Building a Convolutional Neural Network to predict the Steering Angle for an Autonomous Car

Mehul Kohli
Smrithi Ajit
Brianna Lawton

Self-Driving Car Group 1

May 9, 2019

Final Project Report for
ME 592X

Iowa State University
Spring 2019

ABSTRACT

The authors built Convolutional Neural Network models trained with images collected from the camera mounted on the self-driving car to predict it's steering angle. Predicting the steering angle and applying correction at each step through a feedback loop thus helping to correct the motion of the car to keep it on track is a very important attribute of self-driving cars. The idea of behavioral mapping or imitation learning was applied to the problem. A prototype model with the architecture described in (*1*) was trained and tested on Udacity driving simulator data in the proof of concept phase. An experimental model was built to take the inputs from the center, left and right cameras separately through separate networks and then directed to the output. The performance of this model was compared to the model architecture proposed in the Nvidia research (*1*) and was found to have comparable efficiency. The model was tested for efficiency by using the model to run the Udacity simulator car in autonomous mode and checking the number of human interventions that may be required to keep the vehicle on track per run. In the second phase two sets of models were built to predict steering angle for real world data (*2*). In this phase the authors compared the performance of a hybrid model that uses a combination of a pre-trained model VGG16 with additional layers from their own model with their own model and base model having only VGG16.Though, for the entire dataset the model proposed by the authors performed better that the counterparts it was observed that when the data was halved the performance of the hybrid model was better suggesting that the hybrid model had better efficiency if the data was limited.

Keywords: self-driving cars, predict steering angle, Udacity Simulator,VGG16,hybrid model

1 Introduction

As we dawn into an era of Artificial Intelligence, the biggest research innovation in this area is the Self driving car. The easy availability of data collected during driving from humans makes imitation learning a promising approach towards training self-driving cars to engage in a self - correcting mechanism to stay on the road and within the lane of interest. Convolutional neural networks are one of the simplest deep learning approaches that have been used to realize imitation learning, the reason being that in CNNs the features are learnt automatically from training samples, a significant advantage from their older counterparts. Since CNNs use convolutional kernels they need to learn fewer parameters(*3*). Research efforts have been focused primarily on training the car to best emulate the human driver by imitating the human driver behavior at every point of time, yet removing any inconsistencies and aggressiveness that may be typical to a human driver. In this context, several authors have combined the concepts of imitation learning or behavioral mapping with deep learning to come up with different training models that work in coordination with control units to make sure that the self driving car stays on the road and understands what correction needs to be applied to the steering angle to bring it back on track if it should go off. The same idea was used by the authors of the project to work on similar lines.

2 Related Work

Bojarski et.al(*1*) developed a convolutional neural network to map the images from front facing camera to steering commands. The DAVE-2(DARPA Autonomous Vehicle) system was trained using data obtained from three cameras mounted on the data-acquisition car and given below in Fig 1 is a block schematic of the data collection and corrective feedback system of DAVE-2 system. The images from the left, right and center camera fed to the Convolutional neural network. The feedback mechanism compares the system predicted steering angle with the desired steering angle to calculate the error which is fed back to the Convolutional Neural Network helping it adjust it's training weights through back propagation to reduce the error between predicted and actual output. The model used in the proof of concept phase by the authors is that suggested by Bojarski et.al.

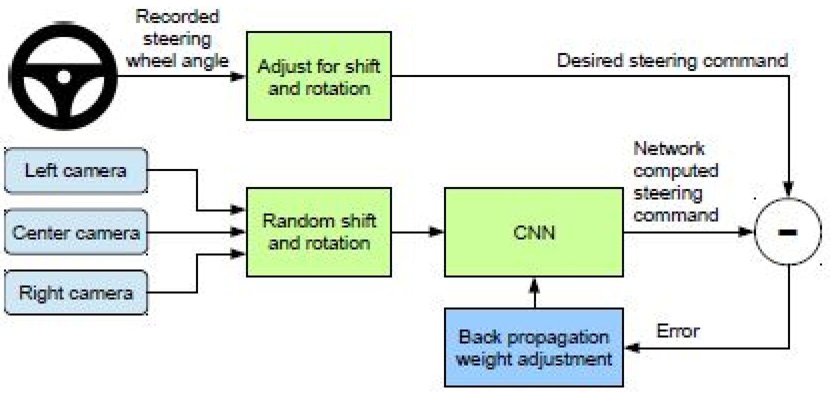

Recorded steering wheel angle
Adjust for shift and rotation
Desired steering command
Left camera
Center camera
Right camera
Random shift and rotation
CNN
Network computed steering command
-
Back propagation weight adjustment
Error

Fig 1: Schematic describing working of DAVE-2

Felipe Codevilla et.al(*3*) also utilized the concept of imitation learning but with the inclusion of a representation of the expert's opinion. Shuyang Du et.al(*4*) utilized the concept of transfer learning that is based off the idea that the lower layers of models trained on very large datasets like Imagenet can be used for other datasets as they extract features well. Shuyang Du et.al blocked the weights of the lower 15 layers of the ResNet50 from updating and the output of ResNet50 was connected to a stack of fully connected layers containing 512, 256, 64, and then the output layer.

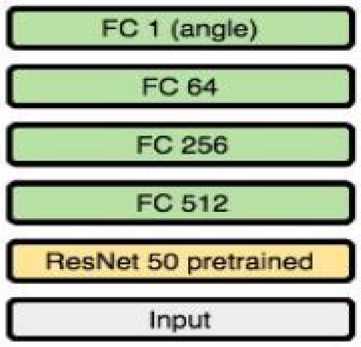
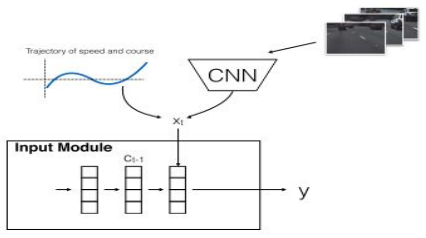


Fig 2: Model proposed by Shuyang Du et.al

The authors took inspiration from this idea to build of the hybrid model that uses the pre-trained model VGG16 after removing the last layer and preventing it's remaining layers from updating weights and adding trainable layers on them.

Huazhe Xu(*5*) and Fernando et.al (*6*) worked on similar lines with visual input encoded with a CNN and while the former merged the CNN output with the speed and angular velocity trajectory and passed it through an LSTM layer the latter encoded the visual input and the steering wheel angle trajectory and passed through separate LSTM models so that their long term history is mapped via two separate external memories and merged current LSTM outputs with memory outputs.



