Simulation of Self Driving Car

Shraddha Manchekar, Bhargav Parsi, Nikhil Thakur, Kelly Bielaski

Computer Science Department, University of California, Los Angeles

bparsi@g.ucla.edu
nikhilt44@g.ucla.edu
smanchekar@ucla.edu
kellybielaski@ucla.edu

***Abstract*— For the past decade, there has been a surge of interest in self-driving cars. This is due to breakthroughs in the field of deep learning where deep neural networks are trained to perform tasks that typically require human intervention. CNN's apply models to identify patterns and features in images, making them useful in the field of Computer Vision. Examples of these are object detection, image classification, image captioning, etc. In this project, we have trained a CNN using images captured by a simulated car in order to drive the car autonomously. The CNN learns unique features from the images and generates steering predictions allowing the car to drive without a human. For testing purposes and preparing the dataset the Unity based simulator provided by Udacity was used.**

***Keywords*— autonomous driving, deep learning, Convolutional Neural Network (CNN), steering commands, NVIDIA, end-to-end learning, deep steering**

I. Introduction

In recent years, autonomous driving algorithms using low-cost vehicle-mounted cameras have attracted increasing research endeavours from both, academia and industry. Various levels of automation have been defined in autonomous driving. There's no automation in level 0. A human driver controls the vehicle. Level 1 and 2 are advanced driver assistance systems where a human driver still controls the system but a few features like brake, stability control, etc. are automated. Level 3 vehicles are autonomous, however, a human driver is still needed to monitor and intervene whenever necessary. Level 4 vehicles are fully autonomous but the automation is limited to the operational design domain of the vehicle i.e. it does not cover every driving scenario. Level 5 vehicles are expected to be fully autonomous and their performance should be equivalent to that of a human driver. We are very far from achieving level 5 autonomous vehicles in the near future. However, level - 3/4 autonomous vehicles are potentially becoming a reality in the near future. Primary reasons for drastic technical achievements in this fields are technical breakthroughs and excellent research being done in the field of computer vision and machine learning and also the low-cost vehicle-mounted cameras which can either independently provide actionable information or complement other sensors. Many vision-based drivers assist features have been widely supported in the modern vehicles. Some of these features include pedestrian/bicycle detection, collision avoidance by estimating the front car distance, lane departure warning, etc. However, in this project, we target autonomous steering, which is a relatively unexplored task in the field of computer vision and machine learning.

In this project, we implement a convolutional neural network (CNN) to map raw pixels from the captured images to the steering commands for a self-driving car. With minimum training data from the humans, the system learns to steer on the road, with or without the lane markings.

This report is organized as follows. Section II contains a brief overview of the relevant works developed in the past years. Section III and IV elaborate on the data collection and data preprocessing part of the project respectively. Section V explains the deep learning model that we used and Section VI describes the system architecture. System performance is evaluated in section VII where section VIII and IX discuss the future work and conclusion.

II. Related Work

The DAVE system was created by DARPA [1] and used images from two cameras as well as left and right steering commands to train a model to drive. It demonstrates that the technique of end-to-end learning can be applied to autonomous driving. This means that the intermediate features such as the stop signs and lane markings don't have to be annotated or labelled for the system to learn. DAVE is an early project in the field of autonomous driving. In the context of current technology, a huge portion relied on wireless data exchange because the vehicle couldn't carry the computers and power sources for the system, which contrasts the lighter equipment that exists today. The architecture of this model was a CNN made up of fully connected layers that stemmed from networks previously used in object recognition.

The ALVINN system [2] is a 3-layer back-propagation network built by a group at CMU to complete the task of lane-following. It trains on images from a camera and a distance measure from a laser range finder to output the direction the vehicle should move. ALVINN's model uses a single hidden layer back-propagation network.

We replicated a study by NVIDIA [3]. The system uses an end-to-end approach where the data is first collected in multiple different environmental conditions. The data is then augmented to make the system robust to driving off center and to different potential environments. The next step is training the network on this data. The network architecture is a total of 9 layers starting with convolutional layers and followed by fully-connected layers. This is the network that we attempted to replicate.

