Computer Vision (EE554) Final Project Report

Savinay Nagendra (sxn265)

May 1, 2018

Abstract

In this project, Convolutional Neural Networks have been explored in the context of end-to-end learning for self-driving cars. The objective of the project is to achieve mapping between raw road image pixels to steering commands, and hence to achieve autonomous driving. The problem has been formulated as a Regression model, where the steering angle predicted by the model is compared with the human generated steering angle to generate a Mean Squared Error (MSE). This error is back-propagated through the network to update the weights of the convolutional neural network, which extracts important features from the road images and thus, trained to achieve end-to-end learning. A total of six models have been trained on two different datasets. One of the dataset has been generated by me by driving a car in a simulated environment, while the other dataset has been collected by Nvidia's Dave 2 system by driving on actual roads. The images have been pre-processed and augmented to increase the performance of training and to avoid the problem of over-fitting. A comprehensive analysis has been made on the performance of each model. Convolution filter visualization has also been made to visualize the features that the conv nets are learning. The models have been quantitatively compared and the corresponding results have been compared.

Contents

1 Project Title

Behavioral Cloning - Steering Angle Prediction for Self Driving Cars

2 Problem Statement

To achieve end-to-end learning for self-driving cars by training a Convolutional Neural Network to map raw pixels from a single front-facing camera directly to steering commands.

The objective of this project is to use a tensor of image frames of roads captured by three cameras (left, right and center) mounted behind the windshield of an automobile and corresponding steering commands from a manual driver, as training data to train a Convolutional Neural Network. The system is modelled as a **Regression** problem. The trained model of the network is used to predict steering angles for the automobile on unseen road image frames, hence achieving end-to-end self driving.

3 Introduction

Autonomous machines has always been a significant topic of interest in Pattern Recognition and Computer Vision. Prior to the widespread adoption of Deep Learning, most pattern recognition tasks were performed using an initial stage of hand-crafted feature extraction followed by a classifier. With the break of Big-Data, higher computation power and the revolutionizing technology of deep learning, the features are automatically learnt by networks called Convolutional Neural Networks (CNNs). The CNN approach is especially powerful in image recognition tasks as 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.

From the pioneering ALVINN [7] from Dean A. Pomerleau et al., self driving has been explored from various domains. But, a reliable and scalable system could not be built at a larger scale until the advent of Convolutional Neural Networks. With no feature extraction and minimum data pre-processing, end-to-end learning is achievable using CNNs to build robust systems.

In this project, Convolutional Neural Network architectures have been explored in the context of Autonomous driving to achieve end-to-end learning by giving raw road images as inputs and getting the steering commands as output, with minimum human training. Since convolutional neural networks extract features from images, the network once trained on a set of images will perform well on similar unseen data as well, which is one of the greatest advantage of using CNNs.

Two datasets have been used in this project: (1) Simulated dataset collected by me, while driving a car on the Udacity simulator powered by unity software. (2) Actual road images collected by Nvidia's Dave 2 system. Three cameras - Left, Right and Center, are installed behind the windshield of an automobile, which is driven on various types

of roads in various lighting conditions. The images from the three cameras as well as the corresponding steering angles are collected at 20FPS. The images, along with their corresponding steering angles are passed into the Convolutional Neural Network as a forward pass. The error in the steering angle between the human driver and the machine prediction is propagated through the network in the backward pass to update the weights of the network through Back-propagation optimization. Three models are trained on Simulated dataset and the other three on the actual dataset. The models trained on the simulated dataset have been tested on the same simulator, where the model drives the car in autonomous mode. Data, before sending into the network is augmented and pre-processed, as discussed in the sections below. The CNN architecture proposed by Bojarski et al. [1] is used as the baseline model since this is the current state of the art on the Nvidia dave 2 dataset. A novel method of extracting temporal information with lower computational expense has also been discussed. The intuition is that this method will be robust in preserving the correlation between steering angles in a block of two shifted video frames. Finally, transfer learning is implemented by Fine tuning ResNet 50 architecture on Keras platform. A comparative analysis of the performance from all the three networks is done by quantitative validation.

4 Goals

1. To train different convolutional neural network architectures, such as 2D CNNs, Networks to extract Temporal information and Transfer Learning using pre-trained networks on augmented and pre-processed road image data sets

2. To evaluate the performance of these networks on a autonomous driving simulated environment

3. To make a quantitative comparative analysis of these networks with the current state-of-the-art architecture.

Hence, the ultimate goal is to empirically achieve end-to-end learning through Convolution Neural networks in self-driving cars by using raw image data to predict steering commands.

5 Motivation and Challenges

Autonomous driving has been a research problem for several decades, where researchers have been trying to achieve a robust and efficient system architecture for self-driving cars.

Why Self-driving?:

1. According to road statistics, 2.6 million people are injured in vehicles every year.

2. Chances of a person dying in a car accicdent in a particular year is approximately 0.011%.

3. Top 4 causes for road accidents are **Distraction, Speeding, Drunk-Driving, Recklessness**.

Hence, research in this domain is significantly important. Many companies in the tech world are working on autonomous driving and trying to implement a robust and reliable system. Potential research topics are Object detection using camera data, obstacle avoidance using radar imaging and sensing, 3D mapping using LIDAR data, Steering angle prediction.
Steering command prediction is one of the most important research topics in the domain of autonomous driving as it involves end-to-end learning and decision making, around which other computer vision applications are integrated.[5][4]

Challenges of Self-driving technology:

There are many challenges and problems to be solved in this domain, but two challenges that are addressed in this project are listed below:

1. Driving safely despite unclear lane markings
2. The ability to operate safely in all weather conditions lighting conditions.

6 Related work

6.1 Related Approach on Pattern Recognition Approach

These are some of the noted systems in the history of self-driving technology for steering command prediction:

1. Autonomous Land Vehicle in a Neural Network (ALVINN) was the pioneering technology in Autonomous End-to-End learning by Pomerleau in 1989. The model used only a fully-connected network and drove on public roads.[7]
2. Defense Advances Research Projects Agency (DARPA) Autonomous Vehice (DAVE), a sub-scale radio control (RC) car driving through a junk filled alley way, 2006
3. NVIDIA's Dave 2 using Convolutional Neural Networks, 2016[6]

Recently, more attempts on using deep CNNs and RNNs to tackle the challenges of video classification, scene parsing and object detection have stimulated the applications of more complicated CNN architectures.

6.2 Related work on the dataset

In 2016, Bojarski et al.[1] (NVIDIA team) have used CNNs to train input camera images to predict the steering wheel angle. They have formulated steering angle prediction as a regression problem and have used three cameras to augment the dataset during training, and thus generalize learning. Correction factor is added to the steering angles corresponding to images collected from left and right cameras. Data augmentation techniques such as adding random rotations to the steering angle have also been applied. Deep network architecture uses five convolutional layers followed by five fully-connected layers.

Udacity launched a self-driving car challenge for using camera data to predict steering wheel angle. Udacity[2] has open sourced its simulator, which can be used to collect data, as well as the actual driving dataset collected for this challenge by NVIDIA Dave 2. Many solutions by teams participating in this challenge used CNN architectures similar to the one used by Bojarski et al.[1] They report deeper networks (such as ResNet and VGGNet) preform worse for this regression problem than shallower architecture. The winning team used a combination of 3D CNNs and LSTMs to achieve mean error of 2.6 degrees error.

Gaurav Bansal et al.[3] (Toyota ITC Team) have also explored this problem using the udacity dataset from a different challenge. They have explored this problem as a regression problem as well and have devised image sharing between Vehicle to Vehicle (V2V) communications. They achieve a mean error of 3.5 degrees.

However, the subset of datsets used by all of these approaches is not specified. Also, the network architectures and hyper-parameters used by the these researchers are not evident, hence making this problem open to pruning to achieve better results.

7 State-of-the-art

As an extension to DAVE and with an inspiration from ALVINN, **DAVE-2** was built by NVIDIA to create a robust system for driving on public roads with end-to-end learning technology using CNNs. Currently, the performance of the convolutional network architecture built by NVIDIA on DAVE-2 is the State-of-the-art on this dataset..

NVIDIA DAVE 2

The results from this paper [1] have been used as the baseline for this project. Dave 2 model from NVIDIA has been used to train a convolutional neural network to map raw pixels from a single front-facing camera directly to steering commands.This model has achieved the best result on this dataset, with a Mean Squared error of approximately **10%**. With minimum training data from humans, the system learns to drive in traffic on local roads with or without lane markings and on highways. It is also able to operate in areas with unclear visual guidance such as in parking lots and on unpaved roads.
Since a convolution neural network has been used, the system automatically learns internal representations of the necessary processing steps such as detecting useful road features with only the human steering angle as the training signal.

NVIDIA DevBox and Torch 7 have been used for training and NVIDIA DRIVE PX self-driving car computer, running Torch 7 has been used for determining where to drive. The system operates at 20 Frames per second (FPS).

Figure 1 shows a simplified block diagram of the collection system for training data for Dave 2. The training data is a tensor of images sampled from the video from the three cameras and corresponding steering angles. The data has been time stamped. Data augmentation such as random shifts and random rotations have been performed through viewpoint estimation to enable the car to learn from mistakes and not drift off roads.

Figure 2 shows the model diagram used for training. Backpropagation algorithm is used to minimize the root mean squared error between the input from human driver and

output from the CNN. Figure 3 shows the model used for testing, where only the image from center camera has been used to predict the steering angle. 4 shows the CNN architecture used by NVIDIA Dave 2.

The data flow in the block diagrams 2 and 3 has been used in all the six models in this project.

