Self Driving Car using CNN

Neel Patel Sayee Shruthi

April 20, 2021

Abstract

The evolution in Computer Vision these days have gone beyond imagination. A few years ago, traditional vision algorithms were used to analyse camera footage while decision making needs manually constructed behaviours. In recent years, Convolutional Neural Networks(CNNs) are being widely used in Image Processing and Classification Tasks. The main objective of this project is to build minimal version of self driving car using deep learning algorithms which can predict the steering angle based on image. Database that contains images taken from road and the angle of steering wheel for each image are given as input to model. The model learns the angle from turns in the each image and predicts the steering angle for any image.

1 Introduction

Self Driving Cars have always been a trending and demanding topic in the field of computer vision. We have companies like Tesla, Uber, Waymo who are pioneers of autonomous driving technology. These companies uses data from multiple sensors like camera, GPS, Light Detection and Ranging (LIDAR), Radio detection and ranging(RADAR) and ultrasonic sensors. Among these sensors, LIDAR and cameras fixed on the vehicle are significant to determine 2D environment and 3D geometry around the vehicle to avoid collisions. Although LIDAR sensors are expensive than others, companies like Waymo use LIDAR for better accuracy. These sensory inputs are fed to a computing box which controls car operations such as steering system, braking system, wipers, indicators, signals and gas pedal. The above technological advancements takes traffic rules and collisions into account before deploying which allows drivers to leave hands off the steering while driving. [1]

With recent advances with deep learning, Convolutional Neural Network(CNN) has brought revolution in the field of computer vision. CNN is fundamentally used for feature extraction, which can extract useful attributes from pre-trained examples. Using this approach based on CNN, we can implement autonomous model vehicle which maps pixels from front camera data to steering commands and can be considered as regression problem. The network automatically learns maximum variable features from the input and hence requires minimum human intervention for training and prediction. After training, given a realistic image from camera input the model can generate steering control command. In this project, we tried to build a simplified and minimal version of self driving car using CNN. We used sequence of front camera images as input which is reasonable in size as compared to LIDAR data, which is expensive to process and large in size. After processing this data and training a model, we predicted sequence of values of steering angle. Furthermore, we displayed a steering wheel image and rotated the image according to predicted output degrees for making it more interactive.

2 Loading the Data Set

There are many public data sets available in varied time ranges and features obtained during data collection like steering angle, brake frequency, acceleration and LIDAR.An open source self driving system startup, commaai[3]has connected a bunch of hardware on the vehicles and collected data which is 45GB compressed and 80GB uncompressed. A slightly larger data set from udacity[4] concatenates data under different climatic conditions which is approximately 200GB and includes variety of information like braking and acceleration. A large data set is available by the company Baidu from a new project Apollo[5] which is open data platform and their data is mostly human annotated.

In this project, we used relatively smaller data set which was manually collected by driving a car for 25 minutes. A dash camera was fixed on the car and on-board diagnostics(OBD) port was hacked to collect the steering angle for every frame of image. This data set which is approximately 2.2GB was uploaded on GitHub by an individual named SullyChen[6] and licensed for research and education purpose. One of the main reason for using SullyChen's data-set for this model is the small size of the data which took 4-6 hours for training on personal computer. More complex and huge datasets would require high computational power and more time for training. Moreover, the proposed model can be replicated later on significantly larger data-sets with ease.

The data-set is a compilation of 25 minutes of manual driving video on different environments namely paring lot, multi-lane roads, high-traffic roads etc. The video is broken down into images with 30 frames per second which would sum up to 45406 images. Moreover, data-set also contains text file named data.txt which contains information of each image file name and it's corresponding steering angle in degrees. This image data and steering angle is loaded to the model by converting steering angle to radian for training.

