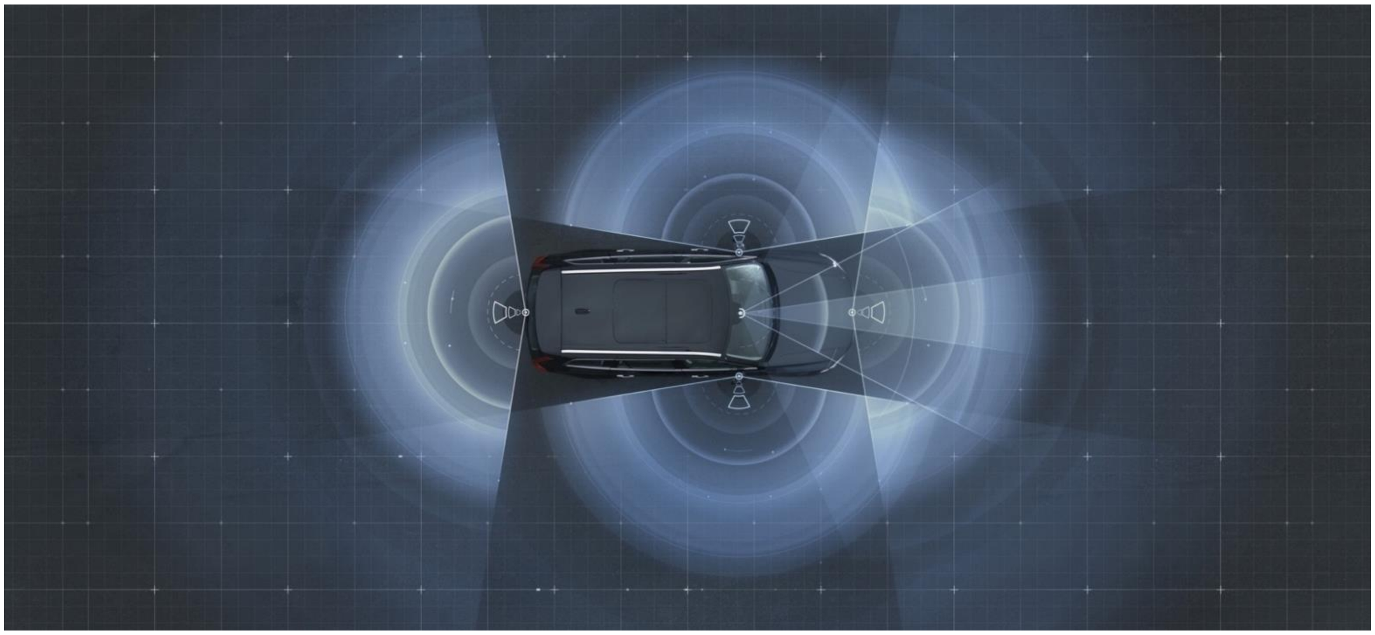
Steering Angle Prediction for Autonomous Cars

- Using Image Recognition & Deep Nets

IDS 576 – Project Report



By: **Shrey Nautiyal | Sarthak Bicchawa | Neha Malhotra**

Contents

1. Business goal

The goal of this project is to apply deep learning techniques in computer vision such as Convolutional Neural Networks(CNN) to predict steering angles for autonomous driving. The model will use the video frames from Udacity driving data to predict steering angles as a regression problem.

We intend to minimize the error between the predicted steering angle and the actual one produced by a human driver.

2. Motivation

The motivation of the project is to eliminate the need for hand-coding rules and instead create a system that learns how to drive by observing. Predicting steering angle is an important part of the end-to-end approach to self-driving car and would allow us to explore the full power of neural networks. Image frames from cameras mounted to the windshield of a car are used to predict the appropriate steering angle using deep learning. For example, using only steering angle as the training signal, deep neural networks can automatically extract features from images to help position the road and make the prediction.