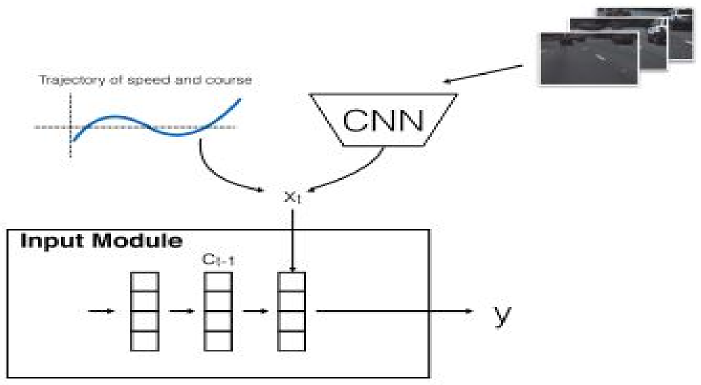


Fig3:(left) Architecture proposed by Huazhe Xu et.al and (right) Architecture proposed by Fernando et.al

3 Experimental Platform

Models were trained on a laptop with the following specs:
GPU1 Specs:

- Nvidia GeForce GTX 1050
- Cuda Cores: 640
- Graphics Clock: 1354 MHz
- Total available graphics memory: 8134 MB
- Shared system memory: 4038 MB

GPU2 Specs:

- Nvidia GeForce GTX 1080Ti
- Cuda Cores: 3584
- Graphics Clock: 1582 MHz
- Total available graphics memory: 11GB
- Shared system memory: 11GB

4 Methodology

The project was carried out in two phases namely: the Proof of Concept Phase where the authors used Udacity Simulator data and the main phase where the authors used real data. The model development included splitting the data into test and train sets followed by training and testing.
The first phase was carried out to test a prototype model(with architecture described in (*1*) capable of predicting the steering angle with the help of the left, right and center view of the road. The prototype was tested for performance by running the Udacity simulator on autonomous mode. The second phase was carried out on real data that had only center images. Two types of models were developed: one model built entirely from scratch and another one (hybrid) with a pre-trained model, VGG16 with all the pre-trained layers frozen exempting the last one. The last layer was removed and three layers were added. This hybrid model's performance and robustness was evaluated on the grounds of least mean squared error between predicted steering values and actual ones. The hybrid model's performance was compared against a base model having VGG16 pre-trained model with an output layer and the author's model under two different scenarios: under the availability of complete data and availability of half the data.

4.1 Dataset description

4.1.1 Udacity Simulator dataset:

In the training phase the authors drove the simulator and the data(images) from the left, center and right images were segregated into separate folders. A log file was generated containing the location of the images and the values of the steering angle, throttle, brake and speed.

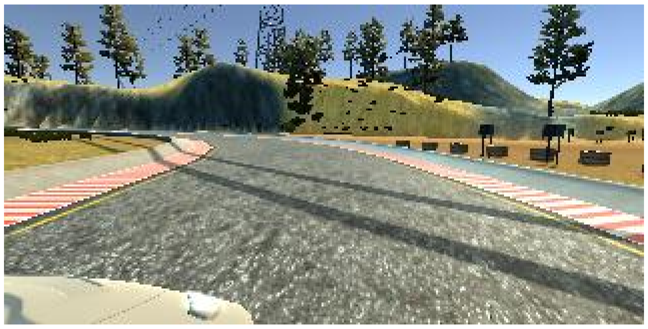


Fig 4: right view

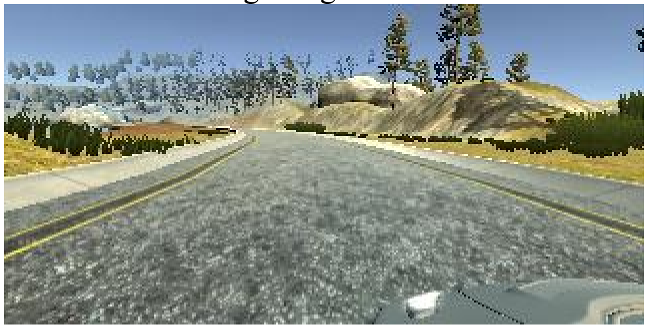


Fig 5: left view

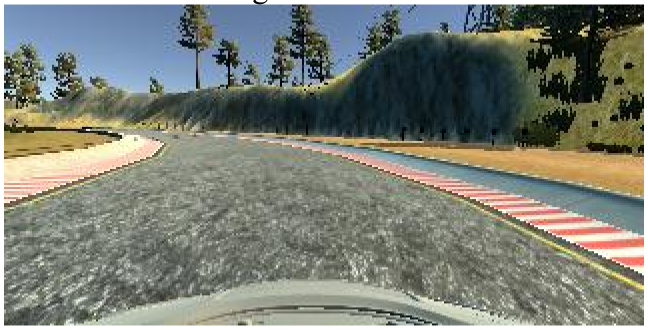
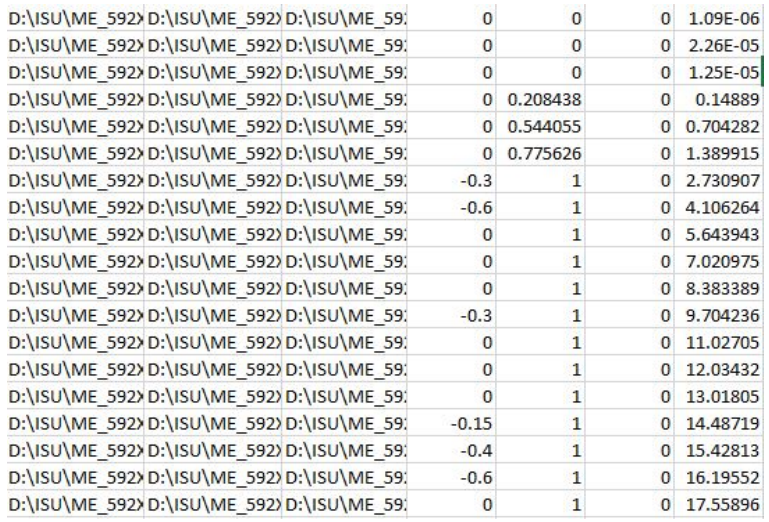


