EEE4016 – Electric Vehicles

Winter-Semester 2021-22

Title: ***Self Driving Car Simulation***

Team Members:

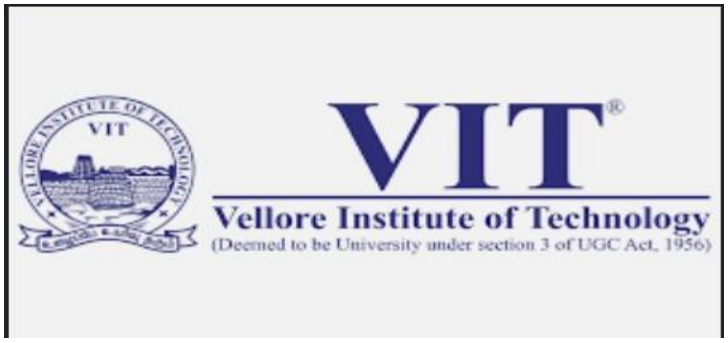
Shivendra Rai (19BEE0057)

Rohit Kumar (19BEE0035)

Divyansh Singh Raghav (19BEE0067)

Submitted to:

CHITRA A



Abstract

Autonomous cars are the future smart cars anticipated to be driver less, efficient and crash avoiding ideal urban cars of the future. This project provides an opportunity to develop a selfdriving car with the proper steering angle to maintain the car in the lane. To teach machines what to do through interactions with the environment, the algorithms based on reinforcement learning is used. The application of reinforcement learning has a high relevance and importance for driving as it is dependent on interactions with the environment. The model will learn the steering angle from the turns in the image and will finally predict the steering angle for unknown images. It is an example of behavioural cloning. The goal of behavioural cloning is to collect data while exhibiting good behavior and then train a model to mimic that behavior with the collected data. Based on the proposed approach an autonomous driving car simulation is realized in this project. This simulator is employed to test the performance of autonomous vehicles and evaluate its self-driving tasks.

Objectives

Over the period of time the traffic on roads has increased which has put a strain on our transport system. Moreover, with the advent of AI in every sector we felt that solving the problems of road traffic by automation of cars would sharply increase the efficiency of the Transport Sector. For humans it is becoming tough and stressful to drive on roads of cities. We felt that we could come up with a solution which could not only increase efficiency but also decrease the human efforts of driving. Self-driving capability will bring positive change to our whole society, such as providing transportation for people who aren't able to drive because of physical impairment or age. The main objective of our project is to reduce human work and increase the efficiency of vehicles.

Problem Statement

This project provides an opportunity to develop a self-driving car with the proper steering angle to maintain the car in the lane. To teach machines what to do through interactions with the environment, the algorithms based on reinforcement learning is used. The application of reinforcement learning has a high relevance and importance for driving as it is dependent on interactions with the environment. The model will learn the steering angle from the turns in

the image and will finally predict the steering angle for unknown images. It is an example of behavioural cloning. The goal of behavioural cloning is to collect data while exhibiting good behaviour and then train a model to mimic that behaviour with the collected data. Based on the proposed approach an autonomous driving car simulation is realized in this project.

Introduction

In the past years, we have seen millions of people have lost their lives in traffic accidents. A wide majority of these accidents are caused because of human mistakes. With each passing year the number of traffic accidents are increasing drastically, it has been estimated that the number of these cases would increase by two times as of now. Street transport is one of the most perilous methods for transport accessible today, yet regularly a large number of individuals use it to travel. The high pace of mishaps is ascribed to the moderate reaction time of drivers and weariness.

Traffic blockages are caused because of the moderate reaction of human drivers. Self-ruling driving would thus be able to lessen the number mishaps worldwide by a huge sum. The car business is an extraordinary industry. So as to keep the travellers' security, any mishap is unsatisfactory. In this manner, the dependability and security must fulfil the stringent standard. The exactness and strength of the sensors and calculations are required amazingly accuracy during the time spent self-driving vehicles.

As of late, the fast advancement of artificial intelligence innovation, particularly deep learning, has made a significant leap forward in fields such as picture acknowledgment and canny control. Deep learning procedures, ordinarily, for example, convolutional neural networks, are generally utilized in different sorts of picture preparing, which makes them reasonable for self-driving applications while supervised learning has a significant issue where it requires a gigantic measure of marked information and is as yet helpless to human inclination.