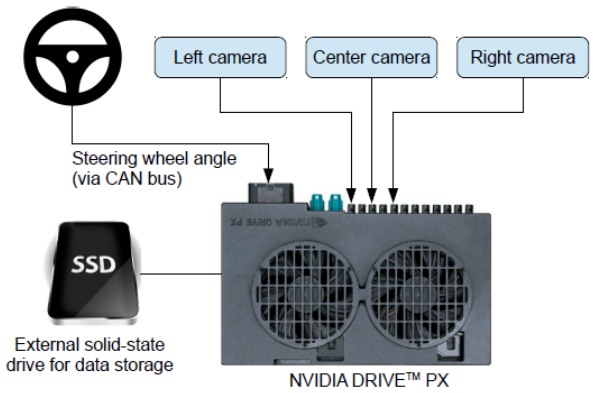


Figure 1: High level view of Data Collection System

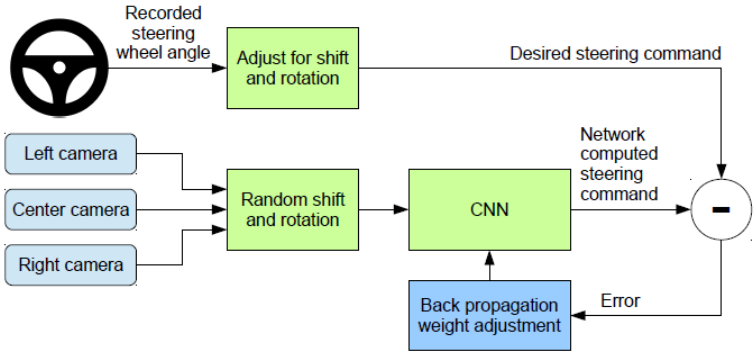


Figure 2: Training Model for the Neural Network

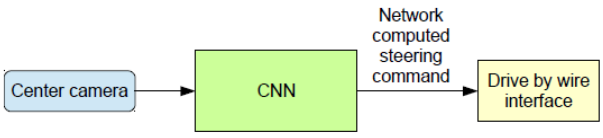


Figure 3: The trained network is used to generate steering commands from a single front-facing center camera

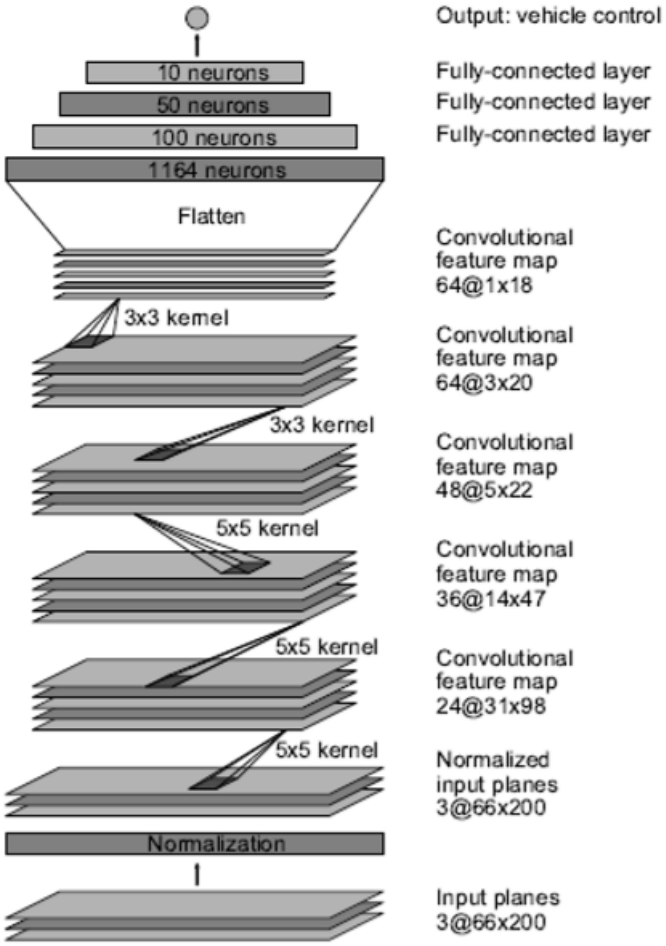


Figure 4: NVIDIA Dave 2 Network Architecture

8 Dataset

Two datasets have been used in this project:

8.1 Simulated Dataset

8.1.1 Data Collection

Data is collected using Udacity self driving simulator, powered by unity software. There are two modes of operation - **Training mode** and **Autonomous mode**. The data is collected in training mode by manually driving the simulated car around the track for **6 laps**. This simulator is built to mimic Nvidia's Dave 2 system. There are 3 front facing cameras on the simulated car - Left, Right and Center. The images from the three cameras and the corresponding steering angles are collected at **20** Frames per second (FPS), which are stored in a csv log file. This csv log file has been used as a Data frame to train the neural networks. Figure 5 shows the simulator environment homepage, depicting training and autonomous modes. Figure 6 shows the images from left, center and right cameras.

Figure 5: Simulated environment homepage

Figure 6: Simulated Images from left, center and right cameras (from left to right)

8.1.2 Data Statistics

1. A total of 24108 images have been collected.
2. Each image is of size $160 \times 320 \times 3$
3. Data is divided into train and validation sets in the ratio 80:20 for all networks
4. The histogram of steering angles is shown in the figure 7. It can be seen that most of the steering angles in the data are distributed around zero, as the car is driven straight for most of the time during data collection. In other words, the data is highly biased and cannot be directly used for training.

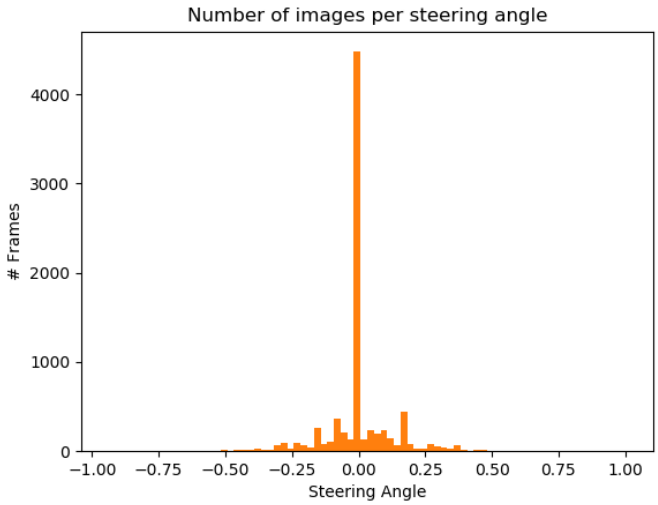


Figure 7: Histogram of steering angles before data augmentation

8.2 Actual Dataset by Nvidia

8.2.1 Data Collection

The dataset is collected by NVDIA Dave 2 system by driving in various traffic and lighting conditions. Three cameras - Left, Right and Center, installed behind the wind shield of an automobile is used to collect image frames and corresponding steering angles at **20** FPS. The car is driven for a total of **28.23** minutes, whose split up can be seen below. Training images have been extracted from 5 different videos recorded by the automobile:

1. 221 seconds, direct sunlight, many lighting changes. Good turns in beginning, discontinuous shoulder lines, ends in lane merge, divided highway

2. discontinuous shoulder lines, ends in lane merge, divided highway 791 seconds, two lane road, shadows are prevalent, traffic signal (green), very tight turns where center camera cant see much of the road, direct sunlight, fast elevation changes leading to steep gains/losses over summit. Turns into divided highway around 350s, quickly returns to 2 lanes.

3. 99 seconds, divided highway segment of return trip over the summit

4. 212 seconds, guardrail and two lane road, shadows in beginning may make training difficult, mostly normalizes towards the end

5. 371 seconds, divided multi-lane highway with a fair amount of traffic

A set of sample images can be seen in figure 8.

Figure 8: Center camera Images collected by Dave 2, each image shows a different lighting condition.

The steering angles are modelled as $\frac{1}{r}$, where r is the turning radius to make the system independent of cat geometry. The data was extracted from ROSbag files using a docker interface. Only center images have been used for training Models 5, while all the images have been used for training Models 4 and 6.

8.2.2 Data Statistics

1. There are a total of 101397 images

2. Data is split into train and validation sets in the ratio 80:20

3. Histogram of steering angles and concatenated steering angles over the five videos have been shown in figures 9 and 10 respectively. It can be observed that maximum steering angles are centered around zero, which makes this data set also biased and hence, it cannot be fed into the networks for training.

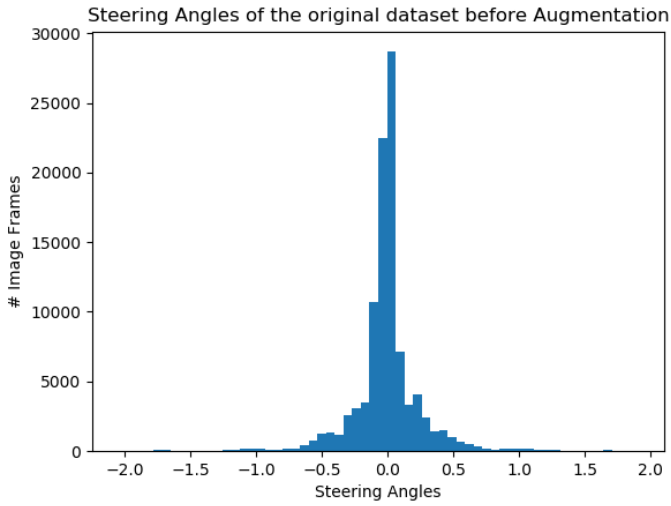


Figure 9: Histogram of steering angles before augmentation

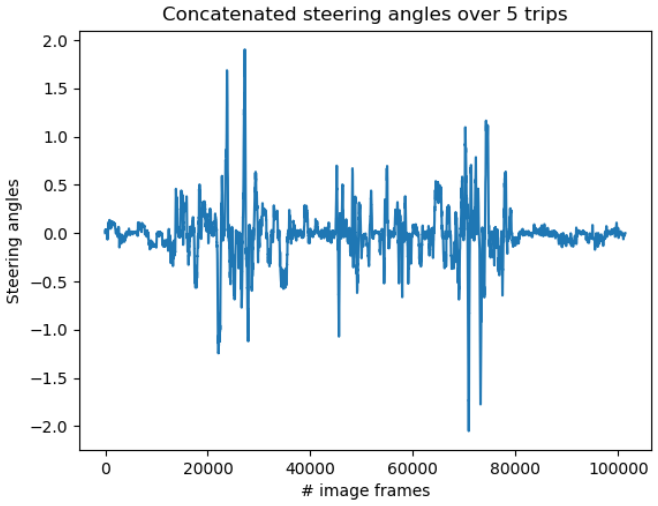


Figure 10: Concatenated steering angles over 5 videos

9 Methods

9.1 Data Augmentation

9.1.1 Simulated Dataset

Data Augmentation for Model 1

Data Augmentation is necessary to teach the car to recover from edges of the roads and avoid over-fitting. The augmentations listed in this section are taken from the paper [1] The following augmentations are implemented on randomly sampled images in batches of 64 through an Image batch generator, which helps in bringing a diversity factor in training:

1. **Horizontal flip**: The image is flipped along the y-axis, and the corresponding steering angle is reversed, hence achieving a reflection transformation.