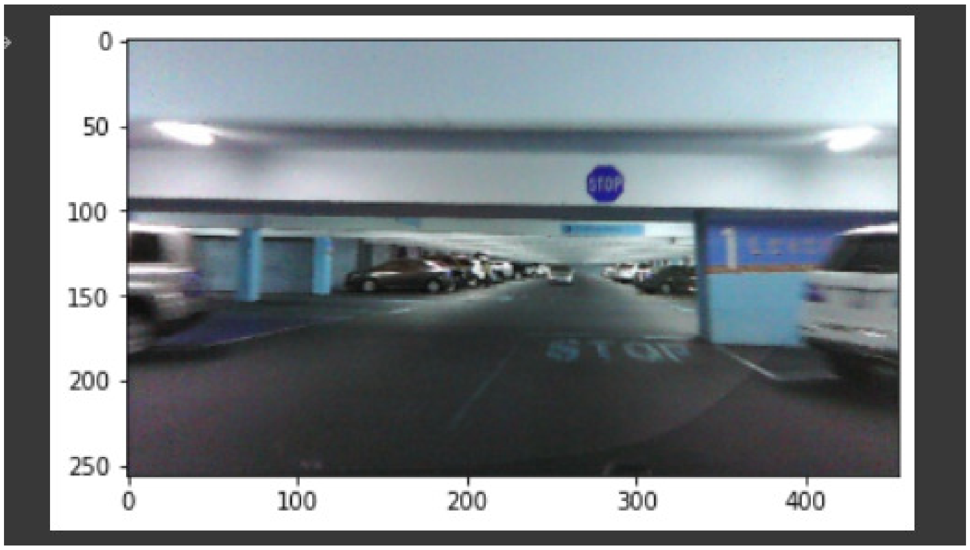


Figure 1: Original image from dataset

Once the temporal data is loaded, we find the steering angle is 0 for first 23 images and then, there is a slight change in steering angle which indicates road curvature. The histogram is plot for train and test data-set in blue and orange color. The non-overlapping graphs of train and test indicates there is a difference in train and test data. There are approximately 16,000 occurrences of zeros in the dataset which indicates that the user is travelling on straight road most of the times. Conversion of steering angle to from degrees to radian helps us in increasing accuracy and decreasing variance of the data.

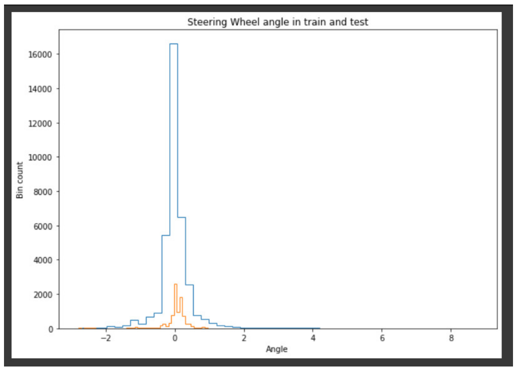


Figure 2: Visualizing the data

3 Pre-processing the Images

Image Pre-Processing helps in removing the unnecessary data and decreasing the training time required. Firstly, we start by cropping the lower part of image which contains pixels for road and it's curvature which is useful for training the data. Hence, the image is zoomed to focus on road part. Then we resize the image to make it of size which can be given to model for training. Along with these we normalize the pixel values by dividing it with 255. Moreover, we performed basic augmentation on training data. We augmented the data randomly by flipping of image or zooming in or changing brightness of image to get better accuracy. We perform this using splitting the image pixel from 150 and resize() function from cv2[7] library. For flipping we used flip() function from cv2 library and we used image augmenters library[8]

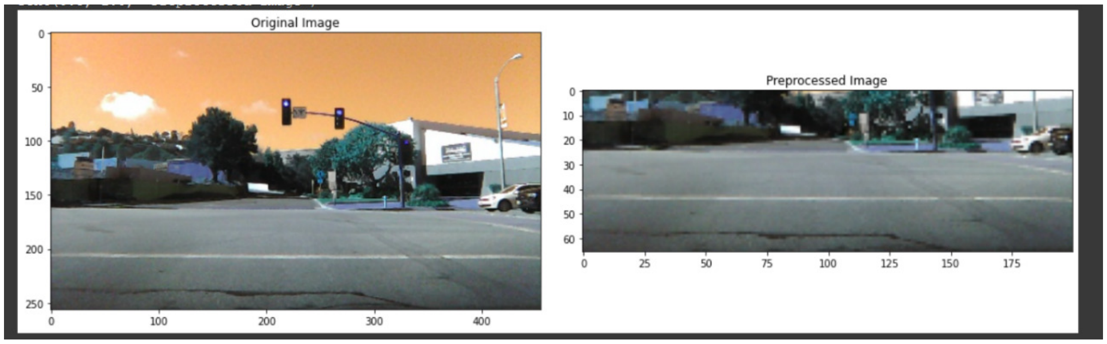
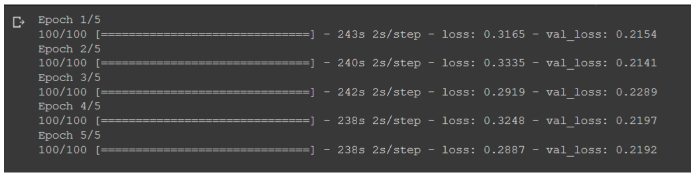