Fig 6: center view



D:\ISU\ME_592	D:\ISU\ME_592	D:\ISU\ME_59	0	0	0	1.09E-06
D:\ISU\ME_592	D:\ISU\ME_592	D:\ISU\ME_59	0	0	0	2.26E-05
D:\ISU\ME_592	D:\ISU\ME_592	D:\ISU\ME_59	0	0	0	1.25E-05
D:\ISU\ME_592	D:\ISU\ME_592	D:\ISU\ME_59	0	0.208438	0	0.14889
D:\ISU\ME_592	D:\ISU\ME_592	D:\ISU\ME_59	0	0.544055	0	0.704282
D:\ISU\ME_592	D:\ISU\ME_592	D:\ISU\ME_59	0	0.775626	0	1.389915
D:\ISU\ME_592	D:\ISU\ME_592	D:\ISU\ME_59	-0.3	1	0	2.730907
D:\ISU\ME_592	D:\ISU\ME_592	D:\ISU\ME_59	-0.6	1	0	4.106264
D:\ISU\ME_592	D:\ISU\ME_592	D:\ISU\ME_59	0	1	0	5.643943
D:\ISU\ME_592	D:\ISU\ME_592	D:\ISU\ME_59	0	1	0	7.020975
D:\ISU\ME_592	D:\ISU\ME_592	D:\ISU\ME_59	0	1	0	8.383389
D:\ISU\ME_592	D:\ISU\ME_592	D:\ISU\ME_59	-0.3	1	0	9.704236
D:\ISU\ME_592	D:\ISU\ME_592	D:\ISU\ME_59	0	1	0	11.02705
D:\ISU\ME_592	D:\ISU\ME_592	D:\ISU\ME_59	0	1	0	12.03432
D:\ISU\ME_592	D:\ISU\ME_592	D:\ISU\ME_59	0	1	0	13.01805
D:\ISU\ME_592	D:\ISU\ME_592	D:\ISU\ME_59	-0.15	1	0	14.48719
D:\ISU\ME_592	D:\ISU\ME_592	D:\ISU\ME_59	-0.4	1	0	15.42813
D:\ISU\ME_592	D:\ISU\ME_592	D:\ISU\ME_59	-0.6	1	0	16.19552
D:\ISU\ME_592	D:\ISU\ME_592	D:\ISU\ME_59	0	1	0	17.55896

Fig7: Log file with steering angle, throttle, brake and speed

4.1.2 Real data(*7*)

The dataset included 63000 images totaling to about 3.1GB of data.The data was recorded around Ranchos Palos Verde and San Pedro, California. The authors have chosen the data for 07/01/2018.The data folder also contained a log file listing the timestamps of the images and the corresponding steering angles.

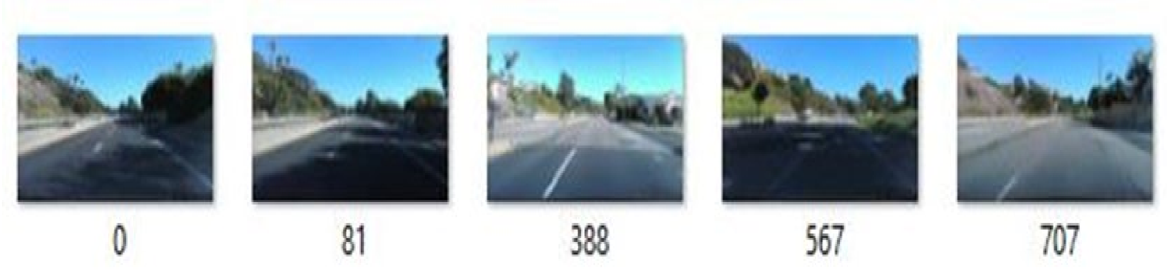
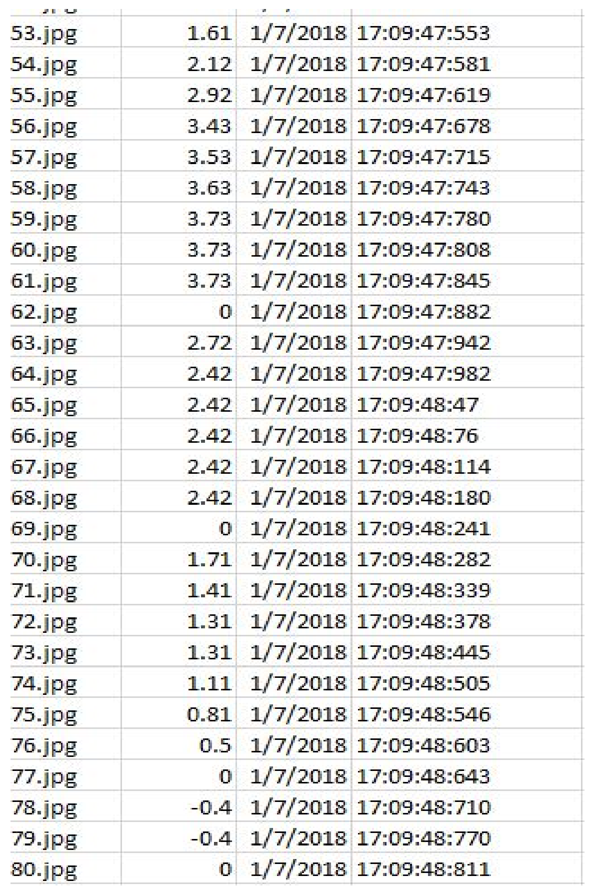


Fig 8: Sample of images obtained for 07/01/2018([Ref:https://github.com/SullyChen/driving-datasets](https://github.com/SullyChen/driving-datasets))



53.jpg	1.61	1/7/2018	17:09:47:553
54.jpg	2.12	1/7/2018	17:09:47:581
55.jpg	2.92	1/7/2018	17:09:47:619
56.jpg	3.43	1/7/2018	17:09:47:678
57.jpg	3.53	1/7/2018	17:09:47:715
58.jpg	3.63	1/7/2018	17:09:47:743
59.jpg	3.73	1/7/2018	17:09:47:780
60.jpg	3.73	1/7/2018	17:09:47:808
61.jpg	3.73	1/7/2018	17:09:47:845
62.jpg	0	1/7/2018	17:09:47:882
63.jpg	2.72	1/7/2018	17:09:47:942
64.jpg	2.42	1/7/2018	17:09:47:982
65.jpg	2.42	1/7/2018	17:09:48:47
66.jpg	2.42	1/7/2018	17:09:48:76
67.jpg	2.42	1/7/2018	17:09:48:114
68.jpg	2.42	1/7/2018	17:09:48:180
69.jpg	0	1/7/2018	17:09:48:241
70.jpg	1.71	1/7/2018	17:09:48:282
71.jpg	1.41	1/7/2018	17:09:48:339
72.jpg	1.31	1/7/2018	17:09:48:378
73.jpg	1.31	1/7/2018	17:09:48:445
74.jpg	1.11	1/7/2018	17:09:48:505
75.jpg	0.81	1/7/2018	17:09:48:546
76.jpg	0.5	1/7/2018	17:09:48:603
77.jpg	0	1/7/2018	17:09:48:643
78.jpg	-0.4	1/7/2018	17:09:48:710
79.jpg	-0.4	1/7/2018	17:09:48:770
80.jpg	0	1/7/2018	17:09:48:811

Fig 9: log file with image number, timestamp and steering angle in degree

4.1 Phases of Work

4.1.1 Proof of concept phase Description

The Udacity simulator helped us generate the data required for training and testing the self-driving car models. In this phase, three different datasets were used for training the model with Nvidia architecture: one with Udacity provided simulation data, the second with the data we generated through driving the simulator and third a mix of the Udacity provided data and the data we generated through driving the simulator. The model performance in the three cases were tested by using the model to drive the simulator car in the autonomous mode.