Hardware & Software Requirements

Operating System – Windows or MacOS

RAM – min of 4GB

Hard Disk Space – 500GB

Software required – Udacity Game Simulator, Anaconda notebook, Atom text editor or VS code text editor

Working Principle

1. Use OpenCV to detect lane line
    i. Convert image to grayscale for edge detection
    ii. Noise Reduction or Smoothening by gaussian filter using a kernel to prevent false edges.
    iii. Canny method to identify edges in our image. iv. Specifying lane lines in image by finding the region of interest.
    v. Hough Transform to detect straight lines in our region of interest thus identifying the lane lines.
    vi. Optimizing: instead of having multiple lines we can average out their slope and y intercept into a single line.

2. Use OpenCV to detect lane line
    i. Optimizing: instead of having multiple lines we can average out their slope and y intercept into a single line. ii. Download traffic dataset.
    iii. Identify the Traffic Signal using CNN.

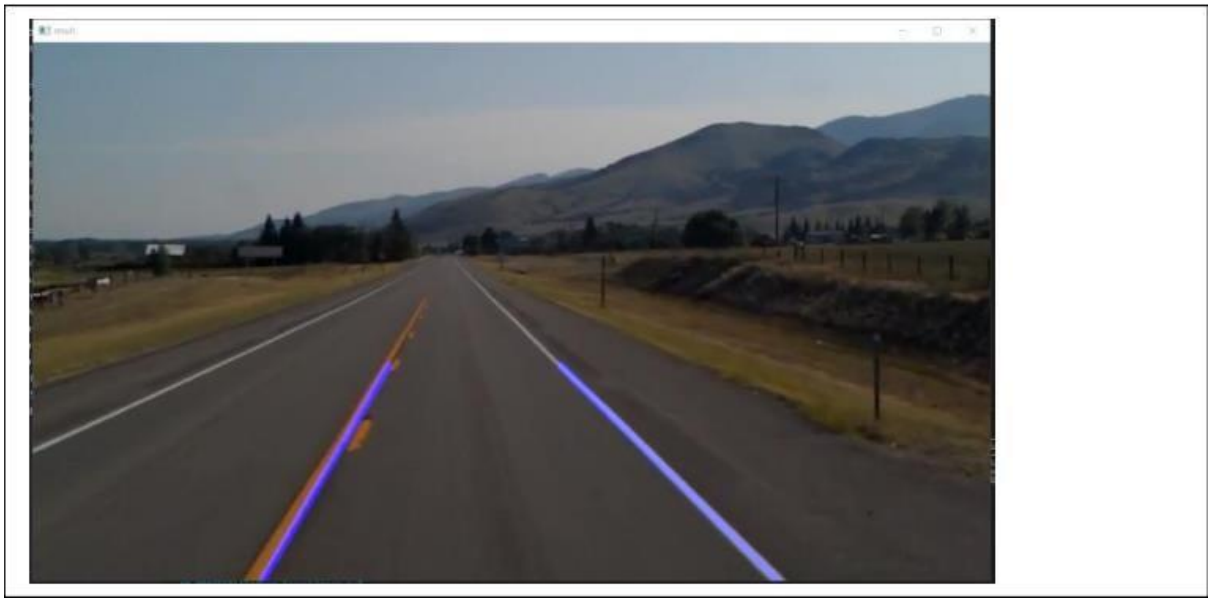
3. Behavioural Cloning
    i. Download Udacity self-driving simulator.
    ii. Open the simulator and train the car by manually driving the car using keyboard keys. The simulator captures each frame with steering angle.
    iii. Using the concept of regression, we predict the steering angle.

4. Testing the data on the Simulator models on a training dataset; if the data is trained accurately, then the car is driving autonomously.

Design Equations

- Firstly, we created a lane identification model using OpenCV.
- To differentiate between binary classes, we trained a perceptron based Convolutional Neural Network.
- By seeing the complex nature of our dataset, we trained a Deep Neural Network.
- Build an autonomous fully functional car to drive on its own on unknown tracks.

Identify Lane lines demo picture:



Module Description

Lane Detection (Basic):

- Used OPenCV to analyze the images and identify lanes. Later on implemented the same lane identification function on a stream of video. The other two techniques that were used are Hough Transformation and Canny edge detection. These two techniques help finding the colour contrast on image thereby highlighting the lanes • Technology used was OpenCV and Computer Vision

Behavioural Cloning:

- Used the technique of behavioural cloning with Tensorflow and Keras. We trained a network (Convolution Neural Network). Trained the model on Udacity Car Simulator and deployed it. Moreover, in the CNN we used techniques such as regularization and dropout for optimum results and making it suitable to run on different tracks
- Technologies used were Deep Learning, Keras, and Convolutional Neural Networks.

