International Journal of Research Publication and Reviews

Journal homepage: www.ijrpr.com ISSN 2582-7421

Self Driving Cars Using Deep Learning

Rahul

N/A, India

ABSTRACT

The research 's primary objective is the development of a simulated model for driverless cars, with the eventual goal of integrating this model as software into real-world autonomous vehicles. This endeavor encompasses several key components, including the enhancement of image quality without compromising its integrity, improving the accuracy of turn detection, and enabling the autonomous vehicle to navigate on a generalized track with consistent training and validation accuracy. Additionally, the research focuses on fundamental capabilities such as mapping, tracking, and local planning, all of which are essential for autonomous driving. To address safety and collision avoidance, obstacle and curb detection methods are being implemented, as well as road vehicle tracking techniques to handle diverse traffic scenarios. The final goal is to create a robust autonomous vehicle capable of autonomously performing tasks such as lane changes, parking, and U-turns, effectively utilizing innovations in obstacle detection, curb detection, and vehicle tracking. These objectives collectively aim to propel autonomous driving technology towards practical real-world applications. However, it is crucial to consider the legal, ethical, and safety aspects in parallel with technological advancements when working on autonomous vehicles.

Introduction

In this research , the NVIDIA model, developed by NVIDIA, plays a pivotal role. This model is available in Python's model library and is a deep neural network. The model is structured as a sequential neural network, comprising 5 Convolution2D layers, 4 dropout layers, and 4 dense layers. Additionally, a flatten layer is incorporated to convert the image matrix into a one-dimensional array. These architectural choices are guided by specific characteristics.

For the dataset, Udacity provides a car simulator where a track is available for manual driving. The simulator includes a record button, which, when activated, allows the selection of a folder to store images. As the car is manually driven on the track, the simulator records images at each moment and associates them with the corresponding car steering angle. The simulator features three cameras capturing images from the left, right, and center perspectives. To create a comprehensive dataset covering various driving scenarios and angles, it is recommended to complete at least three laps in both the forward and reverse directions on the track.

Data preprocessing is a crucial step. To avoid introducing bias into the self-driving simulator, certain high-frequency dataset values are removed. Specifically, since driving on the center of the track is common, the dataset often contains a high number of 0-degree steering angle values. To prevent the model from becoming biased toward predicting a 0-degree angle constantly, some of the 0-degree angle values are dropped from the dataset. This preprocessing step helps ensure the model's predictions are more balanced and capable of handling a variety of steering angles, reducing the risk of crashes.

Scope of the research motivation

The scope of this research is centered on two primary motivations. Firstly, it seeks to create a system capable of delivering enhanced images, and the aspiration is to make this technology accessible to a wide audience by scaling the system for widespread use. Secondly, the research is closely aligned with the ongoing research efforts in the field of driverless cars. The overarching goal is to transform the software model developed here into real-life applications for autonomous vehicles, thereby reducing traffic rule violations and road accidents. The research 's flexibility allows for the integration of different training models to continually improve the vehicle's ability to accurately detect turns. The autonomous vehicle developed as part of this research is expected to operate on diverse tracks while maintaining consistent training accuracy. Furthermore, the system aims to provide not only enhanced image outputs but also statistical insights, which can have various practical applications. Crucially, the research 's focus is to ensure that image quality remains high, even during real-time image processing, without compromising the integrity of the original images.

Dataset

The dataset splitting is a crucial step in creating a well-balanced and effective model while preventing overfitting. Without a validation set, the model risks overfitting to the specific track on which the dataset is created, limiting its generalizability to other tracks.To further enhance the model's robustness, variations of the images are augmented. This includes generating zoomed images, adjusting brightness levels, and flipping images. These augmented variations help the model adapt to different scenarios and improve its generalization capabilities.

In terms of image preprocessing, the RGB images in the dataset are transformed into the YUV color space. YUV is preferred for its efficient coding and bandwidth reduction compared to RGB. Additionally, the images are blurred using the Gaussian blur function from OpenCV and resized to crop out unimportant background scenery. Each pixel is then normalized by dividing by 255, ensuring equal priority for all pixels and reducing values to a range of 0 to 1. The training process involves training the model for a specified number of epochs, allowing it to adjust and refine its predictions. This training aims to eliminate bias and enhance the model's efficiency, making it more suitable for generalized track scenarios.

The model is then simulated using an open-source simulator built with Unity3D. To establish real-time communication and gather data, Flask backend is utilized in conjunction with Socket.io, a JavaScript real-time communication library. This setup allows for the collection of real-time data on tilt, angle, coordinates, and speed at each moment, which is essential for the model's performance evaluation and further refinement.

Main principle behind

The main principle behind the tracking in this context is centered on the notion that a vehicle's movement can be seen as a continuous displacement process. As the vehicle progresses, so do the changes in the lane lines, which are indicative of the vehicle's movement on the road. These changes are typically reflected in the slope of the lane lines. When tracking the lane lines, the idea is to compare the slope of the lane lines in the current frame to those in the previous frame. The critical insight here is that these slopes should not vary significantly if the vehicle maintains its position relative to the lane. This concept allows for the confinement of the search for lane lines within a limited area near the previously detected lane line. This approach effectively reduces the computational load and processing required to locate and track lane lines within the region of interest, contributing to the efficiency of the tracking system.

Working

The working of the system involves several key components and principles. The detection and tracking of lane lines in the road environment are fundamental to the system's operation. The system's ability to accurately detect the position of lane lines is primarily based on the polar angle of these lines within the detection area. When the polar angle is within this area, the system can rapidly and precisely locate the lane lines. However, during situations such as turns, lane changes, or shifts in the camera's position, the lane lines may exceed the detection area, resulting in deviations in the detection results.

To address this challenge, a modified Hough transform is employed. Traditional Hough transform methods involve traversing each point in the image at every angle, which can be computationally intensive. The modified Hough transform, on the other hand, focuses on the vanishing point and the limited pixels around it. This modification enhances the real-time performance of the algorithm, making it more efficient and responsive in detecting and tracking lane lines under various conditions.

A critical component of the system is the custom Convolutional Neural Network (CNN). This deep neural network is designed with a sequential model structure, incorporating 5 Convolution2D layers, 4 dropout layers, and 4 dense layers. Additionally, a flatten layer is included to convert the image matrix into a one-dimensional array. This CNN is a key element in processing and analyzing the image data, enabling the system to make predictions and adjustments based on the input data.

The system's overall performance and efficiency are further optimized through an algorithm that decomposes the input image. This decomposition technique, known as segment decomposition, enhances the computational efficiency of the Hough Transform for straight lines detection. It allows for parallel processing, which is particularly beneficial when dealing with large image datasets and real-time applications.

Deep learning is a subset of machine learning that focuses on teaching computers to perform tasks that come naturally to humans, such as learning from experience. Unlike traditional machine learning algorithms that rely on predetermined equations as models, deep learning algorithms use computational techniques to "learn" information directly from data. This approach is particularly well-suited for image recognition tasks, playing a crucial role in solving problems like facial recognition, action detection, and advanced driver assistance technologies including autonomous driving, lane detection, pedestrian detection, and autonomous parking.

Deep learning achieves its capabilities through the use of neural networks, which are computational models inspired by the biological nervous system. These networks consist of multiple layers of processing units, each performing nonlinear operations and working in parallel. The layers in neural networks help create hierarchies of features, enabling the system to learn and represent complex patterns directly from raw data. This technology has revolutionized fields such as computer vision and natural language processing, making it a powerful tool for solving a wide range of real-world problems that involve processing and understanding data.

Unity3D Gaming Engine:

nity3D is a versatile gaming engine developed by Unity Technologies, initially introduced in 2005 at Apple Inc.'s Worldwide Developers Conference with a focus on Mac OS X. Over time, it has evolved to support more than 25 different platforms. Unity3D offers a user-friendly development environment and serves as a powerful cross-platform 3D engine, making it suitable for both beginners and experienced developers. Its broad compatibility allows game and application creation for various platforms, including mobile devices, desktop computers, web browsers, and gaming consoles.

In the context of creating the environment for applications like simulations or games, a tool called OpenRoadEd is used to generate the environment in a specific format called .xodr. This process involves defining different elements of the road, such as straight sections, arcs, and spirals, through the road settings panel. These road components are then interconnected to form the complete track or environment. Textures and visual elements can also be incorporated into the environment, either using existing textures or by adding custom textures to the image library. By saving the generated geometry, the environment is automatically transformed into the .xodr format, which can be used within Unity3D for applications like driving simulations, providing a realistic and immersive experience.

