking, the ROI can be used to track the movement of the object over time.

Fig4.5. : Region of interest

4.3.6. BINARY NUMBERS & BITWISE_AND :

- Binary numbers are a base-2 numbering system that uses only two digits, 0 and 1, to represent numbers. In contrast, the decimal system is a base-10 system that uses ten digits, 0 through 9.

- In computing, binary numbers are used to represent data and instructions. Each digit in a binary number is called a bit, which is short for binary digit. The value of a bit can be either 0 or 1, and it represents the presence or absence of an electrical signal in a circuit.

- Bitwise AND is a binary operation that takes two binary numbers of equal length and performs a logical AND operation on each pair of corresponding bits. The result is a binary number that has a 1 in each bit position where both input numbers have a 1, and a 0 in all other positions.

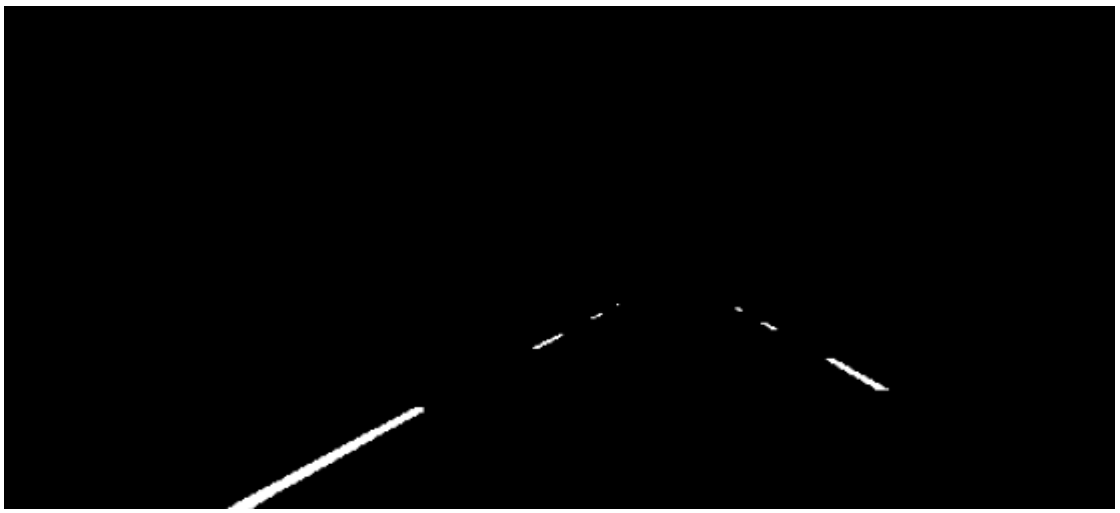


Fig4.6.: Bitwise AND operation is performed for region of interest

4.3.7. Line Detection - Hough Transform :

- Line detection is the process of identifying straight lines in an image. This is a fundamental problem in computer vision and has many applications such as object detection, lane detection in autonomous driving, and edge detection.

- The Hough transform is a technique used in computer vision to detect lines in an image. It was developed by Paul Hough in the early 1960s. The Hough transform converts the image space to a parameter space where lines are represented by their polar coordinates (θ, r). The parameter space is then discretized into a grid and each pixel in the image space votes for the lines that pass through it in the parameter space. The lines with the most votes are considered the detected lines in the image.

- The Hough transform is robust to noise and can detect lines even when they are broken or interrupted. It can also detect multiple lines in an image. However, it is computationally expensive and may not perform well in detecting curved or non-linear lines.

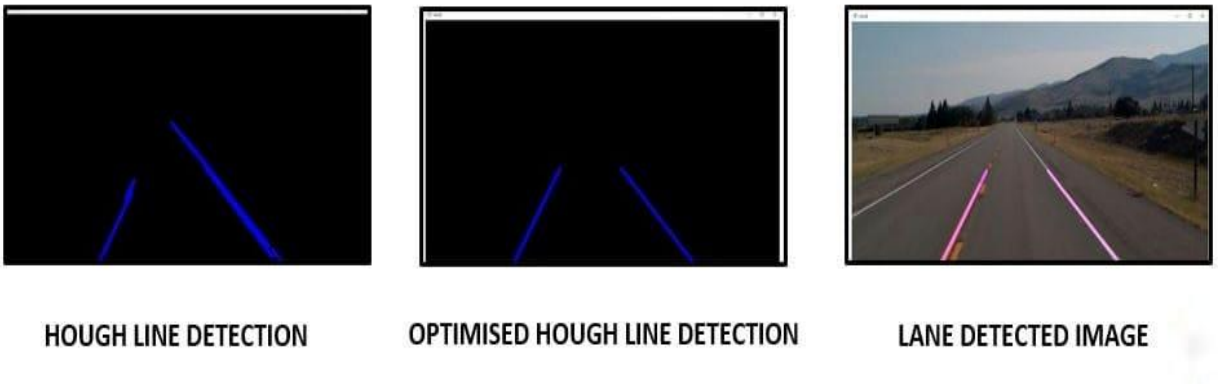


Fig4.7. : Lane Detection Output

4.3.8. OPTIMIZING :

The Hough Transform is a powerful technique for detecting simple shapes like lines and circles in an image. However, the standard implementation of the Hough Transform can be computationally expensive, especially for large images or when detecting more complex shapes.

- **Reduce the resolution**: Depending on the required accuracy, we can reduce the resolution of the Hough Transform by increasing the spacing

between bins. This can significantly reduce the number of bins required, and therefore reduce the computational complexity.

4.3.9. FINDING LANES ON VIDEO :

Finding lanes on video involves using computer vision techniques to detect and track the lane markings on a road. As video consists of images. We apply the above algorithm to each and every frame of the video.

