International Journal of Scientific Research in Science, Engineering and Technology

Print ISSN - 2395-1990
Online ISSN : 2394-4099

OPEN ACCESS

Available Online at :www.ijsrset.com
doi :https://doi.org/10.32628/IJSRSET2411269

Self-Driving Car Using Simulator

Prof. Shamal Patil[1], Hrushikesh Bhujbal[2], Fatru Shaikh[2], Priya Lokhande[2], Aashika Jain[5]

[1]Professor, [2]Student

Department of Artificial Intelligence & Data Science, ZEAL College of Engineering & Research, Pune, Maharashtra, India

ARTICLEINFO

Article History:

Accepted: 15 April 2024
Published: 28 April 2024

Publication Issue :
Volume 11, Issue 2
March-April-2024
Page Number :
508-516

ABSTRACT

Due to rapid technological growth in transportation, self-driving cars became the topic of concern. The main purpose of this project is to use the CNN and train the neural network in order to drive the car in autonomous mode in a simulator environment. Front camera of a car captures the images and we use those captured images in order to train the model, in short we can say we have used the concept of behavioural cloning. In behavioural cloning, the system tries to mimic the human driving behaviour by tracking the steering angle. That means a dataset is generated in the simulator by a user driven car in training mode, and the deep neural network model then drives the car in autonomous mode.

In one track a car is trained and in other tracks the car drives in autonomous mode. The dataset for Track 1, which was straightforward to drive and had good road conditions, was utilized as the training set for the automobile to drive itself on Track 2, which has abrupt curves, barriers, heights, and shadows are all things to consider. Image processing and other augmentation techniques were utilized to solve this difficulty, allowing for the extraction of as many data and features as feasible. In the end, the vehicle performed admirably on Track 2. In the future, the team hopes to achieve the same level of accuracy using real-time data.

Keywords: Self-driving car, CNN, Image Processing, Dataset Generation, Real Time Data, Augmentation Techniques

I. INTRODUCTION

The purpose of this project is to build a better autonomous driver which is capable of driving itself without falling off the track which uses all the functionalities of the car provided like brake, accelerator etc.

The main challenge is to mimic the driving behavior of the human driver on the simulator provided by Udacity. Here we train the neural networks so that we can mimic the behavior which is called behavior cloning technique.

There are two tracks and two modes in the simulator: training mode and autonomous mode. The dataset is

created by the user when driving the automobile in training mode in the simulator. The "excellent" driving data is another name for this dataset. The deep learning model is then tested on the track to see how it performs after being trained with the user data. Another difficulty is transferring the results to various tracks.

That is, the model is trained using the dataset produced on one of the simulator's tracks, and it is then tested on the other track.

II. METHODOLOGY

2.1 Data Collection

The data collection process involved utilizing a simulator environment to capture images from the front camera of a car. The simulator provided a realistic virtual driving experience, allowing for the generation of a diverse dataset representative of various driving scenarios. The car's front camera recorded images as the vehicle traversed through different tracks with varying road conditions, including straight stretches, curves, barriers, heights, and shadows.

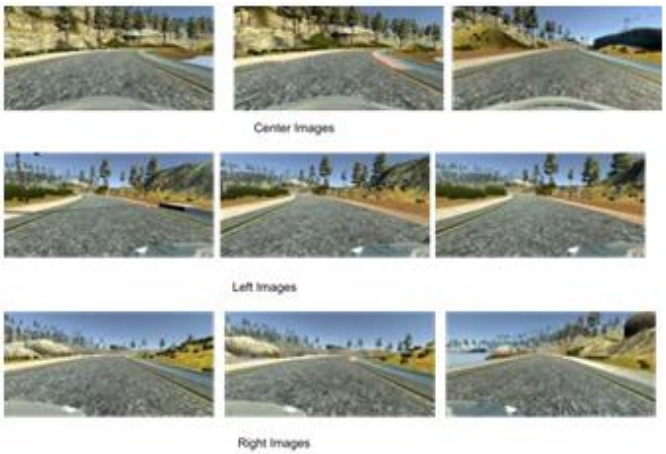

Fig - 1

2.2 Preprocessing

Prior to training the neural network model, the collected images underwent preprocessing steps to enhance their suitability for training. This preprocessing included normalization to standardize pixel values, resizing to ensure uniform image dimensions, and data augmentation techniques to increase the diversity of the dataset. Augmentation techniques such as random rotation, flipping, and brightness adjustment were employed to simulate real-world variations in lighting and perspective.