Supervised learning

Supervised learning is a machine learning paradigm that involves working with a finite subset of training examples, denoted as XTrain, which represents a subset of the image space X. Each example in XTrain is associated with corresponding ground truth data, such as steering angles, brake, and throttle information, represented as y(i) with respect to x(i), where i ranges from 1 to n. The supervised learning model, denoted as Mw, is trained using the training set T, which comprises pairs of input data (x(i)) and their corresponding ground truth labels (y(i)), with i ranging from 1 to n.

Artificial Neural Networks (ANNs) form a group of machine learning models inspired by the biological neural networks found in the central nervous system of mammals. These biological neural networks consist of interconnected neurons that process and evaluate information through electrochemical interactions. ANNs attempt to mimic this information processing structure using computers. An artificial neuron, the basic unit of ANNs, receives one or more inputs, denoted as xi, which are multiplied by corresponding weights wi. These weighted inputs are then summed up along with a threshold term θ, known as the bias. The result of this summation is passed through a non-linear function φ, known as an activation function, to produce an output, represented as y. The operation of an artificial neuron can be expressed as:

$y = \varphi (\sum xi \cdot wi + \theta)$

The ANN is represented as a model Mw, consisting of learnable weights w and activation functions φ. The goal of training the ANN is to adjust the weights w to learn and approximate the underlying patterns and relationships between inputs and outputs in the training data, enabling it to make predictions or classifications on new, unseen data.

Existing System

In the existing self-driving car systems, machine learning algorithms are categorized into four main types:

Regression Algorithms: Regression algorithms are primarily used for predicting events. Self-driving cars employ Bayesian regression, neural network regression, and decision forest regression. In regression analysis, the relationship between variables is estimated, and the impact of these variables is compared on different scales. Regression algorithms utilize repetitive aspects of the environment to create a statistical model of the relationship between an image and the position of a specific object within that image. This model enables rapid online detection through image sampling and can gradually expand its knowledge to recognize other objects with minimal human intervention.

Pattern Recognition Algorithms (Classification): Advanced driver-assistance systems (ADAS) collect images with a wealth of data from the surrounding environment. Pattern recognition algorithms, also known as data reduction algorithms, filter out irrelevant data to identify images containing specific categories of objects. These algorithms play a crucial role in preprocessing the sensor data by detecting object edges, fitting line segments, and circular arcs to these edges. By combining these features, they recognize objects. Commonly used pattern recognition algorithms in self-driving cars include support vector machines (SVM) with histograms of oriented gradients (HOG), principal component analysis (PCA), Bayes decision rule, and k-nearest neighbor (KNN).

Cluster Algorithms: Cluster algorithms are used to discover structure within data points. In cases where the images from ADAS are unclear or classification algorithms miss an object due to low resolution or limited data points, it becomes challenging for the system to detect and locate objects. Clustering techniques focus on leveraging the inherent data structures to group data into clusters with the most common attributes. K-means and multi-class neural networks are popular clustering algorithms in autonomous cars.

Decision Matrix Algorithms: Decision matrix algorithms are employed for decision-making in self-driving cars. These algorithms systematically identify, analyze, and rate the performance of relationships between sets of values and information. Gradient boosting (GDM) and AdaBoosting are commonly used decision matrix algorithms. They determine the car's actions, such as making left or right turns, braking, or accelerating. Multiple independently trained decision models generate predictions that are combined to make overall decisions while minimizing errors.

DRAWBACKS IN EXISTING SYSTEM:

The existing self-driving car systems are not without their drawbacks. Some of the key limitations and challenges include:

Processing Power: Deep learning, which plays a central role in self-driving systems, demands a significant amount of computing power. Powerful GPUs (graphical processing units) are typically required to handle the extensive image processing and data processing needs. Companies like Nvidia and Intel are positioning themselves as leaders in providing the necessary computing power for intelligent vehicles. However, challenges remain in creating low-cost GPUs that operate within the energy consumption and thermal constraints required for mass-market vehicles. Bandwidth and synchronization issues also pose challenges.

Available Training Data: End-to-end learning systems, in particular, require vast amounts of training data to predict a wide range of driving scenarios and ensure safety. Some experts argue that at least a billion kilometers of training data from real-world road scenarios are needed to draw meaningful conclusions about a vehicle's safety. Moreover, the training data needs to be diverse to be effective, as repeatedly driving the same kilometer will not suffice.

Safety: Safety is a significant challenge, especially concerning deep neural networks. These networks can be highly sensitive to so-called adversarial perturbations, where minimal modifications to camera images, such as resizing, cropping, or lighting changes, can lead to misclassification. Ensuring the safety of self-learning algorithms like deep learning remains an unsolved problem. Existing automotive safety standards like ISO 26262 do not provide clear guidelines for assessing the safety of self-learning algorithms, making standardization a challenge. Safety incidents, such as the 2016 Tesla autopilot accident, highlight the ongoing need for research and development in this area, as the system failed to recognize a truck due to sensor interference, resulting in a crash.

PROPOSED SYSTEM

The proposed system involves the utilization of Neural Network Regression Algorithms, a type of algorithm known for its predictive capabilities. Regression analysis, which forms the basis for these algorithms, assesses the relationship between multiple variables and evaluates the effects of these variables on different scales. The key determinants in regression analysis include the shape of the regression line, the type of dependent variables, and the number of independent variables.

In the context of Advanced Driver Assistance Systems (ADAS), images from cameras or radar sensors play a crucial role in actuation and localization. However, one of the major challenges for any algorithm is to develop an image-based model for feature selection and prediction. Regression algorithms address this challenge by leveraging the repeatability of the environment to create a statistical model that relates an object's position in an image to the image itself. This statistical model, allowing for image sampling, enables fast online detection and can be learned offline. It can be extended to recognize and predict the positions of other objects without extensive human intervention. Regression algorithms provide outputs regarding an object's position and a level of trust in the object's presence. They can also be employed for short-term prediction and long-term learning.

Regression algorithms suitable for self-driving cars include decision forest regression, neural network regression, and Bayesian regression, among others. Neural networks, which are widely used in the field, can be applied to regression, classification, or unsupervised learning tasks. They handle unlabeled data, classify data, or predict continuous values after supervised training. Neural networks often use a form of logistic regression in their final layer to transform continuous data into discrete variables such as 1 or 0.

To implement this system, the code will be written in Python3, and Google Collaboratory will be used as the development platform. Google Collaboratory is a free cloud-based platform that supports writing and running code in Python. Importantly, it provides access to GPUs, enhancing the speed and efficiency of code execution, making it a cost-effective and powerful tool for artificial intelligence and machine learning applications in self-driving cars.

ADVANTAGES OF PROPOSED SYSTEM

The proposed system offers several advantages:

High Accuracy: The system is expected to achieve a high level of accuracy in various tasks, including object localization, object detection, and movement prediction. This accuracy is crucial for the safe and efficient operation of self-driving cars.

Uncompressed Picture Quality: The system aims to maintain the quality of images without compression. This is important for ensuring that the visual data captured by cameras or sensors retains its fidelity, allowing for precise analysis and decision-making.

Generalized Environment Compatibility: The proposed system is designed to work effectively in a wide range of environments. It can adapt to and perform well in different scenarios, making it suitable for deployment in a variety of driving conditions.

ALGORITHM’S AND APPROACH:

In terms of algorithms and approaches, machine learning methods are categorized into four main classes: decision matrix algorithms, cluster algorithms, pattern recognition algorithms, and regression algorithms. It's important to note that some machine learning algorithms can be versatile and used to accomplish multiple subtasks. For example, regression algorithms can be applied for tasks such as object localization, object detection, and predicting object movements. This versatility allows for efficient and integrated use of algorithms to handle different aspects of self-driving car functionality.

Model Architecture: The model is designed as a sequential neural network. It includes 5 Convolutional 2D layers for feature extraction, 4 dropout layers for regularization, and 4 dense layers for further processing. Additionally, there is a single flatten layer added to convert the image matrix into a one-dimensional array.

Activation Function: The ELU (Exponential Linear Unit) activation function is employed in this model. ELU is a modified version of the ReLU (Rectified Linear Unit) activation function. It returns the same value for inputs greater than or equal to zero (x >= 0) and a value computed as alpha times the exponential of x minus 1 for inputs less than zero (x < 0). This non-linearity helps prevent plateau or constant (stagnant) loss values during the training of deep neural networks.

Training: The training process involves using the .fit function with the YUV format training images as input to the model. During training, the loss values for both the training and validation datasets are compared and plotted over the epochs. The objective is to have the loss value for the validation dataset become lower than that of the training dataset by the end of the training process. This ensures that the model is generalized and free from overfitting and underfitting issues.