4.1.2 Main Phase Description

In the second phase, the authors tried to develop a model to predict the steering angle based on real images. The real data, however had the limitation of having only center images and therefore, our architecture had to be readjusted to take only a single feed of center images. However, the authors wanted to attempt a hybrid model that combines a pre-built model, namely VGG 16 with a few convolutional layers from our model. The hybrid model performance was not as good as our model when trained on the entire dataset. However, when we trained it on half the dataset the hybrid model outperformed our model illustrating that pre-built models like VGG16 can be very useful in building a robust model if the data available for training may be small.

4.2 Model Architecture Description

4.2.1 General Architecture of a Convolutional Neural network

The convolutional layer helps extract the features from the images and the pooling layer helps reduce the parameters when images are large. Spatial pooling is down sampling i.e dimensionality of the map get reduced but important information is retained. Relu or rectilinear Convolution is a mathematical operation that takes the image and the filter matrices as inputs and multiply them.

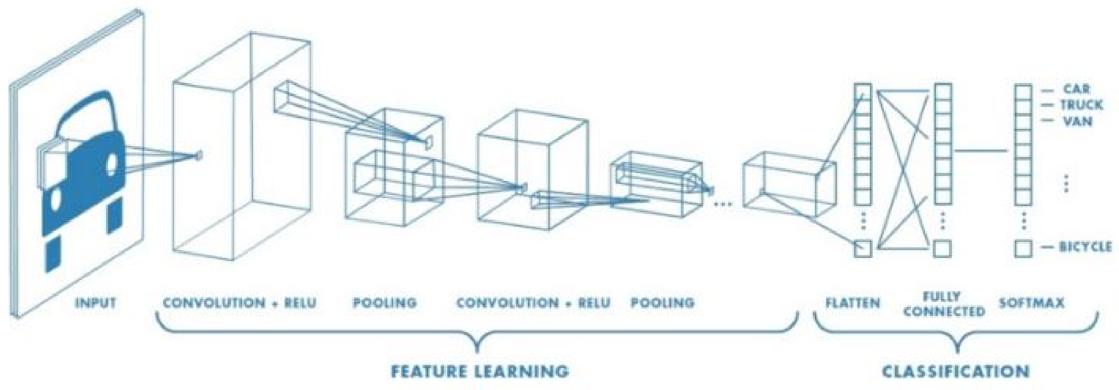


Figure 2 : Neural network with many convolutional layers

Fig10:Convolutional Neural network(https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148)

4.2.1 Proof of Concept Phase

Two architectures were experimented in this phase: the architecture suggested in Nvidia's research and an experimental model developed by the authors.The experimental model architecture is shown in Figure 1. The model developed was capable of taking input images from left, center and right cameras separately unlike the Nvidia architecture(*2*) that takes a single feed of images with either the left, right or center image chosen at random. Each set of images are directed through independent networks (3 sets of convolutional layers) and each of the independent networks had three sets of convolution-max pooling layers followed by a flattening layer (each). The outputs were concatenated and directed to the output layer as illustrated in the architecture shown in Fig (11). The models were trained on the Udacity Simulator data and the performance and results of this model were compared with those obtained from the Nvidia Architecture(*1*). The architecture suggested by researchers in Nvidia shown in Fig (12) served as the baseline for comparison of the model developed by us. The model reported a loss of 0.012 comparable to that proposed by Nvidia which reported a loss of 0.0099. The Nvidia architecture involved 5 sets of convolutional layers with 24, 36,48,64,64 neurons respectively followed by three dense layers and an output layer. The flattening layer helps convert 2 dimensional arrays into a single continuous linear vector.