Fig4.8. : Lanes on road from the input video are detected

4.4. Image Recognition

4.4.1. FLOWCHART :

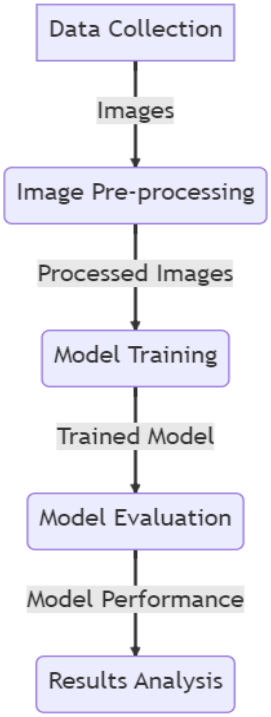


Fig 4.9.: Flow chart of Image recognition

This flowchart explains the procedure involved in Image recognition

4.4.2. Algorithm :

- Cloning a Git repository containing German traffic sign images.
- Importing the necessary libraries for building a convolutional neural network (CNN) model for image classification. The libraries imported include NumPy for numerical operations, Matplotlib for data visualization, Keras for building the CNN model, and OpenCV for image processing.
- Splitting the loaded data into input features and labels.The extracted input features are stored in X_train, X_val, and X_test variables, while the corresponding labels are stored in y_train, y_val, and y_test variables.
- Checking if the dimensions of the images are 32 x 32 x 3, which is the expected size of the input images.
- Visualizing the dataset by displaying a grid of images for each traffic sign class, along with its class ID and sign name.

  num_classes = 43

Fig 4.10. : Dataset containing grid of images for each traffic sign class, along with its class ID and sign name.

- Preprocess the images by converting them to grayscale, performing histogram equalization, and scaling the pixel values to be between 0 and 1. You reshape the preprocessed data to be of shape (num_samples, 32, 32, 1), where the last dimension represents that the image is grayscale.

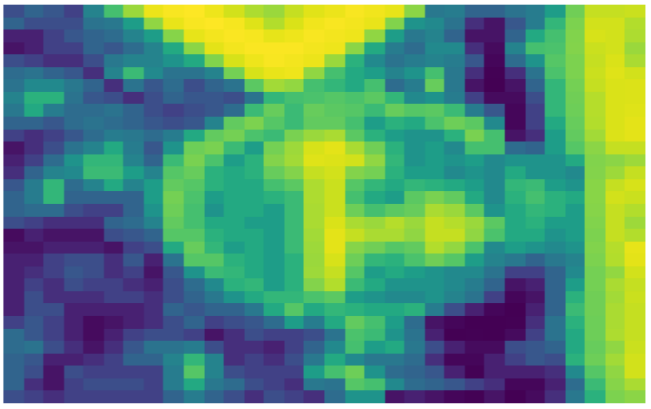


Fig4.11. : Preprocessing of image

- The datagen object creates a generator that performs data augmentation on the training data. The augmentation techniques used include random shifts in the width and height dimensions, zooming, shearing and rotating.Data augmentation is a technique where we artificially create new training data by applying transformations to the original images, such as rotating or flipping them. This can help to increase the amount of training data available to the model and improve its ability to generalize to new images.

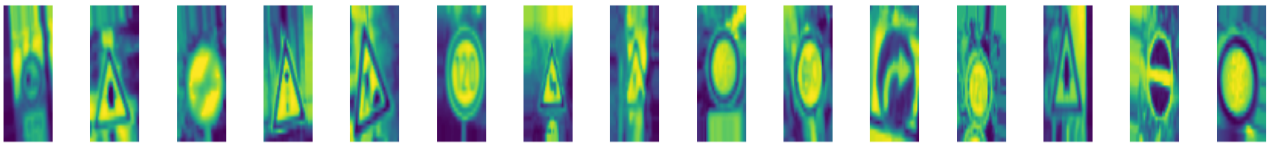


Fig 4.12. : Data Augmentation for selected image

- Implementation of the LeNet-5 architecture with some modifications to fit the problem of classifying traffic signs. The first convolutional layer has 30 filters of size 5x5 and uses ReLU activation. Then, a max pooling layer with size 2x2 is applied. The second convolutional layer has 15 filters of size 3x3 and also uses ReLU activation, followed by another max pooling layer with size 2x2. Finally, the output of the convolutional layers is flattened and passed to a fully connected layer with 500 units and ReLU activation, followed by a dropout layer with a rate of 0.5 to prevent overfitting. The output layer has 43 units, corresponding to the 43 classes of traffic signs, and uses softmax activation to output a probability distribution over the classes. The model is compiled with the Adam optimizer, a learning rate of 0.01, and the categorical cross entropy loss function, which is appropriate for multiclass classification problems. The accuracy metric is also used to monitor the performance of the model during training.

- Modified_model() : The modifications include increasing the number of filters in the initial convolutional layers, adding more convolutional layers, and adjusting the number of neurons in the fully connected layers. The function also compiles the model using the Adam optimizer with a learning rate of 0.001.

- The modified model has a total of 378,023 parameters. The first layer has 1,560 parameters, followed by the second layer with 90,060 parameters. The third layer, a MaxPooling2D layer, has no parameters. The fourth layer has 16,230 parameters, and the fifth layer has 8,130 parameters. The sixth layer, another MaxPooling2D layer, has no parameters. The Flatten layer has no parameters. The seventh layer, a Dense layer, has 240,500 parameters, and the eighth layer has 21,543 parameters.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 28, 28, 60)        1560

 conv2d_1 (Conv2D)           (None, 24, 24, 60)        90060

 max_pooling2d (MaxPooling2D  (None, 12, 12, 60)       0
 )

 conv2d_2 (Conv2D)           (None, 10, 10, 30)        16230

 conv2d_3 (Conv2D)           (None, 8, 8, 30)          8130

 max_pooling2d_1 (MaxPooling  (None, 4, 4, 30)         0
 2D)

 flatten (Flatten)           (None, 480)               0

 dense (Dense)               (None, 500)               240500

 dropout (Dropout)           (None, 500)               0

 dense_1 (Dense)             (None, 43)                21543

=================================================================
Total params: 378,023
Trainable params: 378,023
Non-trainable params: 0
```

- Fitting the modified_model. model will be trained for 10 epochs.