Advanced Lane Finding:

- To make our model more suitable for different unique roads we used different techniques such as distortion, rectification colour transformation and thresholding of gradient. These techniques help to track the vehicle displacement and keep it within the road. Moreover, these techniques also help to overcome challenges such as shadows and pavement identification.
- Technologies used were OpenCV and Computer Vision

Other Vehicle Detection:

- Used techniques such as Histogram of Oriented gradient and support vector machine to detect different vehicles on road. We implemented the pipeline using a deep neural network and performed the necessary detection. As the data was collected from the simulator. The video collected from the same was used for training and optimizing the model.

- Technologies used were Computer Vision, Deep Learning and OpenCV.

Simulation Diagram

Gathering Data

To gather the data for our automated car we used Udacity car simulator. In the same we manually drove the car through a track which provides us with images from three angles left, right and centres. Then these images are fed into the model for training. The one thing that we have to note here is each image also has a steering angle of that particular instance giving us the idea of the turn which car will make. The images are captured for every instance making the whole dataset for our model.

Training Model

We then fed this data to our model. The main thing that we need to train is steering angle at every instance. The car started learning the prediction of steering angle based on our annual driving in Track 1. But Track 1 had some biases resulting in uneven data. We first removed biasness from our data as shown below in order to make it suitable for every track. Later on, the model effectively adjusted the steering angle based on different situations.

Testing Model

We then ran our car on a different track. Our car was able to identify the suitable steering angle and was moving ahead without any hindrance. This track was used to test the car. It performed really well.

This technique of behavioural cloning is very demanding these days because this forms the basis of self-driving cars in real life as well. Cars are typically driven on road manually by a driver then they are trained to follow the same instructions on test tracks. With increasing demand and technology self-driving cars have a very bright future.

Udacity Simulator

Training mode was used to drive the car ourselves and gather training data we used to train the model to train the neural network to simulate our behavior. And then autonomous mode was used to test our neural network to drive the car on its own after we've collected our

training data and checked the performance and the accuracy at which our car drives autonomously on the track.

Training Mode

The different parts of the track had different textures, edges, and borders like the border on the bridge in comparison to the border of sand have different curvatures layouts and landscapes throughout the track. All of these unique features were extracted with the convolutional neural network and was associated with a corresponding steering angle through a regression-based algorithm.

Once we finished the three laps, we did another couple of laps in the reverse direction to gather more data to generalize the model. The reason for this was to guarantee that our data is balanced as the track was left turn biased meaning when driving forwards, we were mostly steering left and thus driving the car in one direction around the track skewed our data to the side. This would have created a problem for the neural network as it could bias the model towards always predicting left turns which would have made the car become biased towards driving left all the time crashing into the edges.

To solve this problem, we drove the car in the opposite direction where we were taking mostly right turns. Thus, the left and right steering angles in our data became more balanced. The training data is the set of images in each training dataset and each set of images has a label a steering angle. We used the training images in order to train the neural network to predict the appropriate steering angles for the car to drive on its own. Whenever there was a straight path mostly the steering angle was zero since it went straight whereas whenever there was a curvature, the steering angle was negative so as to denote a left turn and positive which denoted a right turn.

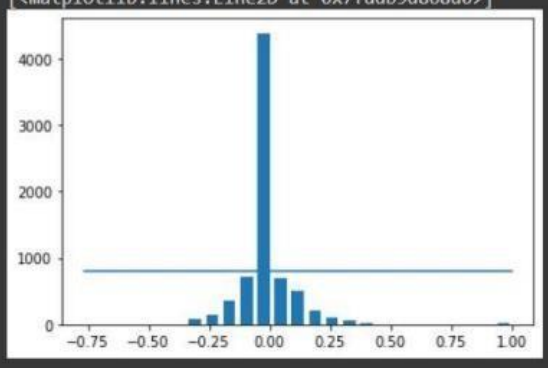
In our dataset there were a large number of the data where the steering angles were zero. If we would have trained the model based on this data it could have biased our model towards predicting zero angle and would have created a problem for the neural network. Thus, the car would have become biased towards driving straight all the time. To improve this situation, we rejected all the samples above a certain threshold limit and made sure that it was not biased towards a specific steering angle.