The model is trained for a total of 10 epochs, which has been found to yield good results for your research . Additionally, 32 images are processed in each batch during training, which contributes to faster training.

Model Architecture: The model is designed as a sequential neural network. It includes 5 Convolutional 2D layers for feature extraction, 4 dropout layers for regularization, and 4 dense layers for further processing. Additionally, there is a single flatten layer added to convert the image matrix into a one-dimensional array.

Activation Function: The ELU (Exponential Linear Unit) activation function is employed in this model. ELU is a modified version of the ReLU (Rectified Linear Unit) activation function. It returns the same value for inputs greater than or equal to zero (x >= 0) and a value computed as alpha times the exponential of x minus 1 for inputs less than zero (x < 0). This non-linearity helps prevent plateau or constant (stagnant) loss values during the training of deep neural networks.

Training: The training process involves using the .fit function with the YUV format training images as input to the model. During training, the loss values for both the training and validation datasets are compared and plotted over the epochs. The objective is to have the loss value for the validation dataset become lower than that of the training dataset by the end of the training process. This ensures that the model is generalized and free from overfitting and underfitting issues.

The model is trained for a total of 10 epochs, which has been found to yield good results for your research . Additionally, 32 images are processed in each batch during training, which contributes to faster training.

Libraries:

NumPy: NumPy is a fundamental library for scientific computing with Python. It provides efficient multidimensional array operations, making it essential for tasks like data manipulation and processing.

Matplotlib: Matplotlib is a powerful plotting library for Python. It's particularly useful for creating visualizations and graphs, which can be valuable for analyzing and presenting data.

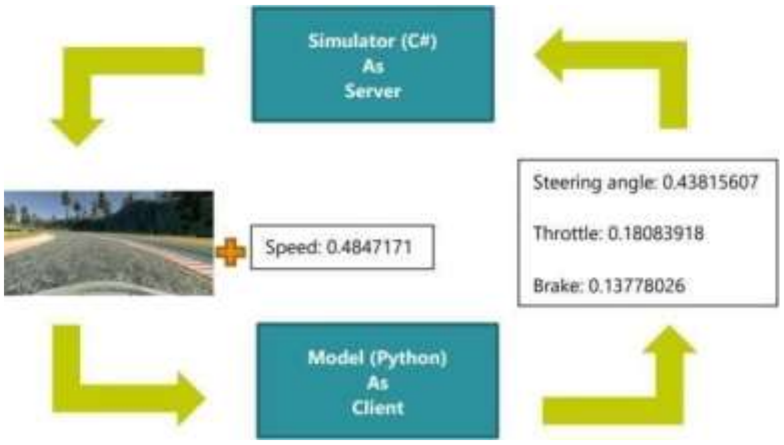
OS: The OS module is a standard Python library that allows you to interact with the operating system. It's useful for tasks like file handling and directory operations.cv2/OpenCV: OpenCV-Python is a computer vision library that provides tools for various image processing and computer vision tasks. It's particularly relevant for working with image data, which is crucial in your research .Keras with TensorFlow as Backend: Keras is a high-level neural network library that allows for rapid prototyping and experimentation with deep learning models. It can run on top of TensorFlow, which is a popular deep learning framework.

Scikit-Learn (Sklearn): Scikit-Learn is a machine learning library for Python. It offers a wide range of machine learning algorithms for classification, regression, and clustering. It's a valuable tool for building and evaluating machine learning models.

Pandas: Pandas is a data manipulation library that's built on top of NumPy. It's particularly useful for handling and analyzing tabular data, making it a key tool for working with datasets in your research .

DESIGN APPROACH AND DETAIL

SYSTEM ARCHITECTURE



DETAILED DESIGN

The communication and integration between the Udacity framework, the autonomous car model, and the simulator are essential for the successful implementation of your self-driving car system. Here's an overview of the communication process and the tools involved: Communication Interface: To establish real-time bidirectional communication between the model and the simulator, a Python script acts as an interface. This script is responsible for exchanging data between the two systems.

Socket-IO Package: The Socket-IO package is used to facilitate the communication between the model (Python) and the simulator (C#). Socket-IO is originally written in JavaScript and is well-suited for real-time applications. It enables fast and reliable data exchange.

Client-Server Setup: In this communication setup, the simulator acts as the server, while the model (Python script) acts as the client. The Python script sends requests to the server (simulator) using a defined socket number (e.g., 4567) and waits for responses.

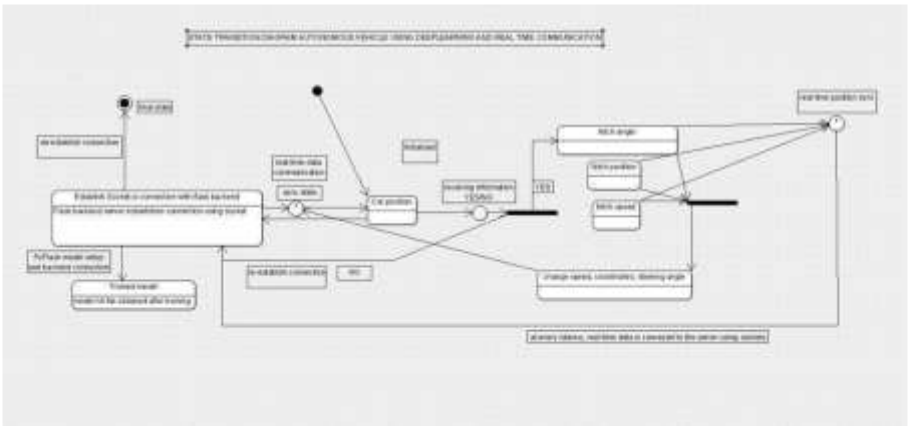
Server-Side (C#): The server-side is implemented in C# within the simulator. It listens for and accepts requests from the client (model) and establishes the connection.

Data Exchange: Once the connection is established, data is transmitted between the model and the simulator. This includes controlling values such as the steering angle, throttle, and speed of the car. The Python script also saves images for later analysis and data validation.

Model Predictions: After receiving the data from the simulator, the Python script passes it to the model for processing. The model generates predictions for the new steering angle, throttle, and brake event magnitude.

Feedback to Simulator: The predicted controlling values are sent back to the simulator via the established communication link. These values are used to make real-time adjustments to the car's behavior by the C# code in the simulator.

