Deep Learning for Autonomous Car Driving

Saransh Jain
University of California, San Diego
California, CA 92092
saj039@eng.ucsd.edu

Gautam Nagpal
University of California, San Diego
California, CA 92092
gnagpal@eng.ucsd.edu

Jaskaran Singh
University of California, San Diego
California, CA 92092
jsvirdi@eng.ucsd.edu

Sudeep Nagabhirava
University of California, San Diego
California, CA 92092
snagabhi@eng.ucsd.edu

Shreyas Udupa Balekudru
University of California, San Diego
California, CA 92092
sudupaba@eng.ucsd.edu

Abstract

Deep neural networks, especially Convolutional Neural Networks (CNN) are now widely used for solving computer visions problems. One such problem of huge interest today is autonomous car navigation. In this project, we experimented with various deep neural networks to autonomously drive a car inside a simulator provided by Udacity. In this paper, we first implemented CNN for the autonomous driving task and analyzed its performance.Then, we implement CNN with Long Short Term Memory (LSTM) network to enable the model to make better decisions with some context present in its memory about the decisions it made in the recent past. We also experimented with a deep reinforcement learning model by trying to get the autonomous driving system to learn steering using policy gradient.

1 Introduction

1.1 Problem Statement and Motivation

An autonomous car/self-driving car is a vehicle that is capable of sensing its environment and navigating without human input. Many such vehicles are being developed, but as of February 2017 automated cars permitted on public roads are not yet fully autonomous. They all require a human driver at the wheel who is ready at a moment's notice to take control of the vehicle.

Autonomous cars use a variety of techniques to detect their surroundings, such as radar, laser light, GPS, odometry, and computer vision. Advanced control systems interpret sensory information to identify appropriate navigation paths, as well as obstacles and relevant signage.

Among the potential benefits of autonomous cars is a significant reduction in traffic collisions;the resulting injuries; and related costs, including a lower need for insurance. Autonomous cars are also predicted to offer major increases in traffic flow; enhanced mobility for children, the elderly, disabled and poor people; the relief of travelers from driving and navigation chores; lower fuel consumption; significantly reduced needs for parking space in cities; a reduction in crime; and the facilitation of different business models for mobility as a service, especially those involved in the sharing economy.

A classification system based on six different levels (ranging from none to fully automated systems) was published in 2014 by SAE International. This classification system is based on the amount of driver intervention and attentiveness required, rather than the vehicle capabilities, although these are very closely related.

SAE automated vehicle classifications:

- Level 0: Automated system has no vehicle control, but may issue warnings.
- Level 1: Driver must be ready to take control at any time. Automated system may include features such as Adaptive Cruise Control (ACC), Parking Assistance with automated steering, and Lane Keeping Assistance (LKA) Type II in any combination.
- Level 2: The driver is obliged to detect objects and events and respond if the automated system fails to respond properly. The automated system executes accelerating, braking, and steering. The automated system can deactivate immediately upon takeover by the driver.
- Level 3: Within known, limited environments (such as freeways), the driver can safely turn their attention away from driving tasks, but must still be prepared to take control when needed.
- Level 4: The automated system can control the vehicle in all but a few environments such as severe weather. The driver must enable the automated system only when it is safe to do so. When enabled, driver attention is not required.
- Level 5: Other than setting the destination and starting the system, no human intervention is required. The automatic system can drive to any location where it is legal to drive and make its own decisions.

In this project, we try to create a Level 2 autonomous system for driving a car in a simulated environment. We use deep neural networks to accurately predict the current steering angle to turn the car using raw pixel data from images taken from the dashboard camera as the input to our system.

The goal of the project is to teach a car how to drive without us explicitly training the system to detect lane markings on roads or using computer vision features like SIFT, SURF etc.

We perform our experiments in a simulator provided by Udacity. By analyzing the images obtained from the simulator at each time step, we aim to predict the steering angle to keep the car on track in the simulator.

1.2 Modifications to existing approaches

The previous models used for solving Udacity's simulator self driving challenge mostly are deep convolutional networks. Here we experiment and improvise with different convolutional neural networks (parameters ranging from 1 million to 2.1 million) and a combination of Convolutional and Recurrent neural networks (parameters ranging from 2.1 million to 47 million). The motivation behind using a combination of CNN and RNN to help predict the steering angle is Oliver Cameron's blog-post which explains how to get started with building a basic deep CNN model and how just a single network which takes raw input (camera imagery) and produces a direct steering command, is considered to be the holy-grail of current autonomous vehicle technology. The motivation behind experimenting with a Deep Reinforcement Learning model is Andrej Karpathy's blog-post, in which he has documented on how a deep reinforcement model can learn to play Atari's Pong using raw pixel data and policy gradient.

2 Background

Here, we try to solve part of this challenge where our task is to predict the steering angle given the current image frame (taken from one of dashboard left, center and right cameras). After predicting the steering angle frame by frame, we aimed at running the car autonomously on the simulator.