Recently, a paper by a couple of IEEE researchers introduced quite a different neural network architecture that also takes the temporal information into account [4]. They achieved this in practice by a combination of standard vector-based Long Short-Term Memory (LSTM) and convolutional LSTM at different layers of the proposed deep network. Consecutive frames usually have a similar visual appearance, but subtle per pixel motions can be observed when the optical flow is computed. Conventional image convolutions, as those adopted by state-of-the-art image classification models, can shift along both spatial dimensions in an image, which implies that they are essentially 2-D. Since these convolutions operate on static images or multi-channel response maps, they are incapable of capturing temporal dynamics in videos. The authors adopted a spatio temporal convolution (ST-Conv) that shifts in both spatial and temporal dimensions therefore applying the convolution in 3 dimensions dimensions as opposed to the traditional 2-D process.

A similar paper also proposed the idea to incorporate temporal information in the model to learn the steering information [5]. In this paper the authors demonstrate quantitatively that a Convolutional Long Short-Term Memory Recurrent Neural Networks (C-LSTM) can significantly improve end-to-end learning performance in autonomous vehicle steering based on camera images. Inspired by the adequacy of CNN in visual feature extraction and the efficiency of Long Short-Term Memory (LSTM) Recurrent Neural Networks in dealing with long-range temporal dependencies our approach allows to model dynamic temporal dependencies in the context of steering angle estimation based on camera input.

III. Data Collection

Fig. 1 Udacity Simulator

We used Udacity's self-driving car simulator for collecting the data. This simulator is built in Unity and was used by Udacity for the Self-Driving Nanodegree program but was recently open-sourced [6]. It replicates what NVIDIA did in the simulation. We can collected all our data from the simulator. Using our keyboard to drive the car, we were able to instruct the simulated vehicle to turn left, right, speed up and slow down. Another important aspect is that this simulator can be used for training as well as testing the model. Hence, it has two modes: (i) Training mode, and (ii) Autonomous mode as shown in Fig. 1.

The training mode is used to collect the data and the autonomous mode is used to test the model. Additionally, there are two types of tracks in the simulator - the lake track and the jungle track. The lake track is relatively smaller and easy to handle the car when compared with the jungle track as shown in Fig. 2 and Fig. 3. The simulator captures data when the car is driven around the track using left and right keys to control the steering angles and up and down arrows to control the speed.

Fig. 2. Udacity Simulator: The lake track

From this, the simulator generates a folder containing images and one CSV file. The image folder contains three images for every frame captured by the left, center and right camera and every row in the CSV file contains four metrics - steering angle, speed, throttle and brake, for every captured frame. Fig. 4, Fig. 5. and Fig. 6 show the left, center and right image, for one frame.

Fig. 3. Udacity Simulator: The jungle track

Fig. 4. Left image

Fig. 5. Center image

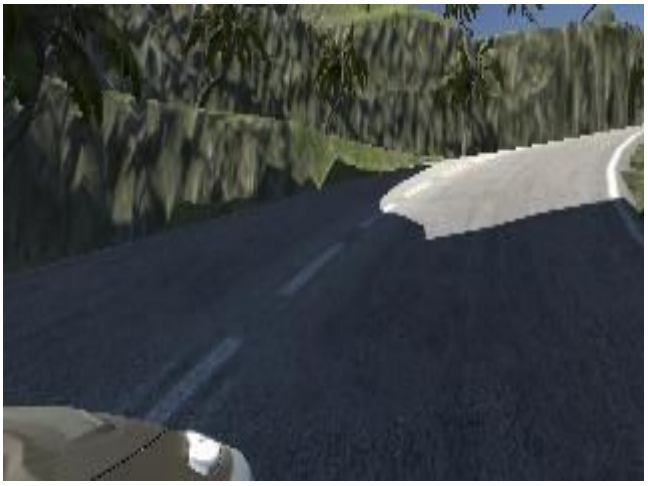

Fig. 6. Right image

IV. Data Preprocessing

The data that we collect i.e. the captured images are preprocessed before training the model. While preprocessing, the images are cropped to remove the sky and front portion of the car. The images are then converted from RGB to YUV and resized to the input shape used by the model. This is done because RGB is not the best mapping for visual perception. YUV color-spaces is a much more efficient coding and reduces the bandwidth more than RGB capture can.