SEQUENCE DIAGRAM



ACTIVITY DIAGRAM

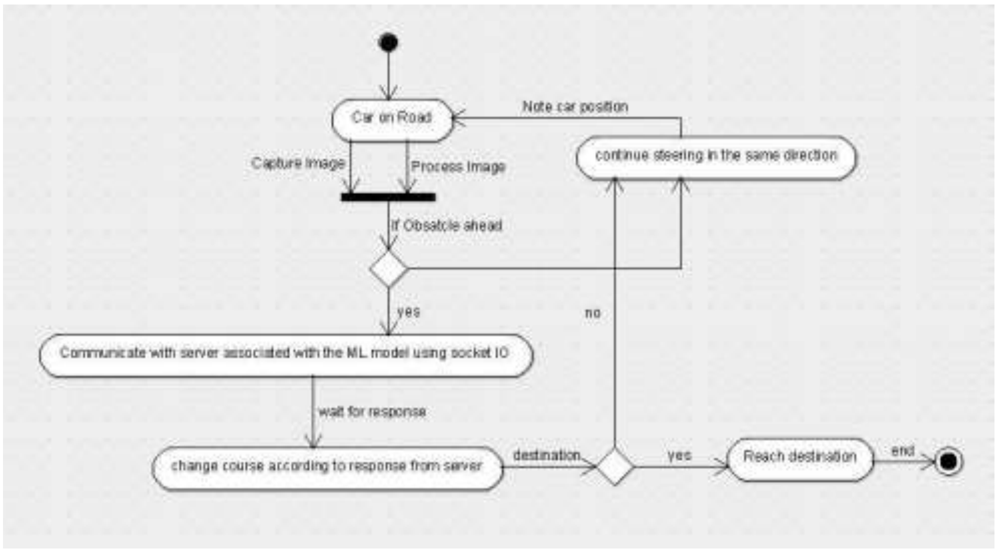
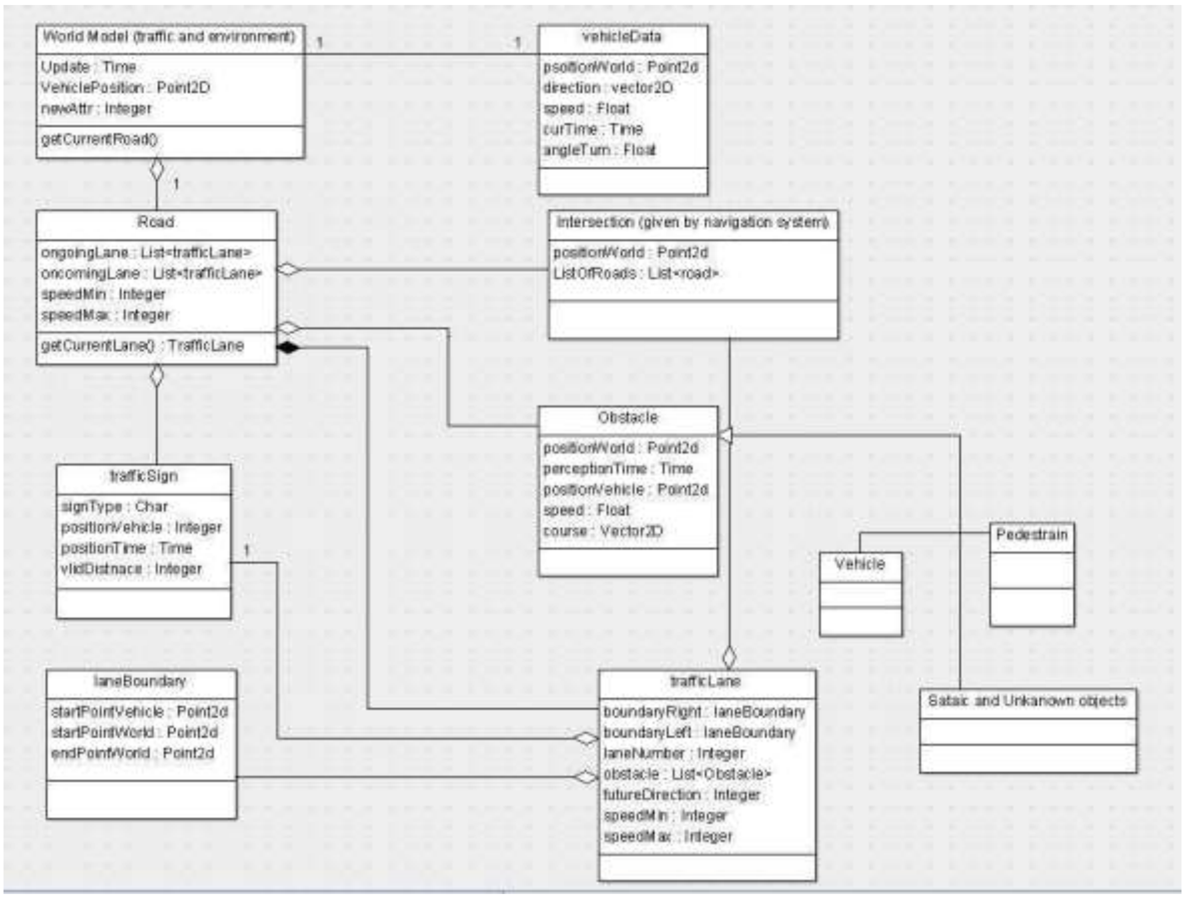
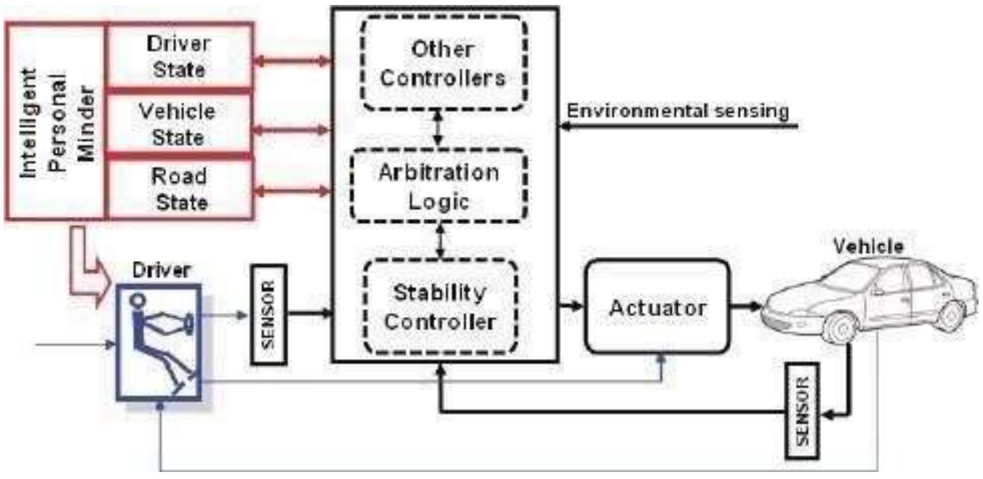

Car on Road
Note car position
continue steering in the same direction
Capture Image
Process Image
If Obsatcle ahead
yes
no
Communicate with server associated with the ML model using socket IO
wait for response
change course according to response from server
destination
yes
Reach destination
end

Figure 5.2.2

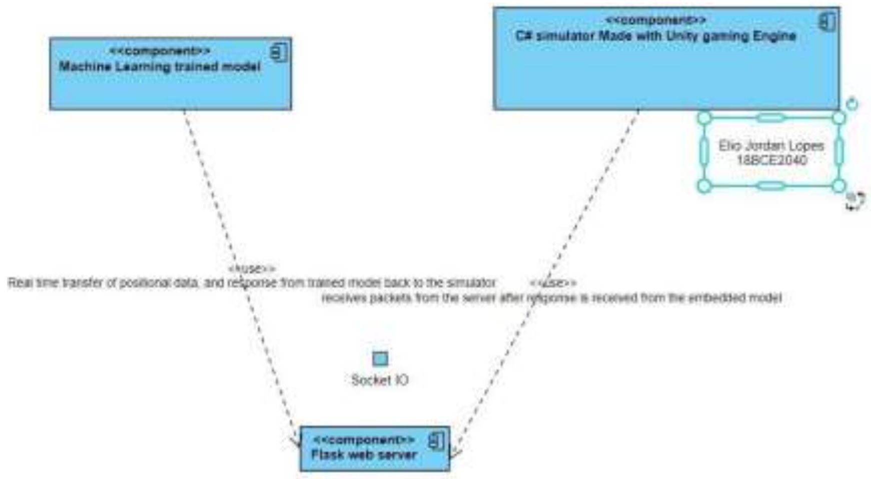
CLASS DIAGRAM



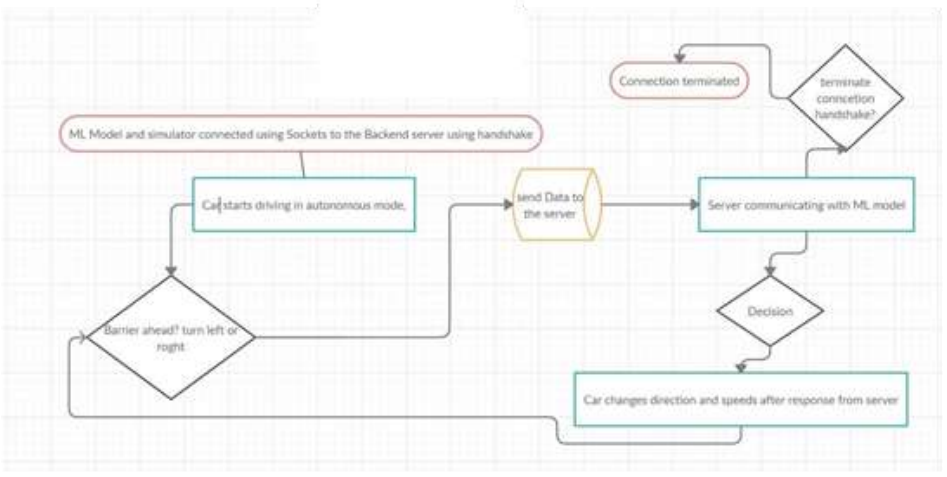
BLOCK DIAGRAM



COMPONENT DIAGRAM


<<component>> Machine Learning trained model
<<component>> C# simulator Made with Unity gaming Engine
Elio Jordan Lopes 18BCE2040
<<use>>
Real time transfer of positional data, and response from trained model back to the simulator
<<use>>
receives packets from the server after response is received from the embedded model
Socket IO
<<component>> Flask web server

DATA FLOW DIAGRAM


Connection terminated
terminate conncetion handshake?
ML Model and simulator connected using Sockets to the Backend server using handshake
Car starts driving in autonomous mode.
send Data to the server
Server communicating with ML model
Decision
Barrier ahead? turn left or roght
Car changes direction and speeds after response from server