2.1 Literature referenced

The problem on Udacity was conceptualized by Nvidia in their paper [1] which is also documented on their blog. In this blog, Nvidia introduces the DAVE2 collection system for training data and also

describes a deep convolutional model architecture which can be used to predict steering angle based on a particular frame. The network has about 27 million connections and 250 thousand parameters. It has five convolutional layers and three fully connected layers.

Since, this is part of Udacity's nano-degree program, Udacity also released the winning solution model architectures after the competition. Most of these models, use deep-convolution network alone and report that performance of a model heavily depends on training data.

2.2 Conceptual framework for previous work

The previous work for this problem is primarily centric around the applications of Convolutional Neural Networks. The framework utilizes real life data provided by Udacity for different daylight conditions. The image data is then fed into the CNN for training with the output being the steering angle for a particular image. The very fact that CNNs are able to learn different features of the road ahead to analyse what represents a left turn, right turn or a straight road helps in predicting the corresponding steering angle. We visualise this learning capability later to show the impact of CNNs for solving this problem. When the CNN has learned sufficient real life scenarios it is able to predict steering angles for all together unknown tracks.

3 Model

We have experimented with various models. Each of the model architectures shown in the following figures was generated using a Keras package, specifically for this purpose. Here are some of the interesting configurations:

- CNN with 1 million parameters: This network architecture as shown in Figure 1 has three convolutional layers and three dense layers. It totals around 1 million parameters. This is the simplest deep convolutional network that we implemented. The key features of this model are as follows:
  - No explicit data preprocessing is done. The first three layers are used instead to preprocess the data. First layer for cropping the image, second layer for resizing the image and the third layer for batch normalization.
  - Three convolution layers with exponential linear units and dropout layers in between.
  - Two fully connected layers in the end.

- CNN with 2 million parameters: This network architecture as shown in Figure 2 has five convolutional layers and five dense layers. It totals around 2 million parameters. The key features of this model are as follows:
  - Explicit data preprocessing is done in this case. Image cropping, random shear, left-right flip, random-gamma and image resizing are done prior to feeding the images to the network.
  - There are 5 convolution layers with relu activations and max-pooling layers in between.
  - Next, after flattening the output of convolutions, there are 5 fully connected layers which help predict the steering angle.

- CNN and LSTM with 47 million parameters: The network architecture as shown in Figure 3. As stated before this model was open-sourced in Oliver Cameron's blog-post. This was one of the best models used in the predictive task of real-time self-driving where apart from steering angle, the throttle and speed also needs to be predicted in order to navigate the vehicle as expected. We adopted this model and modified it so that it receives a different size input and the model only predicts the steering angle. The key features of this model are:
  - This model has a convolutional neural network attached to an LSTM.
  - The CNN is responsible for learning 128-dimensional feature representation of image frames while the LSTM is used to predict the current steering angle while maintaining context of steering angle and image frames of the recent past.

– The key difference in the CNN part of this model from the previous two models is that, residual networks are used in the CNN part of this model for easy propagation of gradient.