This video [3] from Nvidia highlights their efforts of teaching a car how to drive using only cameras and deep learning. Their DAVE-2 deep learning system is capable of driving in many different weather conditions, avoiding obstacles, and even going off-road. Such learning systems in autonomous driving can lead to safe and efficient driving which will reduce accidents caused due to human errors.

Self-driving vehicle control system would have to determine steering wheel angle, brakes, and acceleration in any driving environment. Convolutional Neural Networks (CNN) have been recently proposed as an effective solution for predicting steering wheel angles for self-driving cars. In this project, we have formulated steering angle prediction as a regression problem and have used open-source driving data sets released by Udacity to evaluate various CNN architectures.

3. Background & Literature

Autonomous driving technologies have enhanced exponentially over the last decade. Dynamic firms like Tesla, Waymo, Uber, etc. have been a huge driving force behind it. Artificial Intelligence technologies on a massive scale have been applied to autonomous driving combining deep learning techniques like CNN, RNN, VAE, GAN, Reinforcement learning, etc. Self-driving vehicle control system would have to determine steering wheel angle, brakes, and acceleration in any driving environment. Udacity [1] is hosting an open source Self Driving Car challenge to break down the problem of making cars autonomous. They have released a dataset of images taken while driving along with steering angle and ancillary sensor data labels. This image data can be leveraged to predict accurate steering angles for autonomous driving.

Self-Driving cars can have a huge impact on the society. The traditional robotics approach for designing self-driving software has four main components –

- Localization
- Mapping
- Perception
- Path-Planning.

Data from various on-board sensors (cameras, radars, Lidars, GPS, HD maps etc.) is used for realizing such a solution. This approach has severe challenges in many scenarios such as bad-weather conditions, non line-of-sight view (driving around intersections), long range sensing etc. There has been a recent push to use deep learning for designing end-to-end self-driving systems. This approach could learn from the data collected by human driving and effectively try to emulate human driving behavior.

NVIDIA team [3] have used CNNs to train input camera images to predict the steering wheel angle. They have formulated the steering wheel prediction as a regression problem and have used three cameras (center, left and right) to augment the data set during training, and thus generalize learning. The center camera sees the middle of the road, left and right cameras are tilted sideways. Correction factor is added to the steering angles corresponding to images collected from left and right cameras. *Figure 1* below shows a block diagram of NVIDIA training system. Images are fed into a CNN that then computes a proposed steering command. The proposed command is compared to the desired command for that image, and the weights of the CNN are adjusted to bring the CNN output closer to the desired output. The weight adjustment is accomplished using back propagation.

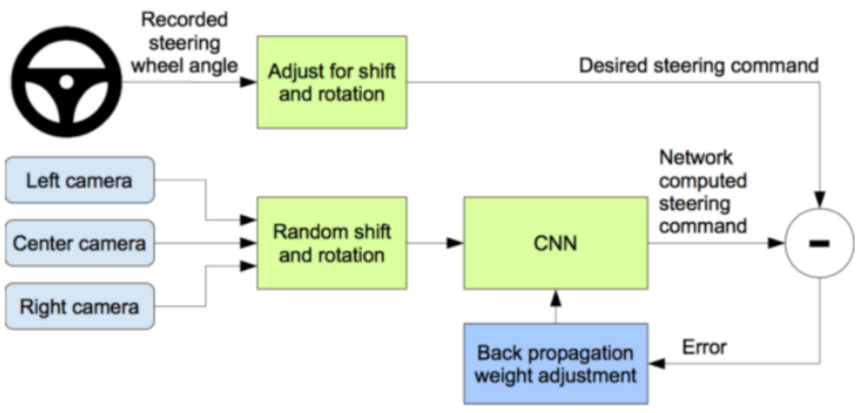

Recorded steering wheel angle
Adjust for shift and rotation
Desired steering command
Left camera
Center camera
Right camera
Random shift and rotation
CNN
Network computed steering command
−
Back propagation weight adjustment
Error

Figure1: Block diagram of NVIDIA Training System

Deep network architecture uses five convolutional layers followed by five fully-connected layers. Udacity launched a similar self-driving car challenge [2] for using camera data to predict steering wheel angle. Udacity has open-sourced driving dataset collected for this challenge. Data-sets has images from 3 cameras (as in [3]) and steering wheel, torque and brake data. Many solutions by teams participating in this challenge used CNN architectures similar to the one used by NVIDIA Team [3]. Using deeper architectures (such as ResNet, VGGNet) were reported performing worse for predicting steering angles than relatively shallower CNN architecture in [3].