```
Epoch 1/10
695/695 [==============================] - 23s 21ms/step - loss: 1.6363 - accuracy: 0.5335 - val_loss: 0.2881 - val_accuracy: 0.9098
Epoch 2/10
695/695 [==============================] - 16s 22ms/step - loss: 0.5226 - accuracy: 0.8385 - val_loss: 0.1035 - val_accuracy: 0.9687
Epoch 3/10
695/695 [==============================] - 14s 21ms/step - loss: 0.3100 - accuracy: 0.9035 - val_loss: 0.0704 - val_accuracy: 0.9782
Epoch 4/10
695/695 [==============================] - 14s 21ms/step - loss: 0.2366 - accuracy: 0.9258 - val_loss: 0.0534 - val_accuracy: 0.9844
Epoch 5/10
695/695 [==============================] - 15s 21ms/step - loss: 0.1914 - accuracy: 0.9403 - val_loss: 0.0593 - val_accuracy: 0.9830
Epoch 6/10
695/695 [==============================] - 15s 21ms/step - loss: 0.1647 - accuracy: 0.9492 - val_loss: 0.0445 - val_accuracy: 0.9866
Epoch 7/10
695/695 [==============================] - 15s 22ms/step - loss: 0.1496 - accuracy: 0.9537 - val_loss: 0.0405 - val_accuracy: 0.9891
Epoch 8/10
695/695 [==============================] - 15s 22ms/step - loss: 0.1344 - accuracy: 0.9582 - val_loss: 0.0322 - val_accuracy: 0.9907
Epoch 9/10
695/695 [==============================] - 14s 21ms/step - loss: 0.1213 - accuracy: 0.9624 - val_loss: 0.0288 - val_accuracy: 0.9925
Epoch 10/10
695/695 [==============================] - 15s 21ms/step - loss: 0.1140 - accuracy: 0.9645 - val_loss: 0.0418 - val_accuracy: 0.9873
```

- Model seems to be performing very well with an accuracy of 98.73% on the validation set after 10 epochs. The training accuracy is also quite high, at 96.45%, indicating that the model is not overfitting.

4.5. RESULTS :

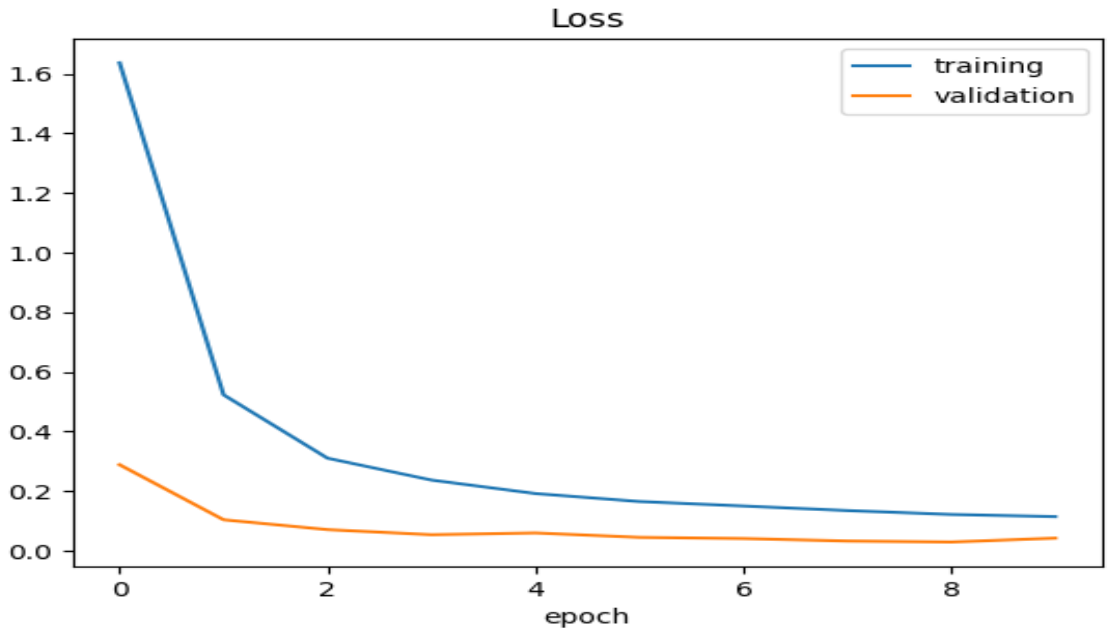


Fig 4.13. : Graph representing Loss vs Epoch

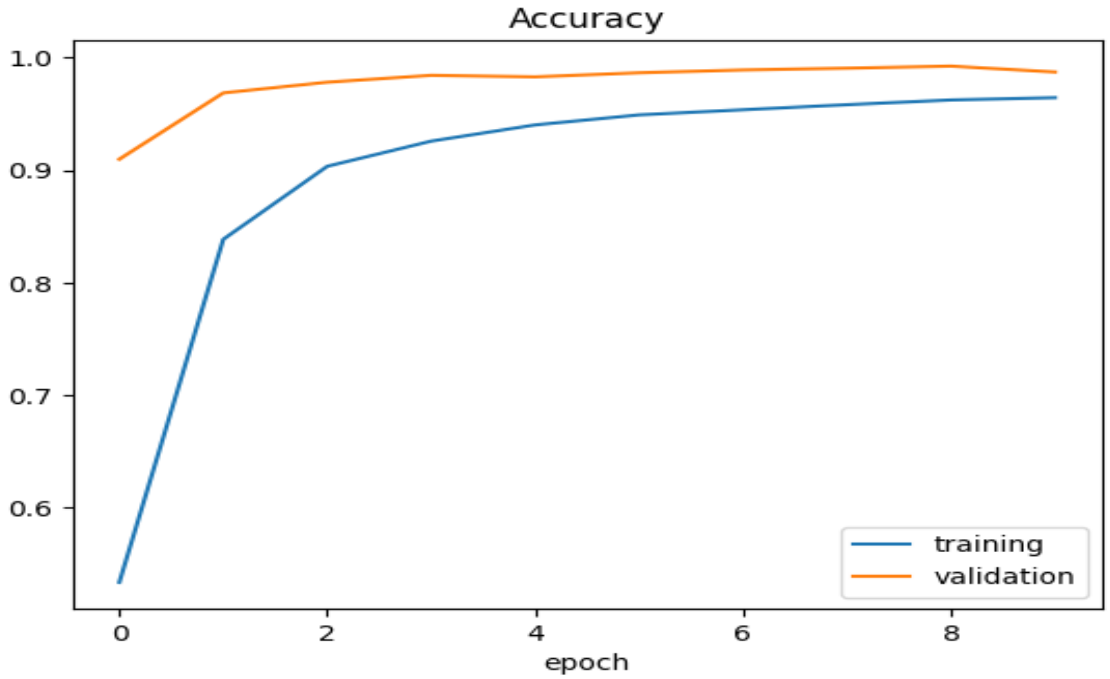


Fig 4.14.: Graph representing Accuracy vs Epoch

The training accuracy is the accuracy of the model on the training data, while the validation accuracy is the accuracy of the model on a separate validation dataset, which is not used in training.

During the training process, the model tries to learn the patterns and features from the training data, and this is reflected in the training accuracy. As the training progresses, the training accuracy tends to increase.

However, the model may not generalize well to new, unseen data, and this is where the validation accuracy comes in. The validation accuracy gives an estimate of how well the model can perform on new data, and it is used to prevent overfitting, which occurs when the model memorizes the training data too well and fails to generalize to new data.

Ideally, the training accuracy and validation accuracy should increase together as the model learns to extract relevant features from the data. However, if the model starts to overfit, the training accuracy may continue to increase while the validation accuracy plateaus or even decreases.

Therefore, the relationship between the training accuracy and validation accuracy can indicate if the model is overfitting or underfitting. If the training accuracy is much higher than the validation accuracy, it may be a sign of overfitting, and the model may need to be regularized or optimized to prevent overfitting.

We can observe that, Validation accuracy is higher than training accuracy, This indicates that the model is not overfitting, as it is able to generalize well to unseen data. This can be a good sign and suggest that the model is robust and not simply memorizing the training data.

4.6. OUTPUT:

input:

Fig-4.15. : Road symbol as input

output :