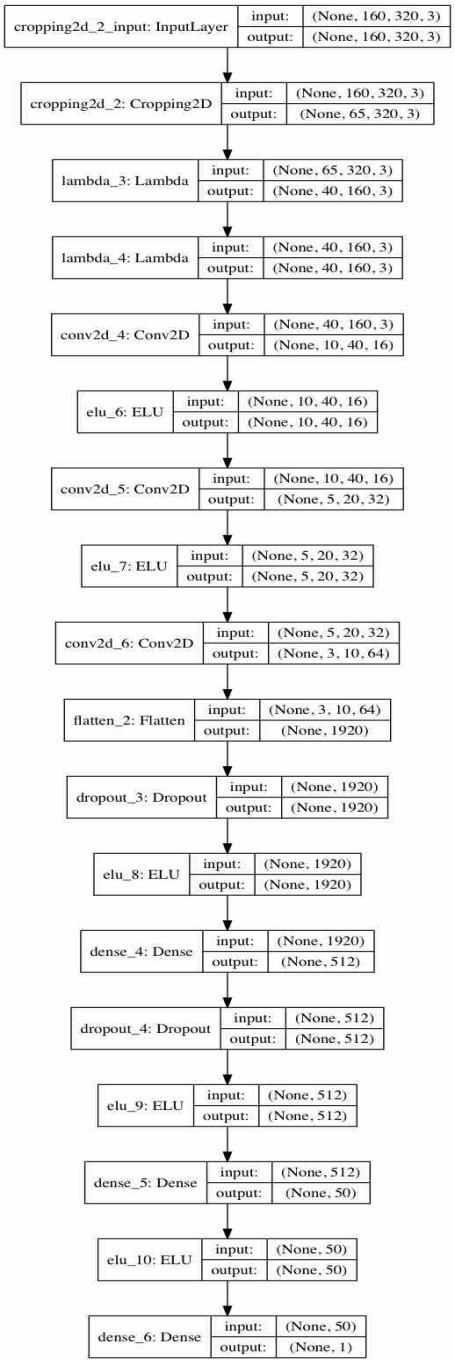
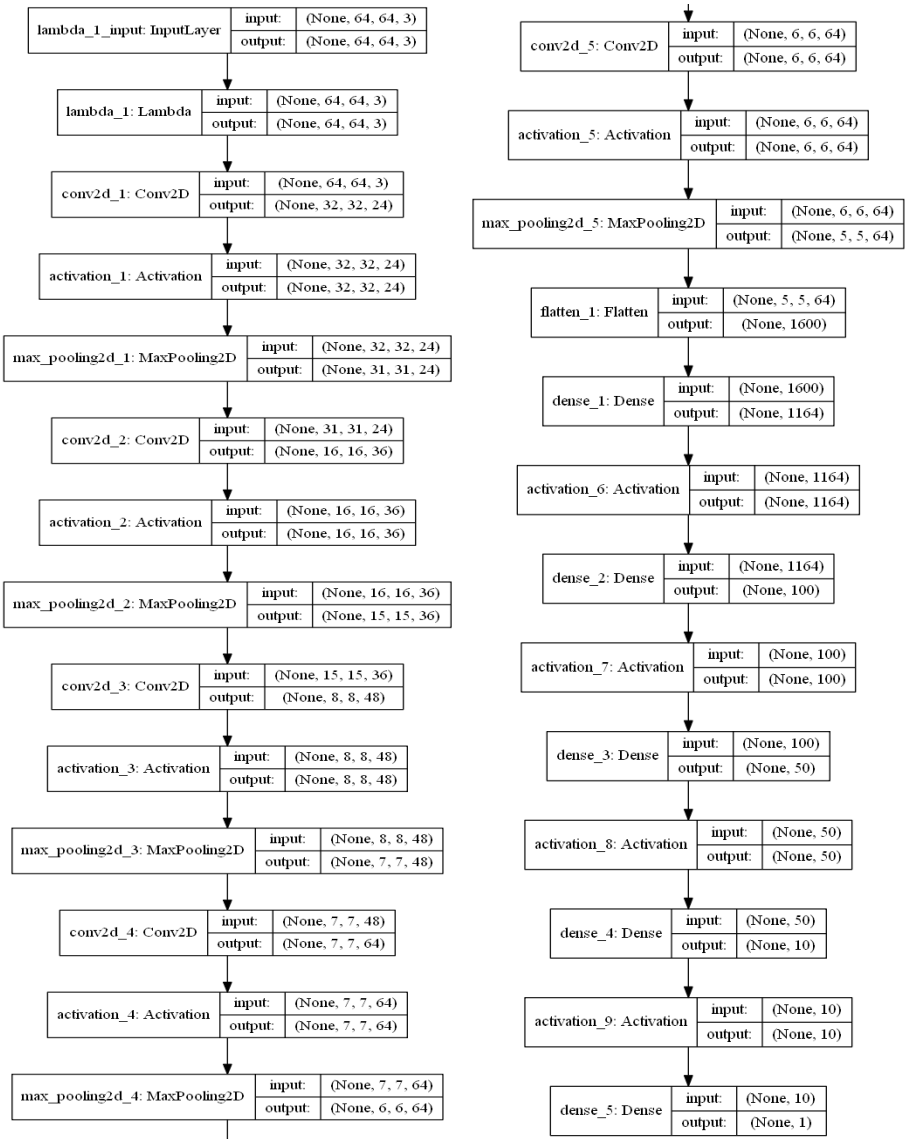


Figure 1: CNN (1 million parameters).



(a) Network Architecture Part I (b) Network Architecture Part II

Figure 2: CNN (2 million parameters).