2.3 Model Architecture

The architecture of the convolutional neural network (CNN) used in this research consisted of multiple layers designed to extract relevant features from the input images and make predictions regarding steering angles. The CNN architecture comprised convolutional layers for feature extraction, followed by pooling layers for dimensionality reduction, and fully connected layers for steering angle prediction. Specific configurations, including the number of layers and filter sizes, were determined through experimentation to optimize model performance.

2.4 Training Procedure

The training procedure involved feeding the preprocessed images into the CNN model to learn the mapping between input images and corresponding steering angles. The Adam optimization algorithm was employed to minimize the mean squared error loss function, which quantified the disparity between predicted and actual steering angles. A batch size of [specify batch size] and a predefined number of epochs [specify number of epochs] were used during training to iteratively update the model parameters and improve performance.

Fig – 2

2.5 Validation and Testing

To comprehensively evaluate the effectiveness of the trained model, a systematic approach was adopted, involving the segregation of the dataset into distinct subsets for validation and testing purposes. This division facilitated rigorous scrutiny of the model's performance under various conditions.

The validation dataset played a crucial role in the iterative training process by serving as a means to monitor the model's performance and prevent overfitting. By periodically assessing the model's performance on this subset during training, adjustments could be made to optimize its learning dynamics and prevent it from memorizing noise in the training data. This iterative refinement process ensured that the model's predictive capabilities were honed to generalize well to unseen data.

Subsequently, the testing dataset served as the ultimate litmus test for the model's generalization ability. By evaluating the model on previously unseen data, its capacity to make accurate predictions in real-world scenarios was rigorously assessed. Performance metrics such as accuracy, mean squared error, and root mean squared error were meticulously computed to quantitatively gauge the model's predictive prowess across various evaluation criteria.

This comprehensive evaluation framework provided valuable insights into the model's strengths and limitations, enabling informed decisions regarding its suitability for real-world deployment. Additionally, it facilitated the identification of areas for further improvement, guiding future research efforts aimed at enhancing the robustness and efficacy of autonomous driving systems.

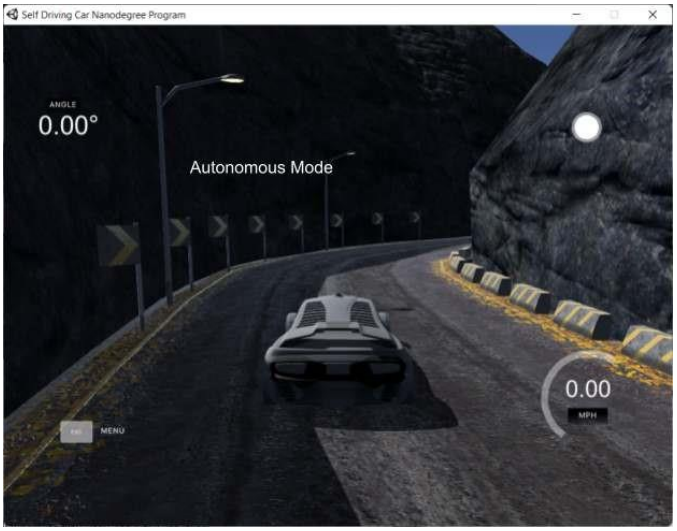

Fig - 3

III. BACKGROUND

CNN is a type of feed-forward neural network computing system that can be used to learn from input data. Learning is accomplished by determining a set of weights or filter values that allow the network to model the behavior according to the training data. The desired output and the output generated by CNN initialized with random weights will be different. This difference (generated error) is back propagated through the layers of CNN to adjust the weights of the neurons, which in turn reduces the error and allows us to produce output closer to the desired one .

