Self-Driving Car

Himanshu Devatwal 210050065 | Ayan Minham Khan 210050028 | Islavath Nithin 210050071 | Adithya Rao 210050060 | Ravi Shankar Meena 210050135

[a]*AI ML Course Project*

Abstract

Self-driving cars has become a trending subject with a significant improvement in the technologies in the last decade. The project purpose is to train a neural network to drive an autonomous car agent on the tracks of Udacity's Car Simulator environment. Udacity has released the simulator as an open source software and enthusiasts have hosted a competition (challenge) to teach a car how to drive using only camera images and deep learning. Driving a car in an autonomous manner requires learning to control steering angle, throttle and brakes. Behavioral cloning technique is used to mimic human driving behavior in the training mode on the track. That means a dataset is generated in the simulator by user driven car in training mode, and the deep neural network model then drives the car in autonomous mode. Ultimately, the car was able to run on Jungle Track generalizing well. The project aims at reaching the same accuracy on real time data in the future.

1. Introduction

CNNs have revolutionized pattern recognition. Prior to the widespread adoption of CNNs, most pattern recognition tasks were performed using an initial stage of hand-crafted feature extration followed by a classifier. The breakthrough of CNNs is that features are learned automatically from training examples. The CNN approach is especially powerful in image recognition tasks because the convolution operation captures the 2D nature of images. Also, by using the convolution kernels to scan an entire image, relatively few parameters need to be learned compared to the total number of operations.

In this project, we describe a CNN that goes beyond pattern recognition. It learns the entire processing pipeline needed to steer an automobile.

2. Overview

Figure 1 shows a simplified block diagram of the collection system for training data for our system. Three cameras are mounted behind the windshield of the data-acquisition car. Time-stamped video from the cameras is captured simultaneously with the steering angle applied by the human driver. In order to make our system independent of the car geometry, we represent the steering command as 1/r where r is the turning radius in meters. We use 1/r instead of r to prevent a singularity when driving straight (the turning radius for driving straight is infinity). 1/r smoothly transitions through zero from left turns (negative values) to right turns (positive values).

Training data contains single images sampled from the video, paired with the corresponding steering command (1/r). Training with data from only the human driver is not sufficient. The network must learn how to recover from mistakes. Otherwise the car will slowly drift off the road. The training data is therefore augmented with additional images that show the car in different shifts from the center of the lane and rotations from the direction of the road.

Images for two specific off-center shifts can be obtained from the left and the right camera. Additional shifts between the cameras and all rotations are simulated by viewpoint transformation of the image from the nearest camera. Precise viewpoint transformation requires 3D scene knowledge which we don't have. We therefore approximate the transformation by assuming all points below the horizon are on flat ground and all points above the horizon are infinitely far away. This works fine for flat terrain but it introduces distortions for objects that stick above the ground, such as cars, poles, trees, and buildings. Fortunately these distortions don't pose a big problem for network training. The steering label for transformed images is adjusted to one that would steer the vehicle back to the desired location.

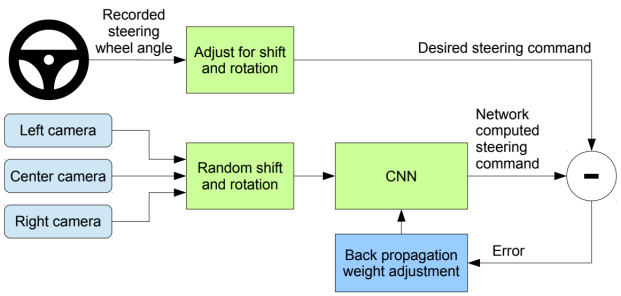

Figure 1: Training the neural network.

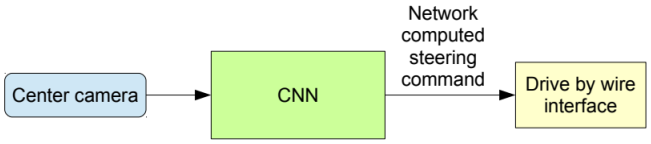

Figure 2: The trained network is used to generate steering commands from a single front-facing center camera.