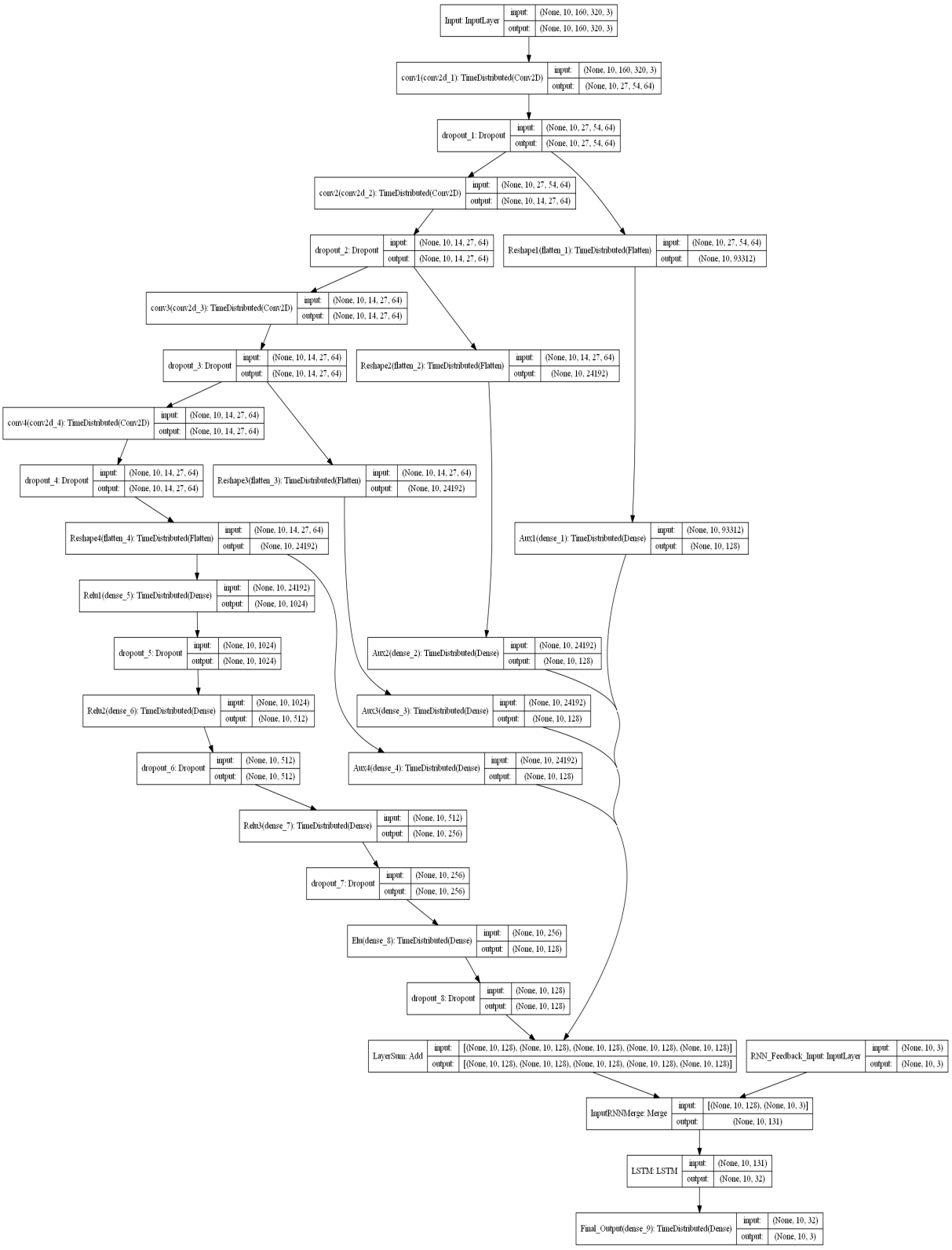


Figure 3: CNN with RNN model architecture (47 million parameters).

3.1 Project Progress Timeline

We have been moving forward in steps as follows:

- Collected and studied documents of the concerned literature.
- Implemented small CNN network with 1 million parameters.
- Preprocessed and augmented data using techniques like *shear*, *gamma*, *cropping*, etc.
- Implemented Deeper CNN networks (2 million parameters)
- Moved to an RNN based approach to analyse temporal nature of predictions
- Experimentation with Reinforcement Learning models

3.2 Data

Data generated using Udacity's simulator.

It is easy to generate data from Udacity's simulator. The car can be driven in Training Mode for a few laps which records a stream of frames from a camera mounted on the dashboard of the car sampled at 10fps. There is a label file which has an entry in it corresponding to each video frame which gives the steering angle corresponding to the image.

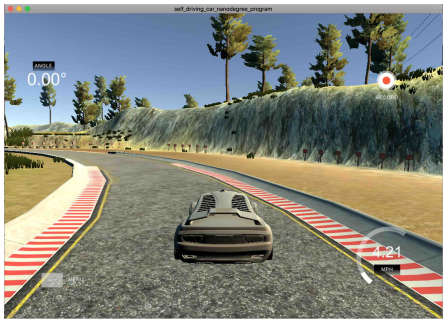
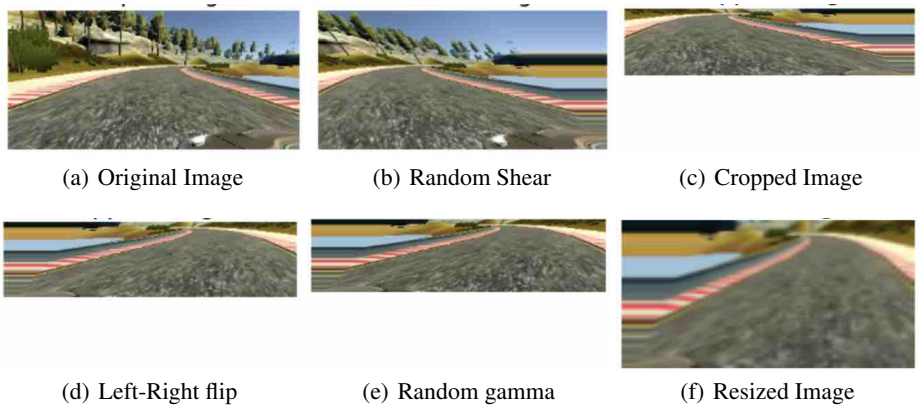


Figure 4: Screenshot of driving screen

3.3 Data preprocessing and Augmentation

Usually for a task like this, training data plays a crucial task and it is very important to generalize the data well for the model to learn efficiently. Using just the raw images from the simulator data was not giving us good performance even after trying different model architectures. This motivated us to spend considerable amount of time for preprocessing the images. This blog mentioned some performed operations like:

- Random-shear: Images are selected with 0.9 probability for random shearing. An angle adjustment is calculated to correspond with the sheared input.
- Crop: In the original image, 35% of it from the top and 10% from the bottom is removed.
- Random-flip: Original images with 0.5 probability are selected to be flipped along with their steering angles. This is because in the Udacity track, there are more left turns than right turns. However, for the CNN and LSTM model, this step was skipped because temporal learning will be lost.
- Random-gamma: This adjusts the luminosity of the image by randomly selecting γ, a hyper-parameter to adjust the brightness.