2. **Random Translation**: The image is shifted randomly in the horizontal and vertical axes, and corresponding steering angles are corrected.

3. **Random shadows**: Some part of the images are darkened to create a shadow effect, so that the model becomes robust to darkness.

4. **YUV transformation**: To account for the defects while capturing an image, images are converted to YUV domain. This increases the stability of the model.

5. **Cropped and Resized**: The images are cropped from the center and resized to $66 \times 200 \times 3$.

Finally, all the augmentations are combined with a random permutation before feeding into the model using a threaded keras generator. A Correction factor of 0.25 for steering angles corresponding to the left and right images have been added to and subtracted from the lest and right camera images respectively to equalize the histogram and make the model unbiased. Figure 11 shows the above augmentations applied on a random image from the center camera.

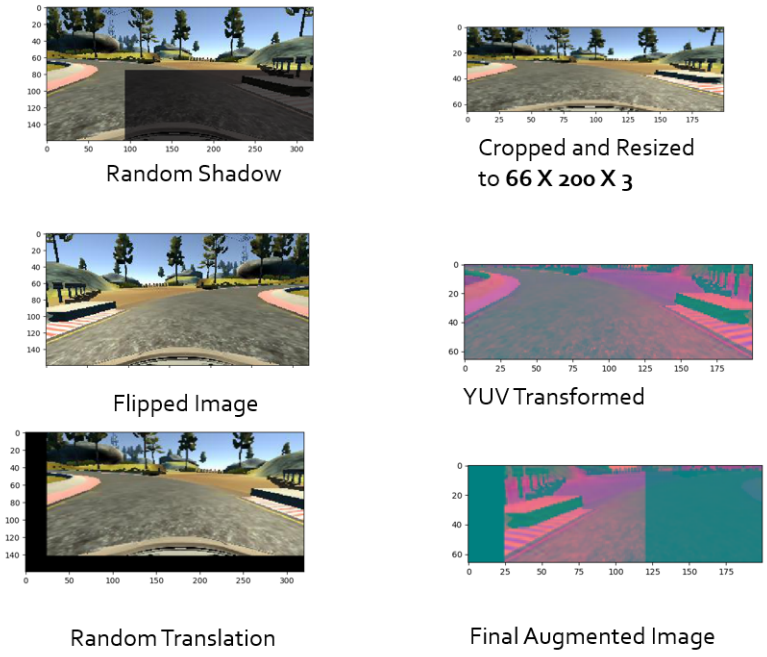


Figure 11: Data Augmentations

Data Augmentations for Models 2 and 3

The following data augmentations were implemented for Models 2 and 3:

1. **Random Channel Shift**: One of the tricks in augmenting an image is to shift its color channel by a small fraction. This function randomly shifts input image channel in the range [-0.2, 0.2] by using keras image processing apis.

2. **Horizontal Flip**: The image is flipped along the vertical axis and the corresponding steering angle is reversed.

3. **Random Translation**: Image is randomly translated in the vertical and horizontal axes. The corresponding steering angles are also corrected.

4. **Brightness Change**: The brightness of the images have been changed in HSV domain.

5. **Crop and Resize**: Finally, the images have been cropped and resized to $64 \times 64 \times 3$.

Compared to the first model, the number of augmentation methods applied on these two models is lesser. The images are randomly sampled and converted to batches by a Batch generator, with batch size 64. Figures 12 shows the augmented images and figure 13 shows a sample batch of 10 images, with all augmentations combined. The same Correction factor for left and right cameras has been used, as that of Model 1.

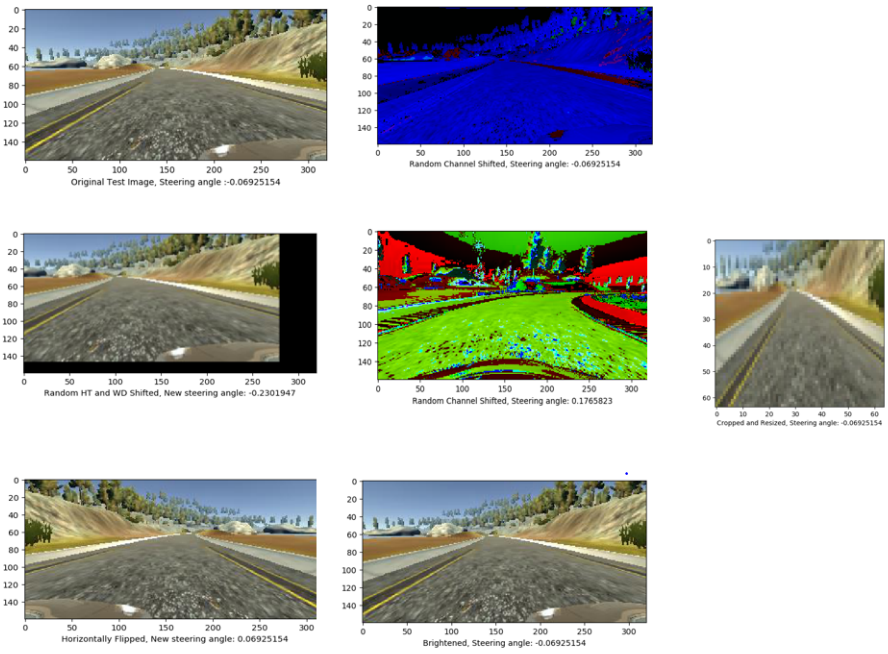


Figure 12: Different Augmentations

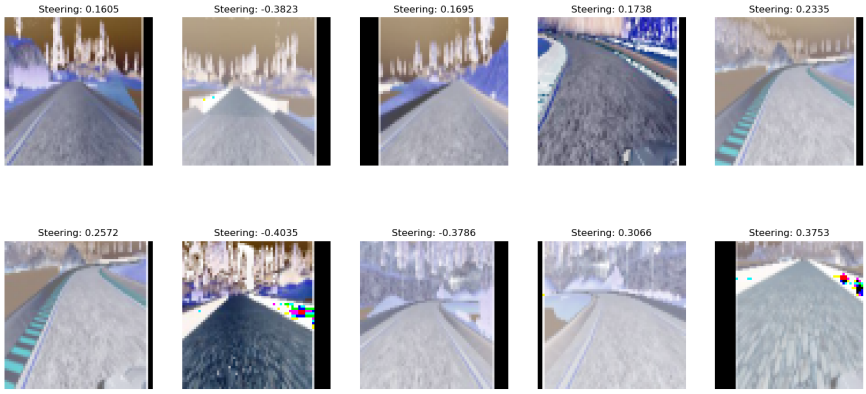


Figure 13: Sample batch with combined augmentations

The histogram of steering angles before data augmentation was shown in the figure 7. Figure 14 shows the histogram of steering angles after the above augmentations were applied . It can be observed from figure 7 that the steering angles before augmentation were concentrated around 0 degrees, as most of the driving is on a straight road, without much curvature in the roads. After the steering angle correction factor has been added and the Data Augmentation has been carried out, it can be observed from figure 14 that the steering angles are more distributed, hence making the model unbiased.

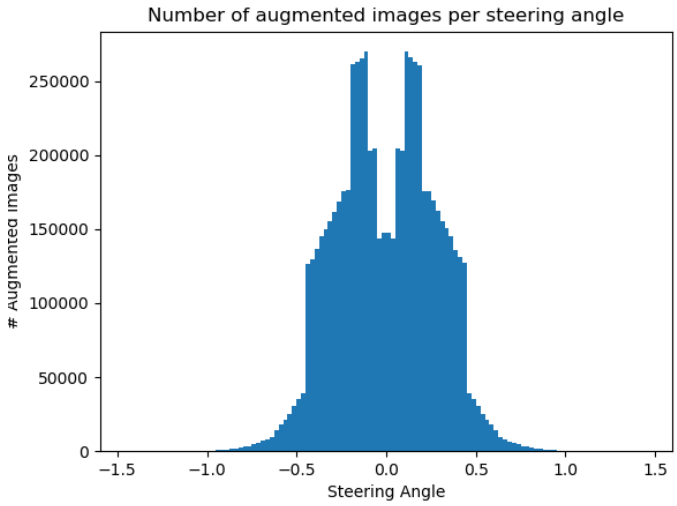


Figure 14: Steering Angles after Data Augmentation

9.1.2 Actual Dataset by Nvidia

This dataset contains information about driving on different types of roads under various lighting conditions. A sample of these images is shown in figure 8. It can be observed that the images inherently have different brightness, contrast and shadow augmentations. So, care has been taken not to over-augment data, which might lead to poor performance of the networks on this dataset. The three Nvidia models Model 4, 5 and 6 were trained on a small random subset of data containing 500 images with three different sets of Augmentations:

1. **Low:** Only Crop and Resize to 66×200 for models 4 and 5 and to 224×224 for Model 6. So, essentially, the images do not undergo any augmentation a n d are fed directly in this set.