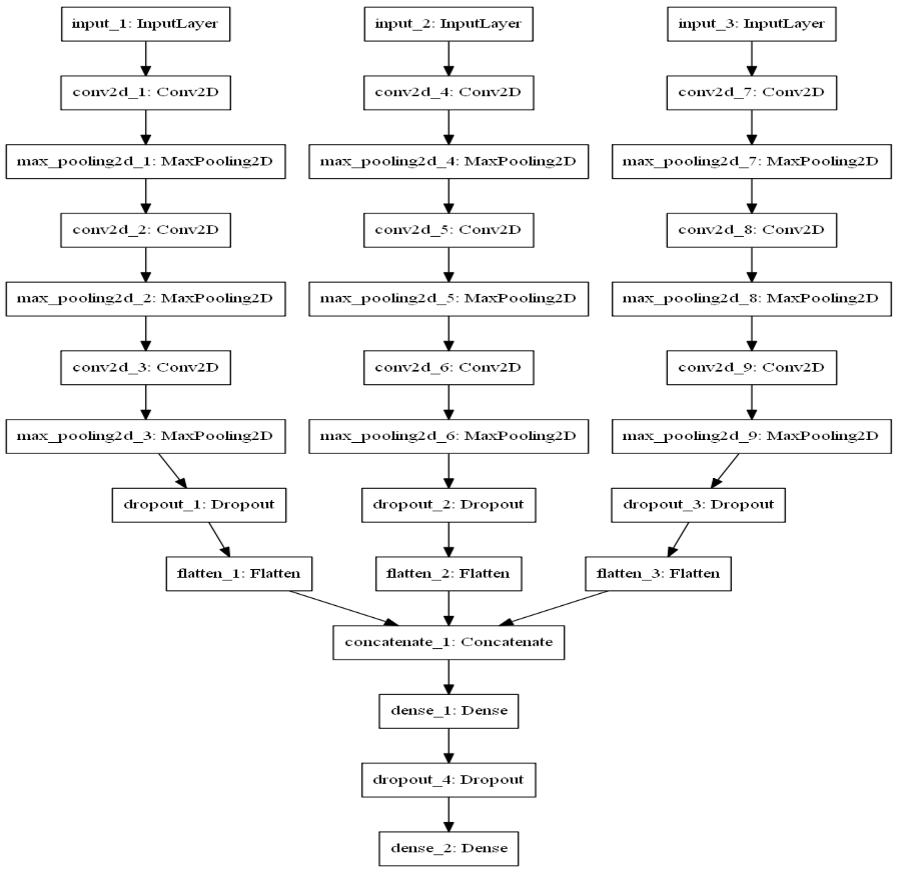


Fig11: Our experimental model modeled on Udacity simulator data

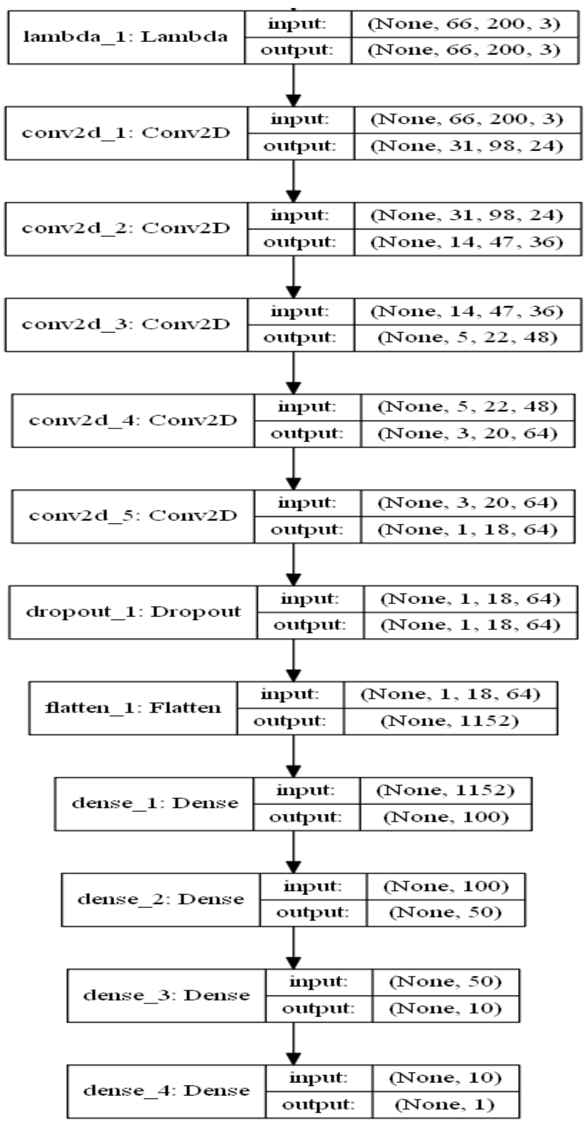


Fig 12: Architecture mentioned in reference(*1*)

4.2.2 Main Phase

Our experimental model was readapted to take only the center images and make steering angle predictions based on them. It consisted of three sets of convolutional- max pooling layers followed by three sets of dense layers and an output layer. The model was built on a trial and error basis by adding and removing layers to see better efficiency or minimum loss.

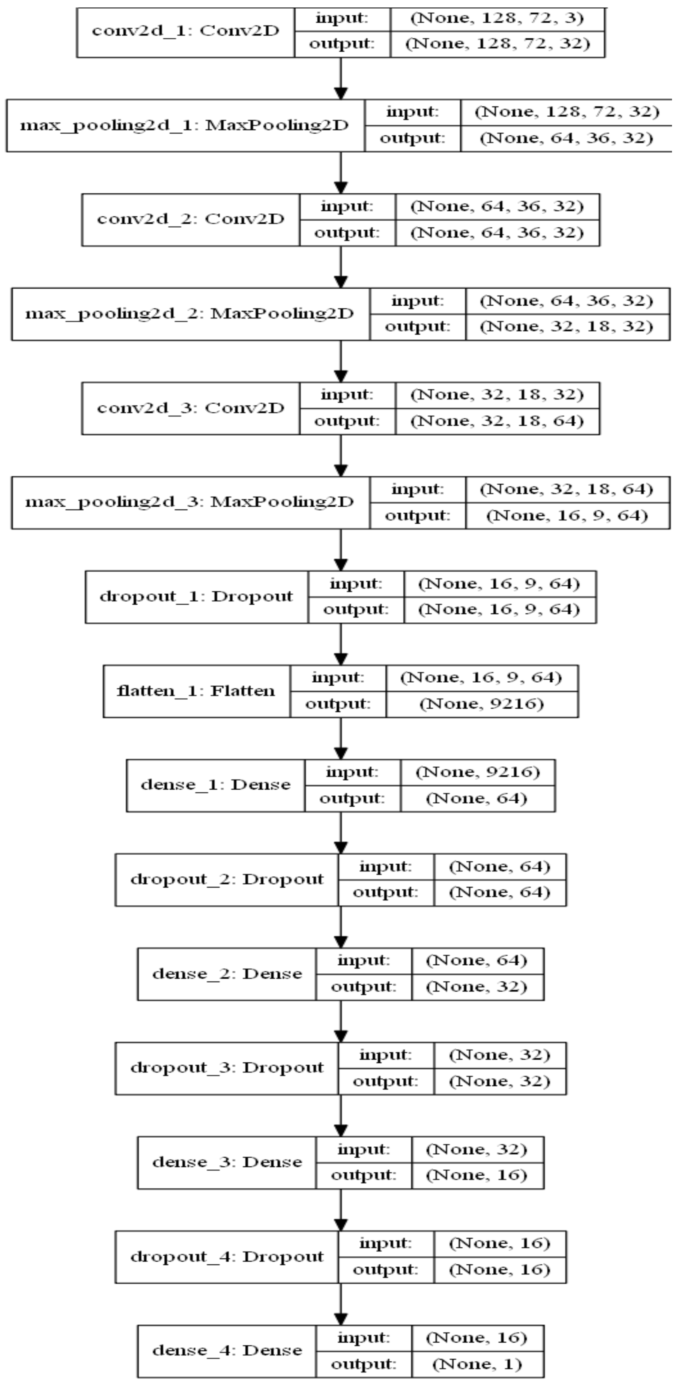


Fig13: our experimental model modeled on real world data fed with only center images