(a) Original Image (b) Random Shear (c) Cropped Image

(d) Left-Right flip (e) Random gamma (f) Resized Image

Figure 5: Data Preprocessing and augmentation

3.4 Input and Output of the Model

The input to the model is the re-sized image obtained from the simulator at each time-step. The original image with dimensions of 160 x 320 is resized to 64 X 64 and has 3 color channels (RGB). The model's output is a single number depicting the predicted steering angle for the current frame. The output is a real number in the range [-1, 1] corresponding to the scaled steering angles in the simulator in the range [-25°, 25°]. A negative angle corresponds to a left turn while a positive angle corresponds to a right turn. The output is generated at every frame depending on the input frame image for the deep CNN models. For the model with CNN and LSTM, the output generated at every frame also depends on pictures of frames in previous 10 time steps.

3.5 Objective Function

The predicted steering angle is a real number in the range [-1, 1]. We seek to maximize the likelihood of the predicted steering angle. Towards this end, we use the mean squared error as the cost function. The goal is to minimize this squared loss function.

$$MeanSquaredError = \frac{\sum_{i=1}^{n}(y - y')^2}{n}$$

where n is the number of training samples, y is the actual steering angle and y' is the predicted steering angle.

3.6 Softwares Used

The experiments for this project are performed on the simulator provided by Udacity [3][4]. The deep neural network models are compiled and trained using Keras which is a high-level neural networks API for Theano and Tensorflow, which are open-source Python libraries for machine intelligence.

4 Experiments and Results

4.1 Experiments Description

Since we were predicting steering angle based on input images from the simulator, we first attempted to train a CNN with around 1 million parameters. We used images obtained from the center camera of the car for training as these were the only images available in the simulator during autonomous mode. The label for every frame denotes the steering angle. An analysis of these label values shows that the training data is highly biased to 0° as shown in the Figure 4.1. During training, we also plotted graphs to visualize how training and validation losses vary over epochs. After training, we stored the model and its weights. We then ran the simulator on autonomous mode by using the steering angle predicted by the model and keeping the throttle value fixed.

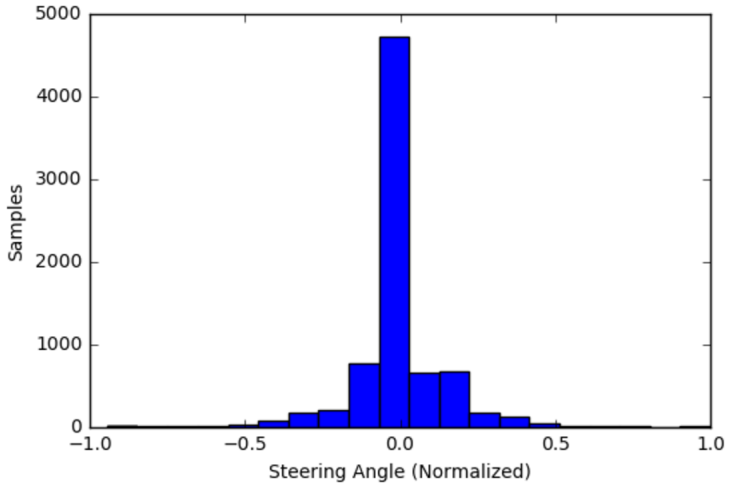


Figure 6: Histogram representing steering angles in the dataset

We performed the same experiments using the different models mentioned in the earlier sections. We compared the performance of the models based on how well the car ran in autonomous mode on the simulator and based on the validation loss obtained after training on several epochs. We also visualized hidden layer activations from the two CNN models used to analyze what the model is representing in the internal hidden layers.

We then trained a CNN model enhanced with LSTM. We used 10 consecutive image frames along with their corresponding driving angles as input. Hence, the LSTM is fed with **10x128** dimensional image representation where **128** is the size of the output representation by CNN and along with it we give a **10x1** vector of past steering angles. We analyzed the validation loss vs number of epochs for this model. However, when we ran our trained model in autonomous mode the performance was not satisfactory. This can be perfected by tuning of hyper-parameters and with more amount of training data.