2. **Moderate**: Random Flips, Random Translations, Crop and Resize. 30% of the randomly chosen images are augmented and the rest are sent in directly. All images are cropped an d resized.

3. **High**: Brightness shift, Translations, Flips, YUV transformation and Crop and Resize. Again 30% of the images undergo random augmentations and the rest are sent in directly.

The model with **Low** augmentation gave the least averaged error on this subset of data. The table of averaged errors is shown in Table 15

LOW	MODERATE	HIGH
0.019	0.030	0.048

Figure 15: Table of averaged errors for different sets of Augmentations

Hence, **Low** augmentation is used for Models 4, 5 and 6.

It can be observed from figure 9 that the steering angles before augmentation are concentrated around the origin, which makes the data highly biased. The car will never learn to recover from sharp turns if this data is used to train the networks. In addition to **Low** augmentation, to make the distribution of steering angles unbiased, the steering angles in the range -0.05 to 0.05 have been removed. Figure 16 shows the histogram of steering angles after augmentation for the actual Nvidia dataset.

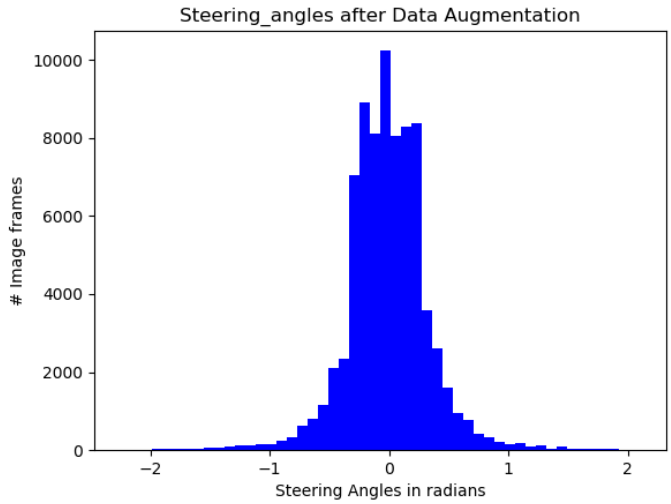


Figure 16: Histogram of steering angles after augmentation

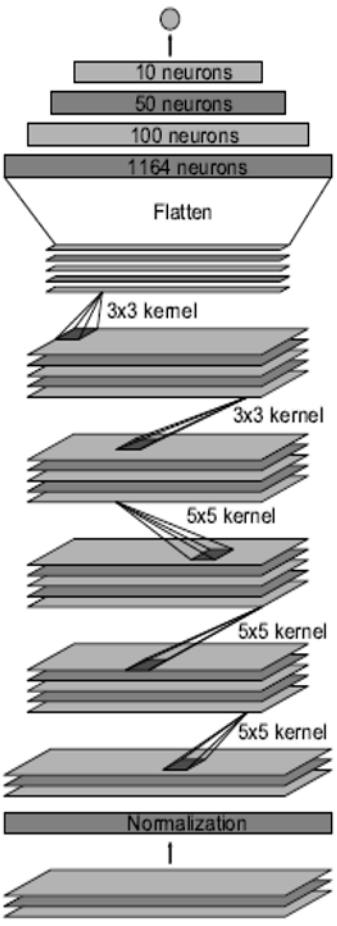
It is very clear from figure 16 that the steering angles distribution is no longer biased and can be used to train the networks.

For Models 4 and 6, images from all three cameras have been used, while for model 5, images only from the center camera have been used.

9.2 Network Architectures on Simulated Dataset

9.2.1 Model 1: Original Nvidia Model

The network is shown in image 17. This architecture was constructed by Bojarski et.al and his team at Nvidia [1], which is the state of the are on this dataset, as mentioned in 7 with the best test accuracy. All the data augmentations suggested in the paper have been used for this model. The hyper-parameters such as learning rate, steering angle correction factors etc. have not been provided in the paper. These have been formulated using cross-validation on smaller subsets of data.



Layer (type)	Output Shape	Param #
lambda_1 (Lambda)	(None, 66, 200, 3)	0
conv2d_1 (Conv2D)	(None, 31, 98, 24)	1824
conv2d_2 (Conv2D)	(None, 14, 47, 36)	21636
conv2d_3 (Conv2D)	(None, 5, 22, 48)	43248
conv2d_4 (Conv2D)	(None, 3, 20, 64)	27712
conv2d_5 (Conv2D)	(None, 1, 18, 64)	36928
dropout_1 (Dropout)	(None, 1, 18, 64)	0
flatten_1 (Flatten)	(None, 1152)	0
dense_1 (Dense)	(None, 100)	115300
dense_2 (Dense)	(None, 50)	5050
dense_3 (Dense)	(None, 10)	510
dense_4 (Dense)	(None, 1)	11

Total parameters: 252,219

Figure 17: Model 1 architecture

The model has 252,219 trainable parameters, with a single drop out layer. The input layer takes an image of size $66 \times 200 \times 3$ RGB. The images have been normalized to pixel values between [-1 1], before passing into the convolution filters. There are 5 convolution filters, each with no padding and a stride of (2,2). Finally, the output of Conv5 goes into a flattening layer, and further passed into 4 Fully connected networks, with the final layer having a single neuron. This neuron outputs the steering angle, calculated by the model.

The Mean Squared error between the model output and human steering input is calculated, which is used to Minimize this error by the process of Back-Propagation

Training Details

1. Loss Function: Mean Squared Error (MSE)
2. Optimizer: Adam
3. Epochs: 20
4. Learning rate: 0.0001
5. Samples per epoch: 5000
6. Batch Size = 64

9.2.2 Model 2: Changed Nvidia Model

Model 1 architecture has been changed to take the inputs of shape $64 \times 64 \times 3$. Further, this model uses different augmentations and also has additional drop-out layers. Other than these two, all the details of the models are the same as Model 1. From figures 22 and 23, it can be observed the original Nvidia Model is over-fitting the data. The intuition for changing the model is that the Nvidia Model was originally designed to handle about 1 lakh images, each of size $640 \times 480 \times 3$. So, for the simulated dataset, a model with lesser parameters must give better results. Figure 18 shows the architecture of Model 2, with 143,419 trainable parameters.

Training Details

1. Loss Function: Mean Squared Error (MSE)
2. Optimizer: Adam
3. Epochs: 10
4. Learning rate: 0.0001
5. Samples per epoch: 1000
6. Batch Size = 64

lambda_1 (Lambda)	(None, 64, 64, 3)	0
Conv1 (Conv2D)	(None, 30, 30, 24)	1824
Conv2 (Conv2D)	(None, 13, 13, 36)	21636
conv2d_1 (Conv2D)	(None, 5, 5, 48)	43248
dropout_1 (Dropout)	(None, 5, 5, 48)	0
conv2d_2 (Conv2D)	(None, 3, 3, 64)	27712
dropout_2 (Dropout)	(None, 3, 3, 64)	0
conv2d_3 (Conv2D)	(None, 1, 1, 64)	36928
dropout_3 (Dropout)	(None, 1, 1, 64)	0
flatten_1 (Flatten)	(None, 64)	0
dense_1 (Dense)	(None, 100)	6500
dense_2 (Dense)	(None, 50)	5050
dense_3 (Dense)	(None, 10)	510
dense_4 (Dense)	(None, 1)	11

Total parameters: 143,419

Figure 18: Changed Nvidia Model

9.2.3 Model 3: Comma.ai Model

Model 3 has been taken from the architecture by Comma.ai, which gave good results on the original Nvidia Dave 2 dataset.The architecture of the model has been portrayed by figure 19. The model has 3 convolution layers, but the number of filters in each layer is significantly higher compared to Models 1 and 2. It has two drop out layers 1 Fully connected layer, FC-512 and finally a regression layer with one neuron which outputs the steering angle. It is a smaller network in depth when compared to Models 1 and 2, but has 592,497 trainable parameters, which is significantly higher than Models 1 and 2.

Training Details

1. Loss Function: Mean Squared Error (MSE)
2. Optimizer: Adam
3. Epochs: 30
4. Learning rate: 0.0001
5. Samples per epoch: 5000

6. Batch Size = 64

Layer (type)	Output Shape	Param #
lambda_1 (Lambda)	(None, 64, 64, 3)	0
Conv1 (Conv2D)	(None, 16, 16, 16)	3088
Conv2 (Conv2D)	(None, 8, 8, 32)	12832
Conv3 (Conv2D)	(None, 4, 4, 64)	51264
flatten_1 (Flatten)	(None, 1024)	0
dropout_1 (Dropout)	(None, 1024)	0
elu_1 (ELU)	(None, 1024)	0
FC1 (Dense)	(None, 512)	524800
dropout_2 (Dropout)	(None, 512)	0
elu_2 (ELU)	(None, 512)	0
output (Dense)	(None, 1)	513

Total parameters: 592,497

Figure 19: Comma.ai Architecture

9.3 Network Architectures on Actual Nvidia Dataset

9.3.1 Model 4: Original Nvidia Model

The architecture of Model 4 is shown in figure 17. This architecture is directly taken off the paper [1]. The augmentations applied on this network have been explained previously. The images from all the three cameras have been used. The left and right steering angles are appended with a steering angle correction to make the distribution of steering angles unbiased.