3. Convolutional Neural Networks

Images are fed into a CNN which then computes a proposed steering command. The proposed command is compared to the desired command for that image and the weights of the CNN are adjusted to bring the CNN output closer to the desired output. The weight adjustment is accomplished using back propagation.

4. Data Collection

Udacity has built a simulator for self-driving cars [Bro17]. There are two modes for driving the car in the simulator: (1) Training mode and (2) Autonomous mode. The training mode gives you the option of recording your run and capturing the training dataset. The simulator's feature to create your own dataset of images makes it easy to work on the problem. Some reasons why this feature is useful are as follows:

- The simulator has built the driving features in such a way that it simulates that there are three cameras on the car. The three cameras are in the center, right and left on the front of the car, which captures continuously when we record in the training mode.

- The stream of images is captured, and we can set the location on the disk for saving the data after pushing the record button. The image set are labelled in a sophisticated manner with a prefix of center, left, or right indicating from which camera the image has been captured.

- Along with the image dataset, it also generates a data-log.csv file. This file contains the image paths with corresponding steering angle, throttle, brakes, and speed of the car at that instance.

We will specify samples we want to remove using looping construct through every single bin by iterating through all the steering data. We will shuffle the data and remove some from it as it is now uniformly structured after shuffling.The output will be the distribution of steering angle that are much more uniform. There are significant amount of left steering angle and right steering angle eliminating the bias to drive straight all the time. Then we will divide our data into training set and validation set.

5. Data Augmentation and Pre-processing

We augment the data by adding artificial shifts and rotations to teach the network how to recover from a poor position or orientation. The magnitude of these perturbations is chosen randomly from a normal distribution. The distribution has zero mean, and the standard deviation is twice the standard deviation that we measured with human drivers. Artificially augmenting the data does add undesirable artifacts as the magnitude increases.

We continued by doing some image processing. We cropped the image to remove the unnecessary features, changed the images to YUV format, used gaussian blur, decreased the size for easier processing and normalized the values.

6. Experimental Configurations

The tweaking of parameters and rigorous experiments were tried to reach the best combination. Though each of the models had their unique behaviors and differed in their performance with each tweak, the following combination of configuration can be considered as the optimal:

- The sequential models built on Keras with deep neural network layers are used to train the data.

- 80% of the dataset is used for training, 20% is used for testing.

- Epochs = 10, i.e. number of iterations or passes through the complete dataset. Experimented with larger number of epochs also, but the model tried to "overfit". In other words, the model learns the details in the training data too well, while impacting the performance on new dataset.

- Batch-size = 100, i.e. number of image samples propagated through the network, like a subset of data as complete dataset is too big to be passed all at once.

- Learning rate = 0.0001, i.e. how the coefficients of the weights or gradients change in the network.

There are different combinations of Convolution layer, Time-Distributed layer, MaxPooling layer, Flatten, Dropout, Dense and so on, that can be used to implement the Neural Network models.

7. Network Architecture of Final Model

The design of the network is based on the NVIDIA model, which has been used by NVIDIA for the end-to-end self driving test. As such, it is well suited for the project.It is a deep convolution network which works well with supervised image classification / regression problems. As the NVIDIA model is well documented, I was able to focus how to adjust the training images to produce the best result with some adjustments to the model to avoid overfitting and adding non-linearity to improve the prediction.

7.1. NVIDIA Model

We have used end-to-end learning system developed by NVIDIA , DAVE-2 [B+16], with some modifications to suit our purpose. NVIDIA Model Architecture is shown in Figure 3.

We train the weights of our network to minimize the mean squared error between the steering command output by the network and the command of either the human driver, or the adjusted steering command for off-center and rotated images. The network consists of 9 layers, including a normalization layer, 5 convolutional layers and 3 fully connected layers. The input image is split into YUV planes and passed to the network.