In this project, we use a convolutional architecture based deep learning solution for designing self-driving software. Specifically, we use camera data from vehicles to train on the steering wheel angle. We have considered this as a regression problem for steering angle prediction. We use an open-sourced driving dataset released by Udacity [1]. We have cropped images in the driving dataset to focus on the section of the image most relevant to steering wheel angle learning. We have used techniques to augment the data collected from additional cameras on the car. We have used image flipping to further augment our dataset. Our results show that Convolutional Neural Networks (CNN)-based deep learning solutions can be designed to effectively predict steering wheel angle.

4. Applications & Significance

End-to-end solutions like this, where a single network takes raw input (camera imagery) and produces a direct steering command, are considered the holy-grail of current autonomous vehicle technology and are speculated to populate the first wave of self-driving cars on roads and highways. By letting the car figure out how to interpret images on its own, we can skip a lot of the complexity that exists in manually selecting features and drastically reduce the cost required to get an autonomous vehicle on the road by avoiding expensive LiDAR-based solutions.

Large amounts of Image frames can be used as input to CNN by data sharing between vehicles (V2V). This would allow vehicles to share raw camera images which could be used for deep learning applications. Automotive industry believes that V2V communications, which allows vehicles to share data among each other can prevent more than 80% of vehicle related crashes. Other mainstream benefits of autonomous cars are –

- Increase in Safety
- Less Traffic
- More Free Time
- Better Transportation Service
- Better Health
- Reduced Emissions
- Increased demand for new jobs

5. Data Description

5.1 Overview

The dataset we used is provided by Udacity, which is generated by NVIDIAs DAVE-2 System [3]. Specifically, three cameras (left, right and center) are mounted behind the windshield of the data-acquisition car. Time-stamped video from the cameras is captured simultaneously with the steering angle applied by the human driver. Steering wheel angle, brake, acceleration, GPS data was also recorded in the experiments.

Datasets on the Udacity page [7] were in 6 ROSBAG files in the form of a Torrent. Following steps were taken to extract the data from ROSBAG format into .jpg images and .csv files –

1. Downloading VirtualBox
2. Installing Ubuntu 16.04 on VirtualBox
3. Installing Docker - link: https://docs.docker.com/engine/installation/linux/ubuntulinux/
4. Downloading rwightman's repository [6] to VirtualBox
5. Downloading torrent to your local machine
6. Setting up shared directories between your VirtualBox and local machine
7. Running rwightman's reader [6] to extract files

After installing Docker on virtual box, rwightman's run-bagdump.sh script was executed to generated directories in the required the format. Images were extracted in 3 folders – left, right and center and a csv file had the labeled data.

Udacity released the data sets [7] from 5 trips with a total drive time of 28.23 minutes (1694 seconds). Camera images are collected at a rate of around 20 Hz. The image size is 480 X 640 X 3 pixels and total data set is 3.63 GB. The data set contains a total of 101,397 frames and corresponding labels including steering angle, torque and speed. The center camera has a total of 33,808 image frames.