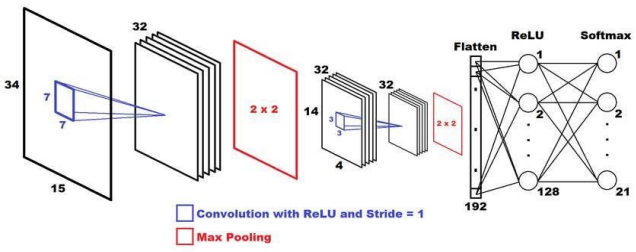

Fig – 4

CNN is good at capturing hierarchical and spatial data from images. It utilizes filters that look at regions of an

input image with a defined window size and map it to some output. It then slides the window by some defined stride to other regions, covering the whole image. Each convolution filter layer thus captures the properties of this input image hierarchically in a series of subsequent layers, capturing the details like lines in image, then shapes, then whole objects in later layers. CNN can be a good fit to feed the images of a dataset and classify them into their respective classes.

IV. SIMULATOR AND DATASET

Udacity Simulator

Udacity has provided the simulator for training the car in the specific path and the platform to run the car through our prepared model of autonomous driving . It is an open source for the software enthusiasts to work in this field so that it could be applied in a real-time environment. Unity, a video game creation platform, is used to create it. The simulator is highly user friendly and has a changeable resolution and controls configuration.

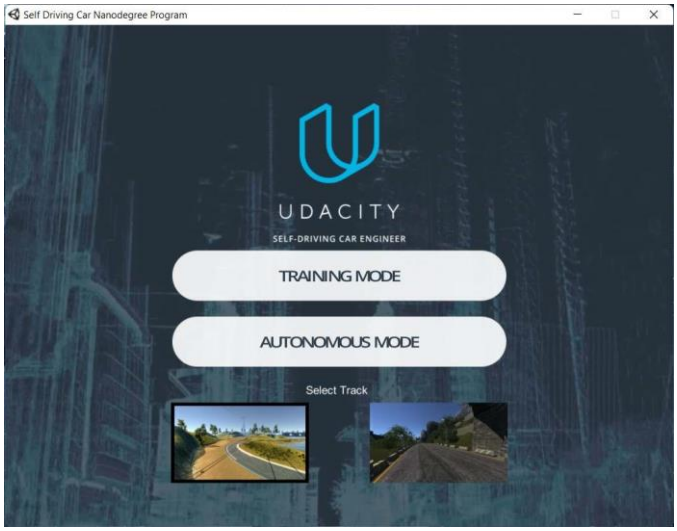

Fig –5

V. NETWORK ARCHITECTURE

Various Network architectures are used to determine the steering angle so that the car can be driven in the autonomous mode smoothly. In architectures, Time-Distributed Convolution layers, MaxPooling, Flatten, Dropout, Dense, and other layers were ordered in series, and various combinations of Time-Distributed Convolution layers, MaxPooling, Flatten, Dropout, Dense, and other layers were utilized. The best-performing ones are highlighted in further detail. The parameters required to create them may be found in the model listings. The architectural figures present a high-level view of the layers utilized to construct the models.

Model 1:

```
def nvidia_model():
  model = Sequential()
  model.add(Conv2D(24,(5,5), strides=(2, 2), input_shape=(66, 200, 3), activation='elu'))
  model.add(Conv2D(36, (5,5), strides=(2, 2), activation='elu'))
  model.add(Conv2D(48, (5,5), strides=(2, 2), activation='elu'))
  model.add(Conv2D(64, (3,3), activation='elu'))

  model.add(Conv2D(64, (3,3), activation='elu'))
  model.add(Dropout(0.5))
  model.add(Flatten())

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

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

  model.add(Dense(10, activation = 'elu'))

  model.add(Dense(1))

  optimizer = Adam(learning_rate=1e-4)
  model.compile(loss='mse', optimizer=optimizer)
  return model
model = nvidia_model()
print(model.summary())
```

Fig –6

Network architecture containing various layer is shown below:

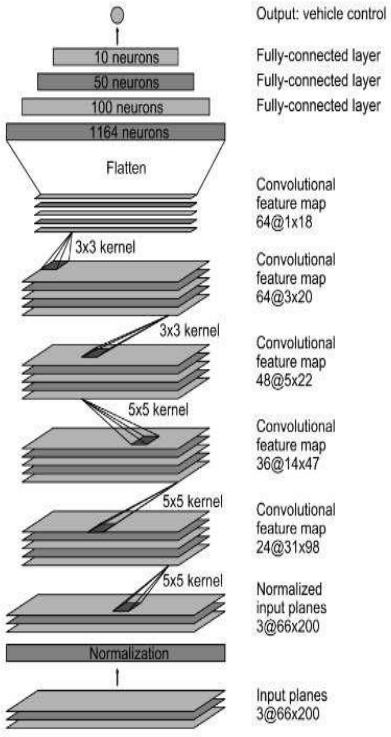

Fig - 7

VI. AUGMENTATION AND IMAGE PRE-PROCESSING

Image Preprocessing

In our preprocessing phase, we focused on enhancing the quality of captured images and extracting relevant features for lane detection. While these enhancements may not directly contribute to the training output, they play a crucial role in improving the robustness of our model. Here's an overview of the preprocessing steps:

1. **Gray Conversion (COLOR_RGB2GRAY)**:
- Convert color images to grayscale to simplify processing and reduce computational complexity.
2. **Blur Images (GaussianBlur)**:
- Apply Gaussian blur to smooth the images and reduce noise, resulting in a more uniform gradient and fewer abrupt changes.
3. **Canny Edge Detection**:
- Utilize the Canny edge detection algorithm to highlight changes in gradient pixels, providing a clear delineation of edges in the image.

Augmentation

The challenge of generalizing the car's behavior across diverse tracks necessitated the use of augmentation techniques during training. These techniques simulate various real-world conditions and enhance the model's adaptability. Here are the augmentation approaches employed:

1. **Zoom**:
- Adjust the image zoom level to simulate different perspectives and increase the model's ability to anticipate upcoming road conditions.
2. **Panning**:
- Crop a portion of the image containing relevant road features, ignoring extraneous information from the surrounding environment. Approximately 30% of the top section of the image is cropped during training.
3. **Brightness Adjustment**:
- Mimic varying lighting conditions by adjusting the brightness of images, allowing the model to learn to navigate through different weather conditions, such as bright sunny days or low-light environments.
4. **Flipping**:
- Flip images horizontally by 180 degrees to augment the dataset with mirrored representations of road features. This helps the model generalize better to road configurations involving left and right turns.

VII. OBSERVATIONS

The result that has been observed during the training phase from the architectural model we have used is value loss or accuracy.

Accuracy

After training the model with some of the dataset and passing other data for testing we have come up with the performance measure which reflects the loss over Epoch.

Epoch:

Epoch is an arbitrary cutoff used to divide training into different phases, which is important for recording and periodic assessment. It is commonly described as "one run over the complete dataset." When the fit technique of Keras models is used with validation data or validation split, evaluation is performed at the conclusion of each epoch.

```
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,
                             validation_steps=200,
                             verbose=1,
                             shuffle=1)
```

Fig – 8

```
import sys
Epoch 1/10
300/300 [==============================] - 429s 1s/step - loss: 0.0853 - val_loss: 0.0646
Epoch 2/10
300/300 [==============================] - 430s 1s/step - loss: 0.0667 - val_loss: 0.0582
Epoch 3/10
300/300 [==============================] - 429s 1s/step - loss: 0.0647 - val_loss: 0.0581
Epoch 4/10
300/300 [==============================] - 429s 1s/step - loss: 0.0617 - val_loss: 0.0549
Epoch 5/10
300/300 [==============================] - 427s 1s/step - loss: 0.0597 - val_loss: 0.0529
Epoch 6/10
300/300 [==============================] - 424s 1s/step - loss: 0.0560 - val_loss: 0.0501
Epoch 7/10
300/300 [==============================] - 423s 1s/step - loss: 0.0541 - val_loss: 0.0427
Epoch 8/10
300/300 [==============================] - 428s 1s/step - loss: 0.0525 - val_loss: 0.0407
Epoch 9/10
300/300 [==============================] - 429s 1s/step - loss: 0.0501 - val_loss: 0.0415
Epoch 10/10
300/300 [==============================] - 427s 1s/step - loss: 0.0485 - val_loss: 0.0373
```