After selecting the final set of frames, the data is augmented by adding artificial shifts and rotations to teach the network how to recover from a poor position or orientation. While augmenting, we randomly choose right, left or center images, randomly flip the images left/right and adjust the steering angle. The steering angle is adjusted by +0.2 for the left image and -0.2 for the right image. Using the left/right flipped images is useful to train the recovery driving scenario. We also randomly translate the image horizontally or vertically with steering angle adjustment. The horizontal translation can be useful for handling scenarios with difficult curves. The magnitude of these changes is chosen randomly from a normal distribution. The distribution has a zero mean. We applied these augmentations using a script from the Udacity repository.

Augmented images are added into the current set of images and their corresponding steering angles are also adjusted with respect to augmentation performed. The primary reason for this augmentation is to make the system more robust, thus, learning as much as possible from the environment by using diverse views with diverse settings.

V. Deep Learning Model

The deep learning model we prepared is based on the research done by NVIDIA for their autonomous vehicle [3]. The model comprises of the following important layers.

A. Convolutional Layer

Convolutional layer applies the convolution function (filter) on the input image and produces a 3D output activation of neurons. This layer helps to find various features of the image which is used for classification. The number of convolutional layers in the network depends on the type of application. The initial convolutional layers in the networks help detect the low level features of the image which are simple and the convolutional layers further help in detecting the high-level features of the image.

B. Max Pooling Layer

Pooling/Down sampling layer helps in reducing the number of parameters/weights in the network and helps in reducing the training time without losing any specific feature information of the image. This layer produces a smaller image than the input image by downsampling the image using pooling of neurons. There are different types of pooling like max pooling, average pooling, L2-norm pooling etc but max pooling is the one which is most widely used.

C. Dense Layer

Dense layer or Fully connected layer is the same as normal neural network layer in which all the neurons in this layer are connected to each neurons from previous layer. This layer is generally designed as the final layer in the convolutional

neural network.

The entire architecture diagram is shown in Fig. 7 There are 5 convolutional layers with varying number of filters and sizes and a dropout after that to handle overfitting. In the end, 3 dense layers were added followed by the output layer. Adam optimizer was used for parameter optimization with a fixed learning rate. Batch size of 100 was chosen and the number of epochs of 50-60 was experimented with. On a machine without GPU, 16 GB ram Core i5 it took roughly 6 hours of training.

VI. System Architecture

Fig. 8 shows a high-level architecture of the system. After performing data augmentation on the input images, batches are created from them and fed to the CNN model for training. After the training is completed, the model is used to perform prediction on the steering angle and send the predictions to the Udacity Simulator to drive the car in real time.

VII. Performance Evaluation

For performance evaluation during training, mean squared error was used as a loss function to keep a track of the performance of the model.

$$\text{MSE} = (1/n)\sum_{i=1}^{n}(y - y')^2$$

In real life scenarios, while driving on road, the following metric has been proposed in [3]. This metric has been named as autonomy. It is calculated by counting the number of interventions, multiplying by 6 seconds, dividing by the elapsed time of the simulated test, and then subtracting the result from 1.

$$\text{Autonomy} = \frac{1-(\textit{number of interventions}) * 6\ \textit{seconds}\ * 100}{\textit{elapsed time}\ [\textit{seconds}]}$$

To evaluate our system, we ran the autonomous aspect of the simulator on both the jungle and lake track. In the jungle track, the vehicle drove off the rode after 8 seconds of driving. By the above measure of autonomy, this would mean that the car was 87.5% autonomous. We re-trained the model with more data and the vehicle never drove off the path, making it 100% autonomous. The overall driving behavior doesn't appear realistic. It appears to bounce back and forth between the edges of the lane.

During the first run on the lake track, the vehicle doesn't leave the track, but it appears to hug the left side of the lane and that is the direction that the curve in the track is turning towards. The vehicle drives to the left side of the lane and when it runs close to the left lane marking, the vehicle drives back in the direction of the center of the lane until it is around halfway between the center of the lane and the left lane marking, then it drive back to the left side of the lane. This behavior repeats. After re-training the model, the vehicle's driving appears much more smooth. It doesn't hug the left side of the lane, but it appears to drive to one side of the lane and move towards the other. This behavior also repeats. In terms of autonomy, both models appear to be 100% autonomous.