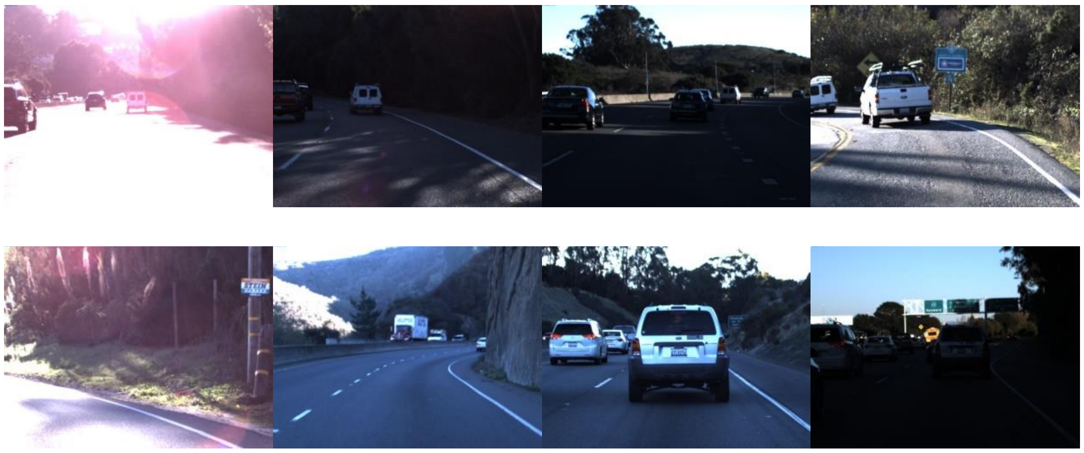


Figure 2: Example images from the data showing different driving conditions

The videos were taken in various light, traffic and driving conditions. The trips were taken at different locations with the following attributes -

1. **Trip1**: (221 seconds) Direct sunlight, many lighting changes. Good turns in beginning, discontinuous shoulder lines, ends in lane merge, divided highway
2. **Trip2**: (791 seconds) Discontinuous shoulder lines, ends in lane merge, divided Highway, two lane road, shadows are prevalent, traffic signal (green), very tight turns where center camera can't 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. **Trip3**: (99 seconds) Divided highway segment of return trip over the summit
4. **Trip4**: (212 seconds) Guardrail and two-lane road, shadows in beginning may make training difficult, mostly normalizes towards the end
5. **Trip5**: (371 seconds) Divided multi-lane highway with a fair amount of traffic

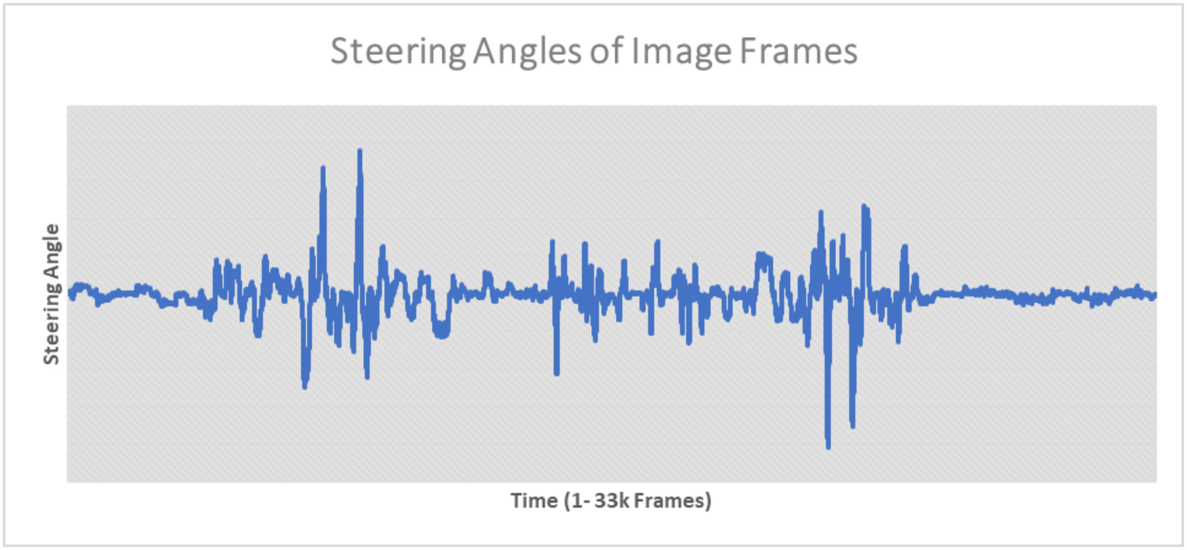
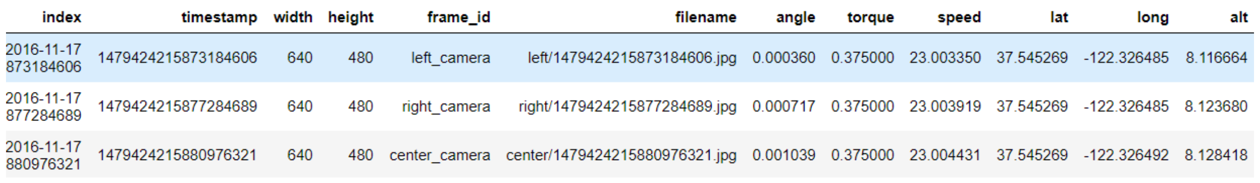