Fig – 9

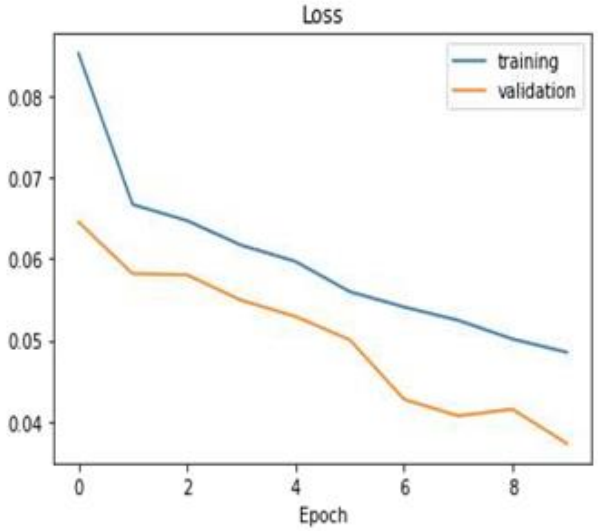

Fig – 10

VIII. CONNECTION

There is Configuration for the Connection between the simulator and the model we have trained earlier. The simulator side is called server whereas the user made a link with python code said to be client. Socket programming is the way where ports listen up and exchange the information between the nodes. python-socketio is the package that we have used to transport the telemetry information back and forth

```
@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__()
    })
```

Fig – 11

IX. RECEIVING THE TELEMETRY INFORMATION

The automatic communication mechanisms from many data sources are referred to as telemetry. Customer experiences are improved through the usage of telemetry data, which is used to track security, application health, quality, and performance.Areas may be regulated and controlled remotely via telemetry. Temperature, pressure, humidity, movement, and illumination, among other sensors, may all be controlled and monitored with it. Telemetry allows measurements to be taken without having to be present, regardless of mobility.

X. DRIVE PERFORMANCE

We evaluate the performance on the metrics of generalization , the working of car in track-2. This may be characterized as how effectively the models forecast values when driving on a track for which they were not trained. The expected steering angle, brakes, and throttle are the values here. It's not something that can be charted, but it can be measured in terms of how far the car can travel on the second track without tipping over. The pace, twists, and track circumstances such as hills, shadows, and so on can all have an impact.The models are only given training on Track 1 data because it is easier, but they are validated on Track 2. As a result, working with Track 2 was quite difficult. During the trial, a few new findings were made that support this claim.

- Most architectures were unable to take the first turn on Track 2 while performing well on Track 1 and providing the greatest accuracy during training (loss over epochs).
- Overfitting for the Track 1 dataset might be the cause. Overfitting occurs when a software attempts to mimic the training data too accurately. The model trains for details and even noise for the data it has passed through in general. This, in turn, has a detrimental influence on the capacity to generalize.

XI. CHANGE MADE TO WORK AROUND THE PROBLEM

- Gaussian filter and augmentation techniques are utilized to do something about this challenge.

XII. FUTURE DIRECTIONS

The current research lays a solid foundation for advancing autonomous driving technology, yet there are several avenues for future exploration and improvement. Here, we outline comprehensive directions for future research:

1. **Real-Time Deployment**:
- Transitioning the developed model from simulation to real-world application is paramount. Integrating the trained neural network into an actual self-driving vehicle and evaluating its performance in real-time driving scenarios will provide invaluable insights into its practical viability and effectiveness.

2. **Robustness Enhancement**:
- Enhancing the model's robustness to various uncertainties, including adverse weather conditions, changing road surfaces, and unexpected obstacles, is critical for ensuring safe and reliable autonomous driving across diverse environments. Investigating robust optimization techniques and sensor fusion strategies can fortify the model against these challenges.