The first layer of the network performs image normalization. The normalizer is hard-coded and is not adjusted in the learning process. Performing normalization in the network allows the normalization scheme to be altered with the network architecture and to be accelerated via GPU processing.

The convolutional layers were designed to perform feature extraction and were chosen empirically through a series of experiments that varied layer configurations. We use strided convolutions in the first three convolutional layers with a 2×2 stride and a 5×5 kernel and a non-strided convolution with a 3×3 kernel size in the last two convolutional layers.

We follow the five convolutional layers with three fully connected layers leading to an output control value which is the inverse turning radius. The fully connected layers are designed to function as a controller for steering, but we note that by training the system end-to-end, it is not possible to make a clean break between which parts of the network function primarily as feature extractor and which serve as controller.

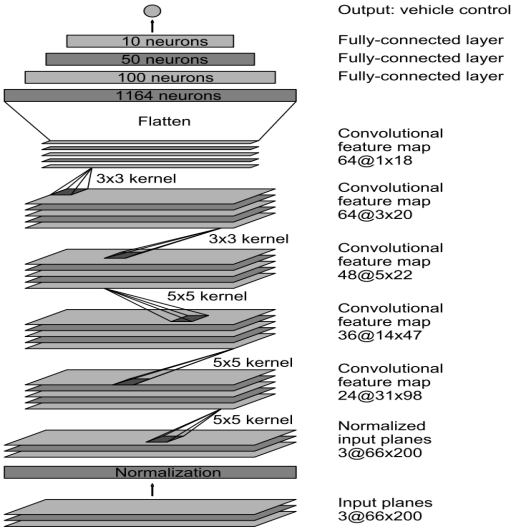

Figure 3: CNN architecture. The network has about 27 million connections and 250 thousand parameters.

7.2. Our Model

Our model, utilizing behavioral cloning, deals with a dataset of approximately 90,000 images. The architecture involves convolutional layers with specific configuration, a 5x5 kernel, and subsampling. Subsequent layers follow similar patterns, with adjustments to filters, kernels, and subsampling as per the NVIDIA model. The model concludes with a dense layer containing a single output node for predicting the steering angle.

The model compilation involves specifying mean squared error as the metric and Adam optimization with a low learning rate. Dropout layers are incorporated to prevent overfitting. The convolutional and fully connected layers are separated by a dropout factor of 0.5. The training process is defined using model.fit(), incorporating training data, validation data, and batch size. The architecture and training parameters align with the NVIDIA model while addressing the specific requirements of required model.

7.2.1. Why ELU?

We can have dead relu this is when a node in neural network essentially dies and only feeds a value of zero to nodes which follows it. We will change from relu to elu. Elu function has always a chance to recover and fix it errors means it is in a process of learning and contributing to the model.

```
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

 flatten (Flatten)           (None, 1152)              0

 dense (Dense)               (None, 100)               115300

 dropout (Dropout)           (None, 100)               0

 dense_1 (Dense)             (None, 50)                5050

 dropout_1 (Dropout)         (None, 50)                0

 dense_2 (Dense)             (None, 10)                510

 dropout_2 (Dropout)         (None, 10)                0

 dense_3 (Dense)             (None, 1)                 11

=================================================================
Total params: 252219 (985.23 KB)
Trainable params: 252219 (985.23 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
```

Figure 4: Modified Model

7.3. Results

The following results were observed for described architectures. We had to come up with two different performance metrics.

- Value loss or Accuracy (computed during training phase)
- Generalizing of Track (Drive Performance)

7.3.1. *Value loss or Accuracy*

The first evaluation parameter considered here is "Loss" over each epoch of the training run. To calculate value loss over each epoch, Keras provides "val_loss", which is the average loss after that epoch. The loss observed during the initial epochs at the beginning of training phase is high, but it falls gradually.