Figure 3: time series of recorded steering angles across the 5 trips.

Stratified samples were taken from each of these trips to generate the train and test data as every trip had different conditions such as turns, light, time of the day, etc.

5.2 Data Snapshot



index	timestamp	width	height	frame_id	filename	angle	torque	speed	lat	long	alt
2016-11-17 873184606	1479424215873184606	640	480	left_camera	left/1479424215873184606.jpg	0.000360	0.375000	23.003350	37.545269	-122.326485	8.116664
2016-11-17 877284689	1479424215877284689	640	480	right_camera	right/1479424215877284689.jpg	0.000717	0.375000	23.003919	37.545269	-122.326485	8.123680
2016-11-17 880976321	1479424215880976321	640	480	center_camera	center/1479424215880976321.jpg	0.001039	0.375000	23.004431	37.545269	-122.326492	8.128418

6. Methodology and Statistical models

6.1 Data Processing and Normalization

- Normalizing and zero-centering image data (x /255.0-0.5)
- After carefully analyzing the image dataset top half (240 pixels) of the images were cropped as it does not provide any useful information on the road (mostly consists of trees, sky etc.)

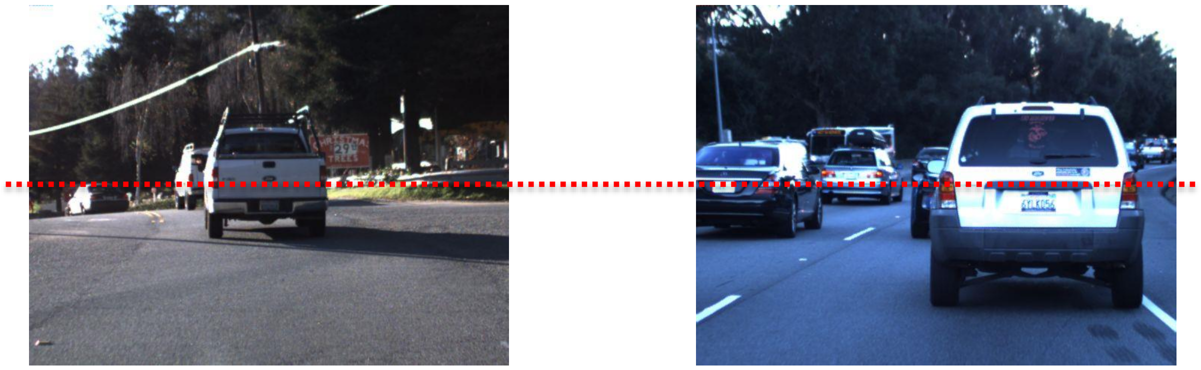


Figure 4: Example images from the data (portion above the red line has been cropped)

- Images with steering angle values below a certain threshold (.05 radians) were removed. This removes the noise in the data collection process and helps balance the dataset better because a large part of the data involves straight driving.

6.2 Data Augmentation

The data from left and right cameras is used to augment the data collected by center camera. Hence, we adjust the steering angle by a fixed threshold to account for the positioning difference between these cameras. These off-center cameras enable training for recovery paths for situations when car might weave from center of the road.