3. **Dynamic Environment Adaptation**:
- Developing adaptive algorithms that enable the model to dynamically respond to changes in the driving environment, such as construction zones, temporary road closures, and pedestrian crossings, will bolster its versatility and responsiveness. This may involve incorporating reinforcement learning mechanisms or advanced planning algorithms capable of handling dynamic scenarios.

4. **Multi-Agent Interaction**:
- Exploring methods for enabling autonomous vehicles to interact intelligently with other agents, including vehicles, pedestrians, and cyclists, in shared spaces is essential for enhancing safety and efficiency in urban driving scenarios. Research in cooperative decision-making, negotiation strategies, and communication protocols can facilitate harmonious interactions among multiple agents on the road.

5. **Continual Learning Frameworks**:
- Implementing continual learning frameworks that enable the model to adapt and improve over time through exposure to new data and experiences is crucial for ensuring continual refinement and optimization of autonomous driving capabilities. This involves developing algorithms capable of incremental learning, knowledge transfer, and adaptation to evolving environments.

6. **Hardware Optimization**:

- Optimizing the hardware architecture and computational efficiency of onboard systems is imperative for meeting the computational demands of real-time processing and decision-making in autonomous vehicles. Research in efficient neural network architectures, hardware acceleration techniques, and energy-efficient computing platforms can facilitate scalable and cost-effective deployment of autonomous driving systems.

7. **Ethical and Regulatory Considerations**:

- Addressing ethical and regulatory challenges associated with autonomous driving, such as liability assignment, privacy protection, and safety standards compliance, necessitates interdisciplinary collaboration and policy development. Research in ethical decision-making frameworks, transparent accountability mechanisms, and regulatory frameworks for autonomous vehicles can ensure responsible deployment and societal acceptance.

By pursuing research in these comprehensive directions, we can continue to advance the field of autonomous driving and accelerate the realization of safe, efficient, and accessible transportation for all.

XIII. CONCLUSIONS

This study began with the models being trained and parameters being tweaked to achieve the greatest performance on the tracks, and then attempting to generalize that performance across multiple songs. Because the models that performed well on one track performed badly on Track 2, picture augmentation and processing were required to accomplish real-time generalization. Substituting recurrent layers for pooling layers might reduce the loss of information and would be worth exploring in the future projects It's fascinating to see how these models are trained using a mix of real-world and simulator data. Then I'll understand how a model may be taught in a simulator and then generalized to the actual world, or vice versa. There are several experimental implementations in the field of self-driving automobiles, and our project contributes to a large portion of them. In a nutshell, we became successful in predicting the steering angle using CNN.

XIV. REFERENCES

[1]. SAE International, "SAE J3016C (Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles)", 2021.

[2]. Lerner, J., &Tirole, J. (2003). Some Simple Economics of Open Source. The Journal of Industrial Economics, 50(2), 197-234.

[3]. Lerner, J., &Tirole, J. (2005). The Economics of Technology Sharing: Open Source and Beyond. Journal of Economic Perspectives, 19(2), 99-120.

[4]. von Krogh, G., &Spaeth, S. (2007). The open source software phenomenon: Characteristics that promote research. Journal of Strategic Information Systems, 16, 236-253.

[5]. Apollo. [Online]. Available: https://apollo.auto/index.html.

[6]. CARLA Simulator. [Online]. Available: https://carla.org/.

[7]. von Krogh, G., & von Hippel, E. (2006). The Promise of Research on Open Source Software. Management Science, 52(7).

[8]. YOLO: Real-Time Object Detection. [Online]. Available: https://pjreddie.com/darknet/yolo/.

[9]. Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In: NIPS.

[10]. LeCun, Y., Bengio, Y., & Hinton, G. E. (2015). Deep learning. Nature.

[11]. Zablocki, E., Ben-Younes, H., Perez, P., & Cord, M. (Year). Explainability of vision-based

autonomous driving systems: Review and challenges.

[12]. Ullman, S. (1980). Against direct perception. Basic Books.

[13]. Redmon, J., &Farhadi, A. (2017). YOLO9000: better faster stronger. In: CVPR.

[14]. Redmon, J., Divvala, S. K., Girshick, R. B., &Farhadi, A. (2016). You only look once: Unified real-time object detection. In: CVPR.