Figure 3: Original Image and Preprocessed Image

4 Developing CNN Architecture

We started by experimenting with different type of CNN architectures, in which we chose ResNet, VGG and InceptionNet but we found accuracy for all the them to be very low. Among these, Resnet had lowest loss and hence, we started tweaking some changes into models to roughly get better idea of it. Whilst in this process, we got loss of approximately 0.28 in 5 epochs as attached in below screenshot.



```
Epoch 1/5
100/100 [==============================] - 243s 2s/step - loss: 0.3165 - val_loss: 0.2154
Epoch 2/5
100/100 [==============================] - 240s 2s/step - loss: 0.3335 - val_loss: 0.2141
Epoch 3/5
100/100 [==============================] - 242s 2s/step - loss: 0.2919 - val_loss: 0.2289
Epoch 4/5
100/100 [==============================] - 238s 2s/step - loss: 0.3248 - val_loss: 0.2197
Epoch 5/5
100/100 [==============================] - 238s 2s/step - loss: 0.2887 - val_loss: 0.2192
```

Figure 4: Training & Testing Loss in Resnet Model

While researching more, we came across research paper from NVidia's end to end learning solution for self driving cars[2] and decided to follow the same CNN structure as it was also using images as an input. Dimensions of input images are 66x200x3 where 66x200 are the width and height of the images and 3 is the number of channels (Red-Green-Blue). The weights in our network were trained to minimise mean square error between original output and the output from network. The network consisted of 9 layers from convulational layers to fully connected dense layers and one normalization layer. Firstly, Normalization is performed for the value of images. After this, 5 convolutional layers works for extracting features and were chosen by series of experiments of various configurations. This layer creates a convolution kernel that is convolved with the layer input to produce a tensor of outputs. Moreover, activation function relu is applied to each layer. First 3 convolutional layers consists of strides of 2 x 2 and kernel size of 5 x 5. Last 2 convolutional layers consists of 3 x 3 kernel size. These layers are followed by 4 fully connected dense layers which reduces number of neurons into one output. Since, this is a regression problem we have applied identity function in the end.

5 Improving Performance

After deciding the model architecture, we started experimenting with hyper-parameters to getting the best model. Following this, we changed a few hyper-parameters like train and test ratio and calculated loss. We used (60,40), (70,30) and (80,20) train/test ratios respectively. From the experiments, we could see that (80,20) split had minimum loss. This could be seen in Figure and other split had higher losses in train and test data. Firstly we