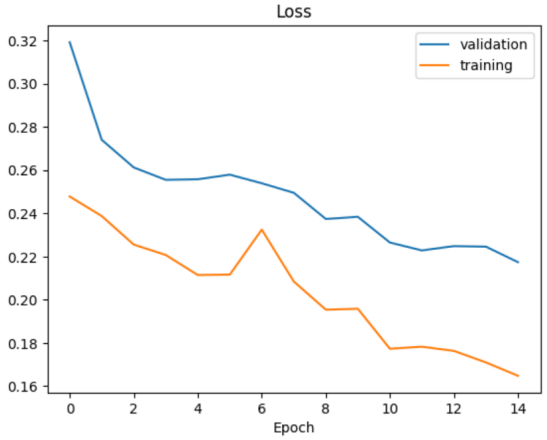

Figure 5: Loss vs Number of Epochs for nvidia model

7.3.2. *Generalizing of Track*

We can train our model on dataset created by Track 1 of Udacity Simulator and test our model on Track 2 in autonomous mode. This part will be further explained in Simulation section.

7.4. *MobileNetV2 Model*

We took the mobilenetv2 [San+19] model which works well for the regression model problems. The basic mobilenetmodelv2 is explained in the following diagram.

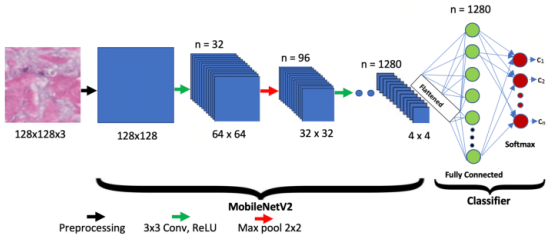

Figure 6: Default mobilenetv2 architecture

We modified the model such that it fits the purpose of behavioural clonning appropriately. The GlobalAveragePooling2D layer performs global average pooling on the spatial dimensions of the input. It reduced each spatial dimension (height and width) to 1 by taking the average over all values in that dimension. This is used to reduce the spatial dimensions before feeding the data into a dense layer.

The Dense layer is a fully connected layer with 100 neurons. Here ELU is used as the activation function. This helped the network learn robust representations.

Dropout layer is used as regularization technique that randomly sets a fraction of input units to zero during training, which helps prevent overfitting by introducing noise and promoting the learning of more robust features.

Finally a Dense layer is added which outputs a real value. This is the predicted angle the car should take at that instant.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
mobilenetv2_1.00_224 (Func   (None, 3, 7, 1280)        2257984
tional)

global_average_pooling2d_1   (None, 1280)              0
 (GlobalAveragePooling2D)