```
1/1 [==============================] - 0s 20ms/step
predicted sign: [22]
```

Fig-4.16. : Successfully predicted Road symbol

Fig-4.17.: Successfully predicted respective class of given road symbol

4.7. Behavioral Cloning

4.7.1. FLOWCHART :

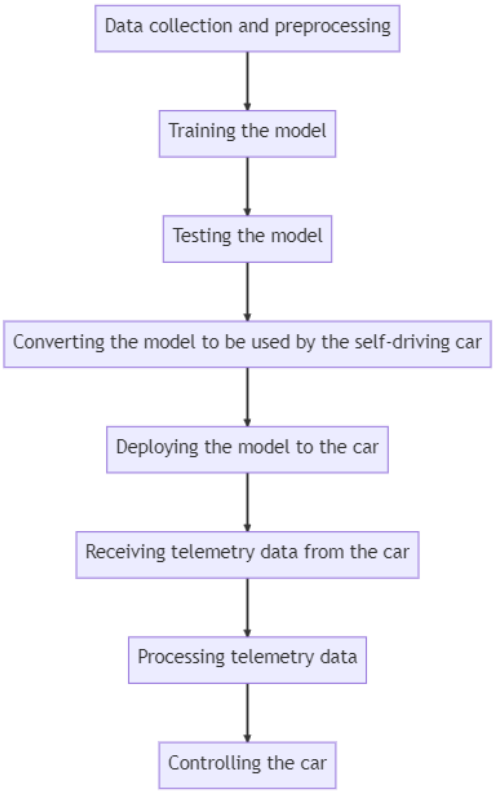


Fig 4.18.: Flowchart of Behavioral cloning

This flowchart describes the procedure involved in behavioral cloning of self driving cars.

4.7.2. ALGORITHM:

- Collect training data: Record images and steering angle data while driving the car manually.

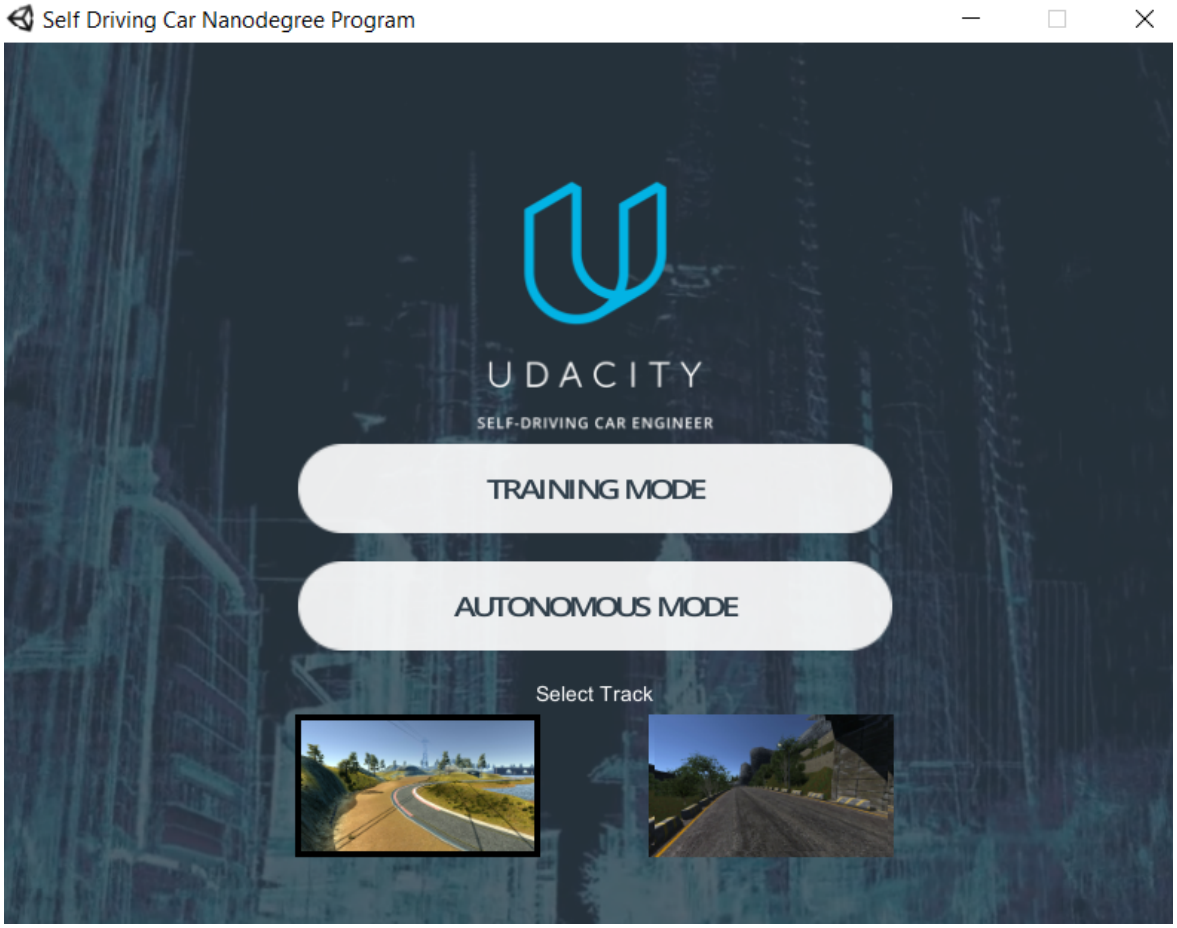


Fig 4.19.: Udacity Software Simulator

- Click on training mode.

Fig 4.20. : Travelling of car in training mode in Udacity self driving car simulator

- Click on record.

Fig4.21. : Saving the recorded data into a folder

- Select a folder and the simulator will start recording to that folder. Drive through the track at least 5 laps and collect data. Upload the data to github.
- Preprocess data: Crop images, convert to YUV color space, blur, and resize. Normalize the image pixel values to be between 0 and 1.
- Split data into training and validation sets.
- Define a deep learning model to predict the steering angle from images.
- Train the model using the training set and validate using the validation set.