Collaborative Diagram

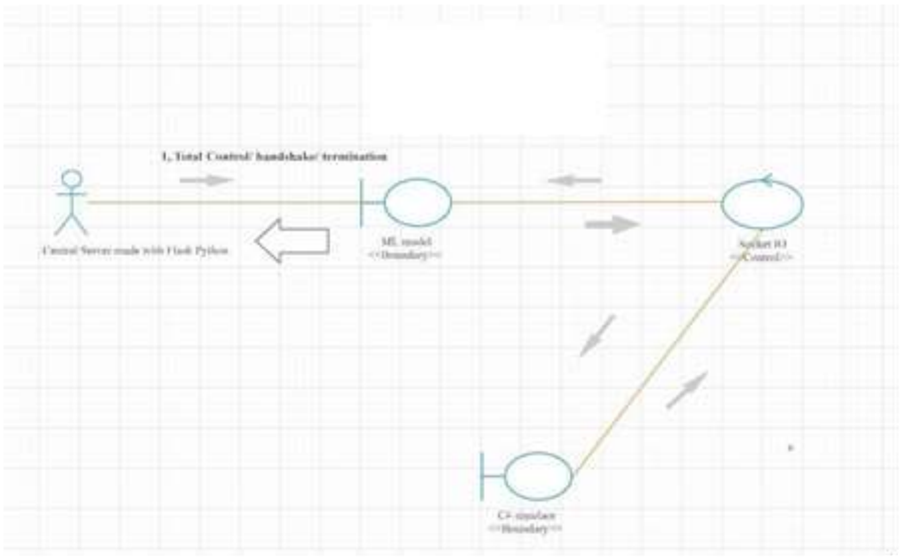
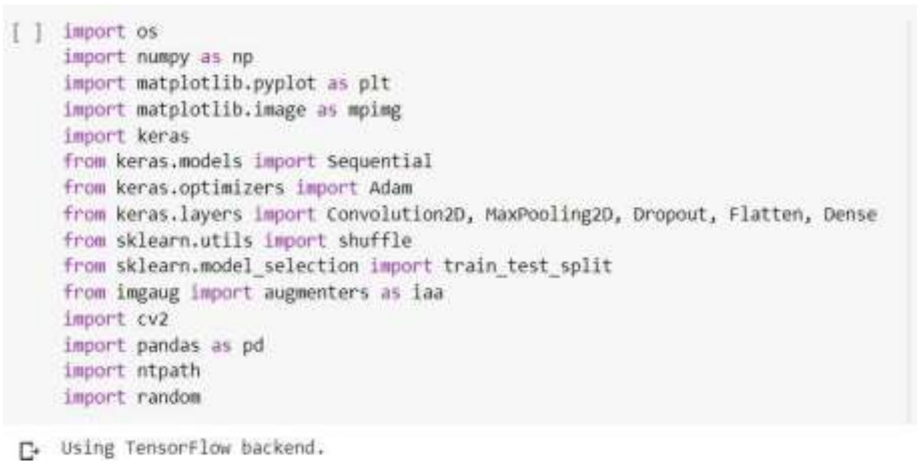

1, Total Control/ handshake/ termination
Central Server made with Flask Python
ML model
<<Boundary>>
Socket IO
<<Control>>
C# simulator
<<Boundary>>

Figure 5.2.7

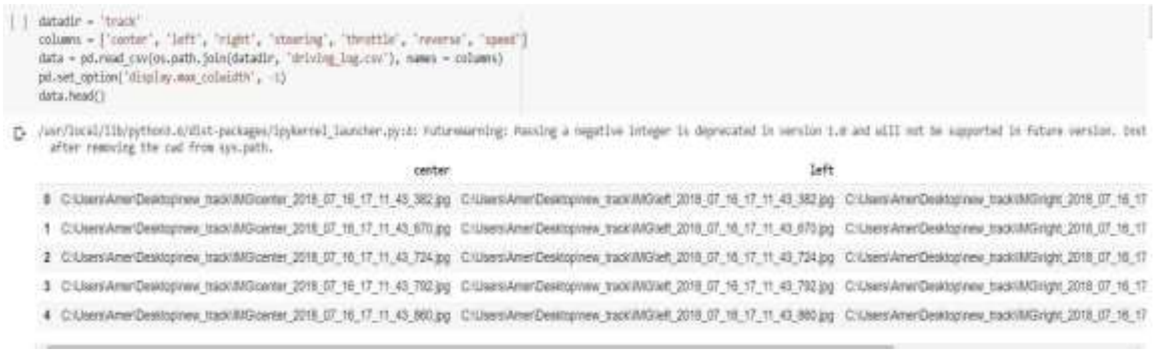
SAMPLE CODE

Importing the libraries:



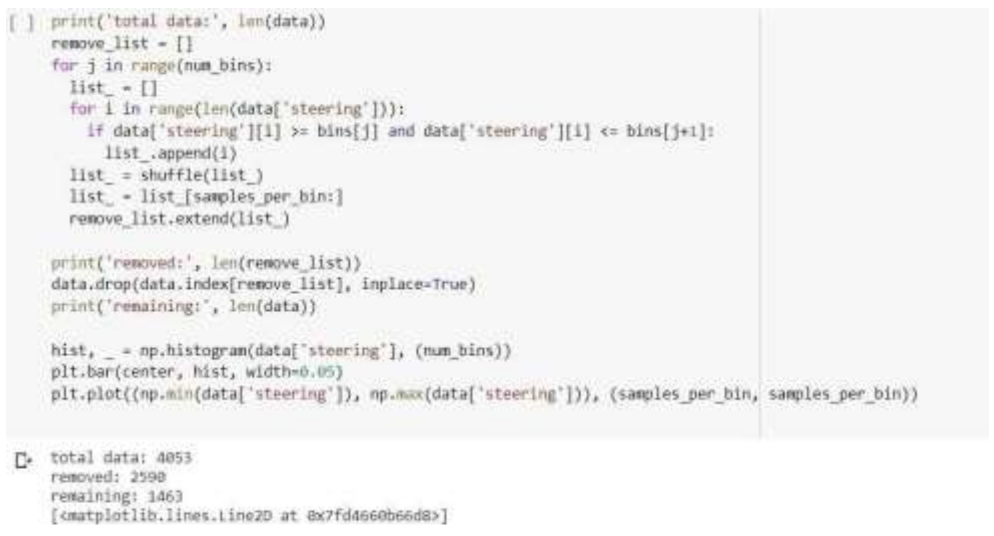
```
[ ] import os
    import numpy as np
    import matplotlib.pyplot as plt
    import matplotlib.image as mpimg
    import keras
    from keras.models import Sequential
    from keras.optimizers import Adam
    from keras.layers import Convolution2D, MaxPooling2D, Dropout, Flatten, Dense
    from sklearn.utils import shuffle
    from sklearn.model_selection import train_test_split
    from imgaug import augmenters as iaa
    import cv2
    import pandas as pd
    import ntpath
    import random

    Using TensorFlow backend.
```

Load data:



```
[ ] datadir = 'track'
    columns = ['center', 'left', 'right', 'steering', 'throttle', 'reverse', 'speed']
    data = pd.read_csv(os.path.join(datadir, 'driving_log.csv'), names = columns)
    pd.set_option('display.max_colwidth', -1)
    data.head()

    /usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:4: FutureWarning: Passing a negative integer is deprecated in version 1.0 and will not be supported in future version. Inst
      after removing the cwd from sys.path.
                                                     center                                                       left
    0 C:\Users\Amer\Desktop\new_track\IMG\center_2018_07_16_17_11_43_382.jpg C:\Users\Amer\Desktop\new_track\IMG\left_2018_07_16_17_11_43_382.jpg C:\Users\Amer\Desktop\new_track\IMG\right_2018_07_16_17
    1 C:\Users\Amer\Desktop\new_track\IMG\center_2018_07_16_17_11_43_670.jpg C:\Users\Amer\Desktop\new_track\IMG\left_2018_07_16_17_11_43_670.jpg C:\Users\Amer\Desktop\new_track\IMG\right_2018_07_16_17
    2 C:\Users\Amer\Desktop\new_track\IMG\center_2018_07_16_17_11_43_724.jpg C:\Users\Amer\Desktop\new_track\IMG\left_2018_07_16_17_11_43_724.jpg C:\Users\Amer\Desktop\new_track\IMG\right_2018_07_16_17
    3 C:\Users\Amer\Desktop\new_track\IMG\center_2018_07_16_17_11_43_792.jpg C:\Users\Amer\Desktop\new_track\IMG\left_2018_07_16_17_11_43_792.jpg C:\Users\Amer\Desktop\new_track\IMG\right_2018_07_16_17
    4 C:\Users\Amer\Desktop\new_track\IMG\center_2018_07_16_17_11_43_860.jpg C:\Users\Amer\Desktop\new_track\IMG\left_2018_07_16_17_11_43_860.jpg C:\Users\Amer\Desktop\new_track\IMG\right_2018_07_16_17
```