After the second time we trained the model, we noticed that the vehicle can drive autonomously, but the behavior was not natural. This is something to keep in mind when refining the system.

VIII. Future Work

We have several ideas to improve the performance of the self driving car, out of which one was implemented due to lack of time. The first idea would be to add the feature of speed to the CNN so that when the simulator in autonomous mode, it is using the predicted speed making the movement appear to be more realistic.

The need for this element to be added came from observing the simulator driving in autonomous mode after training the car and noticing that after it accelerates to the maximum speed, the car will automatically slow down to the minimum speed making it appear unrealistic. (We actually did add this one)

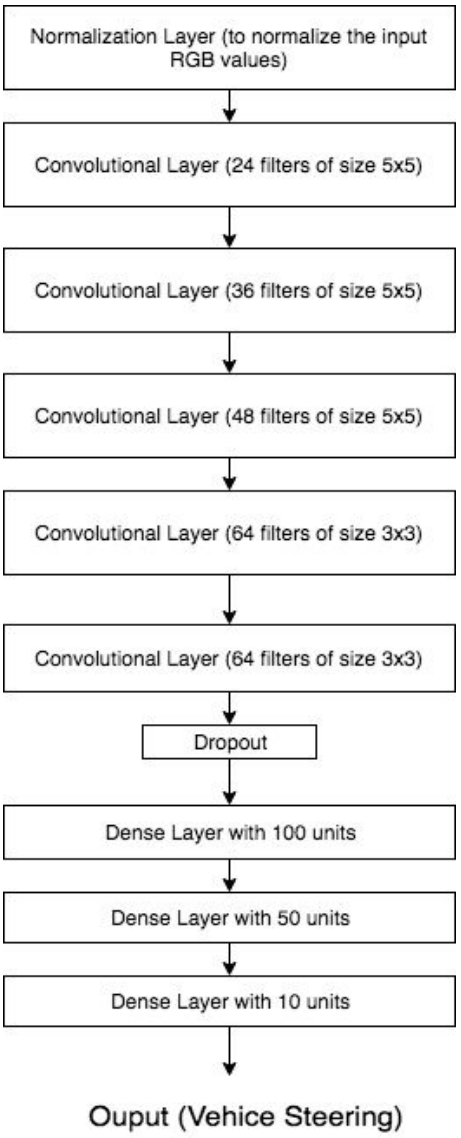

Fig. 7. Deep Network Architecture

Another possible improvement would be to consider each of the cameras separately and create CNN models using each stream of images to create a distinct steering command coming from the left, center, and right model. Then averaging the three of these to get a more accurate prediction. We would expect that the majority of the time, this model would have accurate predictions but if one model predicts a steering angle that is very unlike the other two, then it would skew the steering angle in an unexpected direction. Figure 9 is a graphical representation of this concept.

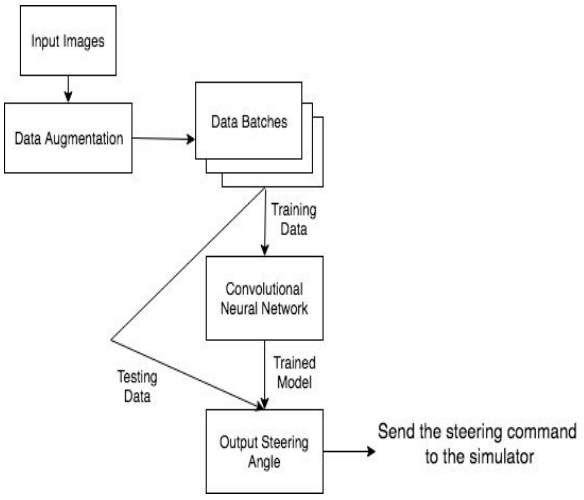

Fig. 8 High-level system architecture

In addition, we would like to add training data where the car recovers from being off the road. In the augmentation step of this process, the system learns how to recover from some small mishaps but never from larger deviations from the road. If the autonomous car is driving completely off the road it will not be able to recover. We want to add training data where the car starts from being off the road and makes its way back. This would require the data recording to be clipped starting when the car is off the road which would cut out the user driving the car off the road because we don't want the model to learn to drive the car off the road, but we do want it to learn driving onto the road.