Training Details

1. Loss Function: Mean Squared Error (MSE)
2. Optimizer: Adam
3. Epochs: 10
4. Learning rate: 0.0001

5. Samples per epoch: 5000

6. Batch Size = 64

9.3.2 Model 5: Nvidia Model for Temporal information extraction

In prediction problems involving a time frame, it becomes crucial for the system to extract useful time based information, which makes the accuracy of predictions higher. This model represents a novel technique to extract useful temporal information without the need for complex, computationally expensive networks such as 3D CNNs or RNNs. Even though these networks might have a superior performance when compared to the model I am suggesting, considering the computational expenses, it is definitely worth considering it. This model is inspired by the idea of Vehicle-to-vehicle (V2V) communications.Automotive industries all over the world have been working on V2V communication technologies for many years. General Motor's Cadillac, Toyota Lexus are some of the many models which have this technology in them. These deployments allow 300 to 400 bytes of messages at a periodic 10Hz frequency with a transmission rate of 300-500 metres.Additionally, cellular industries are planning to launch 5G technologies with automotive use cases. 5G based V2V communications will enable vehicles to share huge amounts of data (Gbps and higher), with high reliability. This would allow vehicles to share raw camera images, which can be used for deep learning applications.

In this model, I have simulated a virtual vehicle that is 2 seconds ahead of the ego vehicle. 2 second from the 2 second rule of driving that is common in the United states. Since i do not have data from 2 vehicles, I have simulated this situation. The image at time [t+40] is given to the network along with the image at time [t].

The error

$$error = I_{[t]} - I_{[t+40]} \tag{1}$$

is fed into the network shown in 17. This model thus does not require the complexity of a 3D CNN or a RNN. It also performs better than the Original Nvidia Model shown in 17

9.3.3 Model 6: ResNet-50 (Transfer Learning)

Fine Tuning, a method of Transfer learning is applied on the dataset using ResNet 50 base model pre-trained on ImageNet weights. All the Fully connected layers of the base network have been removed. The output of the last convolution layer is given to a Flatten() layer to vectorize the features. Four fully connected layers (Dense layers) have been attached following the flatten layer. The last FC-1 layer is the regression layer. Further, except the last 20 layers, all the other layers of the base network have been frozen. The total number of trainable parameters are **9,079,169**. The model has been portrayed in the figure 20.

Layer (type)	Output Shape	Param #
resnet50 (Model)	(None, 1, 1, 2048)	23587712
flatten_1 (Flatten)	(None, 2048)	0
dense_1 (Dense)	(None, 512)	1049088
dropout_1 (Dropout)	(None, 512)	0
dense_2 (Dense)	(None, 256)	131328
dropout_2 (Dropout)	(None, 256)	0
dense_3 (Dense)	(None, 64)	16448
dropout_3 (Dropout)	(None, 64)	0
dense_4 (Dense)	(None, 1)	65

Total params: 24,784,641
Trainable params: 9,079,169
Non-trainable params: 15,705,472

Figure 20: ResNet 50 Architecture

10 Results

The results for the above models have been portrayed in this section.

10.1 Simulated Dataset

These are the results of Models 1, 2 and 3.

10.1.1 Model 1

The model was trained for 20 epochs, beyond which it started to over-fit the data, at a learning rate of 0.0001. Training and Validation losses have been calculated from the dataset. Test Loss has been formulated by driving the simulated car in autonomous mode around the track for one lap, and has been averaged over 3 runs. The results at the end of 20 epochs have been portrayed below:

	Training Loss	Validation Loss	Test Loss
MSE	0.0224	0.0231	0.049
Steering angle error (degrees)	8.58	8.71	12.689

Figure 21: Table of Results - Model1

It can be observed that the Training and Validation Losses decrease with epochs, but the loss is not converging with batches, and it is overshooting.

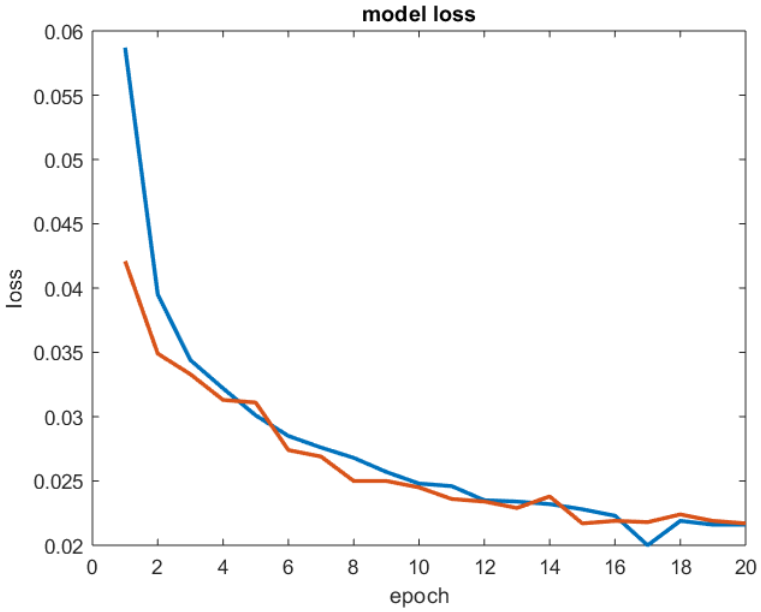


Figure 22: Model Loss vs epochs

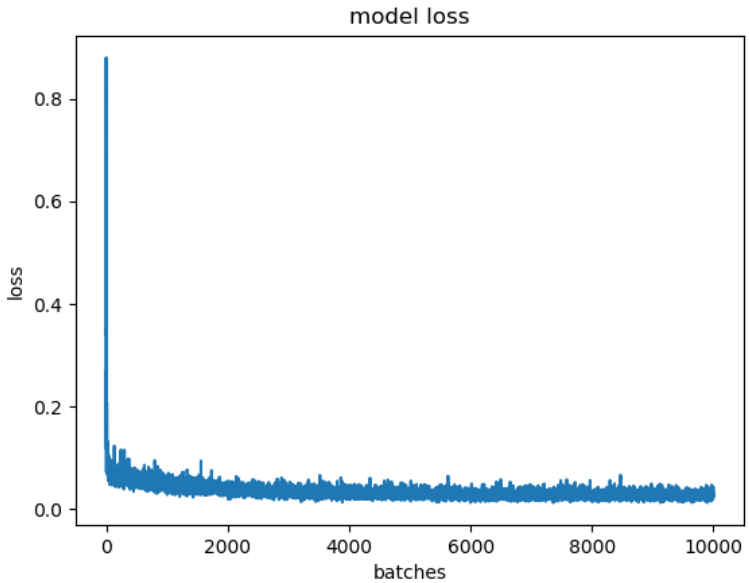


Figure 23: Model Loss vs Batches

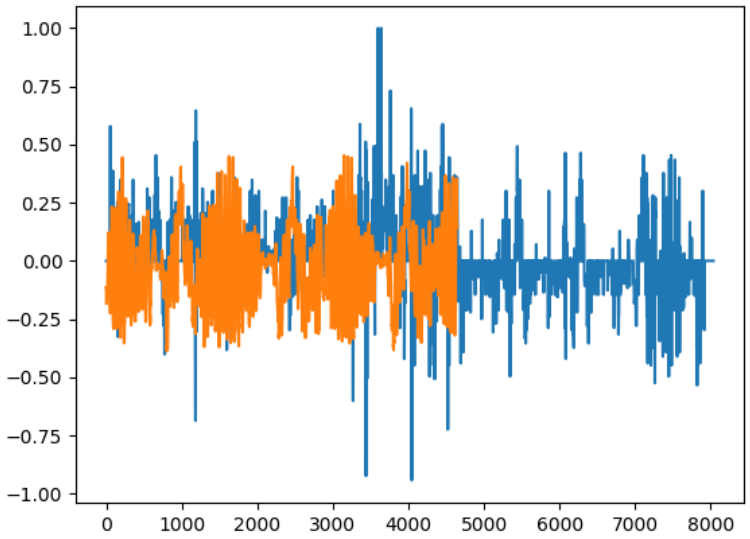


Figure 24: Actual(Blue) vs Predicted(Orange) Steering Angles during Testing

10.1.2 Model 2

Model 2 has been trained for 10 epochs, after which it started to over-fit. The model has been trained for 10 epochs with a learning rate of 0.0001 and a batch size of 64, with 1000 training samples per epochs and 100 validation samples per epoch. It can be observed from figures 26 and 27 that the model loss is going down gradually, which means the model is not over-fitting. This is reconfirmed by figure 28, where the model predicted steering angles is following the human steering angles without over-shooting. The visualizations of convolution layers 1 and 2 are shown in figure 29. It can be observed that the edges are highlighted, which means that the model is learning to drive inside the boundaries of the lane. The results at the end of 10 epochs have been portrayed below:

	Training Loss	Validation Loss	Test Loss
MSE	0.0331	0.0314	0.0283
Steering angle error (degrees)	10.42	10.15	9.638

