International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 08 Issue: 03 | March - 2024 SJIF Rating: 8.176 ISSN: 2582-3930

SELF DRIVING CAR SIMULATION USING CNN NVIDIA TENSORFLOW

Author1:
Pandrameesu Mahendra
B.Tech-CSE-AIML
Sphoorthy Engineering College
Hyderabad, Telangana
pandrameesumahendra@gmail.com

Author2:
A.Mohith
B.Tech-CSE-AIML
Sphoorthy Engineering College
Hyderabad, Telangana
mohithA123@gmail.com

Author3:
B.Sai Vinayak
B.Tech-CSE-AIML
Sphoorthy Engineering College
Hyderabad, Telangana
Saivinyak233@gmail.com

Author4:
Mr.Mohammed Ayazuddin
Asst.professor
Sphoorthy Engineering College

Abstract

Self-driving cars are moving from fiction to reality, but are we ready for it? This study analyses the current state of self-driving cars. Address gaps that need to be filled and identify problems that need to be solved before self- driving cars become a reality on the road. There are 4,444 technological advancements each year, paving the way for the integration of artificial intelligence into automobiles. In this paper, we provided a comprehensive study-on his implementation of self-driving cars based on deep learning. For your convenience and safety, simulate your car with the simulator provided by Udacity. Data for training the model is collected in the simulator and imported into the project to train the model. Finally, we implemented and compared various existing deep learning models and presented the results. Since the biggest challenge for self-driving cars is the autonomous lateral movement of the, the main goal of this white paper is to clone the drive using multilayer neural networks and deep learning techniques. and improve the performance of self-driving cars. Focuses on the realization of his self-driving cars driving under stimulus conditions. Within the simulator, the mimics the images obtained from the cameras installed in his car, the driver's vision, and the reaction that is her steering angle of the car. A neural network trains the deep learning technique based on photos taken by camera in manual mode. This provides the conditions for driving the car in automatic mode using a trained multilayer neural network. The driver imitation algorithm created and characterized in this paper is a deep learning technique centered around the NVIDIA CNN model.

I. Introduction

The rise of artificial intelligence (AI) has enabled 4,444 fully automated self-driving cars. In the near future, all technologies will be available for passengers to ride in self-driving cars. The vehicle takes her passenger from location A to location B, moving her fully automatically without passenger intervention. Technology is one piece of the puzzle that must address before he can put self-driving cars on the road. This paper analyzes current (federal and state) policy regarding autonomous vehicles and identifies gaps that need to be addressed by governments and legislators.

We also review the self-driving car safety features advertised by the vendor. Learn about self-driving car security from a cyberattack perspective. In addition, we analyze the psychological acceptance of self-driving cars by consumers. This whitepaper focuses on the obstacles that have prevented the realization of self- driving cars in his decade. Self-driving cars are the dream of many. The advent of his car in self-driving could mean the dawn of ultimate his driving safety for some, or his newly discovered mobility. For example, think of an elderly relative who

doesn't want anyone behind the wheel. A self-driving car can maintain his mobility and autonomy, but not endanger anyone. Or imagine someone who seems completely unaware that he drinks too much. Now I can see them driving safely and I no longer have to worry about the safety of myself or others who may be affected by poor judgment. Or imagine a disabled person who cannot drive a car today. Imagine the possibilities that will open up for them when self-driving cars become a reality. There are endless examples of how self-driving cars can help people and why they are needed, at least according to self-driving car proponents. His, who are against the idea that machines make the difference between life and death, are increasingly concerned as the auto industry moves closer to self-driving cars.

Fig (1.1) Self Running UI

This is an algorithm tuned for lane recognition in bright, sunny conditions, but the may not perform well in dark, dark conditions. This framework uses a convolutional neural system to provide control points that depend on road images. Essentially, the model is set up to copy human driving behavior. The end-to-end model is less sensitive to changing lighting conditions, as the standard may not actually be written. At the end of this paper, we describe in detail how self-driving cars are controlled. Develop a model of car motion and test the control design with the stimulator.