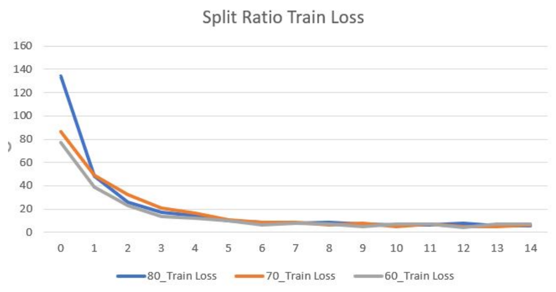


Figure 5: Train Loss of different Split Ratios

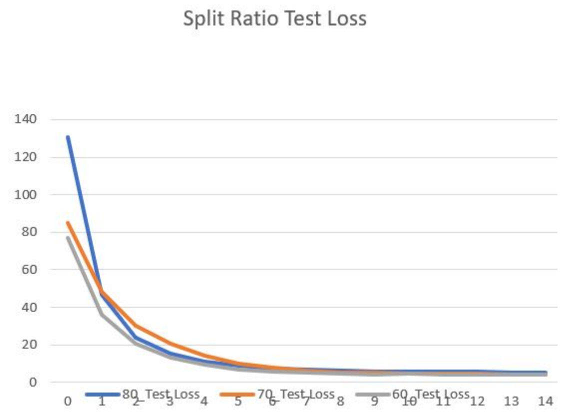


Figure 6: Test Loss of different Split Ratios

changed different optimizers for the models and plotting recorded the loss. Also, we could see that Adam with learning rate 10 power -4 had the highest accuracy and it learned fastest among other optimizers. Adam with learning rate 10 power -3 had oscillating results without reaching local minima. This can be seen in Figure 8. Other optimizers like Adagrad optimizer performed worse .

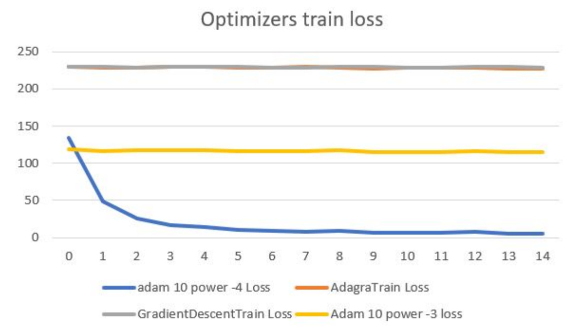


Figure 7: Loss in different Optimizers

For improving performance of the model, we can combine multiple convolutional layer with different number of filters. We can also add multiple classification layers accordingly. Firstly, we tried experimenting with different number of filters in a single Convolutional 2D Layer. After that we generated fully connected layers with hyperparameter changes in dropout value. We could see that droupout with 1 was giving minimum loss .

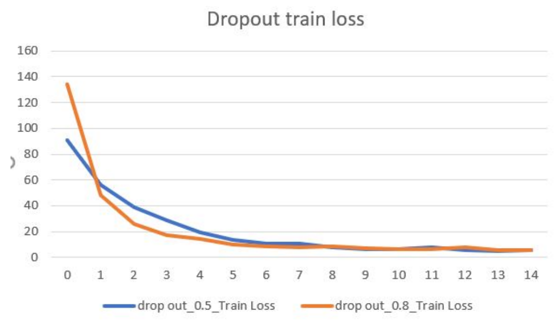


Figure 8: Train Loss of different Dropouts

Since, this is a regression problem so we have applied identity function in the end. We can also applied atan function. It took higher computational power to test with both identity and atan functions. From the experiment we found out that identity function had better accuracy