Figure 25: Table of results Model 2

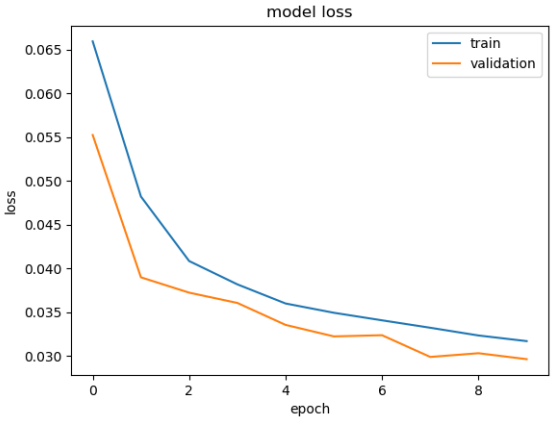


Figure 26: Model loss vs epochs

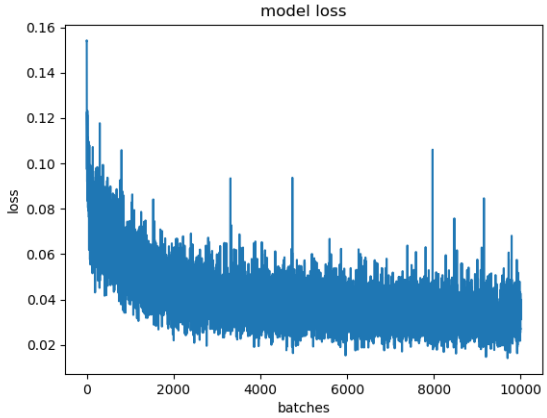


Figure 27: Model loss vs batches

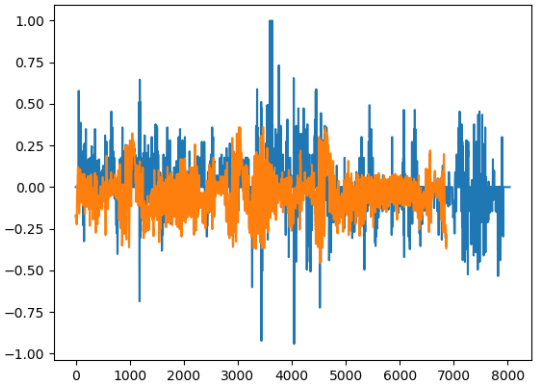


Figure 28: Actual (blue) vs Predicted (orange) steering angles

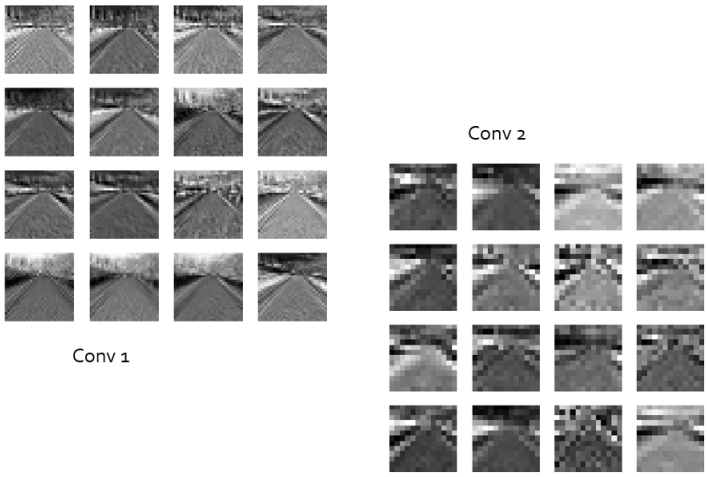


Figure 29: Conv1 and Conv2 visualizations

10.1.3 Model 3

The results of this model have been portrayed in figure **??**. The convolution layer visualizations have been shown in figure 32. It can be observed that the training and validation losses are smaller than that of Model2. But, the predicted steering angles are over-shooting. The model has been trained for 30 epochs at a learning rate of 0.0001 and a batch size of 64, with 5000 samples per epoch, after which it was over-fitting the data. The results at the end of 30 epochs are as follows:

	Training Loss	Validation Loss	Test Loss
MSE	0.0172	0.0176	0.03297
Steering angle error (degrees)	7.51	7.6	10.40

Figure 30: Table of Results - Model 3

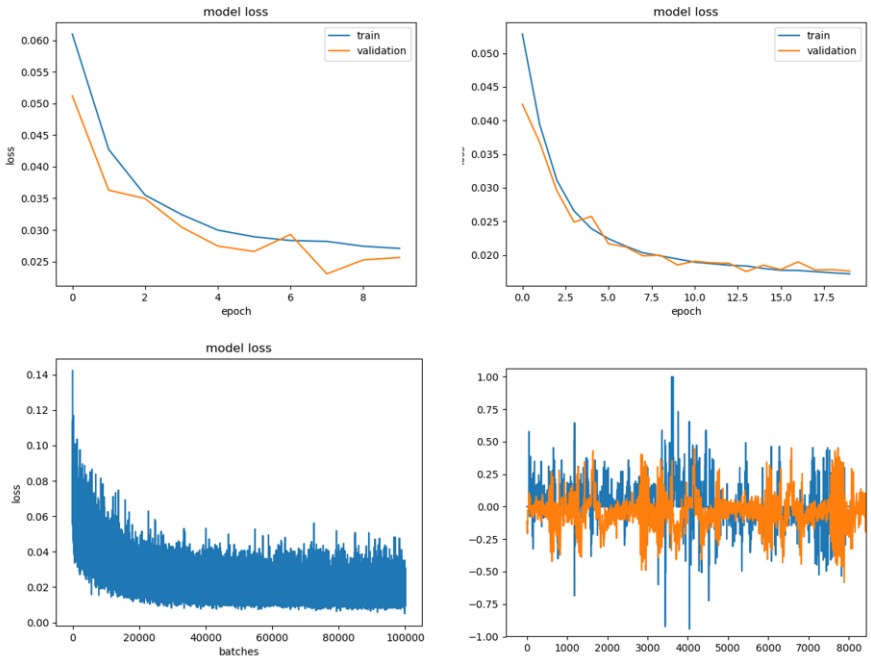


Figure 31: Model 3 results

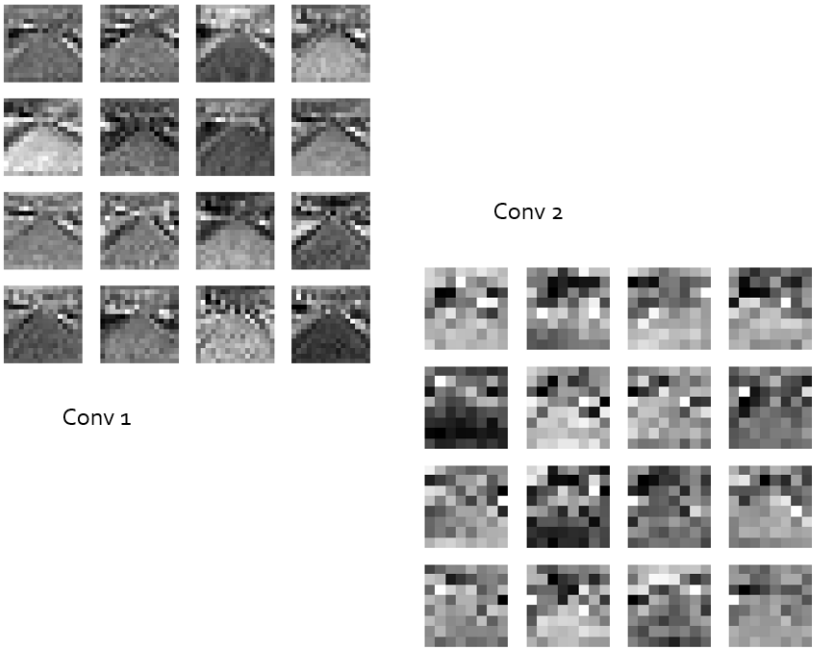


Figure 32: Conv1 and Conv2 visualizations

10.2 Actual Nvidia Dataset

10.2.1 Model 4

The results of this model have been portrayed in figure 34. The convolution layer visualizations have been shown in figure 38. The model has been trained for 10 epochs at a learning rate of 0.0001 and a batch size of 64, with 5000 samples per epoch, after which it was over-fitting the data. The results at the end of 10 epochs are as follows:

	Training Error	Validation Error
MSE	0.0208	0.016
Steering angle error (degrees)	8.263	7.24