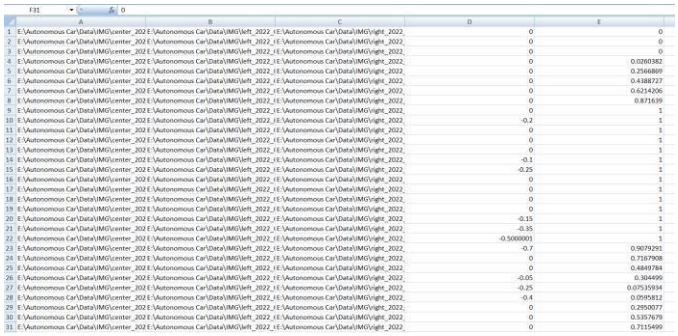
II. Literature Survey (Background study)

Convolutional Neural Networks (CNNs) have painted new perspectives in pattern recognition. Towards large-scale CNN deployments, many of the pattern recognition projects ended with early stages of their own components, which were extracted and tracked by classifiers. The function used acquired knowledge using the CNN growth training example. This method is mainly used in image recognition , as the Convolution activity captures the 2D nature of a given image. Recently, convolutional neural networks (CNNs) are very useful in image processing, and these types of neural networks have applications in various fields. For example, these are; Computer Vision, Face Recognition, Scene Labeling, Action Recognition, etc. CNN is a special architecture of the artificial neural network proposed by Jan Lekun in 1988 for the purpose of effective image recognition. A CNN consists of layers of different types. They are the convolution layer, the sub-discretization layer (subsampling, subsampling), and the fully connected layer of the neural network perceptron. CNNs are a very important part of various neural network architectures. These architectures are used to generate new images, segment images, etc. The CNN's learning algorithm was parallelized with graphics processing units to facilitate fast learning. The DARPA Autonomous Vehicle (DAVE) was used to identify end-to-end learning potential and validate the initiation of the DARPA Ground Robotic Learning (LAGR) program. However, DAVE's success has not been reliable enough to support a complete alternative to the more modular approach for off-road driving: the average distance between collisions was about 20 m. A new application has been launched at NVIDIA. Based on DAVE, we create a powerful system that can learn the entire task of circuit and path monitoring without requiring manual decomposition, labeling, semantic abstraction, path planning, and control. The agenda of this project is to circumvent his requirement to recognize certain characteristics. The transfer of training from common neural network architectures has led to research describing a similar approach to steering angle prediction.
CNN helps extract features from chassis. I would like

to create a network with fewer parameters and train it on artificially modeled data compared to the author of this article. This allows him to use the at efficient computational costs to rapidly create solutions for self-driving cars. Things such as: B. Lane marking, crash barriers, or other vehicles.

III. Methodology

a. DATA SET AND SIMULATOR



For the experiment, we used the Unity-based CarND.

Udacity Simulator - This simulator allows you to simulate his movement of the car in manual and automatic mode. The software is very flexible and allows map customization. We modeled about an hour drive in manual mode in various styles. I tried not to collide with the object and go off track, but you can model styles that are only implemented on special polygons in the real world. Recording and processing movements resulted in the formation of a log containing various modeling parameters. Here the main image is the image containing the simulated left, right, center camera and steering angles. A total of images of about 54000 were obtained. There is no doubt that driving a vehicle with a stimulator is not the same as actually driving a vehicle. There are so many similarities. Given the current state of game design, images

captured in the reproduced situation (roads, markings, scenes) is a good estimate of the images he could actually capture. The Stimulator provides additional safety and comfort. Collecting information is not a problem. Collect information efficiently. Also, even if the model fails, there is no danger to life. The stimulator is the best stage for exploring and improving different model structures. Later we can implement the model in a real car with a real camera, but for that we need a stimulator to work properly.

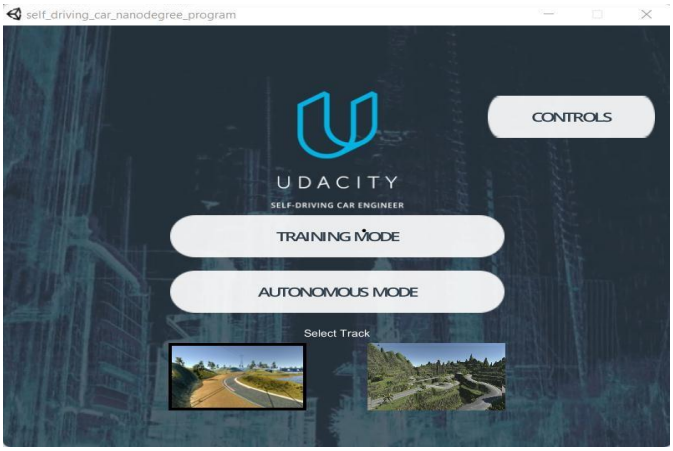
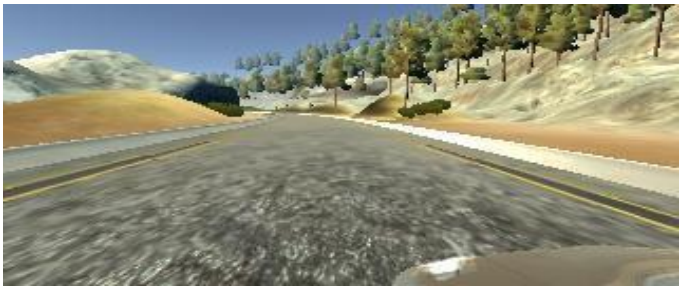