```
Epoch 1/10
300/300 [==============================] - 164s 505ms/step - loss: 0.1800 - val_loss: 0.1231
Epoch 2/10
300/300 [==============================] - 158s 529ms/step - loss: 0.1183 - val_loss: 0.1109
Epoch 3/10
300/300 [==============================] - 153s 510ms/step - loss: 0.1092 - val_loss: 0.0958
Epoch 4/10
300/300 [==============================] - 152s 507ms/step - loss: 0.0998 - val_loss: 0.0880
Epoch 5/10
300/300 [==============================] - 152s 508ms/step - loss: 0.0959 - val_loss: 0.0852
Epoch 6/10
300/300 [==============================] - 151s 505ms/step - loss: 0.0899 - val_loss: 0.0882
Epoch 7/10
300/300 [==============================] - 151s 505ms/step - loss: 0.0892 - val_loss: 0.0862
Epoch 8/10
300/300 [==============================] - 149s 497ms/step - loss: 0.0871 - val_loss: 0.0724
Epoch 9/10
300/300 [==============================] - 149s 497ms/step - loss: 0.0846 - val_loss: 0.0755
Epoch 10/10
300/300 [==============================] - 150s 502ms/step - loss: 0.0835 - val_loss: 0.0704
```

Fig4.22. : Detecting the loss and validation loss

- Evaluate the model's performance on a test set.

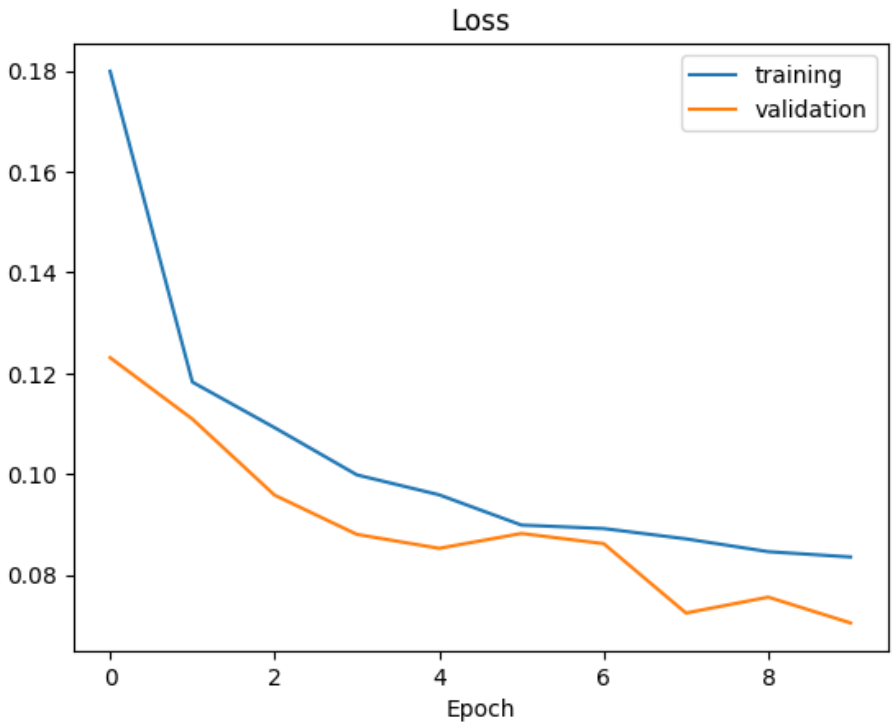


Fig 4.23.: Graph for loss obtained in Training and Validation is plotted

- Save the trained model to a file.
- Define a function to preprocess incoming images in the same way as the training data.
- Load the saved model from the file.
- Define a function to receive telemetry data (image and speed) from the simulator, preprocess the image, and use the model to predict the steering angle.