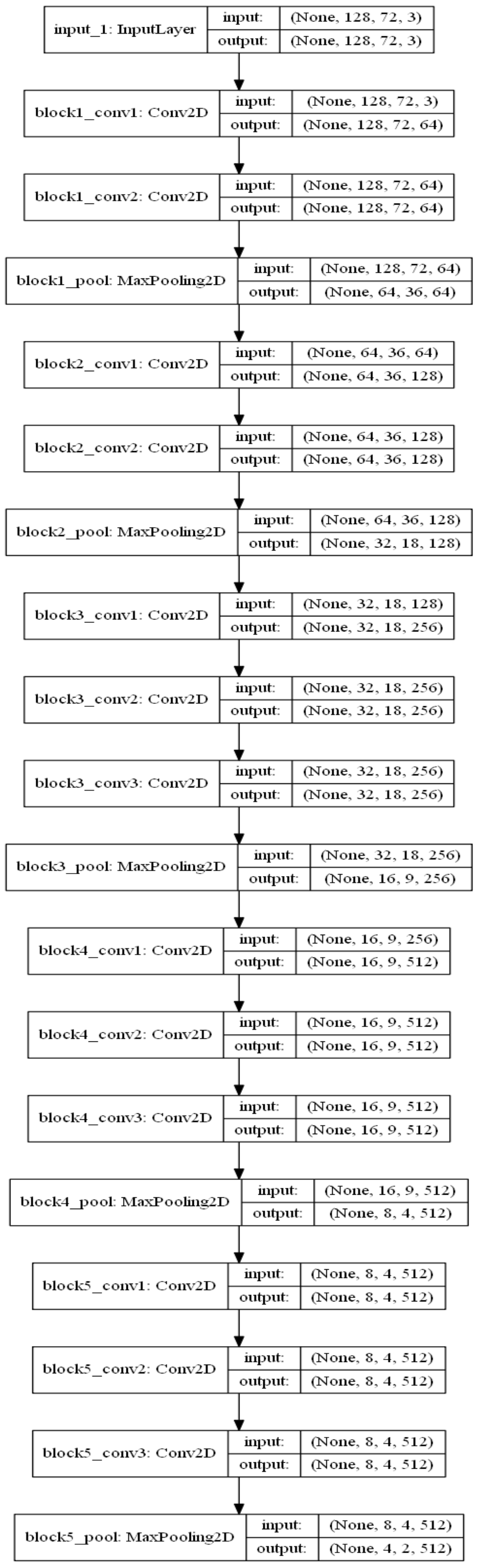

input_1: InputLayer | input: (None, 128, 72, 3) | output: (None, 128, 72, 3)
block1_conv1: Conv2D | input: (None, 128, 72, 3) | output: (None, 128, 72, 64)
block1_conv2: Conv2D | input: (None, 128, 72, 64) | output: (None, 128, 72, 64)
block1_pool: MaxPooling2D | input: (None, 128, 72, 64) | output: (None, 64, 36, 64)
block2_conv1: Conv2D | input: (None, 64, 36, 64) | output: (None, 64, 36, 128)
block2_conv2: Conv2D | input: (None, 64, 36, 128) | output: (None, 64, 36, 128)
block2_pool: MaxPooling2D | input: (None, 64, 36, 128) | output: (None, 32, 18, 128)
block3_conv1: Conv2D | input: (None, 32, 18, 128) | output: (None, 32, 18, 256)
block3_conv2: Conv2D | input: (None, 32, 18, 256) | output: (None, 32, 18, 256)
block3_conv3: Conv2D | input: (None, 32, 18, 256) | output: (None, 32, 18, 256)
block3_pool: MaxPooling2D | input: (None, 32, 18, 256) | output: (None, 16, 9, 256)
block4_conv1: Conv2D | input: (None, 16, 9, 256) | output: (None, 16, 9, 512)
block4_conv2: Conv2D | input: (None, 16, 9, 512) | output: (None, 16, 9, 512)
block4_conv3: Conv2D | input: (None, 16, 9, 512) | output: (None, 16, 9, 512)
block4_pool: MaxPooling2D | input: (None, 16, 9, 512) | output: (None, 8, 4, 512)
block5_conv1: Conv2D | input: (None, 8, 4, 512) | output: (None, 8, 4, 512)
block5_conv2: Conv2D | input: (None, 8, 4, 512) | output: (None, 8, 4, 512)
block5_conv3: Conv2D | input: (None, 8, 4, 512) | output: (None, 8, 4, 512)
block5_pool: MaxPooling2D | input: (None, 8, 4, 512) | output: (None, 4, 2, 512)