Fig (3.1) Udacity Car Simulator



The above are the steering angle images

Fig (3.2) Csv File (Data set)

DATA BALANCING

The original training dataset was biased toward either left or right-hand drive, as the ego-vehicle traversed the track in one direction. In addition, all collected data sets were heavily biased towards zero steering, as the steering angle was reset to zero each time the control button was released.

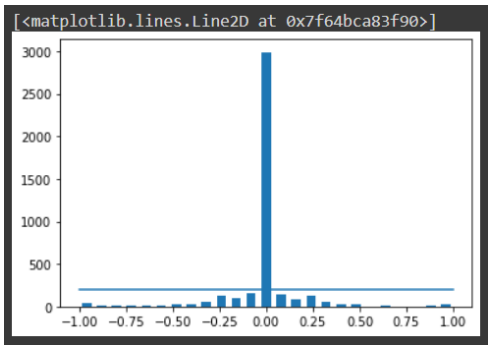


Fig (3.3) Graph for steering angles

To minimize these imbalances, we balanced the data set by removing random portions of the data set containing exactly zero steering angle measurements above a threshold of 400.

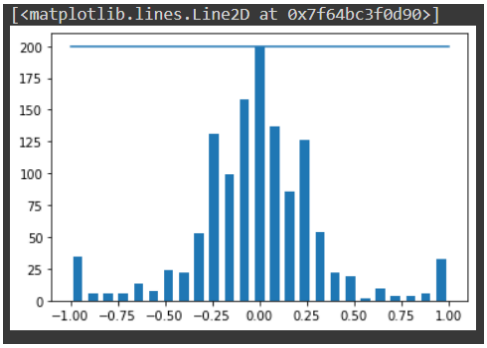


Fig (3.4) Graph for data balancing

b. DATAAUGMENTATION AND PREPROCESSING

Image preprocessing and augmentation are necessary to improve convergence, minimize overfitting, and speed up the training process. The image processing pipeline was implemented in Python using computer vision libraries. In this work, a total of five enhancement techniques were applied to the data set during the training phase: zoom, pan, brightness and flip.