Code

Remove the biasness

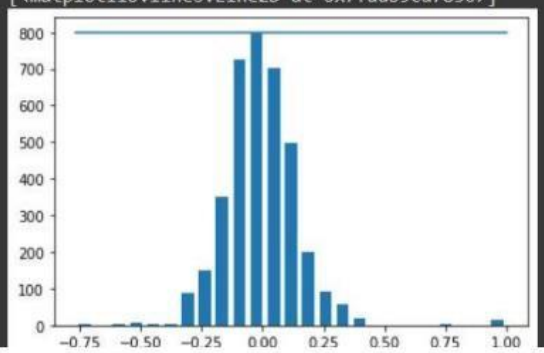
```
[ ] num_bins = 25
    samples_per_bin=800
    hist,bins=np.histogram(data['steering'], num_bins)
    center = (bins[:-1]+ bins[1:])*0.5
    plt.bar(center, hist, width=0.05)
    plt.plot((np.min(data['steering']), np.max(data['steering'])),(samples_per_bin, samples_per_bin))
```

```
[<matplotlib.lines.Line2D at 0x7fddb9d868d0>]
```



```
[ ] print('total_data:', len(data))
    remove_list= []
    for j in range(num_bins):
      list_= []
      for i in range(len(data['steering'])):
        if data['steering'][i] >= bins[j] and data['steering'][i]<=bins[j+1]:
          list_.append(i)
      list_ = shuffle(list_)
      list_= list_[samples_per_bin:]
      remove_list.extend(list_)
    print('removed:', len(remove_list))
    data.drop(data.index[remove_list], inplace=True)
    print('remaining:', len(data))
    hist,_ =np.histogram(data['steering'], num_bins)
    plt.bar(center, hist, width=0.05)
    plt.plot((np.min(data['steering']), np.max(data['steering'])),(samples_per_bin, samples_per_bin))
```

```
total_data: 7308
removed: 3587
remaining: 3721
[<matplotlib.lines.Line2D at 0x7fddb9cd7850>]
```



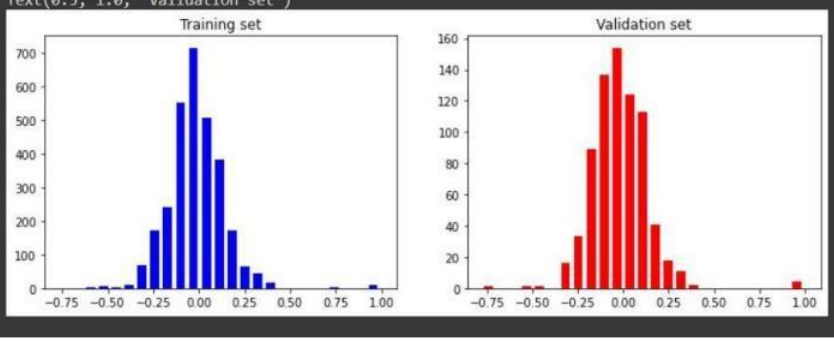
Train_test_split

```
[ ] X_train, X_valid, y_train, y_valid= train_test_split(image_paths, steerings, test_size=0.2, random_state=6)
    print('Training Samples:{}\nValid Samples:{}'.format(len(X_train), len(X_valid)))
```

```
Training Samples:2976
Valid Samples:745
```

```
[ ] fig, axes= plt.subplots(1, 2, figsize=(12,4))
    axes[0].hist(y_train, bins=num_bins, width=0.05, color='blue')
    axes[0].set_title('Training set')
    axes[1].hist(y_valid, bins=num_bins, width=0.05, color='red' )
    axes[1].set_title('Validation set')
```

```
Text(0.5, 1.0, 'Validation set')
```

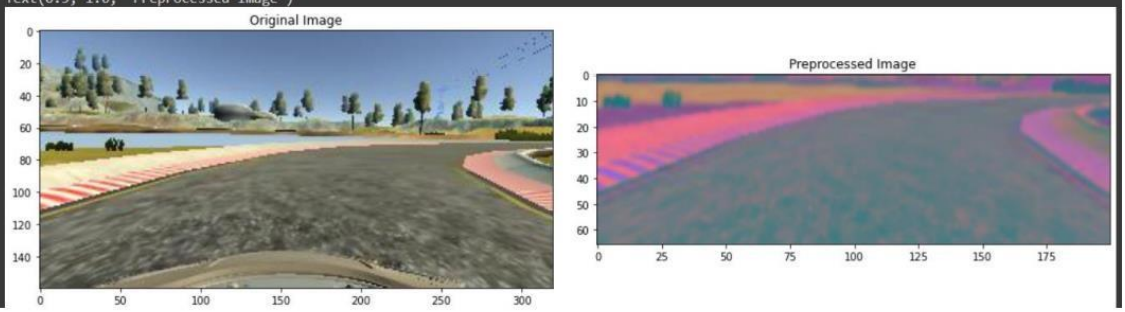


Processing the images

```
[ ]  def img_preprocess(img):
       #img= mpimg.imread(img) #to change
       img= img[60:135, :, :]
       img= cv2.cvtColor(img, cv2.COLOR_RGB2YUV)
       img=cv2.GaussianBlur(img, (3,3), 0)
       img=cv2.resize(img, (200,66))
       img=img/255
       return img
```

```
[ ]  image = image_paths[150]
     original_image= mpimg.imread(image)
     preprocessed_image= img_preprocess(original_image)

     fig, axs =plt.subplots(1,2, figsize=(15,10))
     fig.tight_layout()
     axs[0].imshow(original_image)
     axs[0].set_title('Original Image')
     axs[1].imshow(preprocessed_image)
     axs[1].set_title('Preprocessed Image')
```