As an experiment, we also tried to implement a Deep Reinforcement Learning model using policy gradient to learn how to steer the car in the simulator. We implemented the model with one hidden layer with 200 hidden nodes. The input to the network were the scaled pixel values between 0 and 1 of the image while the output was a sigmoid function depicting the probability of taking a left turn. The hidden layer used a rectified linear unit activation function. The architecture is as shown in 7. All weights in the network were initialized using Xavier initialization. We then ran the car in autonomous mode by predicting the steering angle using forward propagation in the model with the image pixels as input. Once the car crashed (speed became less than a threshold), we updated the weights by backpropagating through the network using the reward obtained. The reward was set to be 0.1 for each second the car stayed on track and a constant -1 for crashing.

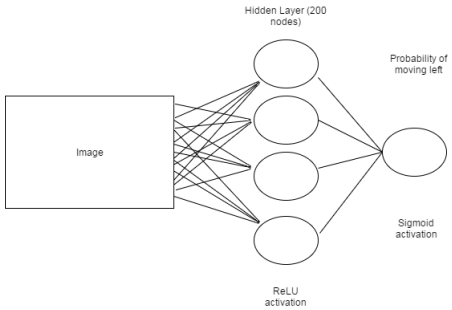


Figure 7: Architecture of the policy network

4.2 Results

4.2.1 CNN Model - 1 million parameters

For the purpose of training this model we used the ADAM [7] optimization algorithm and for checking for convergence we used early stopping technique. This model after training had a validation loss(measured in mean squared error) of 0.01 as shown in Figure 8. We plotted the activations for two images representing left and right turns from our dataset for the intermediate convolutional layer from our network to visualize how the features got trained.To check for the convergence of model and the corresponding loss values we plotted the train and validation loss for each epoch.(Figures 8-11)

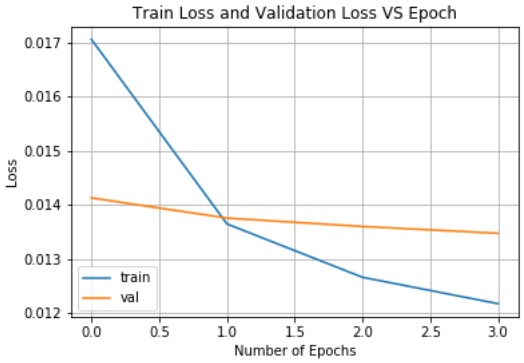
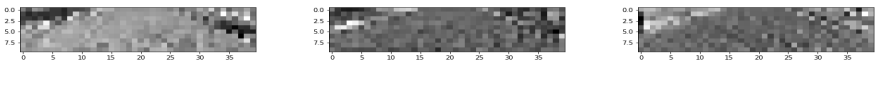


Figure 8: Loss vs Number of Epochs (CNN Model 1)

Figure 9: Right Turn Original Image



(a) (b) (c)

(d) (e)

Figure 10: Hidden Layer Activations for Right Turn (CNN Model 1)

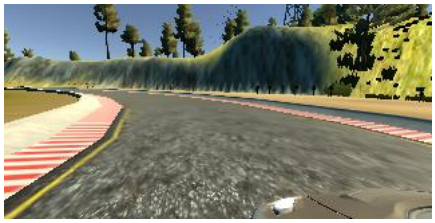
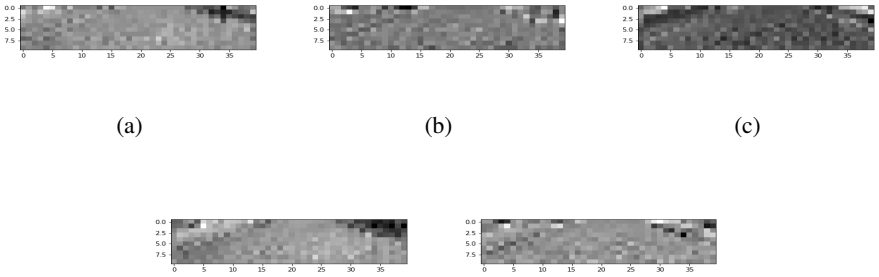


Figure 11: Left Turn Original Image (CNN Model 1)



(a) (b) (c)

(d) (e)

Figure 12: Hidden Layer Activations for Left Turn (CNN Model 1)

4.2.2 CNN Model - 2 million parameters

Similarly, for the purpose of training this model we used the ADAM [7] optimization algorithm. We deduced similar activations for the second CNN model also to make comparison between the two CNN models.

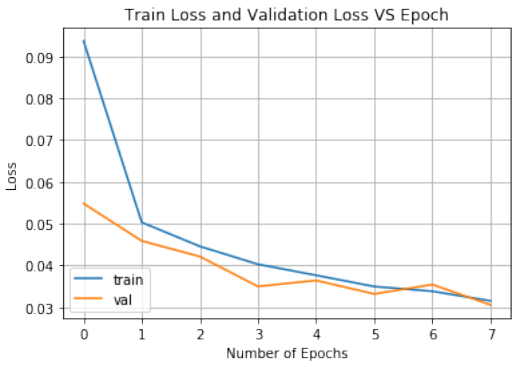


Figure 13: Loss vs Epoch Plot for model (CNN Model 2)

Figure 16: Right Turn Original Image (CNN Model 2)