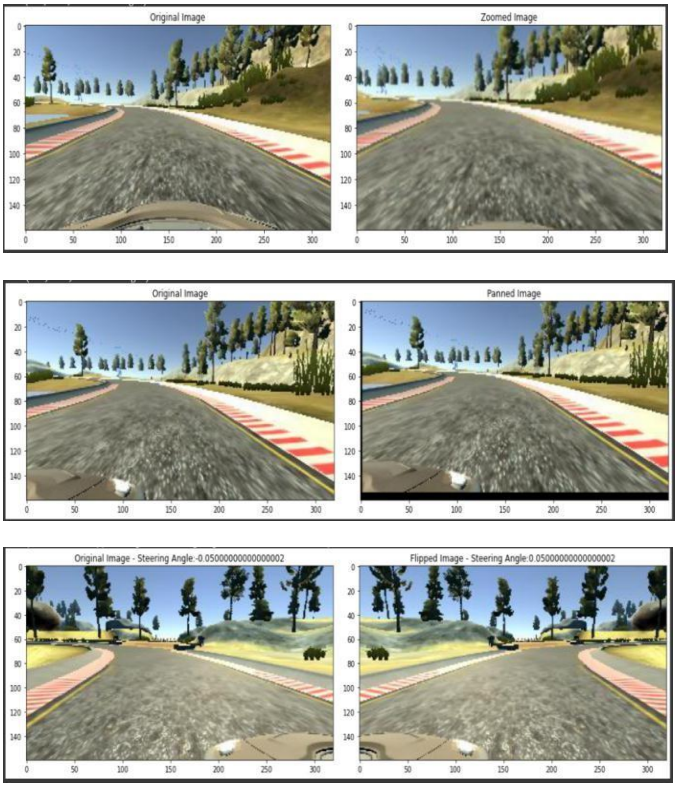


Fig (3.5) Data augmented images

Preprocessing of data for faster and more efficient training and deployment. This work presents a two-stage preprocessing function that performs resize and normalize operations (with moderate centering) on the input image. This is one of the important steps. Because we know there are many areas of the image that we don't use, or we can tell that we don't need to focus on it, so we removed the image. Features such as the Sky Mountain in the photo and the hood of the base vehicle. These parts have absolutely nothing to do with our car determining the steering angle of his. So I trimmed this with a simple NumPy array slice. As you can clearly see in the image, the image on axis is 160x300, the height from 160 to 135 is the car hood, and the range from 60 to 0 is the sky and landscape. So I removed that part from the photo and changed the axis from 60 to 135. This allows them to focus more on its essential functions. Convert the RGB image to YUV using the CV2 library. The advantage of converting images is that YUV images require less bandwidth than RGB. After transforming the image, I smooth it using the GaussianBlur method provided in the CV2 library.

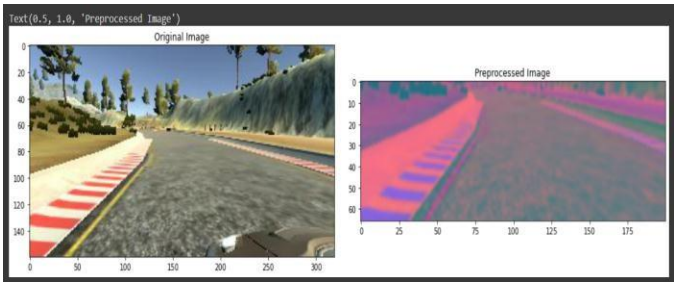


Fig (3.6) Preprocessed image

Crop the image according to the region of interest. Image above shows the original image and image next to it shows the output after the preprocessing phase has been performed.

c. NEURAL NETWORK ARCHITECTURE

The entire network is implemented and trained using Keras and Tensor Flow as backends. The neural network architecture I used is based on a paper published by NVIDIA and is designed to map raw pixels to control commands. The network consists of 9 layers, including a normalization layer, 5 convolution layers, and 3 fully connected layers. The image is normalized with a normalization layer. According to the paper, performing normalization on the network allows the normalization scheme to be tailored to the network architecture and accelerated by GPU processing.