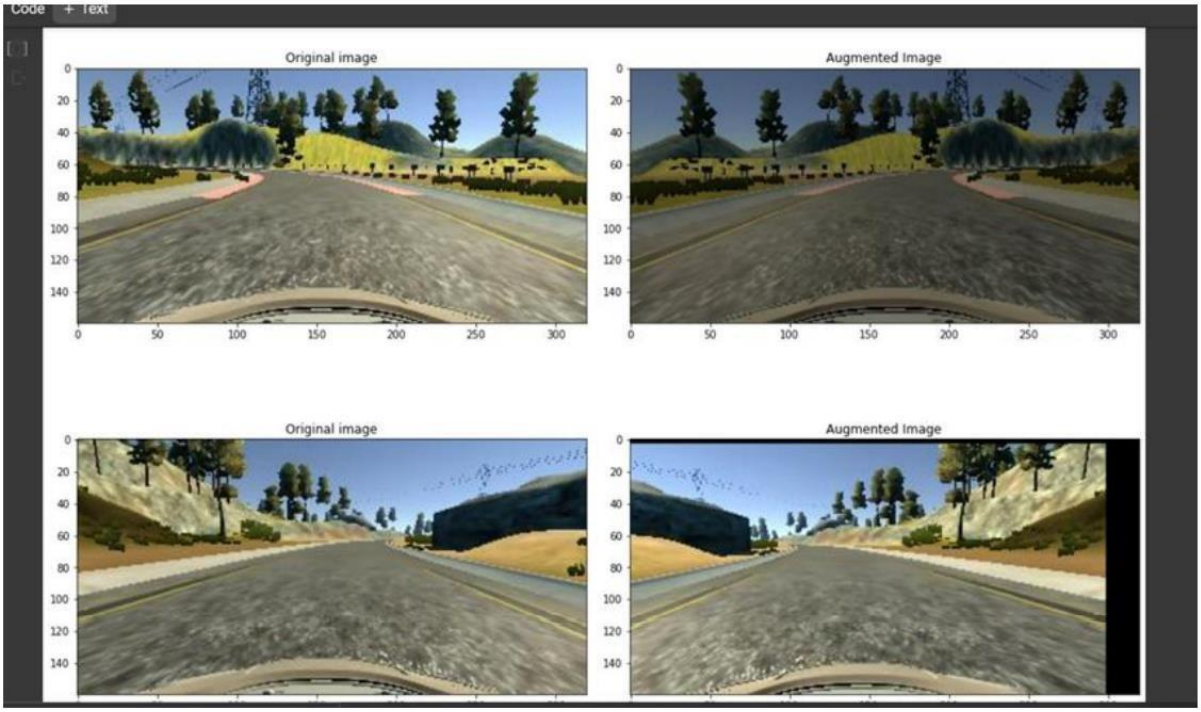
```
Text(0.5, 1.0, 'Preprocessed Image')
```



Transforming Test Train data's images

```
def random_augment(image, steering_angle):
  image=mpimg.imread(image)
  #50% of images will be augmented
  if(np.random.rand() < 0.5):
    image= pan(image)
  if(np.random.rand() < 0.5):
    image= zoom(image)
  if(np.random.rand() < 0.5):
    image= img_random_brightness(image)
  if(np.random.rand() < 0.5):
    image,steering_angle= img_random_flip(image,steering_angle)

  return(image, steering_angle)
```

```
[ ] ncol=2
    nrow=10
    fig, axs = plt.subplots(nrow, ncol, figsize=(15,50))
    fig.tight_layout()
    for i in range(10):
      randnum= random.randint(0, len(image_paths)-1)
      random_image= image_paths[randnum]
      random_steering=  steerings[randnum]

      original_image= mpimg.imread(random_image)
      augmented_image, steering= random_augment(random_image, random_steering)

      axs[i][0].imshow(original_image)
      axs[i][0].set_title('Original image')
      axs[i][1].imshow(augmented_image)
      axs[i][1].set_title('Augmented Image')
```