Specifically, we adjust the steering angles for images from left and right cameras as:
steering_left = steering_center + STEER_CORRECTION
steering_right = steering_left - STEER_CORRECTION

The correction factor is treated as a hyperparameter. Here, we set STEER_CORRECTION to a value of 0.1 radians after cross validation

6.3 Data Sampling

It should be noted that although the training set has 33.8k images, there are only few images in the dataset with very steep turns. This makes training challenging because most examples have low to moderate steering angles, making it hard to learn and predict steep left and right turns.

The dataset is a series of images and there is high correlation between adjacent samples, thus it is important during training to shuffle the training images.

If we randomize the whole dataset and choose a validation set, we might get very similar images in the validation and training sets making it difficult to detect overfitting. Thus we choose stratified samples of the last 20% of each trip for cross validation. This allows us to check how well the model generalizes to unseen images and also helps us capture some steep turns from Trip 2 and Trip 4.

6.4 Model Build

We use the CNN architecture as in the NVIDIA team [3], while formulating steering angle prediction as a regression problem. The Network has ten layers and we have extensively used **Batch Normalization** and **Dropout** (which was not used in [3]). The 10-layer network architecture is as follows:

Layer	Type	Size
1	Conv	5*5, 24
2	Conv	5*5, 36
3	Conv	5*5, 48
4	Conv	3*3, 64
5	Conv	3*3, 64
6	FC	1164
7	FC	100
8	FC	50
9	FC	10
10	FC	1

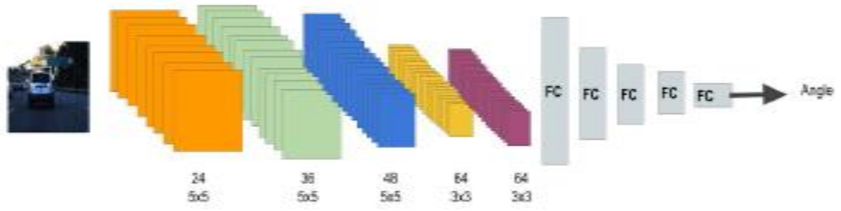


Figure 5: 10-Layer CNN architecture used for steering angle prediction

Batch Normalization: Used after all 5 Conv and Fully connected layers
Dropout: Used after first 4 Fully connected layers. Keep probability parameter of Dropout layer 0.2 identified as the best setting for our model.
Activation: ReLu nonlinearity
Loss: Mean squared error (MSE) loss without any regularization.
Optimizer: Adam

Iterations: These design choices were derived using extensive experimentation with a cross validation set consisting of 20% of the data. For example- Iterations with Batch Normalization layer placed before the ReLU activations were also tested but the configuration performs significantly worse. Hence, we decided to use Batch Normalization after the ReLU activations. Dropout with probability 0.5 also performs worse than probability of 0.2. We also experimented with different number of convolutional and fully-connected layers. Our results show that best performance is achieved by using 5-Conv and 5-FC layer architecture.

7. Analysis & Results

We use Keras for all our experiments with a Tensor Flow backend. The network is trained using the MSE Loss function. Adam Optimizer was used with a learning rate of 1e-3 and no decay rate. Our minibatch size is 64.

We train the network for 10 epochs (due to computational limitations) and got the least loss at 7th epoch. **We achieved a validation loss of 0.0303**. We have plotted the validation and training loss for the network below-