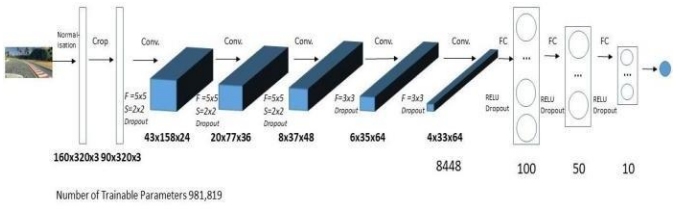


Fig (3.7) Neural Network Architecture

There are 5 levels of convolution used to extract features from images. The first three convolutions are performed with strided convolutions with 2x2 stride and 5x5 kernels. The last two layers of convolution are obtained by strideless convolution performed on a 3x3 kernel. Convolutional layers are followed by three fully connected layers to give output control values. The model is implemented in Keras with Tensorflow as the backend. The final model architecture is shown in the image below.

```
[ ]  model = nvidia_model()
     print(model.summary())

     Model: "sequential"
     _________________________________________________________________
      Layer (type)                 Output Shape              Param #
     =================================================================
      conv2d (Conv2D)              (None, 31, 98, 24)        1824

      conv2d_1 (Conv2D)            (None, 14, 47, 36)        21636

      conv2d_2 (Conv2D)            (None, 5, 22, 48)         43248

      conv2d_3 (Conv2D)            (None, 3, 20, 64)         27712

      conv2d_4 (Conv2D)            (None, 1, 18, 64)         36928

      dropout (Dropout)            (None, 1, 18, 64)         0

      flatten (Flatten)            (None, 1152)              0

      dense (Dense)                (None, 100)               115300

      dense_1 (Dense)              (None, 50)                5050

      dense_2 (Dense)              (None, 10)                510

      dense_3 (Dense)              (None, 1)                 11

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

Fig (3.8) Model Summary

Adam Optimizer was used to update network weights.

d. TRAINING AND VALIDATION

The collected dataset was randomly split into training and validation subsets with a 4:1 ratio (i.e. 80% training data and 20% validation data).The random state of partitioning for each dataset was specially chosen such that the training and validation datasets had minimal deviation from the steering measures.

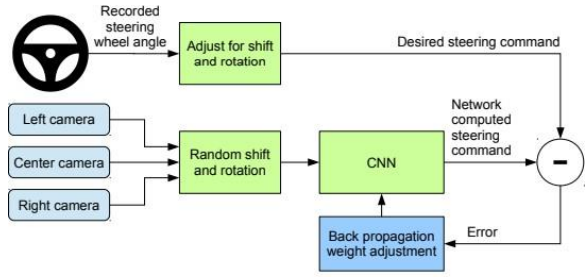


Figure 2: Training the neural network.

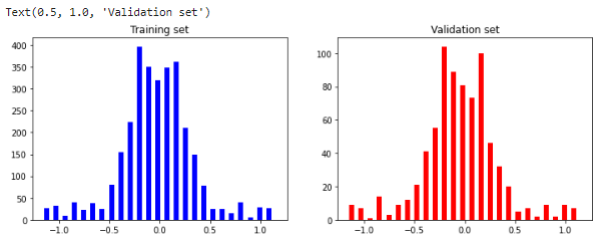


Fig (3.9) Graphs for training and data validation

e. UML DIAGRAMS

A UML diagram is the final result of the whole discussion. Create a complete UML diagram with all the elements and relationships, the diagram represents the system. The visual impact of UML diagrams is the most important part of the overall process. All other elements are used to complete it. UML is a standardized general-purpose modeling language in the field of object-oriented software engineering. Standards are managed and created by the Object Management Group. The goal of UML is to become a common language for modeling object-oriented computer software. UML in its current form consists of his two main components: the metamodel and the notation. Some form of method or process may also be added in the future. Or related, UML.

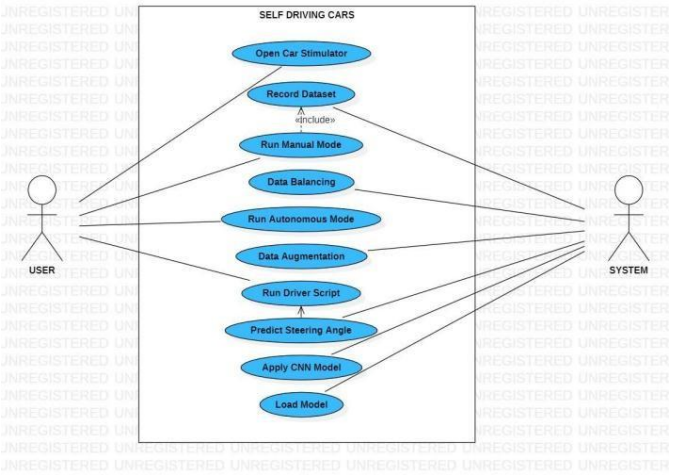


Fig (3.9) Use case diagram

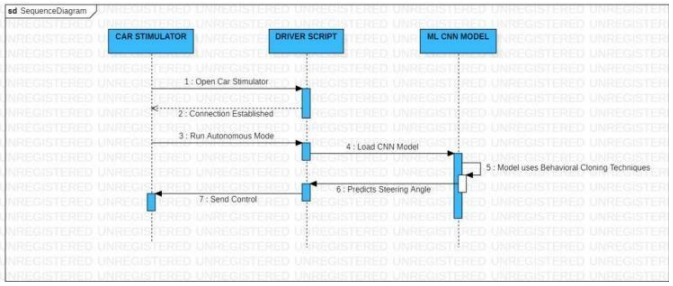


Fig (3.10) Sequence diagram

IV. Results

Simply put, the job of behavioral cloning is to collect information and use the collected data to train a model to mimic the behavior. Behavioral Cloning Leads to Simulated Driving Scenario. In behavioral cloning, we

stimulate a driving scenario implemented using CNN with an activation function that is Elu. The Elu function uses a mean squared error loss of used to validate the data. As expected, performed better after training the data, but the two values in the trained and tested data were very close, leading to the conclusion that the model represents a more general quality. Here is the result after applying the above method: The car drives that route and you can see that the right corner of is the speedometer. A trained model can successfully drive a car on a variety of unknown tracks. This model is able to consistently wrap without failure. Using a large training dataset consisting of different scenarios improves the ability of the model to stay in autonomous mode. Here is the result after applying the above method:

```
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')
```

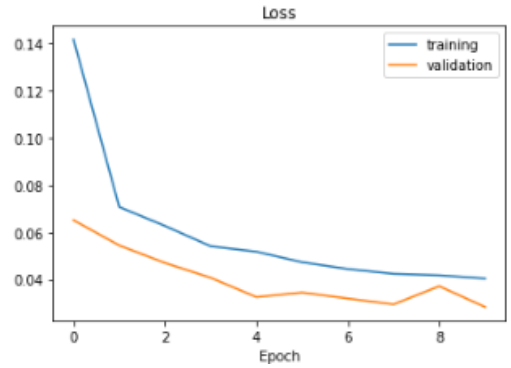


Fig (4.1) Graph for training and validation

The car drives that route and you can see that the right corner of is the speedometer. A trained model can successfully drive a car on a variety of unknown tracks.

This model is able to consistently wrap without failure. Using a large training dataset consisting of different scenarios improves the ability of the model to stay in autonomous mode.

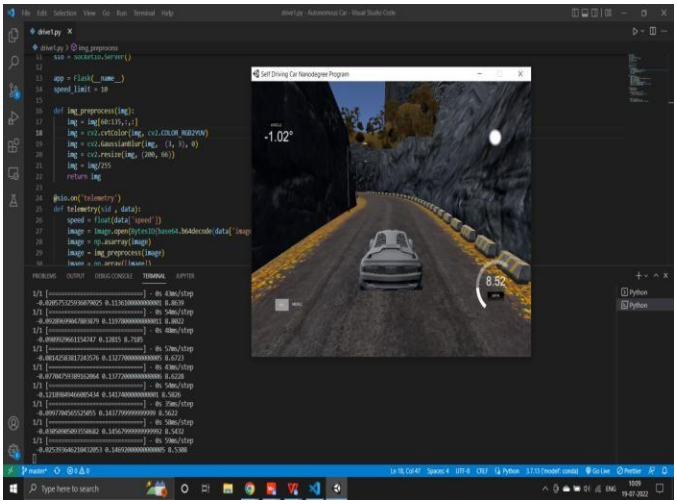


Fig (4.2) Running autonomously on a jungle track

V. Conclusion

Real-world automotive automation is a very large area with many sensors. Our idea is to resolve many of these sensors to his two sensors. This is to detect distance and objects such as stop signs, traffic lights and other obstacles in a single way for Monocular Vision. can be expanded. The prototype focuses on these features he developed on the Model RC car, while the other prototypes focus on only one aspect of it. This article describes how to generate data using the emulator. Data generation on the simulator has been shown to be very efficient. Easily create large datasets for experimentation. This avoids collecting data from the actual world. It is very resource intensive in terms of money and human resources. As an example, I was able to generate 54000 images in one hour without consuming additional resources. A neural network can be trained on synthetic images to predict steering angles for self-driving car motion. CNN can extract his data from camera images and find the dependencies needed for prediction. This article presents a comparison of the convolutional neural network architecture and a small set of parameters that can solve these problems. The resulting network had 3 layers of convolutions and a total of 26,000 parameters. It is possible to avoid overfitting, but had high variability because there was a small set of modeled data for. For example, we reduced the number of parameters and the memory used by neural networks by an order of magnitude compared to other popular architectures, results show that these networks yield high results, are modifiable and complex.

VI. References

[1] Udacity. Open Source Autonomous Vehicles, 2017.

[2] Y. Sun, X. Wang, X. seaweed. Sparsification of neural network connections for face recognition. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 4856-4864.

[3] Pichao Wang, Wanqing Li, Zhimin Gao, Chang Tang: Depth Pooling -Based Massive 3D Action Recognition and Convolutional Neural Networks. arXiv: 1804.01194, 2018.

[4] Y. LeCun B. Boser, J S. Denker, D. Henderson, R.E. Howard, W. Hubbard und L. D. Jackel: Back propagation Applied to Handwritten Zip Code Recognition, Neural Computation, 1(4):541-551, Winter 1989.

[5] He Huang, Phillip S. Yu: An Introduction to Image Synthesis with Generative Hostile Net, arXiv:1803.04469, 2018.

[6] A. Canziani, A. Pasque, E. Curcello An Analysis of Deep NeuralNetwork Models for Practical Applications 、 arXiv:1605.07678 [cs],Apr. 2017.

[7] C. Szegedy、S. Ioffe、V. Vanhoucke and A.A. Alemi. Inceptionv4, Effects of Inception Resnet and remaining connections on training. AAAI Conference on Artificial Intelligence, 2017, p. 4278-4284.