Machine Learning Model

```
def nvidia_model():
  model=Sequential()
  model.add(Convolution2D(24, (5, 5), strides=(2,2), input_shape=(66,200,3), activation='elu'))
  model.add(Convolution2D(36, (5, 5), strides=(2,2), activation='elu'))
  model.add(Convolution2D(48, (5, 5), strides=(2,2), activation='elu'))
  model.add(Convolution2D(64, (3, 3), activation='elu'))

  model.add(Convolution2D(64, (3, 3), activation='elu'))
  #model.add(Dropout(0.5))

  model.add(Flatten())

  model.add(Dense(100, activation='elu'))
  #model.add(Dropout(0.5))

  model.add(Dense(50, activation='elu'))
  #model.add(Dropout(0.5))

  model.add(Dense(10, activation='elu'))
 #model.add(Dropout(0.5))
  model.add(Dense(1))

  optimizer= Adam(lr=0.0001)
  model.compile(loss='mse', optimizer=optimizer)
  return model
```

```
[ ]
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_5 (Conv2D)           (None, 31, 98, 24)        1824

 conv2d_6 (Conv2D)           (None, 14, 47, 36)        21636

 conv2d_7 (Conv2D)           (None, 5, 22, 48)         43248

 conv2d_8 (Conv2D)           (None, 3, 20, 64)         27712

 conv2d_9 (Conv2D)           (None, 1, 18, 64)         36928

 flatten_1 (Flatten)         (None, 1152)              0

 dense_4 (Dense)             (None, 100)               115300

 dense_5 (Dense)             (None, 50)                5050

 dense_6 (Dense)             (None, 10)                510

 dense_7 (Dense)             (None, 1)                 11

=================================================================
Total params: 252,219
Trainable params: 252,219
Non-trainable params: 0
_________________________________________________________________
```

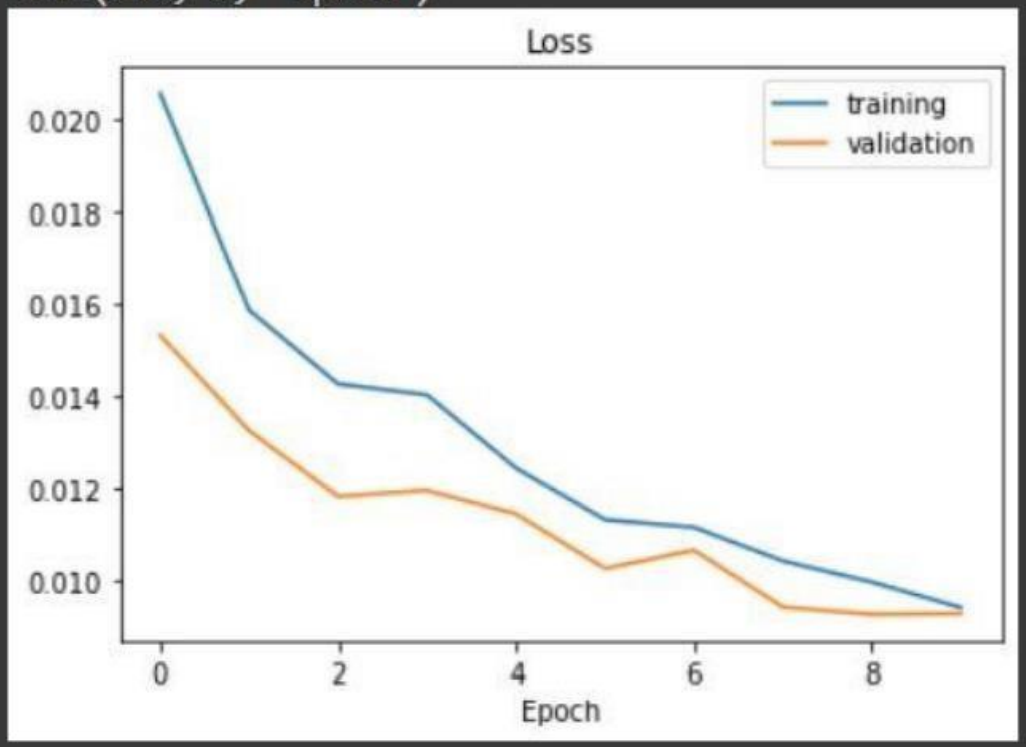
Fitting the Model and Plotting the Results

```
[ ] history = model.fit_generator(batch_generator(X_train, y_train, 100, 1),
                                  steps_per_epoch=300,
                                  epochs=10,
                                  validation_data=batch_generator(X_valid, y_valid, 100, 0),
                                  validation_steps=200,
                                  verbose=1,
                                  shuffle=1)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:7: UserWarning: `Model.fit_generator`
  import sys
Epoch 1/10
300/300 [==============================] - 251s 836ms/step - loss: 0.0206 - val_loss: 0.0153
Epoch 2/10
300/300 [==============================] - 252s 843ms/step - loss: 0.0158 - val_loss: 0.0132
Epoch 3/10
300/300 [==============================] - 252s 844ms/step - loss: 0.0142 - val_loss: 0.0118
Epoch 4/10
300/300 [==============================] - 254s 848ms/step - loss: 0.0140 - val_loss: 0.0119
Epoch 5/10
300/300 [==============================] - 253s 848ms/step - loss: 0.0124 - val_loss: 0.0114
Epoch 6/10
300/300 [==============================] - 253s 848ms/step - loss: 0.0113 - val_loss: 0.0102
Epoch 7/10
300/300 [==============================] - 255s 853ms/step - loss: 0.0111 - val_loss: 0.0106
Epoch 8/10
300/300 [==============================] - 254s 849ms/step - loss: 0.0104 - val_loss: 0.0094
Epoch 9/10
300/300 [==============================] - 254s 849ms/step - loss: 0.0099 - val_loss: 0.0092
Epoch 10/10
300/300 [==============================] - 249s 834ms/step - loss: 0.0094 - val_loss: 0.0093
```

```
[ ]  plt.plot(history.history['loss'])
     plt.plot(history.history['val_loss'])
     plt.legend(['training', 'validation'])
     plt.title('Loss')
     plt.xlabel('Epoch')
```

```
Text(0.5, 0, 'Epoch')
```