Figure 33: Table of results - Model4

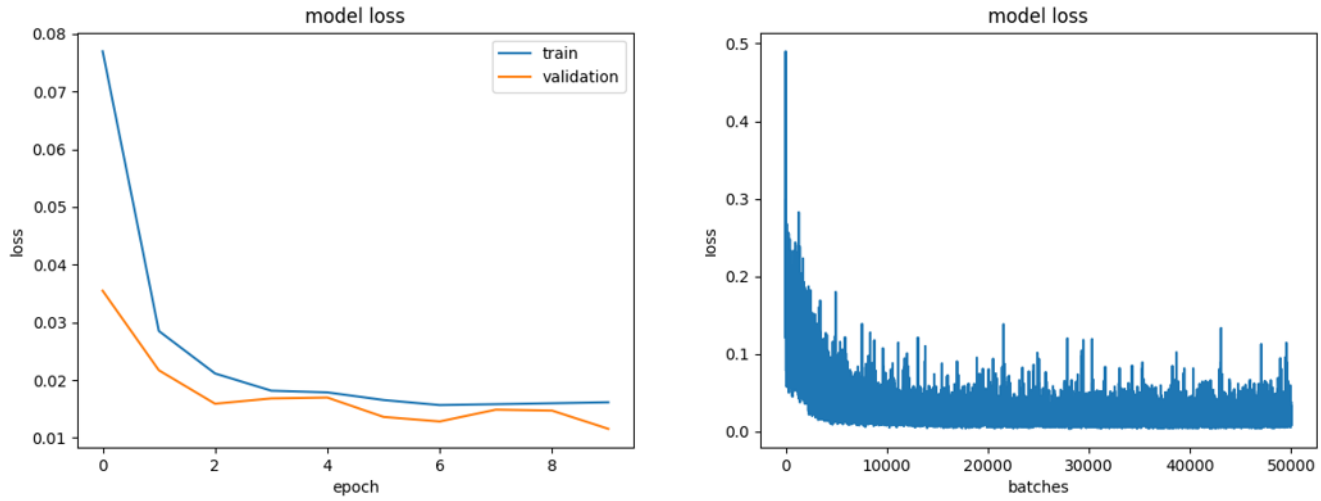


Figure 34: Results of Model 4

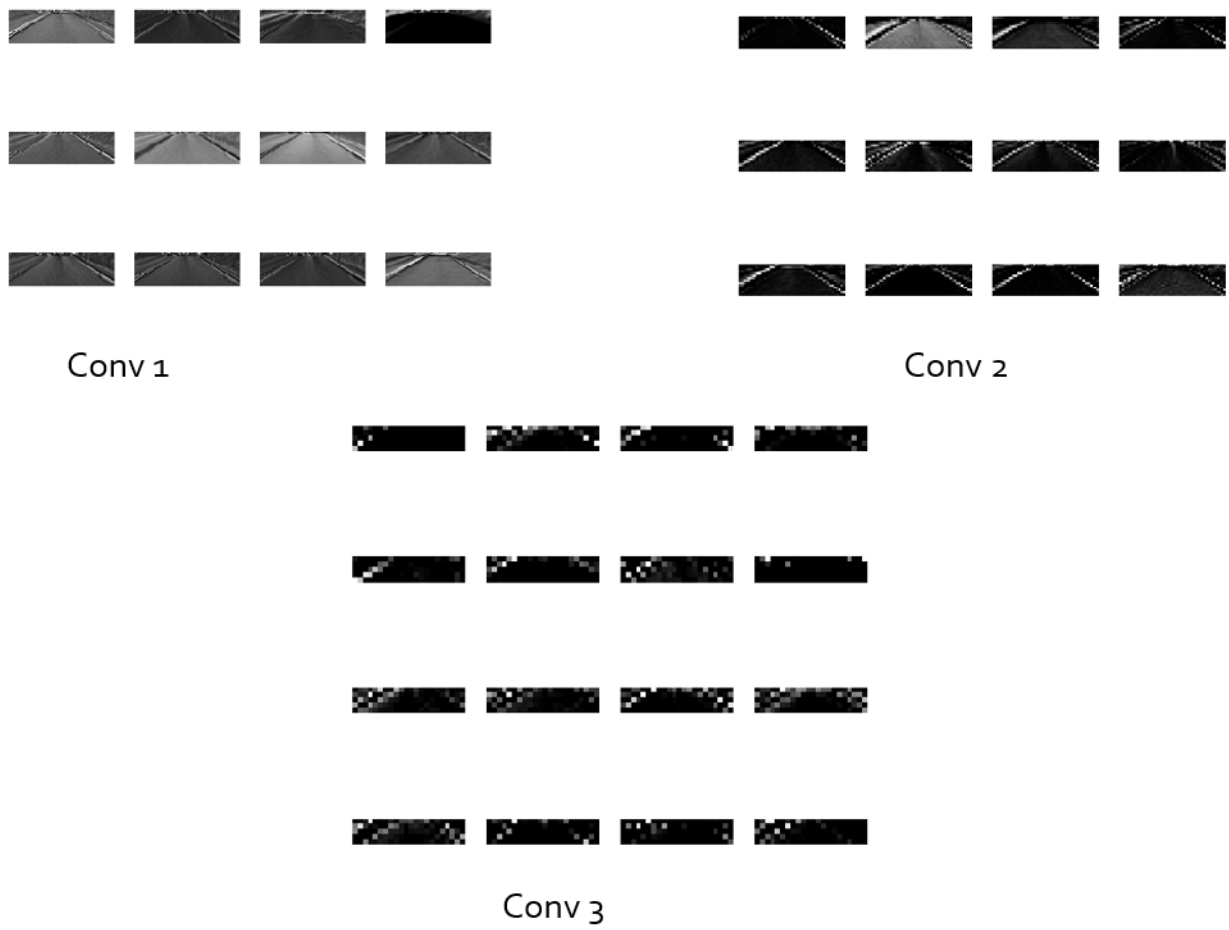


Figure 35: Convolution layer Visualization for Model 4

10.2.2 Model 5

The results of this model have been portrayed in figure 37. The model has been trained for 20 epochs at a learning rate of 0.0001 and a batch size of 64, with 5000 samples per epoch, after which it was over-fitting the data. The model with the least validation loss is chosen. The results at the end of 20 epochs are as follows:

	Training Error	Validation Error
MSE	0.019	0.0068
Steering angle error (degrees)	7.89	**4.72**

Figure 36: Table of results - Model5

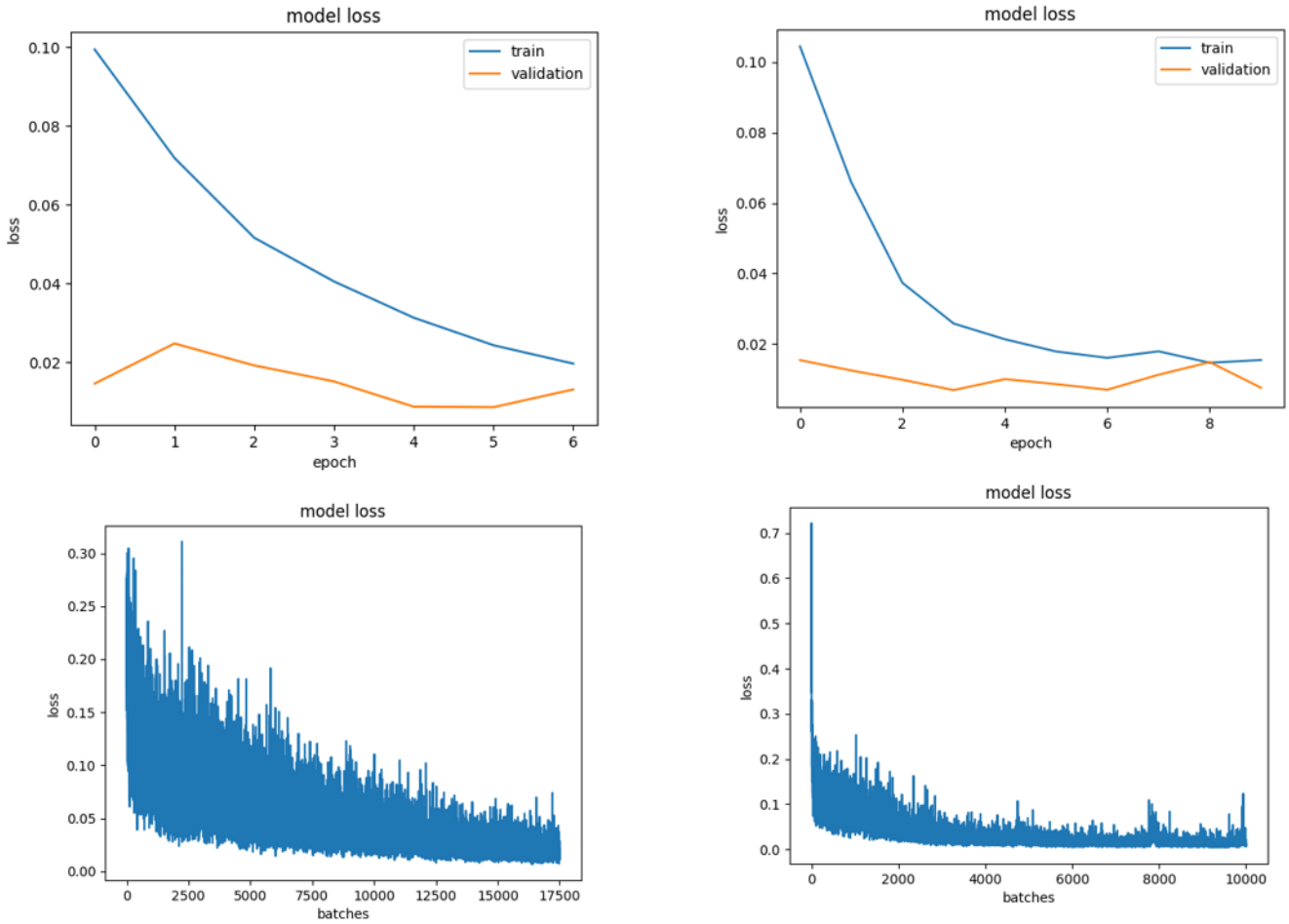


Figure 37: Results of Model 5

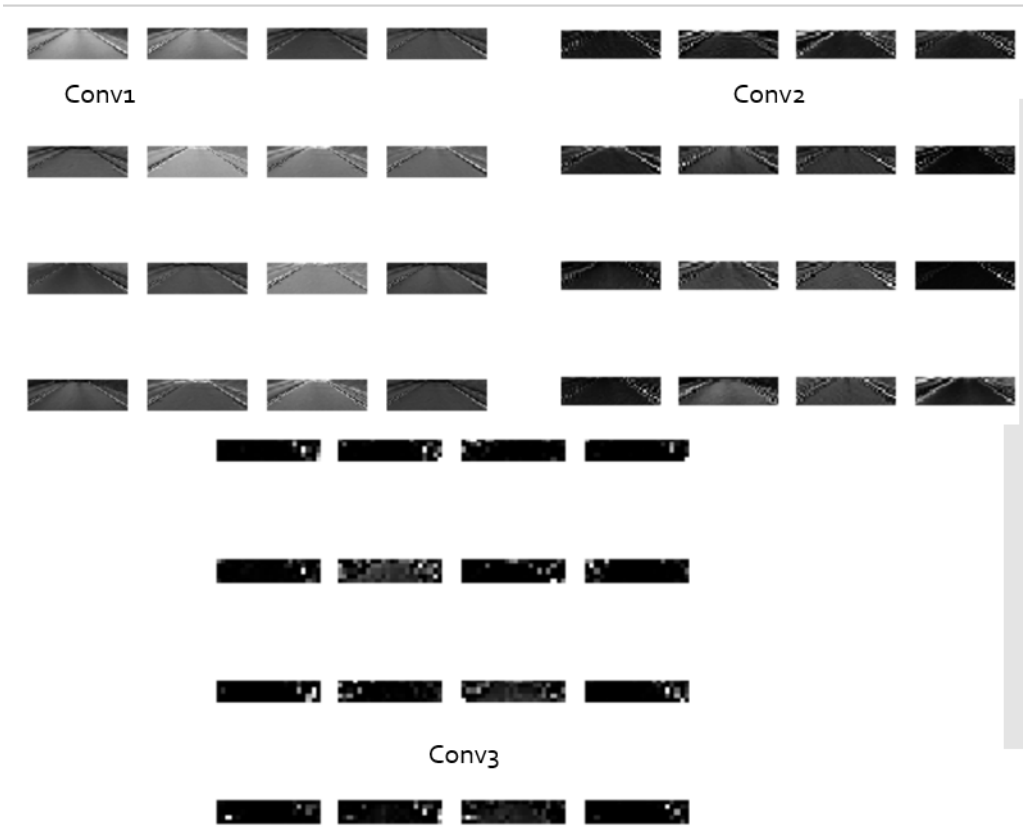


Figure 38: Convolution layer Visualization for Model 5

10.2.3 Model 6

The results of this model have been portrayed in figure 40. The model has been trained for 50 epochs at a learning rate of 0.0001 and a batch size of 64, with 5000 samples per epoch, after which it was over-fitting the data. It can be observed that even after 3 runs and 50 epochs, the model is not converging. The results at the end of 50 epochs are as follows:

	Training Error	Validation Error
MSE	0.055	0.076
Steering angle error (degrees)	13.44	15.8

Figure 39: Table of results - Model6

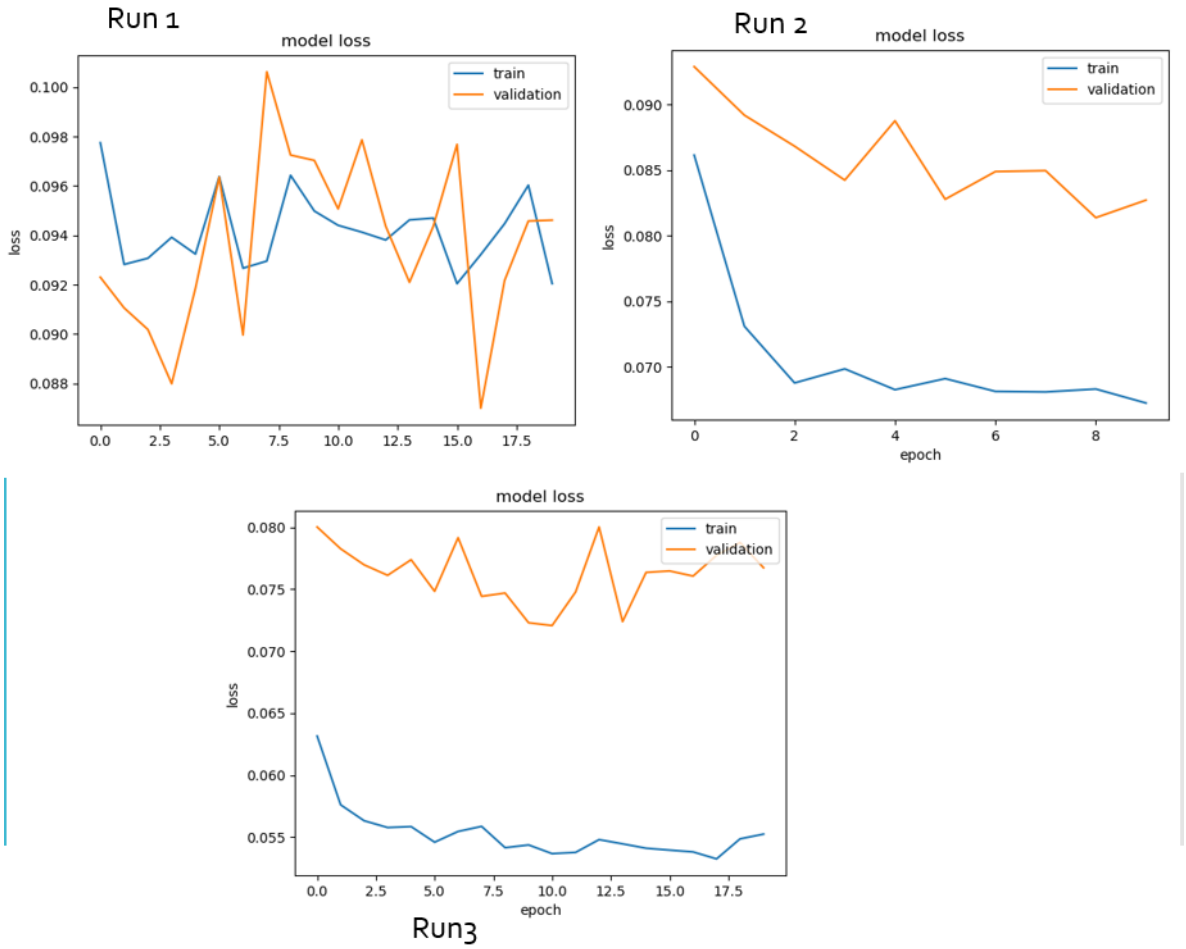


Figure 40: Results of Model 6

11 Discussion

The key objective of this project was to achieve end-to-end learning for self-driving cars using Convolutional Neural Networks. This objective has been successfully achieved by training six different models on two datasets. One important thing to note in this project is that all the six models could not be run through the entire dataset for each epoch due to insufficient time and compute resources. So, the results of these models might not be explicit, and could be imporved with more time and GPU resources. Also, the models run on the actual dataset are not compatible with the driving simulator. So, real-time testing could not be achieved on the models trained on actual dataset.

Besides, even though many models have been constructed by researchers on these two datasets, the hyper-parameters of training and intricate details are not available. Hence, all the models constructed are based on cross-validation results of testig with different settings.

It was observed that the original Nvidia model, which was the state of the art on the Actual dataset did not perform as expected on the simulated dataset and had the highest test error. The model is learning to stay within the sides of the road during testing, but the model is still not robust. The reasons for this might be:

1. The model was originally developed to be trained on over 0.1 million images, but the simulated data has only 24000 images.

2. There might have been redundant data augmentations, which might have caused the poorer performance.

3. Insufficient training

However, after changing the original Nvidia architecture with more drop-out layers, different augmentations and a smaller input size, the model performed the best out of the three, with just 1000 images per epoch. This model imitated the behaviour of the human driver during testing, cruising even sharp curves smoothly, and gave the best test accuracy/least test loss.

Even though the Comma.ai model had over 500,000 parameters to learn, the model still performed than the original Nvidia Model on the simulated dataset. The accuracies of this model were between the two Nvidia architectures. The model gave the least training and validation loss out of the three indicating a hint of over-fitting, as the test-loss increased. But, the model has learnt to maneuver sharp turns, which can be observed in the driving videos.

Consider Model 4. The experiment was performed by using all the 0.1 million images in the dataset, having images from all the three cameras. But, due to insufficient compute resources and less time, I could run the model with only 5000 images per epoch, which seems like it is insufficient. The model can give better accuracies if it is trained for sufficiently longer.

Overall, Model 5 gave the best results, with the lowest validation loss. The model is has a novel technique of extracting temporal information from images in prediction tasks. However, even this model will perform better if trained for a longer time.

Consider Model 6, built with a base of ResNet-50 model. Even after a significant amount of grid search for parameters and using Learning rate scheduler to decay the

learning rate with epochs, the model did not converge, or come close to the training and validation losses of other contemporary models. This model performed the worst when compared to all the other models. The reasons might be :

1. Models 4 and 5 performed well on the dataset, by training only on center images. These had only 250,000 trainable parameters. But, Model 6 had over 9 million trainable parameters, for which the size of this dataset might not have been sufficient.

2. Insufficient Training might have been the problem, which was a result of insufficient compute resources.

3. ImageNet weights might not have been compatible with the road features.

4. Images had too many variations in terms of brightness, contrast and shadow effects.

12 Conclusion

The over all results have been shown here:

		Training	Validation	Test
Simulated Dataset	Model 1	8.58	8.71	12.689
	Model 2	10.42	10.15	9.638
	Model 3	7.51	7.6	10.40
Actual Dataset	Model 4	8.263	7.24	-
	Model 5	7.89	4.72	-
	Model 6	13.44	15.8	-

Figure 41: Summary of results

1. Model 3 gives the best Training and Validation losses on the Simulated dataset.
2. Model 2 gives the best Test loss in real time driving, proving to be the best model off the three.
3. Model 5 (Temporal Model) gives the best results on the Actual dataset, with the lowest Validation loss.
4. Model 6 performs the worst among all the other models, probable reasons for which is listed in Discussions section.

These results are not explicit and might improve with more training. However, with the subset of dataset I have used, the hyper-parameters chosen by me and my intricate details of training, in the course of this project, the results show that the temporal model built by me beats the state of the art Nvidia model in terms of training and validation losses.

13 Future Work

1. Try RNNs and 3D CNNs to compare the performance of these with Model 5
2. Try more hyper parameter search to improve accuracy of the ResNet architecture.
3. Test the models trained on the actual dataset on the simulator.

References

[1] M. Bojarski, D. D. Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba 4, 5, 6, 13, 18, 21 *End to end Learning for Self-Driving Cars,* CoRR, vol. arXiv:1604.07316 , 2016

[2] Udacity Open-source Data sets: https://github.com/udacity/self-driving-car 6

[3] Convolutional Architectures for Self Driving carshttp://cs231n.stanford.edu/reports.html 6

[4] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel., *Backpropagation applied to handwritten zip code recognition.* Neural Computation, Winter 1989. http://yann.lecun.org/exdb/publis/pdf/lecun-89e.pdf. 5

[5] Large scale visual recognition challenge (ILSVRC). URL: http://www.image-net.org/challenges/LSVRC/. 5

[6] Net-Scale Technologies, Inc. Autonomous off-road vehicle control using end-to-end learning, July 2004. URL: http://net-scale.com/doc/net-scale-dave-report.pdf. 5

[7] Dean A. Pomerleau, *ALVINN, an autonomous land vehicle in a neural network. Technical report, Carnegie Mellon University*, 1989. URL: http://repository.cmu.edu/cgi/viewcontent. cgi?article=2874context=compsci. 3, 5

[8] Facebook AI Research *Learning Spatiotemporal Features with 3D Convolutional Networks* ICCV 2015, https://www.cv-foundation.org/openaccess/content - iccv 2015/

[9] Mayer, Nikolaus, et al. "A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2016.