dense_8 (Dense)              (None, 100)               128100

dropout_6 (Dropout)          (None, 100)               0

dense_9 (Dense)              (None, 50)                5050

dropout_7 (Dropout)          (None, 50)                0

dense_10 (Dense)             (None, 10)                510

dropout_8 (Dropout)          (None, 10)                0

dense_11 (Dense)             (None, 1)                 11

=================================================================
Total params: 2391655 (9.12 MB)
Trainable params: 133671 (522.15 KB)
Non-trainable params: 2257984 (8.61 MB)
_________________________________________________________________
```

Figure 7: Modified mobilenetv2 architecture

The architecture consists of several dense layers with ELU activations, interspersed with dropout layers for regularization. The model compilation involves specifying mean squared error as the metric and Adam optimization with a low learning rate. Dropout layers are incorporated to prevent overfitting. The convolutional and fully connected layers are separated by a dropout factor of 0.5.

7.4.1. *Value loss or Accuracy*

To calculate value loss over each epoch, we used Keras "val_loss", which is the average loss after that epoch. The loss observed during the initial epochs at the beginning of training phase is high, but it falls gradually. For the model to not overfit we used 10 epochs.

8. Simulation

We created our dataset by running vehicle in training mode of Udacity Simulator. Since human drivers might not be driving in the center of the lane all the time, we manually calibrate the lane center associated with each frame in the video used by the simulator. We call this position the "ground truth". The simulator transforms the original images to account for departures from the ground truth. Note that this transformation also includes any discrepancy between the human driven path and the ground truth. The simulator accesses the recorded test video along with the synchronized steering commands that occurred when the video was captured.

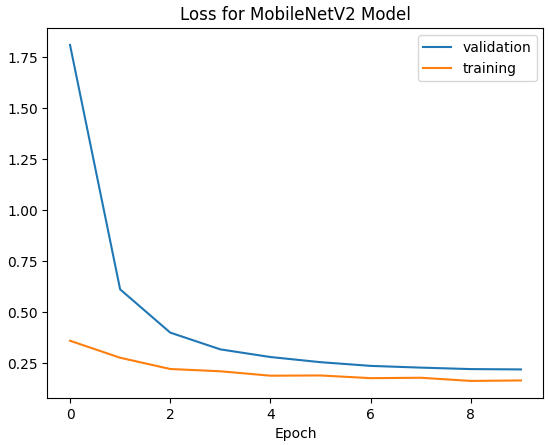

Figure 8: Loss vs Number of Epochs for mobilenetv2 model

The simulator sends the first frame of the chosen test video, adjusted for any departures from the ground truth, to the input of the **trained CNN**. The CNN then returns a steering command for that frame. The CNN steering commands as well as the recorded human-driver commands are fed into the dynamic model of the vehicle to update the position and orientation of the simulated vehicle.

The simulator then modifies the next frame in the test video so that the image appears as if the vehicle were at the position that resulted by following steering commands from the CNN. This new image is then fed to the CNN and the process repeats.

The simulator records the off-center angle (angle from the car to the lane center), the yaw, and the distance traveled by the virtual car. When the off-center angle exceeds chosen angle, a virtual human intervention is triggered, and the virtual vehicle position and orientation is reset to match the ground truth of the corresponding frame of the original test video.

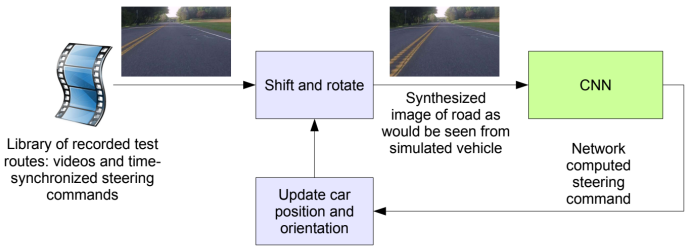

Figure 9: Block-diagram of the drive simulator.