```python
@sio.on('telemetry')
def telemetry(sid, data):
    speed = float(data['speed'])
    image = Image.open(BytesIO(base64.b64decode(data['image'])))
    image = np.asarray(image)
    image = img_preprocess(image)
    image = np.array([image])
    steering_angle = float(model.predict(image))
    throttle = 1.0 - speed/speed_limit
    print('{} {} {}'.format(steering_angle, throttle, speed))
    send_control(steering_angle, throttle)
```

Fig4.24. : Defining a function to receive telemetry data (image and speed)

- Define a function to calculate the throttle value based on the current speed and a target speed.
- Define a function to send control commands (steering angle and throttle) to the simulator using socketio.

```python
def send_control(steering_angle, throttle):
    sio.emit('steer', data={
        'steering_angle': steering_angle.__str__(),
        'throttle': throttle.__str__()
    })
```

Fig 4.25.: Defining a function to send control commands

- Connect to the simulator using socketio and start receiving telemetry data.

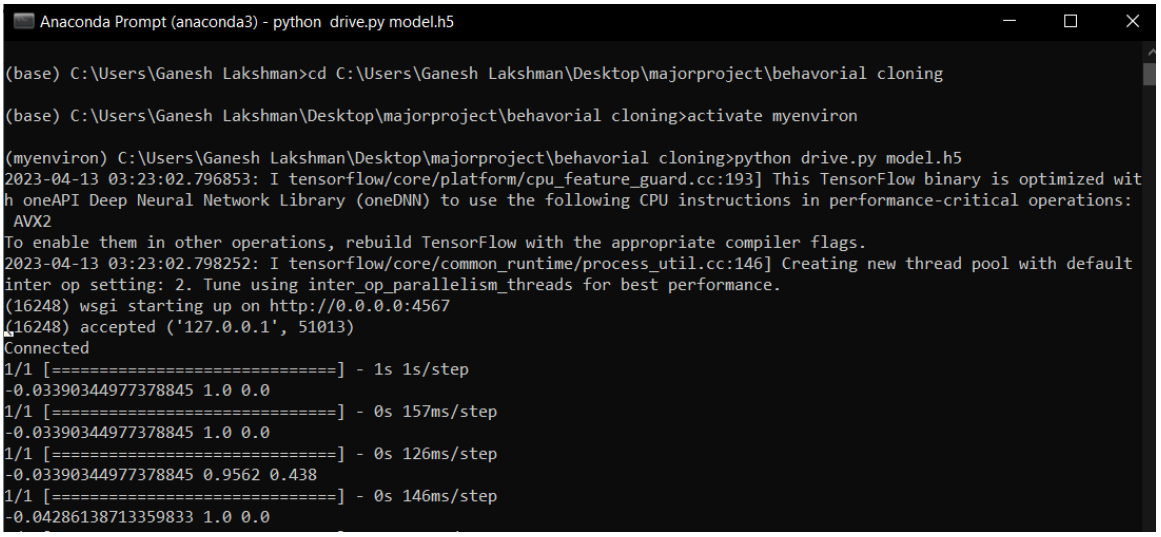


Fig 4.26. : Connecting to the simulator using socket.io

- Preprocess the incoming image, predict the steering angle using the loaded model, calculate the throttle, and send control commands to the simulator.

4.7.3. Output:

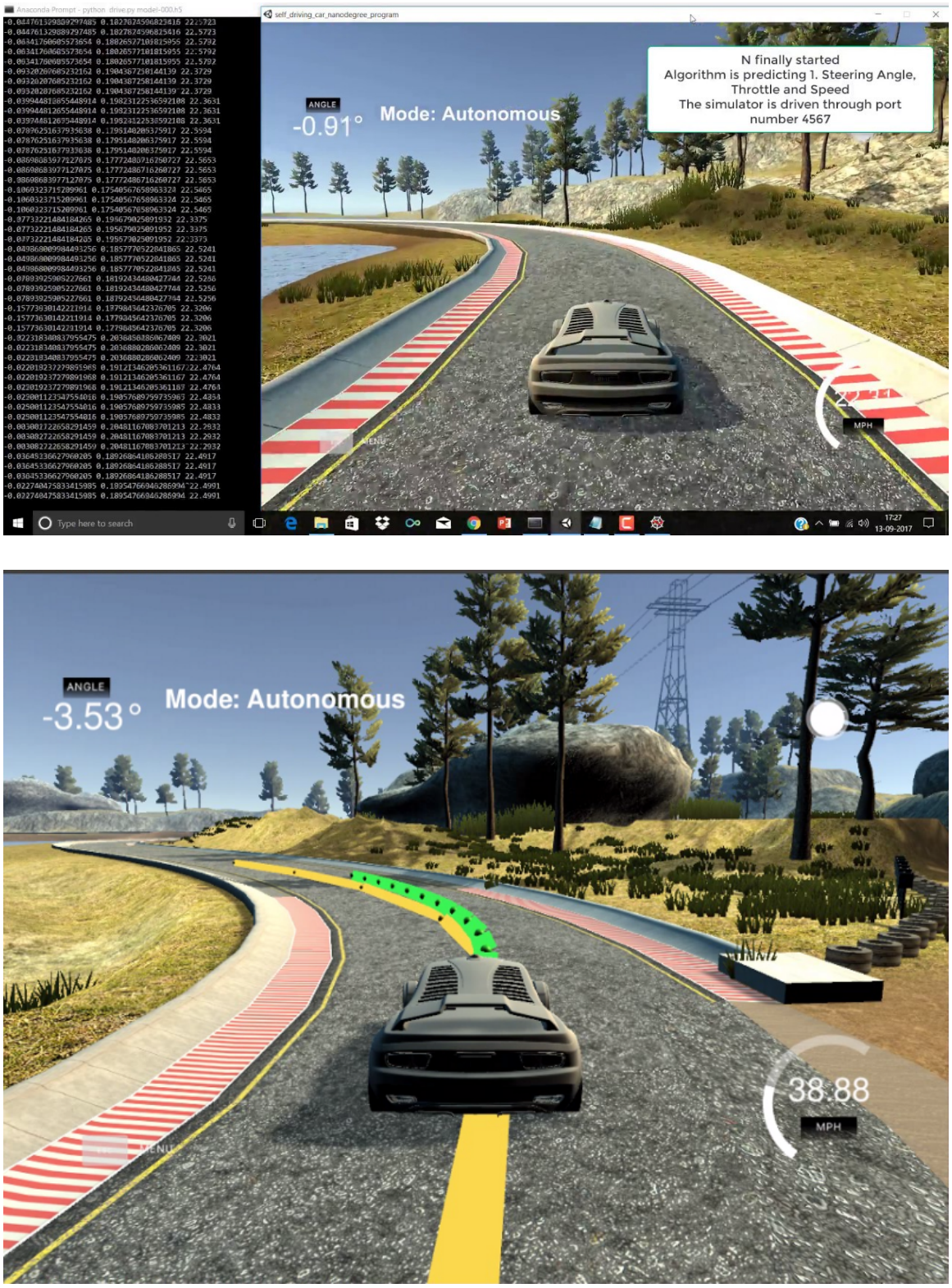


Fig4.27. : Self-driving car travelling successfully

4.8. OBJECT DETECTION AND COLLISION AVOIDANCE

4.8.1. FLOWCHART :

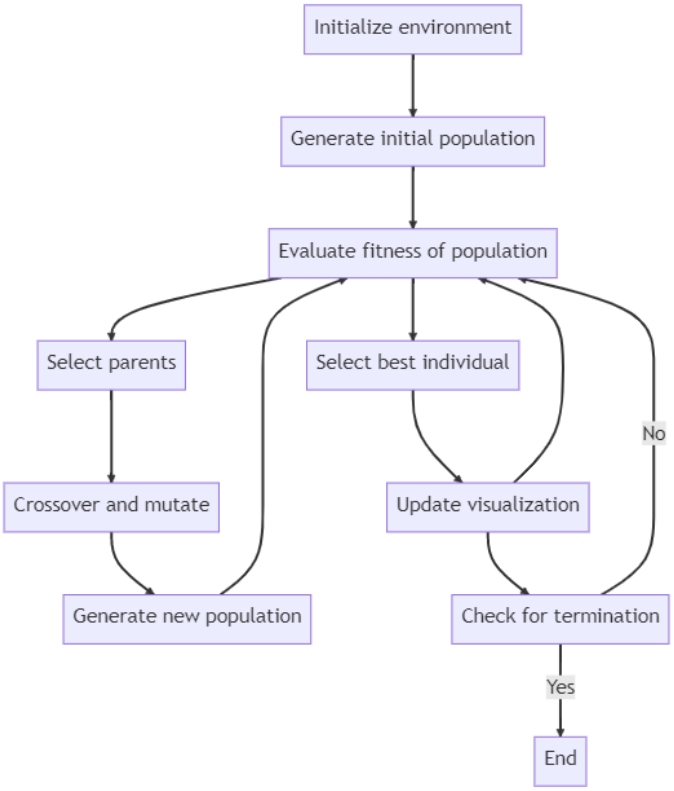


Fig 4.28.: Flowchart of object detection and collision avoidance

This flowchart describes the procedure involved in object detection and collision avoidance.

4.8.2. ALGORITHM :

Genetic Algorithm is a type of optimization algorithm that is inspired by the process of natural selection in genetics. It is a type of heuristic search algorithm that is used to find the optimal solution to a problem by mimicking the process of natural selection.

In genetic algorithm, a population of potential solutions to a problem is randomly initialized, and each individual solution is represented as a string of genes, which can be thought of as parameters or variables. The population then undergoes a process of selection, crossover, and mutation, similar to how genes are selected, crossed over, and mutated in natural selection.

In the selection process, individuals that have a higher fitness score (i.e., those that are closer to the optimal solution) are more likely to be chosen for reproduction. In the crossover process, two individuals are chosen to create a new individual by swapping