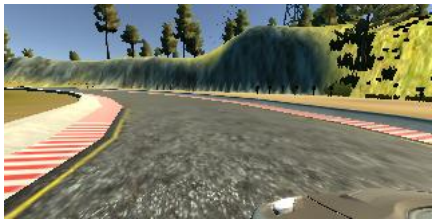
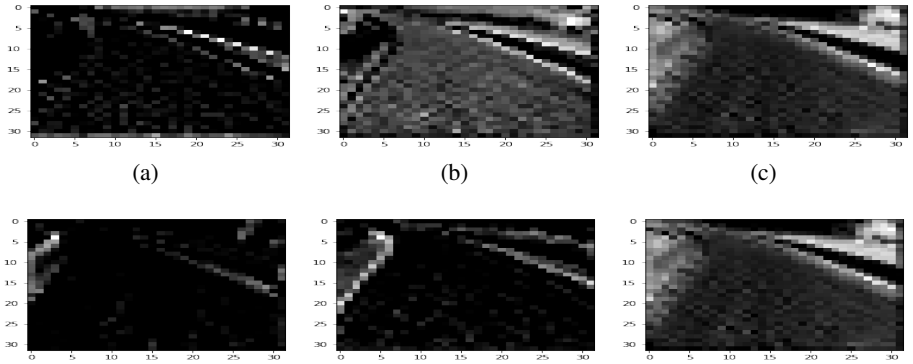


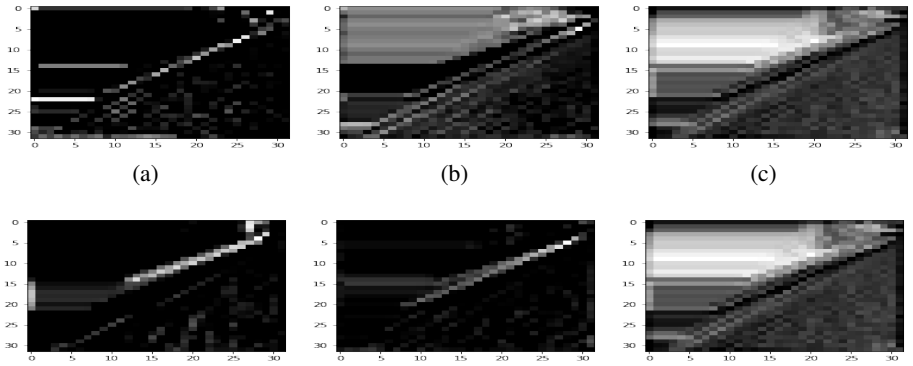
Figure 14: Left Turn Original Image (CNN Model 2)



(a) (b) (c)

(d) (e) (f)

Figure 15: Hidden Layer Activations for Left Turn Image (CNN Model 2)



(a) (b) (c)

(d) (e) (f)

Figure 17: Hidden Layer Activations for Right Turn Image (CNN Model 2)

4.2.3 CNN and LSTM network with 47 million Parameters

We deduced similar plots for CNN and LSTM network, however the internal feature_maps are too small to make any sense, hence plotting activations of just the last two non-output layers instead.

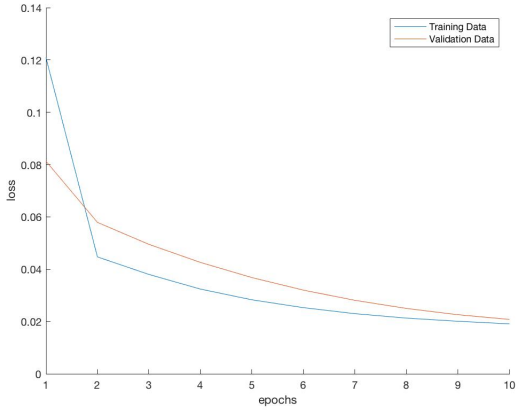


Figure 18: Loss vs Epoch Plot for model (CNN and LSTM)

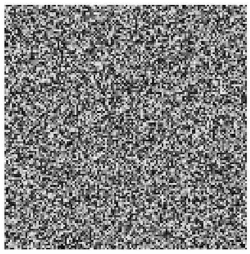


Figure 19: Weights of third last fully connected layer (CNN and LSTM)

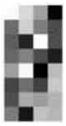


Figure 20: Weights of second last fully connected layer (CNN and LSTM)

Plots of validation loss vs number of epochs is an important way to check the performance of the model as it shows how precise the predicted steering angles are with respect to actual labels. The hidden layer activations for few images were plotted to check what exactly the network was learning from these images.

5 Discussion and Conclusions

The results show that the validation loss was the least for the network with 1 million parameters. However, the best simulated driving experience was noticed with the model with 2 million parameters. This shows that validation loss is not an optimal way to measure the performance of a model on the given dataset. The latter had a validation loss of 0.01 which represents the mean-squared error of the predicted steering angle against the actual values. The simulated driving experience of the two models was very smooth and the car never deviated from its intended path. This clearly shows that

both these models were successful.