9. Conclusion

We have empirically demonstrated that CNNs 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.

Figure 10: Finally our model is driving car in autonomous mode of Udacity Simulator

A small amount of training data from less than two hours of driving on one track of simulator was sufficient to train the car to operate in diverse conditions, on highways, local and residential roads in sunny, cloudy, and rainy conditions of other tracks. The CNN is able to learn meaningful road features from a very sparse training signal (steering alone).

The system learns for example to detect the outline of a road without the need of explicit labels during training.

The use of CNN for getting the spatial features and RNN for the temporal features in the image dataset makes this combination a great fit for building fast and lesser computation required neural networks. Substituting recurrent layers for pooling layers might reduce the loss of information and would be worth exploring in the future projects.

It is interesting to find the use of combinations of real world dataset and simulator data to train these models. Then We can get the true nature of how a model can be trained in the simulator and generalized to the real world or vice versa. There are many experimental implementations carried out in the field of self-driving cars.

Appendix A. Custom Model

We created a Custom model which performed subpar when compared to Nvidia Model. It has following layers

- The first convolutional layer has 6 filters of size (5, 5) with a Rectified Linear Unit (ReLU) activation function. This layer extracts basic features from the input images. The second convolutional layer has 12 filters of size (5, 5) with ReLU activation, capturing more complex patterns.
- The flatten layer is used to convert the output from the convolutional layers into a flat vector, preparing it for the fully connected layers.
- The first fully connected layer has 120 neurons with ReLU activation, capturing high-level features.The first fully connected layer has 120 neurons with ReLU activation, capturing high-level features.
- Dropout layers with dropout rate of 0.5 are added in between every two layers to avoid over-fitting.

Finally the validation loss for this Model is worst than other two models specified above.

Appendix B. Result Comparison of Models

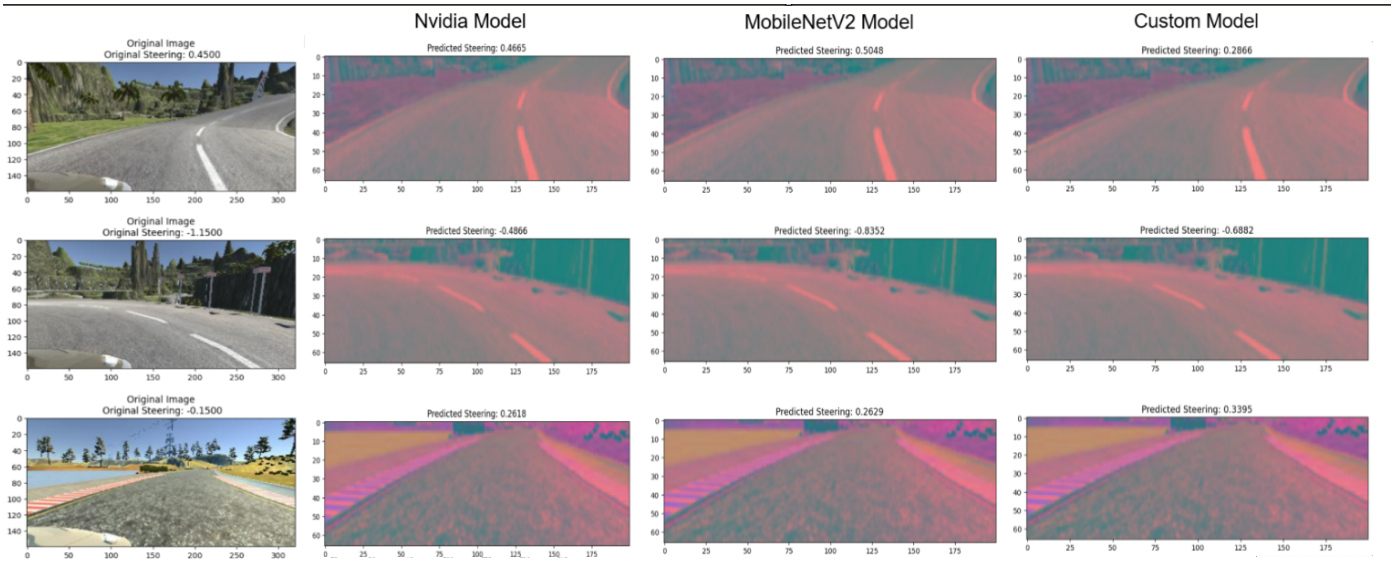

Figure B.11: Comparison between predicted steering angles of three images from dataset

The models try to imitate the behaviour the trainer, the discrepancies are due to model-error and sometimes are also due to human-error of the trainer as we can see the steering given by model seems more accurate than original. As we can see the DAVE-2 model and mobilenetv2 model outputs are closer to each other and are also close to expected output.

References

[B+16] Mariusz B et al. *End-to-End Deep Learning for Self-Driving Cars*. 2016. URL: `https://arxiv.org/pdf/1604.07316v1.pdf`.

[Bro17] Aaron Brown. *self-driving-car-sim*. 2017. URL: `https://github.com/udacity/self-driving-car-sim`.

[San+19] Mark Sandler et al. *MobileNetV2: Inverted Residuals and Linear Bottlenecks*. 2019. URL: `https://doi.org/10.48550/arXiv.1801.04381`.