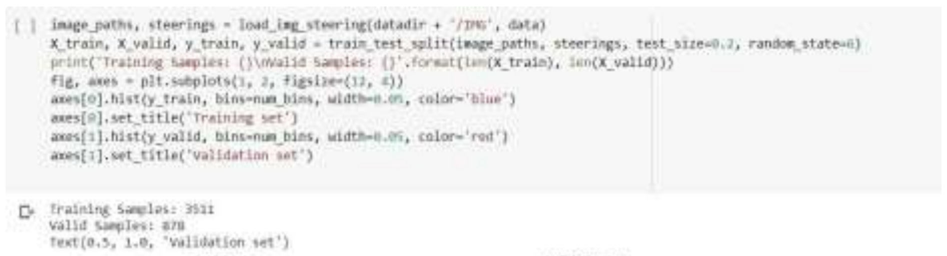
Deleting high frequency 0-degree steering angle values to prevent biased mod:



```
[ ] 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: 4053
removed: 2590
remaining: 1463
[<matplotlib.lines.Line2D at 0x7fd4660b66d8>]
```

Splitting into training and validation set:



```
[ ] image_paths, steerings = load_img_steering(datadir + '/IMG', data)
    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)))
    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')
```

```
Training Samples: 3511
Valid Samples: 878
Text(0.5, 1.0, 'Validation set')
```

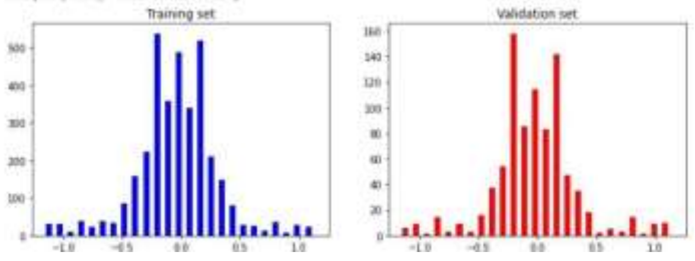
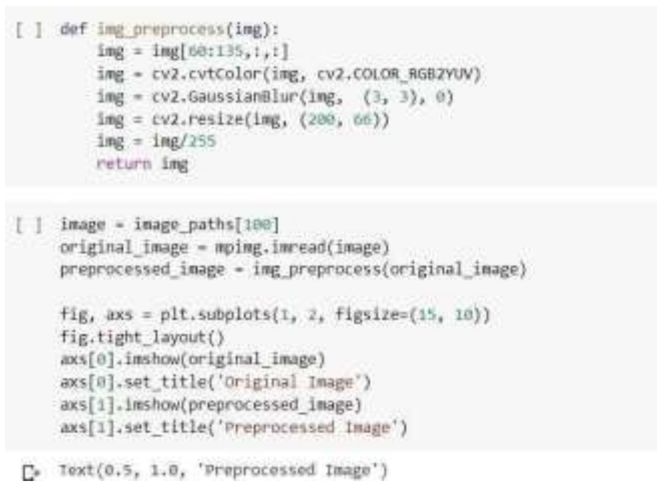


Image Pre-Processing:

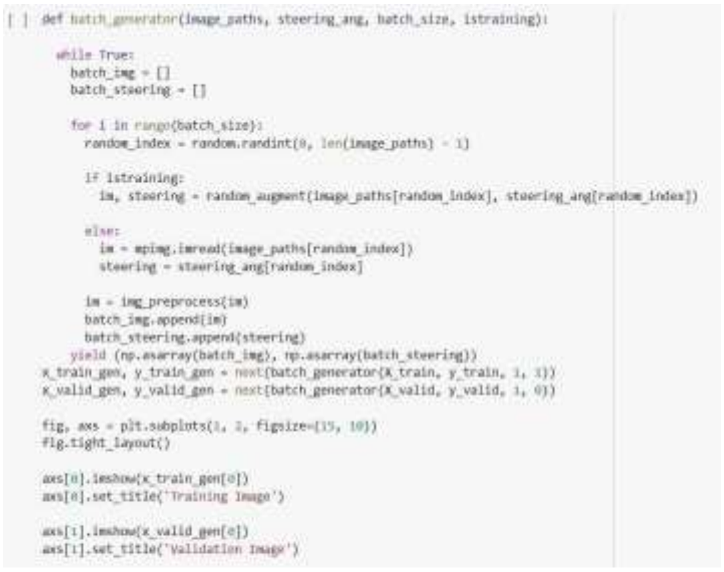


```
[ ] def img_preprocess(img):
        img = img[60:135,:,:]
        img = cv2.cvtColor(img, cv2.COLOR_RGB2YUV)
        img = cv2.GaussianBlur(img,  (3, 3), 0)
        img = cv2.resize(img, (200, 66))
        img = img/255
        return img
```

```
[ ] image = image_paths[100]
    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')
```

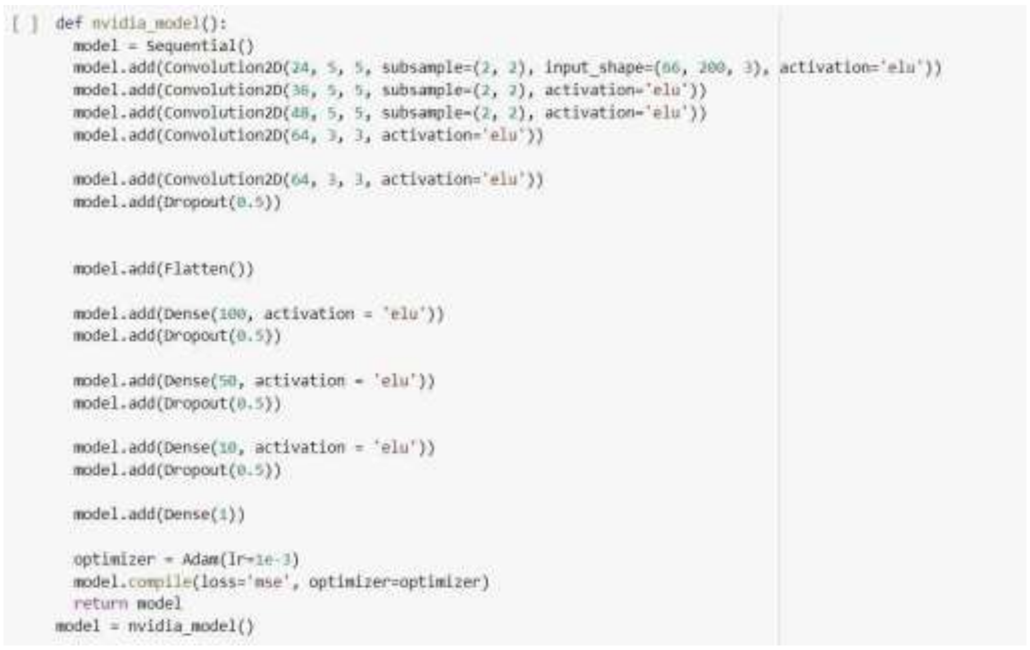
```
Text(0.5, 1.0, 'Preprocessed Image')
```

Batch Generator:



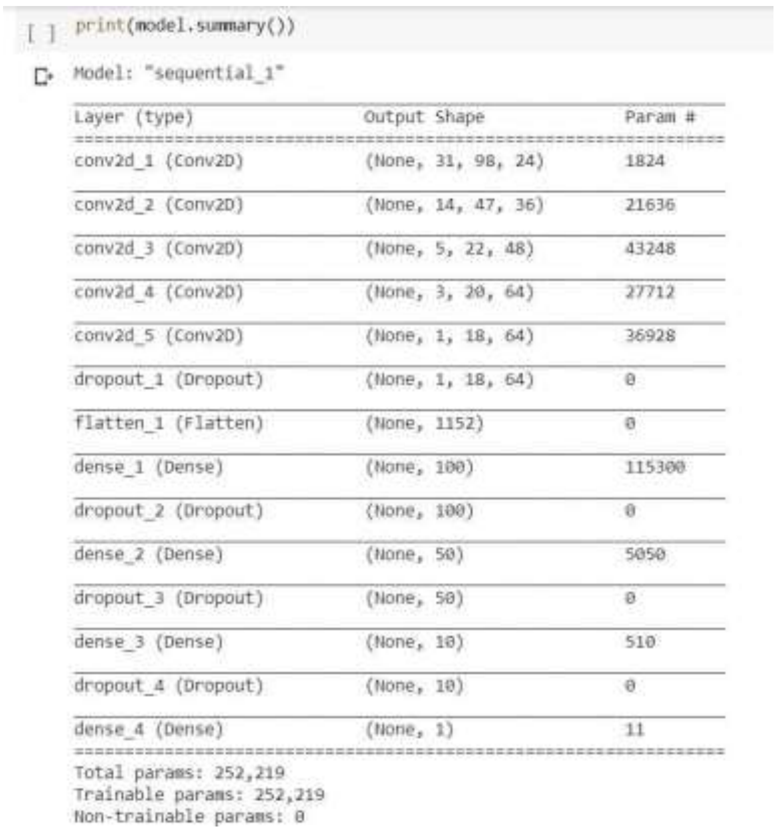
```
[ ]  def batch_generator(image_paths, steering_ang, batch_size, istraining):

       while True:
         batch_img = []
         batch_steering = []

         for i in range(batch_size):
           random_index = random.randint(0, len(image_paths) - 1)

           if istraining:
             im, steering = random_augment(image_paths[random_index], steering_ang[random_index])

           else:
             im = mpimg.imread(image_paths[random_index])
             steering = steering_ang[random_index]

           im = img_preprocess(im)
           batch_img.append(im)
           batch_steering.append(steering)
         yield (np.asarray(batch_img), np.asarray(batch_steering))
     x_train_gen, y_train_gen = next(batch_generator(X_train, y_train, 1, 1))
     x_valid_gen, y_valid_gen = next(batch_generator(X_valid, y_valid, 1, 0))

     fig, axs = plt.subplots(1, 2, figsize=(15, 10))
     fig.tight_layout()

     axs[0].imshow(x_train_gen[0])
     axs[0].set_title('Training Image')

     axs[1].imshow(x_valid_gen[0])
     axs[1].set_title('Validation Image')
```

Figure 6.1.7 Nvidia Model:



```
[ ]  def nvidia_model():
       model = Sequential()
       model.add(Convolution2D(24, 5, 5, subsample=(2, 2), input_shape=(66, 200, 3), activation='elu'))
       model.add(Convolution2D(36, 5, 5, subsample=(2, 2), activation='elu'))
       model.add(Convolution2D(48, 5, 5, subsample=(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=1e-3)
       model.compile(loss='mse', optimizer=optimizer)
       return model
     model = nvidia_model()
```

Model summary:



```
print(model.summary())

Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 31, 98, 24)        1824
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 14, 47, 36)        21636
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 5, 22, 48)         43248
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 3, 20, 64)         27712
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 1, 18, 64)         36928
_________________________________________________________________
dropout_1 (Dropout)          (None, 1, 18, 64)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 1152)              0
_________________________________________________________________
dense_1 (Dense)              (None, 100)               115300
_________________________________________________________________
dropout_2 (Dropout)          (None, 100)               0
_________________________________________________________________
dense_2 (Dense)              (None, 50)                5050
_________________________________________________________________
dropout_3 (Dropout)          (None, 50)                0
_________________________________________________________________
dense_3 (Dense)              (None, 10)                510
_________________________________________________________________
dropout_4 (Dropout)          (None, 10)                0
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 11
=================================================================
Total params: 252,219
Trainable params: 252,219
Non-trainable params: 0
```

Drive Coordinates are plotted in real time (-) indicates left turn and (+) right turn:

-0.0488455705344677 -2.0188300000000003 30.1883

-0.0488455705344677 -2.01879 30.1879

-0.06489957123994827 -2.01879 30.1879

-0.05687129124999046 -2.0187999999999997 30.188

-0.05687129124999046 -2.0187999999999997 30.188

-0.05891359969973564 -2.0188099999999998 30.1881

-0.059988927096128464 -2.01875 30.1875

-0.07196944206953049 -2.01878 30.1878

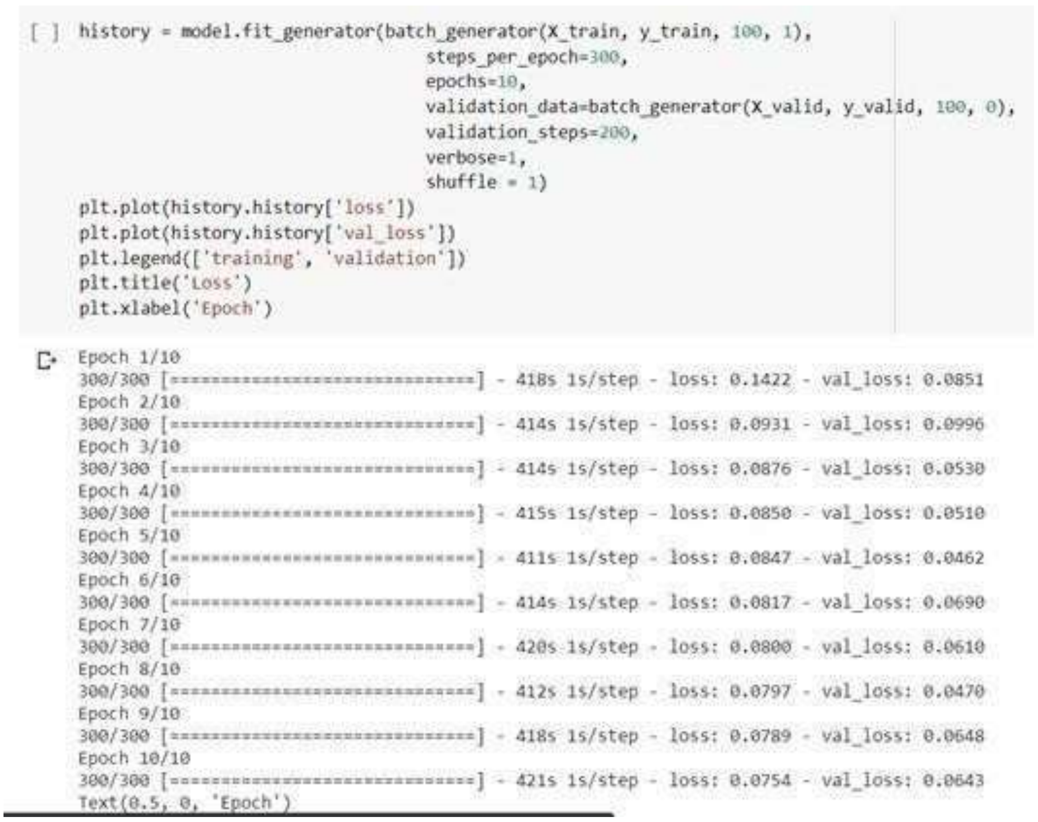
-0.06476251780986786 -2.0188099999999998 30.1881

-0.05087998881936073 -2.0187999999999997 30.188

-0.05087998881936073 -2.01878 30.1878

127.0.0.1 - - [06/Jun/2020 07:23:57] "GET

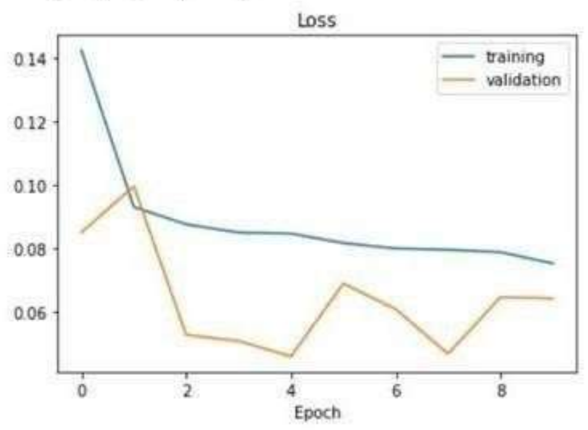
/socket.io/?EIO=4&transport=websocket HTTP/1.1" 200 0 100.167020

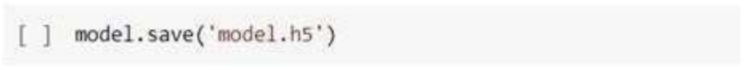


```
[ ] history = model.fit_generator(batch_generator(X_train, y_train, 100, 1),
                                  steps_per_epoch=300,
                                  epochs=10,
                                  validation_data=batch_generator(X_valid, y_valid, 100, 0),
                                  validation_steps=200,
                                  verbose=1,
                                  shuffle = 1)
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.legend(['training', 'validation'])
    plt.title('Loss')
    plt.xlabel('Epoch')
```

```
Epoch 1/10
300/300 [==============================] - 418s 1s/step - loss: 0.1422 - val_loss: 0.0851
Epoch 2/10
300/300 [==============================] - 414s 1s/step - loss: 0.0931 - val_loss: 0.0996
Epoch 3/10
300/300 [==============================] - 414s 1s/step - loss: 0.0876 - val_loss: 0.0530
Epoch 4/10
300/300 [==============================] - 415s 1s/step - loss: 0.0850 - val_loss: 0.0510
Epoch 5/10
300/300 [==============================] - 411s 1s/step - loss: 0.0847 - val_loss: 0.0462
Epoch 6/10
300/300 [==============================] - 414s 1s/step - loss: 0.0817 - val_loss: 0.0690
Epoch 7/10
300/300 [==============================] - 420s 1s/step - loss: 0.0800 - val_loss: 0.0610
Epoch 8/10
300/300 [==============================] - 412s 1s/step - loss: 0.0797 - val_loss: 0.0470
Epoch 9/10
300/300 [==============================] - 418s 1s/step - loss: 0.0789 - val_loss: 0.0648
Epoch 10/10
300/300 [==============================] - 421s 1s/step - loss: 0.0754 - val_loss: 0.0643
Text(0.5, 0, 'Epoch')
```

Fitting the model (with each epoch summary)





```
[ ] model.save('model.h5')
```

Loss comparison between validation and training data

Dataset Collection:

Data for training the self-driving car model is gathered using the Udacity car simulator.