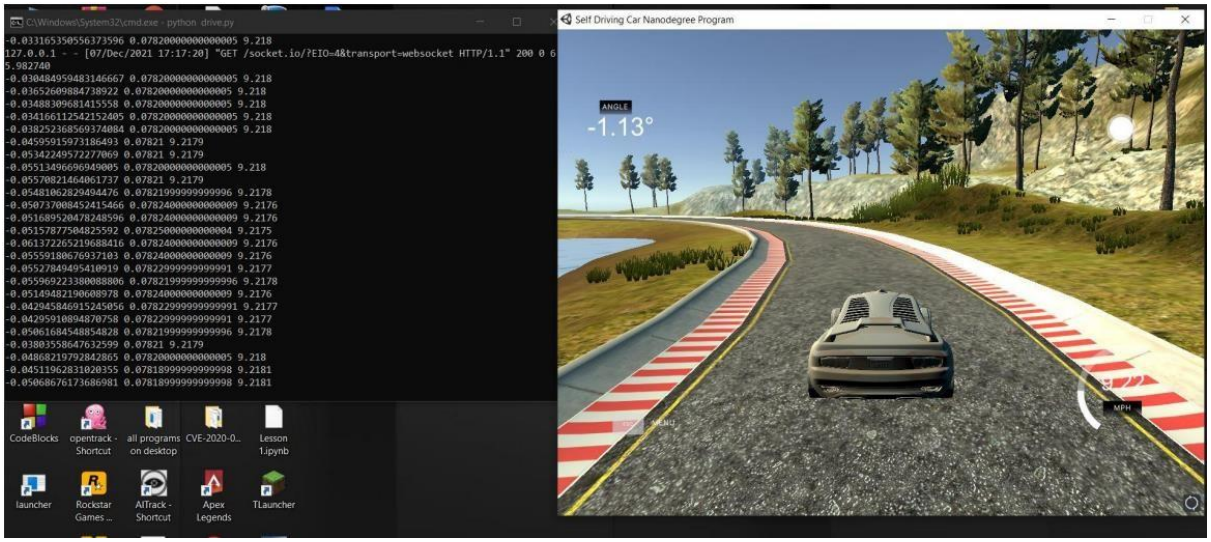
Connecting to Simulator

```
import socketio import eventlet
from flask import Flask import sys
from keras.models import
load_model import base64 from io
import BytesIO from PIL import
Image import numpy as np import
cv2


sio= socketio.Server()
 app = Flask(__name__)
#'__main__'
```