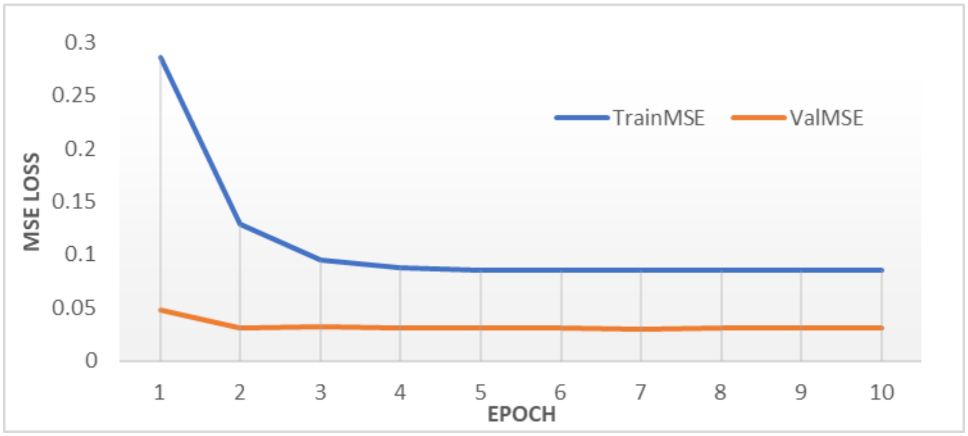


Figure 6: Loss Function for the network

- It should be noted that the training loss is higher than the validation loss. The reason for this is that the training data has a lot more steep turns, and we do not predict very well for such steep turns. Hence, the training loss is skewed by these turns. Validation loss has relatively fewer steep turns.
- It is also interesting to note that the network learns to detect the lane markers. This is because most of the dataset has clear lane markings. This means that in areas without well-defined lanes the predictions may be off.
- We can see that the network tracks the measured steering angle quite well for moderate turns, but for steep turns it is not able to predict huge turns (i.e. large steering angle). Trip 2 and Trip 4 specifically have some very tight turns and attributes to highest prediction errors.

8. Conclusion

8.1 Creative contribution

Data Handling: Extensive data manipulation and extraction performed as the image and sensor data was in ROSBAG format in a torrent file. ROSBAG files were extracted in Ubuntu using dockerfile and python scripts to get images in .png format and labeled data in .csv file. More than 100k images data of 4GB was used for building the network**.**

Model: We used a CNN based 10-layer architecture inspired by NVIDIA team to train camera data for predicting steering wheel angle for self-driving car.

Optimization: We have used various techniques for data-processing (image cropping, removing of low steering angles, normalizing images) and data augmentation (flipping images, using data from left and right cameras). We have extensively used Batch Normalization and Dropout layers in the CNN architecture. Though the training phase was computationally expensive, multiple iterations were performed to tune the parameters.

Analysis & Insight: Our CNN model performs well and achieves a validation loss of .0303. Given the driving data-set is only of 28.33 minutes, our technique of using images with deep learning architecture shows potential of being used in practice. With more training data the model can overcome its current limitations to accurately predict steep turns.

8.2 What went right?

- Application of NVIDIA model architecture using 5 CONV layer and 5 Fully connected layer worked well for our data
- Given the small duration of data, our model performs fairly well.

8.3 What went wrong?

- Efforts to connect with online cloud platforms - Google Colab to leverage GPU support were not fruitful as there is no seamless way to upload large amounts of data on Google Drive. As Google drive is not local to Colab, processing was slower than local system due to frequent data transfer.

8.4 What can be improved?

- Number of EPOCHS can be increased from 10 to achieve better results. We were restricted to 10 epochs due to computational limitations.
- More training data can help the model learn better and reduce the loss.
- In Udacity challenges, CNN-LSTM network seem to improve performance over the CNN-based network used in this project. CNN-LSTM architecture could also be a good avenue for research.

9. References

1. Data and inspirations from Udacity - https://www.udacity.com/self-driving-car
2. https://medium.com/udacity/challenge-2-using-deep-learning-to-predict-steering-angles-f42004a36ff3
3. https://devblogs.nvidia.com/deep-learning-self-driving-cars/
4. https://futurism.com/images/7-benefits-of-driverless-cars/
5. https://github.com/udacity/self-driving-car/blob/master/steering-models/community-models/cg23/README.md
6. https://github.com/rwightman/udacity-driving-reader
7. https://github.com/udacity/self-driving-car/tree/master/datasets/CH2
8. http://cs231n.stanford.edu/reports.html
9. https://selfdrivingcars.mit.edu/
10. https://github.com/choudharydhruv/cs231n_project