The results with the CNN and LSTM model should technically be a lot better as it also considers frames of previous timesteps before prediction, but the performance which we noticed on the simulator was not impressive. This is probably because the model was not trained correctly with respect to the hyper-parameters (such as the number of previous outputs to consider for the LSTM, the number of nodes in the hidden layer, etc.) considering the depth of the network. Also, the dataset used may not be the best for this network. Another potential issue with the implementation of this model in autonomous mode was that there was a delay of a few frames between obtaining the model output and passing this prediction to the simulator via a socket. This proved to be a setback as the model needed accurate temporal data input.

The generalization of the learning depends heavily on the type of training data used. We used the training data generated from the Udacity simulator. Udacity has also released real-life training data which amounts to around 200GB. By using such a dataset, the models can generalize well to any type of tracks. However, we did not use that dataset due to size limitations.

The hidden layer activations for the images clearly show that the network is trying to learn edges or path of the road which is an important factor in predicting the steering angle.

We were not able to obtain meaningful results from our implementation of policy gradient for a deep reinforcement learning model to learn to steer a car using raw pixels. We identified a couple of issues with our implementation. One of the major setbacks that we faced was with the design of the simulator on which we ran our experiments. The simulator does not provide explicit feedback and we had no reasonable way to detect programatically when the car went off track. The simulator also had no reset option on crashing. For the "rollout" procedure, we would have had to manually reset the simulation hundreds of times. We were also unable to restrict the weight changes with higher learning rates. Gradients would either be too high or too low leading to an output of 0 or 1 from the sigmoid activation function in subsequent iterations. With very low learning rates, we were able to train the car on the simulator for a few iterations but the performance was not good, mainly due to the fact that the reward function was not well suited for the learning task. By rewarding the system for every timestep that the car has significant velocity, we were not necessarily motivating it to stay on track. In the simulator, the car could have a significant velocity even when it was off track.

6 Concept and Innovation

We have attempted to train various models to predict steering angle based on simulator images. Using existing datasets, we have trained multiple models. We have implemented a Convolutional Neural Network enhanced with a Long Short Term Memory to enable the model to access the previous decisions to better predict the current steering angle. We have compared the performance of this model to a model that predicts the steering angle just using a deep Convolutional Neural Network. We have also visualized the hidden layer activations in the deep CNN to understand how the car is learning to stay on track in the simulation. We have also experimented with implementing policy gradient for a deep reinforcement learning model to predict steering angle based on images. Although we did not achieve meaningful results with this approach (mainly because the simulator is not designed to give feedback), we suppose that a deeper network with more layers suited for visual inputs may perform better. Also, reinforcement learning may not be the most practical approach to learn to steer a car. This approach is not very feasible when the inputs are the raw pixels from the image frames.

We have been able to demonstrate that an end-to-end level 2 autonomous vehicle can be developed using just visual input cues. With well curated data, the models we have developed would be able to generalize well.

7 Individual Contributions

The project involved a huge amount of experimentation with different models and datasets. We divided the tasks so that everyone was contributing with experimentation.

7.1 Jaskaran Singh

Jaskaran took the responsibility of tuning the first CNN model (1 million parameters). He made sure that the data generated from the simulator was appropriate and trained the model on it. Jaskaran also experimented with different data-augmentation techniques to improve performance and we utilized these techniques across all models.

7.2 Saransh Jain

Saransh was instrumental in setting up the 47 million CNN plus RNN network and training it. Saransh read through the online resources and materialized the complicated network. He trained the model on AWS and tuned it with different datasets. Saransh also implemented the data-augmentation techniques to generate good quality dataset.

7.3 Gautam Nagpal

Gautam worked on implementing the second CNN model (2 million parameters). He trained it on the augmented dataset created earlier and tuned the model to perform well. He ran the models on the simulator on autonomous mode to visualize the performance of each model. He also implemented the hidden layer representations along with Shreyas to visualize model behavior.

7.4 Shreyas Udupa

Shreyas worked with Sudeep to implement the very critical experimentation of the reinforcement learning algorithms. He took inspiration from Andrej Karpathy's blog post and try to incorporate his approach for our use-case. He also contributed towards tuning the CNN plus RNN along with Saransh to try different datasets and time-step combinations.

7.5 Sudeep Nagabhirava

Sudeep worked with Shreyas in implementing the reinforcement learning experimentation. He took the lead on visualizing our models on the simulator by setting up the code to run the model in run-time to predict steering angles from the incoming images from the simulator.

8 References

[1] Bojarski, Mariusz, *End to end learning for self-driving cars.*, arXiv preprint arXiv:1604.07316 (2016). APA

[2] Nvidia Self Driving Car Blog

[3] The Udacity Open-Source Self Driving Car Project

[4] The Udacity Self-Driving Car Simulator

[5] DriveAI- Mastering Autonomous Driving with Deep Learning

[6] Deep Reinforcement Learning : Pong from Pixels

[7]Kingma, Diederik P. and Ba, Jimmy, *Adam: A Method for Stochastic Optimization*, arXiv:1412.6980 [cs.LG], December 2014