some of their genes. In the mutation process, a small random change is introduced to an individual's genes to create a new individual.

Through these processes, the population evolves over generations, with the hope that the individuals will converge towards the optimal solution to the problem. The fitness function, which evaluates how well an individual solution performs, is a key component of the algorithm, and is used to guide the selection process.

Genetic algorithms have been used in a variety of fields, including engineering, computer science, finance, and biology. It is particularly useful in problems where the solution space is large and complex, and where traditional optimization techniques may not be effective.

- Define canvas elements and their contexts to draw objects.
- Create a road object.
- Generate N number of car objects with an "AI" tag.
- Define a dummy car object for traffic.
- Load the best-performing car's neural network model from the local storage.
- Run the animate() function to continuously update the car and road objects' positions.
- For each car, update their position and check for collisions with the road borders and other cars.
- Update the best car based on its y position.
- Draw the road, cars, and traffic objects in the canvas using the draw() method of each object.
- Draw the best car with a higher opacity to distinguish it from other cars.
- Visualize the best car's neural network on the network canvas.
- Call requestAnimationFrame() to run the animate() function repeatedly.
- Provide save() and discard() functions to store and remove the best-performing car's neural network from local storage, respectively.

4.8.3. Output :

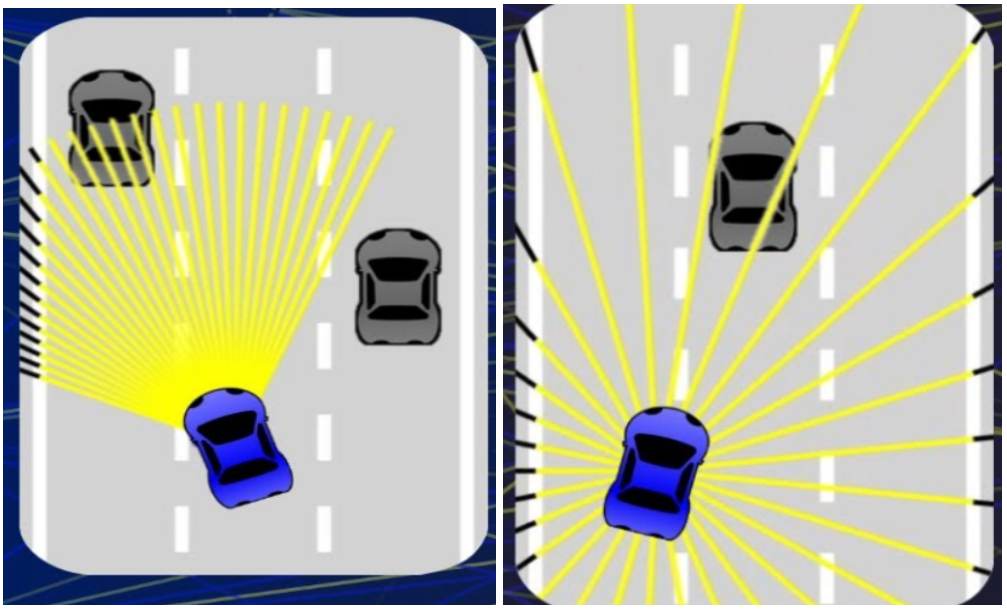


Fig 4.29.: Object detection and collision avoidance

FUTURE SCOPE

As of now, most autonomous vehicles are at Level 2 or Level 3 based on the The Society of Automotive Engineers (SAE) which means that they still require human intervention in certain situations. However, there is ongoing research and development to achieve Level 4 or Level 5 autonomy, which would mean that the vehicle is capable of operating without any human input.

Achieving higher levels of autonomy would require advancements in technology such as

- Integration with advanced sensors: Integration of advanced sensors like LiDAR, RADAR, and ultrasonic sensors to provide more accurate and reliable data for the autonomous driving system.
- Robustness to complex environments: Improving the robustness of the self-driving car system to complex environments such as snow, rain, and fog, where the sensors and cameras might not function as expected.