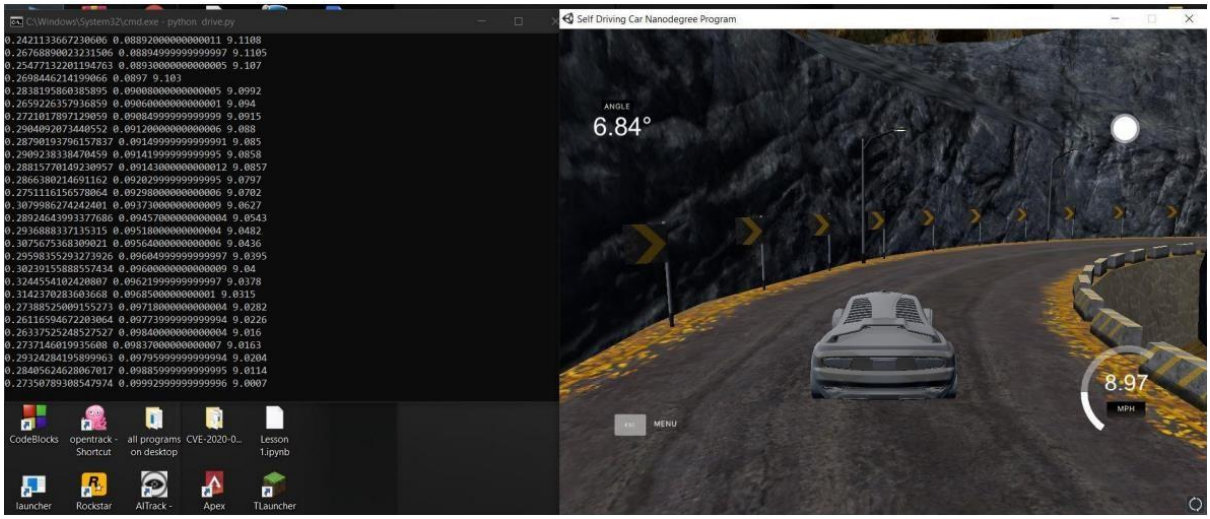
```
speed_limit=10 def
img_preprocess(img):
    img= img[60:135, :, :]   img=
cv2.cvtColor(img, cv2.COLOR_RGB2YUV)
img=cv2.GaussianBlur(img, (3,3), 0)
img=cv2.resize(img, (200,66))
img=img/255   return img

@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)

@sio.on('connect')  def
connect(sid, environ):
    print('Connected')
send_control(0, 0)


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

if __name__=='__main__':
    model = load_model('newmodel2.h5')
print(sys.prefix)
    app=socketio.Middleware(sio,app)
eventlet.wsgi.server(eventlet.listen(('',4567)), app)
```

Simulation Results

Automated Driving – Track 1



Automated Driving – Track 2



Results

After training the data with different datasets and different training parameters we were able to achieve a training loss of "X" and validation loss of "Y". Model doesn't overfit and has decent accuracy.

Here, X is the loss at the last epoch, and Y is the validation loss at the last epoch

Conclusion

We were able to identify lanes successfully on a picture and video. We extracted thew data from the Udacity Car Simulator and implemented the data into our machine learning model. We trained our car on Track 1 and later on tested it on Track 2. This shows that our car is able to move on an unknown track with limited speed.

References

[1] https://developer.nvidia.com/blog/deep-learning-self-driving-cars

[2] ojarski, M., Del Testa, D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., ... & Zhang, X. (2016). End to end learning for self-driving cars. arXiv preprint arXiv:1604.07316.

[3] Bhadani, R. K., Sprinkle, J., & Bunting, M. (2018). The cat vehicle testbed: A simulator with hardware in the loop for autonomous vehicle applications. arXiv preprint arXiv:1804.04347.

[4] Shao, W., & Terzopoulos, D. (2007). Autonomous pedestrians. Graphical models, 69(56), 246-274.

[5] Naghavi, S. H., Avaznia, C., & Talebi, H. (2017, November). Integrated real-time object detection for self-driving vehicles. In 2017 10th Iranian Conference on Machine Vision and Image Processing (MVIP) (pp. 154-158). IEEE.

[6] Xu, N., Tan, B., & Kong, B. (2018). Autonomous driving in reality with reinforcement learning and image translation. arXiv preprint arXiv:1801.05299.

[7] Bagloee, S. A., Tavana, M., Asadi, M., & Oliver, T. (2016). Autonomous vehicles: challenges, opportunities, and future implications for transportation policies. Journal of modern transportation, 24(4), 284-303.

[8] Babaeizadeh, M., Frosio, I., Tyree, S., Clemons, J., & Kautz, J. (2016). Reinforcement learning through asynchronous advantage actor-critic on a gpu. arXiv preprint arXiv:1611.06256.

[9] Lieberman, H. (1997, March). Autonomous interface agents. In Proceedings of the ACM SIGCHI Conference on Human factors in computing systems (pp. 67-74).

[10] Okuyama, T., Gonsalves, T., & Upadhay, J. (2018, March). Autonomous driving system based on deep q learnig. In 2018 International Conference on Intelligent Autonomous Systems (ICoIAS) (pp. 201-205). IEEE. [8] Kang, Y., Yin, H., & Berger, C. (2019). Test your self-driving algorithm: An overview of publicly available driving datasets and virtual testing environments. IEEE Transactions on Intelligent Vehicles, 4(2), 171-185.