Fig14: Hybrid model including VGG16 and three extra layes

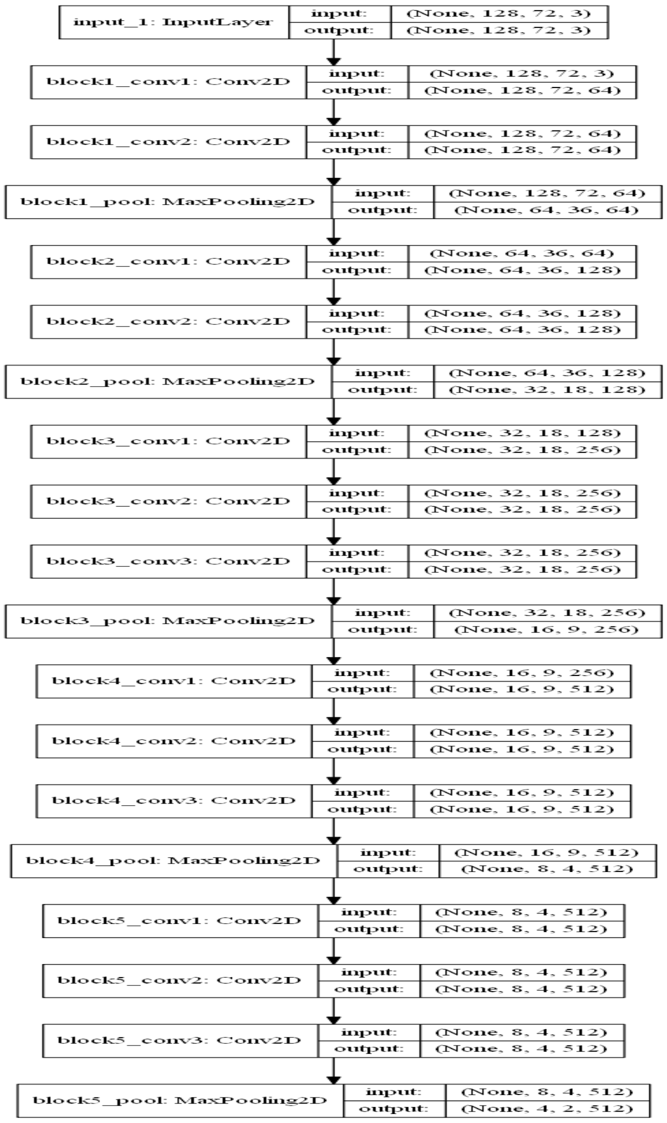

input_1: InputLayer | input: (None, 128, 72, 3) | output: (None, 128, 72, 3)
block1_conv1: Conv2D | input: (None, 128, 72, 3) | output: (None, 128, 72, 64)
block1_conv2: Conv2D | input: (None, 128, 72, 64) | output: (None, 128, 72, 64)
block1_pool: MaxPooling2D | input: (None, 128, 72, 64) | output: (None, 64, 36, 64)
block2_conv1: Conv2D | input: (None, 64, 36, 64) | output: (None, 64, 36, 128)
block2_conv2: Conv2D | input: (None, 64, 36, 128) | output: (None, 64, 36, 128)
block2_pool: MaxPooling2D | input: (None, 64, 36, 128) | output: (None, 32, 18, 128)
block3_conv1: Conv2D | input: (None, 32, 18, 128) | output: (None, 32, 18, 256)
block3_conv2: Conv2D | input: (None, 32, 18, 256) | output: (None, 32, 18, 256)
block3_conv3: Conv2D | input: (None, 32, 18, 256) | output: (None, 32, 18, 256)
block3_pool: MaxPooling2D | input: (None, 32, 18, 256) | output: (None, 16, 9, 256)
block4_conv1: Conv2D | input: (None, 16, 9, 256) | output: (None, 16, 9, 512)
block4_conv2: Conv2D | input: (None, 16, 9, 512) | output: (None, 16, 9, 512)
block4_conv3: Conv2D | input: (None, 16, 9, 512) | output: (None, 16, 9, 512)
block4_pool: MaxPooling2D | input: (None, 16, 9, 512) | output: (None, 8, 4, 512)
block5_conv1: Conv2D | input: (None, 8, 4, 512) | output: (None, 8, 4, 512)
block5_conv2: Conv2D | input: (None, 8, 4, 512) | output: (None, 8, 4, 512)
block5_conv3: Conv2D | input: (None, 8, 4, 512) | output: (None, 8, 4, 512)
block5_pool: MaxPooling2D | input: (None, 8, 4, 512) | output: (None, 4, 2, 512)

Fig15 : VGG16 model with an output layer

RESULTS

The results from the Udacity simulator data have been summarized in the table below. The least mean squared error was obtained for the simulation data obtained from Udacity and the highest error for the data for the driving data obtained by authors. However the authors made an attempt to collect from the track 2 which involved more curves since they felt that training the car to negotiate more curves and training it to handle more adverse conditions would help it drive better. A good tradeoff was obtained with the model that used a combination of Udacity data from track 1 and out data from track 2 reporting a mean squared error if about 0.1582.

Table 1: Summary of model based on Udacity simulator data

Data	Loss=Mean squared error	Total epochs	Number of data points	Trained on
Udacity provided data(prebuilt model)	0.0223	5	6830	Track 1 only
Our own driving data	0.1815	5	23188	Track 2
Combination of Udacity and our driving data	0.1582	5	6114	Both track1 and 2

The model parameters that were chosen for the Phase 2 work have been summarized in the table below. The efficiency of the model was reported based on the mean squared error calculated when predicting the steering angle. The epoch in which the least error was recorded has also been indicated in the table below.

Table 2: Summary based on Real data(complete dataset)

Model	Data	Optimizer	lr(learning rate)	decay	loss	epoch
Model 1: Our model	Real data	Adam	1 e-4	1 e-6	Mean squared error=0.002175	66th epoch
Model 2: VGG+3 hidden layersand output layer	Real data	Adam	1 e-4	1 e-6	Mean squared error= 0.00324	97th epoch
Model3: VGG+1 output layer	Real data	Adam	1 e-4	1 e-6	Mean squared error=0.0048	97th epoch

Table 3: Summary based on Real dataset (half the data)

Model	Data	Optimizer	lr(learning rate)	decay	loss	epoch

Model 1: Our model	Real data	Adam	1 e-4	1 e-6	Mean squared error=0.005496	26th epoch
Model 2: VGG+3 hidden layersand output layer	Real data	Adam	1 e-4	1 e-6	Mean squared error= 0.00273	76th epoch
Model3: VGG+1 output layer	Real data	Adam	1 e-4	1 e-6	Mean squared error=0.003673	97th epoch

The proof of concept gave the authors an idea of how to train a neural network with the images obtained during driving to predict the steering angle. The authors realized the variety that needs to be in the data in order to make the algorithm understand what decisions to take in which situation For eg: if the driver always drove straight without much steering angle variation, the car, in the autonomous mode cannot understand the sharp turns too well. The authors also implemented a hybrid model using a prebuilt VGG16 model. The margin of improvement in performance was not remarkable since the model developed by the authors was already giving good results with mean squared loss 0.002175 (32.8 % less than the hybrid model). However, when the size of the data was reduced o half, the hybrid model with VGG16 outperformed the model with VGG16 by nearly 50% as evident in the 'Model loss' graph. The model loss graphs corresponding to the three models built in the second phase have been shown in Fig 16, 17 and 18.