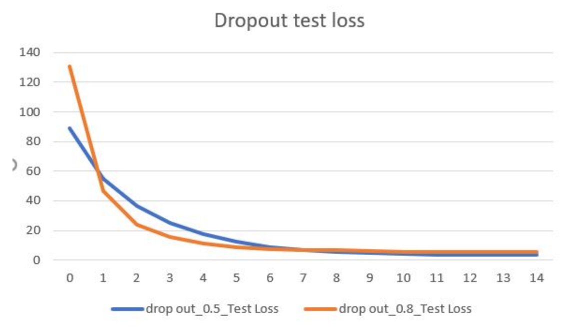


Figure 9: Test Loss of different Dropouts

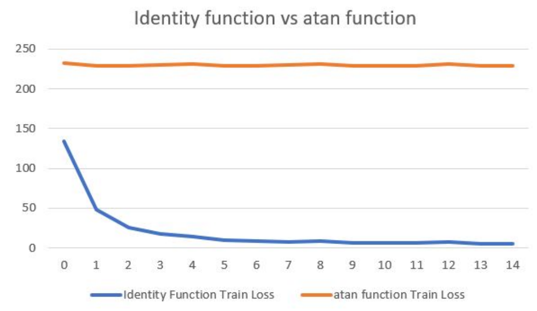


Figure 10: Loss for identity and atan function

6 Training & Testing the model

The final model files after training are stored under a directory as a checkpoint since the training takes a lot of time. We keep storing the model as checkpoint after each epoch. Moreover, we are also storing The train and test loss in an .csv file for reference in the end. Finally, the model is saved as model.ckpt after calculating test loss. It takes approximately 4-6 hours for training depending on the computation power. To avoid over fitting, the total loss we tried to minimize using L2 regularized loss formula. This was done by calculating mean square error+ L2 loss for all weights multiplied by L2 Normalization constant. From the changes in hyper parameters we train the model with Adam Optimizer with learning rate 10 power -4. The loss is stored in a local file for later evaluations. We have trained the model with maximum 30 epochs to predict steering angle and trained to minimum 15 epochs for hyper parameters tuning. The model is saved as model.ckpt after calculating test loss. We could see the loss is significantly reduced from 6.13

to 0.13 at the end. The model can be tested and visualized with the either restoring the saved model or training the whole model. Finally, it shows up a frame window of steering wheel and image sequence of the video which mimics as video being played as shown in figure 11.

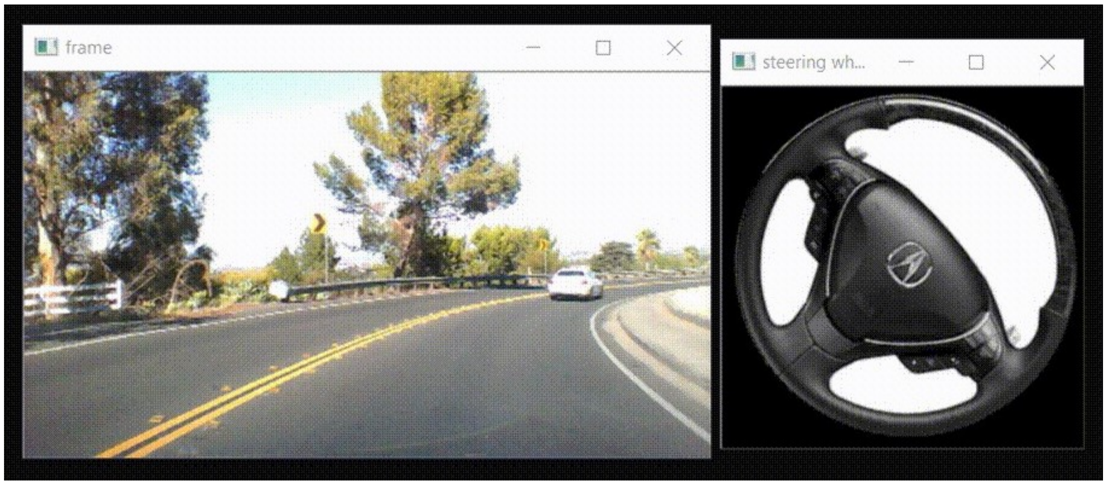


Figure 11: Screenshot of output

Link to our code is here.

7 Conclusion

We were able to learn a lot about CNNs which are state of art techniques in computer vision through lectures and projects. We were able to develop a model that could learn about detecting the outline of road without explicitly providing labels during training. This model can be easily extended on dataset with higher parameters. It can be easily integrated with other key features like braking or acceleration values to make the car fully automated. Moreover, we have used small dataset due to our computation limitation but this model can be easily used on huge datasets. Lastly, even though this model gave an accuracy of 97%, to minimize the risks this can be deployed in real time systems for assisting humans in emergency situations for controlling the car better.

References

[1] A Systematic Review of Perception System and Simulators for Autonomous Vehicles Research
https://www.mdpi.com/1424-8220/19/3/648/htm

[2] End to End Learning for Self-Driving Cars
https://arxiv.org/abs/1604.07316

[3] commai
https://comma.ai

[4] Udacity
https://github.com/udacity/self-driving-car.git

[5] Baidu Apollo
https://apollo.auto/

[6] Driving Datasets - Sullychen
https://github.com/SullyChen/driving-datasets.git

[7] CV2 Library
https://pypi.org/project/opencv-python/

[8] Image augmenters
https://imgaug.readthedocs.io/en/latest/source/examples_basics.html