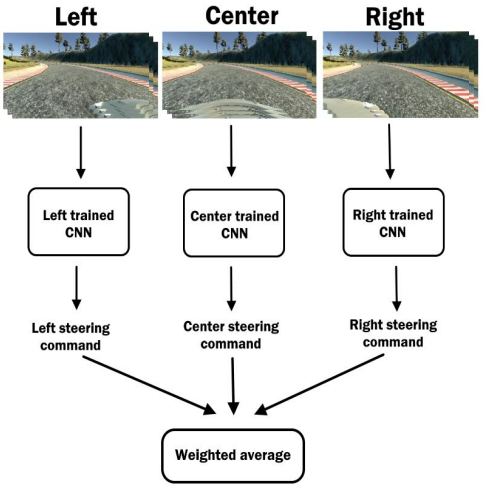

Fig. 9. 3 Model Prediction

Another idea to improve the accuracy of the steering would be to apply the deep steering architecture proposed in [2] to combine standard vector-based LSTM and convolutional LSTM to allow the model to use both spatial and temporal information to extract features.

An opportunity to evaluate the robustness of the system is to create another track other than the predefined jungle and lake track and see if the model trained would be able to drive the car on the track. Also, if it could drive relatively well, it would be interesting to observe the difference in how well it drives on the trained track versus the track that the model hadn't been trained on.

IX. Conclusions

In this project, we were able to successfully predict the steering angles using convolutional neural networks and were able to understand the inner details of convolutional neural networks along with the way they can be tuned. We also demonstrated that CNN's are able to learn the entire task of lane and road following without manual decomposition into road or lane marking detection, semantic abstraction, path planning, and control. A small amount of training data from less than a hundred hours of driving was sufficient to train the car to operate in diverse conditions, on highways, local and residential roads in sunny, cloudy, and rainy conditions. An interesting caveat to this is that the system was able to successfully drive on the roads that it had been trained on. Autonomous systems for vehicles that don't use the Udacity simulator require a greater robustness as they have to take into consideration roads that haven't been driven on and a greater amount of obstacles such as pedestrians and stop signs. However, the task defined in our paper was successfully accomplished.

X. Experimental Setup

The machine configuration for our experiments was as follows:

HARDWARE:

- RAM - 16 GB
- Operating System - OS X - 10.13.3
- Hard disk size - 1 TB

SOFTWARE:

- Python
- Unity 3D
- Keras (Tensorflow Backend)
- Anaconda
- Open CV

References

[1] LeCun, Y., et al. *DAVE: Autonomous off-road vehicle control using end-to-end learning*. Technical Report DARPA-IPTO Final Report, Courant Institute/CBLL, http://www. cs. nyu. edu/yann/research/dave/index. html, 2004.

[2] Pomerleau, Dean A. "Alvinn: An autonomous land vehicle in a neural network." *Advances in neural information processing systems*. 1989.

[3] Bojarski, Mariusz, et al. "End to end learning for self-driving cars." arXiv preprint arXiv:1604.07316 (2016).

[4] Chi, Lu, and Yadong Mu. "Deep Steering: Learning End-to-End Driving Model from Spatial and Temporal Visual Cues." *arXiv preprint arXiv:1708.03798* (2017).

[5] Eraqi, Hesham M., Mohamed N. Moustafa, and Jens Honer. "End-to-End Deep Learning for Steering Autonomous Vehicles Considering Temporal Dependencies." *arXiv preprint arXiv:1710.03804* (2017).

[6] https://github.com/udacity/self-driving-car-sim

[7] Deep learning for Video classification and captioning by Zuxuan Wu, Ting Yao, Yanwei Fu, Yu-Gang Jiang

[8] https://keras.io/backend

[9] https://github.com/llSourcell/How_to_simulate_a_self_driving_car

[10] https://github.com/naokishibuya/car-behavioral-cloning

[11] https://github.com/jeremy-shannon/CarND-Behavioral-Cloning-Project

[12] Visualizing and Understanding Convolutional Networks by Matthew D. Zeiler, Rob Fergus.

[13] Dropout: A Simple Way to Prevent Neural Networks from Overfitting by Nitish Srivastava, Geoffrey Hinton , Alex Krizhevsky, Ilya Sutskever, Ruslan Salakhutdinov.