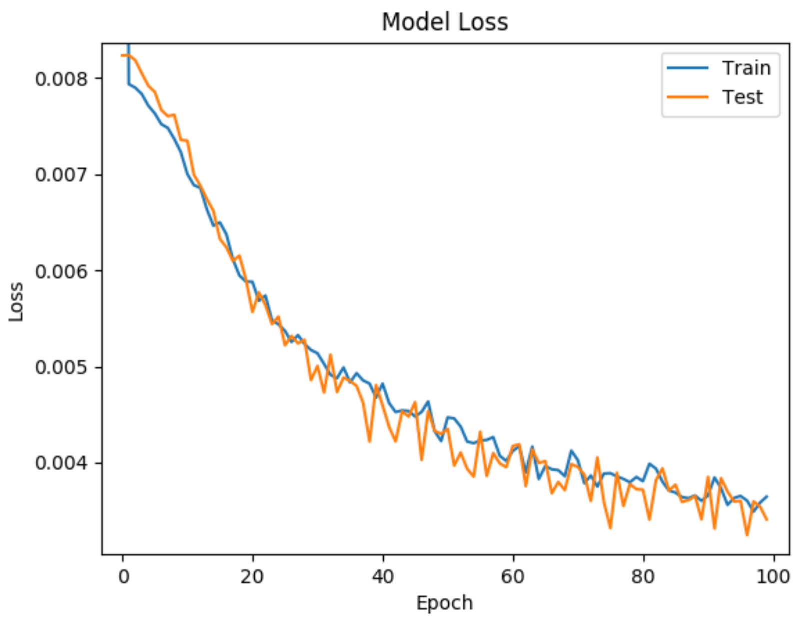


Fig 16: Base model with VGG16 and output layer model loss graph

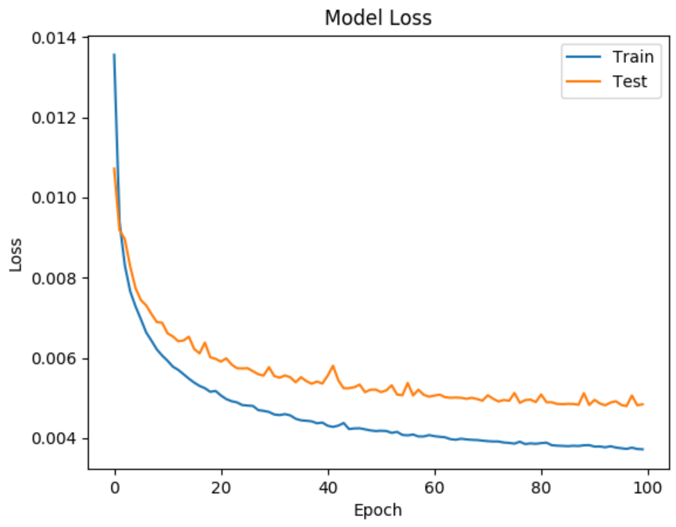


Fig 17: Hybrid model model loss graph

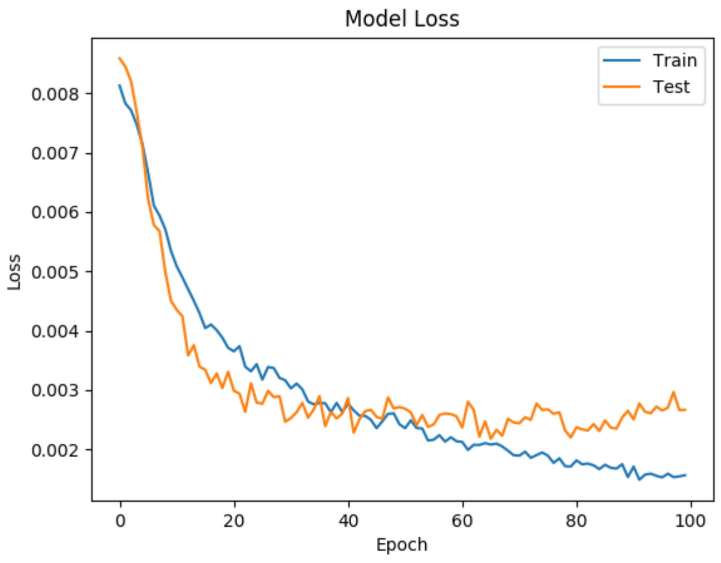


Fig 18: Model developed by authors' model loss graph

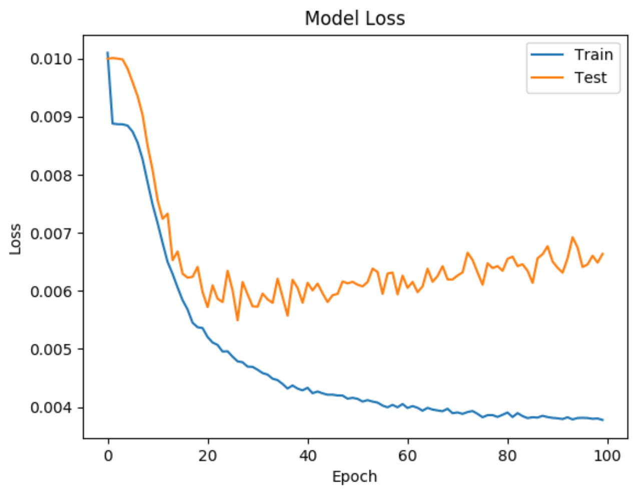


Fig 19: Model loss graph for authors' model(half data)

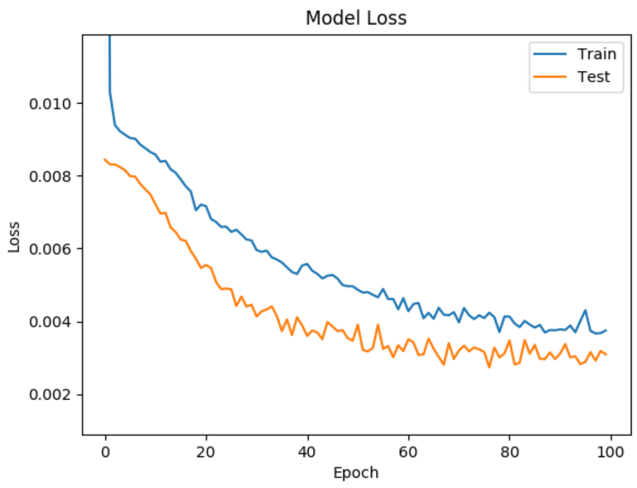


Fig 20: Hybrid model model loss graph(half data)

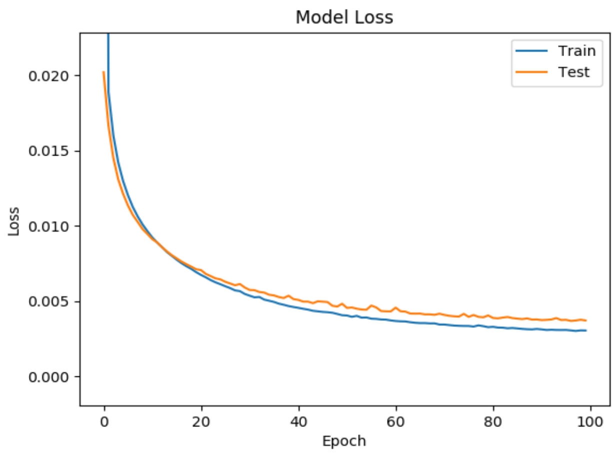


Fig 20: Base model with VGG16 and output layer model loss graph(half data)

CONCLUSION

Building and comparison of the different models' performance gave the authors an intuition of how to utilize convolutional neural networks to do behavioral mapping in self driving cars predicting the steering angle with as much accuracy as possible. The performance of the hybrid model utilizing VGG16 outperformed our model when the data was reduced to half illustrating that such models are useful when required to train on small datasets. Though imitation learning has been applied to lane following, road following and obstacle avoidance when it comes to making decisions like where to turn at a fork as per the desire of the passenger conditional imitation learning may be required as suggested in (*3*) where besides the perceptual input and control signal, a representation of the expert's intention is also given. This could pose as a future scope of the project

REFERENCES

1. Bojarski, M., D. Del Testa, P. Goyal, J. Zhang, D. Dworakowski, L. D. Jackel, B. Firner, M. Monfort, J. Zhao, and K. Zieba. End to End Learning for Self-Driving Cars. pp. 1–9.
2. Https://github.com/llSourcell/How_to_simulate_a_self_driving_car. No Title.
3. Codevilla, F., M. Matthias, and L. Antonio. End-to-End Driving via Conditional Imitation Learning.
4. Du, S., H. Guo, and A. Simpson. Self-Driving Car Steering Angle Prediction Based on Image Recognition.
5. Xu, H., Y. Gao, F. Yu, and T. Darrell. End-to-End Learning of Driving Models from Large-Scale Video Datasets. *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, Vol. 2017-Janua, 2017, pp. 3530–3538. https://doi.org/10.1109/CVPR.2017.376.
6. Fernando, T., and C. Fookes. Going Deeper : Autonomous Steering with Neural Memory Networks.
7. Https://github.com/SullyChen/driving-datasets, H. amazonaws. com/video. udacity-data. com/topher/2016/December/584f6edd_data/data. zi. No Title.