- Implementation of more advanced algorithms and techniques for object detection and collision avoidance, such as reinforcement learning and object tracking.

Additionally, there is also a potential for the use of autonomous vehicles in other industries, such as transportation and logistics. This could include self-driving trucks for shipping and delivery, or autonomous drones for aerial transportation. The possibilities for autonomous vehicles are vast, and there is still much to explore and develop in this field.

CONCLUSION

In conclusion, this project aimed to design and implement a self-driving car system using computer vision and deep learning techniques. The project successfully achieved its objectives by developing and fine-tuning convolutional neural networks for image recognition and lane detection, training a deep neural network for behavior cloning, and implementing real-time control using Flask and Socket.io for communication between the car and server. The project achieved high accuracy rates in image recognition and lane detection tasks, and the knowledge gained in computer vision, deep learning, and autonomous systems can be applied to further develop the technology for future self-driving vehicles. Overall, the project was a success and demonstrated the potential for self-driving cars to revolutionize the transportation industry.

REFERENCES

- P. M. Castro, L. C. de Oliveira and R. C. Dórea, "Development of a Self-Driving Car using Deep Learning," 2020 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC), Porto, Portugal, 2020, pp. 107-112, doi: 10.1109/ICARSC49135.2020.9090331.

- R. Arora, N. Taneja and A. Garg, "Self-Driving Car: A Journey from Perception to Control," 2020 International Conference on Communication, Computing and Networking (ICCCN), Big Island, HI, USA, 2020, pp. 1-6, doi: 10.1109/ICCCN49319.2020.9209777.

- L. H. M. C. Costa, M. P. Ramos, P. R. P. Coelho and A. S. Silva, "Development of a Self-Driving Car Prototype using ROS and Machine Learning," 2020 International Joint Conference on Neural Networks (IJCNN), Glasgow, United Kingdom, 2020, pp. 1-8, doi: 10.1109/IJCNN48605.2020.9207552.

- S. Yang, Y. Han and S. Lee, "A Study on Self-Driving Car Navigation System Using Real-Time Image Recognition," in IEEE Access, vol. 9, pp. 35213-35221, 2021, doi: 10.1109/ACCESS.2021.3069786.

- R. R. Raj, A. Z. A. Basar and N. N. Ariffin, "A Review on Control Approaches for Autonomous Vehicles," in IEEE Access, vol. 8, pp. 26057-26075, 2020, doi: 10.1109/ACCESS.2020.2973116.