The simulator provides a virtual track where a human driver can manually drive the car.

A recording option in the simulator allows you to capture images at each time step during manual driving.

These captured images represent the input data for the model.

The simulator also records the corresponding steering angle for each image, which serves as the target or label for training the model.

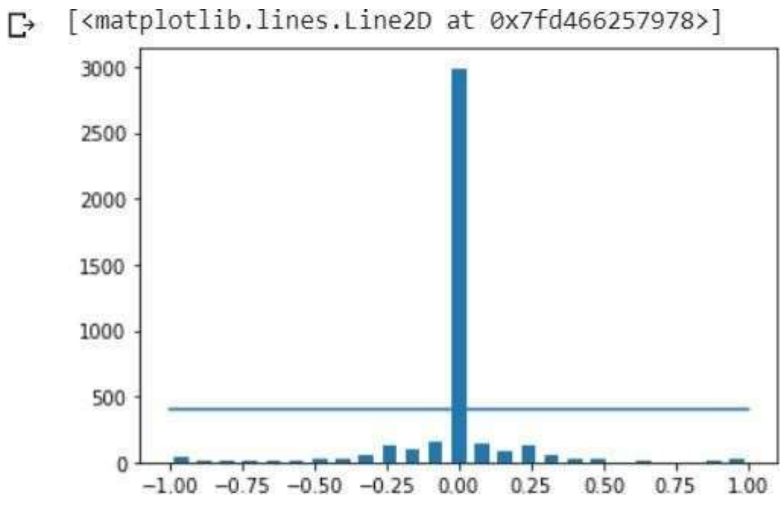
The simulator has three cameras that capture images from the left, center, and right perspectives, providing additional data for training.

Data Preprocessing:

To ensure that the dataset is well-balanced and not biased, you've mentioned that you need to address the issue of high-frequency dataset values, specifically the 0-degree steering angle values.

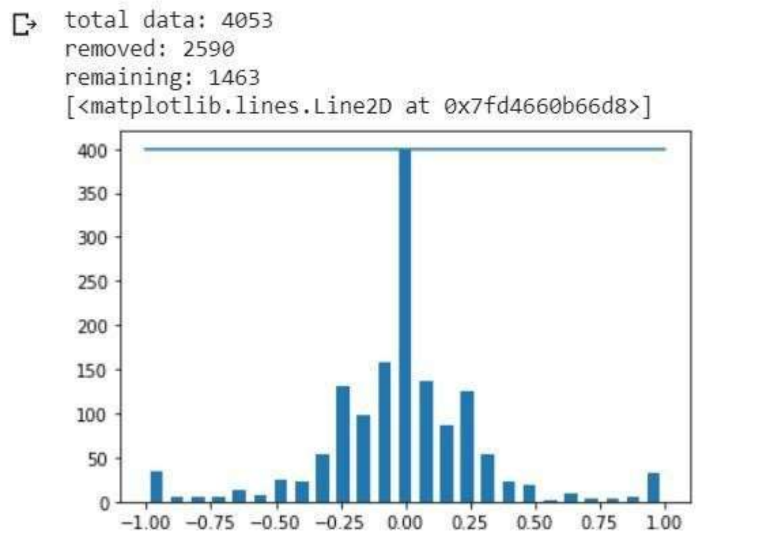
Since most of the driving happens near the center of the track, a large number of 0-degree steering angle values are expected, which could lead to the model predicting 0-degree angles by default, potentially causing issues or crashes.

To mitigate this bias, you've mentioned that you drop some of the 0-degree angle values from the dataset.By removing or reducing the frequency of 0-degree steering angles, you aim to create a more balanced dataset that represents a wider range of steering angles, including turns and curves.



```
[<matplotlib.lines.Line2D at 0x7fd466257978>]
```

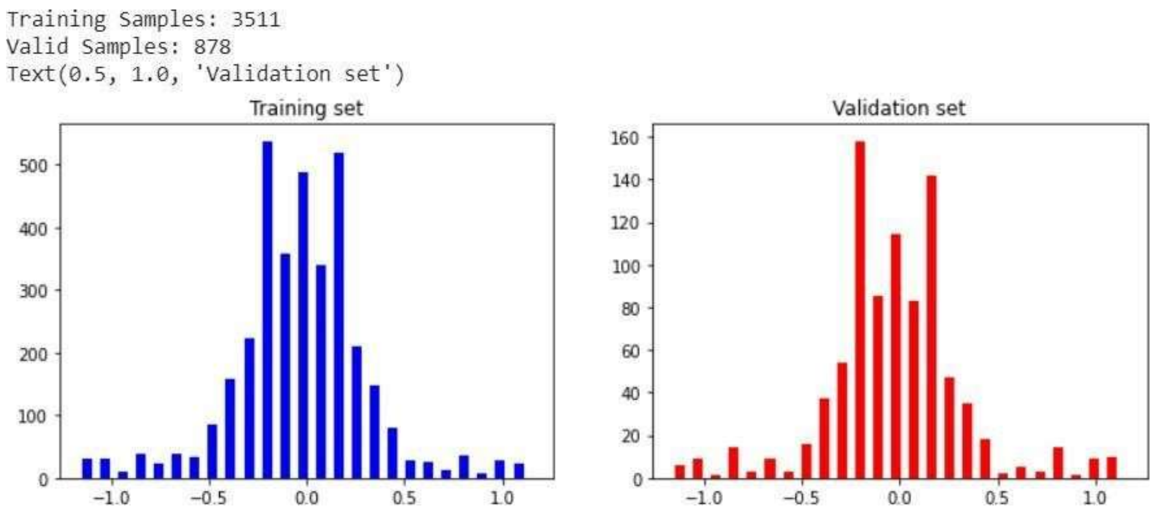
Original steering angle dataset (Frequency of each angle in dataset)



```
total data: 4053
removed: 2590
remaining: 1463
[<matplotlib.lines.Line2D at 0x7fd4660b66d8>]
```

Dataset after deleting some extra 0 degree angle values

Training set and validation set splitting of dataset for better model creation and prevention of overfitting of training data: *If we do not make a validation set then the model will overfit and won't* work well for generalized tracks. It will only work well for the track on which the dataset is created.



```
Training Samples: 3511
Valid Samples: 878
Text(0.5, 1.0, 'Validation set')
```

Conclusion

This research presents a proposed model aimed at realizing the vision of driverless cars. As the world of autonomous vehicles continues to evolve, our work serves as a stepping stone towards the integration of simulated models as practical software solutions in real-life automobiles. The ongoing research in this field is a testament to the collective efforts of millions of data scientists and artificial intelligence experts who are tirelessly working to transition these software models into tangible applications for self-driving cars. The ultimate goal is to not only eliminate the need for human drivers but also to significantly reduce rule violations and, most importantly, the occurrence of road accidents.

Our research demonstrates the feasibility of this approach, offering a stable model that, when simulated, successfully navigates a track while reaching speeds of up to 30 km/h. The model dynamically adjusts the tilt angle to account for right and left turns, ensuring a safe and collision-free journey. Furthermore, our system generates comprehensive positional and angular data at every time point based on both polar and Cartesian coordinates of the car within the simulated environment. This data is stored in a CSV format, ready for future self-driving research and a wide range of analytical experiments. The flexibility of our model opens the door for its extension and adaptation with different training models, thereby enhancing the accuracy of turn detection by autonomous vehicles.