Teaching Cars to Reproduce Human Driving Behavior Using Deep Neural Networks in a Simulated Environment

Zhen Liu and CJ Chung
College of Arts & Sciences, Lawrence Technological University

INTRODUCTION

Human sub-cognitive skills can be captured and reproduced in a computer program. As the human subject performs the skills, his or her actions are recorded. This record is used as input to a learning program, which outputs a set of rules that reproduce the skilled behavior. This method is called behavioral cloning [4]. In this project, we apply this method to teach a car to steer itself by training a deep neural network to predict the steering angles when the car is facing various turns. We train and test the neural network on a virtual car in a simulator instead of a real car, which is usually unavailable or expensive to access in the classroom. The simulator we use is open source Udacity simulator; its two modes allow us to collect training data and test the neural network model with ease. The training data comprise images of road from three cameras mounted on the simulated car, along with corresponding steering angels. The data is used to train and validate a Deep Neural Network's model in Keras with TensorFlow as backend so that it can recognize road configurations and generate appropriate angles. A block diagram of the training process in shown in figure 1. Images are fed into a CNN which computes a proposed steering angle. The steering angle is then compared to the desired steering angle for that image and the weights of the CNN are adjusted using back propagation to bring the CNN output closer to the desired output. Once trained, the network can generate steering angles for the simulated car to drive itself in the simulator. Currently the network with best performance is derived from the NVIDIA architecture with 4 convolution layers, which have 3x3 filter sizes and depths varying from 24 to 64, and 4 fully connected layers. As the project goes further, it will include the use of recurrent neural network (RNN) and/or other deep learning techniques to improve the model.

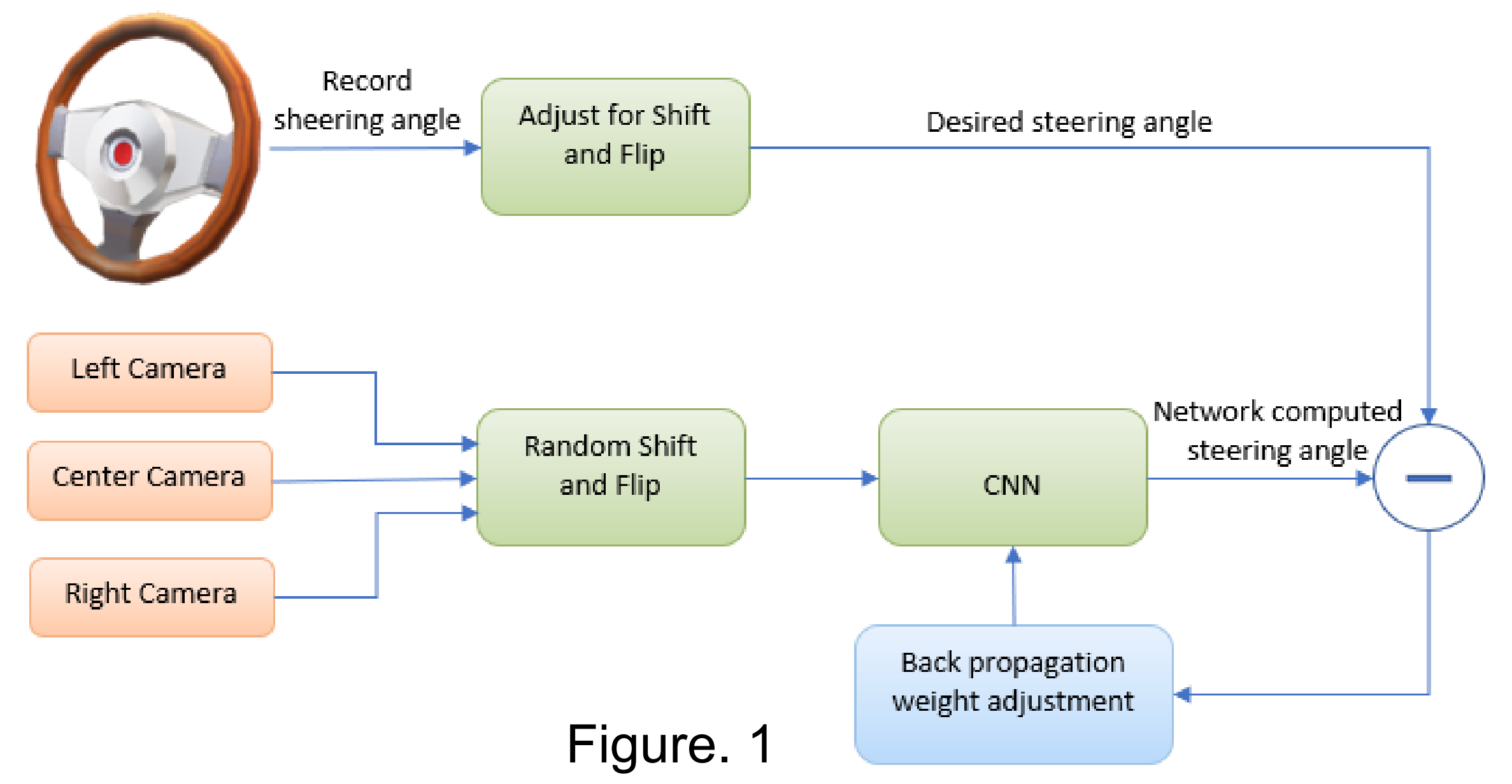
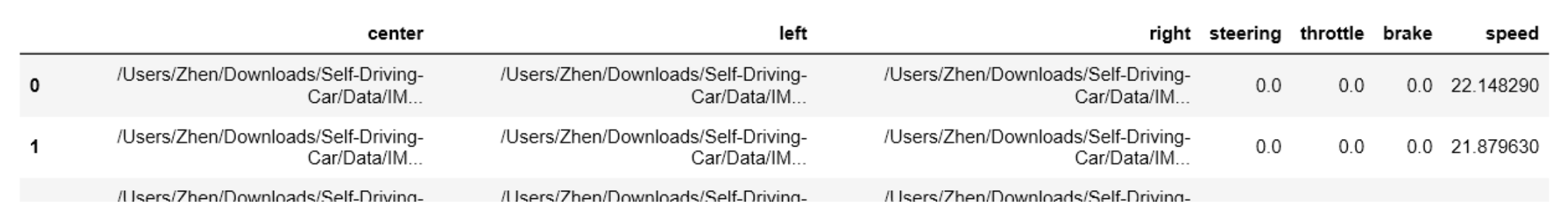


Figure. 1



	center	left	right	steering	throttle	brake	speed
0	/Users/Zhen/Downloads/Self-Driving-Car/Data/IM...	/Users/Zhen/Downloads/Self-Driving-Car/Data/IM...	/Users/Zhen/Downloads/Self-Driving-Car/Data/IM...	0.0	0.0	0.0	22.148290
1	/Users/Zhen/Downloads/Self-Driving-Car/Data/IM...	/Users/Zhen/Downloads/Self-Driving-Car/Data/IM...	/Users/Zhen/Downloads/Self-Driving-Car/Data/IM...	0.0	0.0	0.0	21.879630
	/Users/Zhen/Downloads/Self-Driving-	/Users/Zhen/Downloads/Self-Driving-	/Users/Zhen/Downloads/Self-Driving-				

Figure. 2

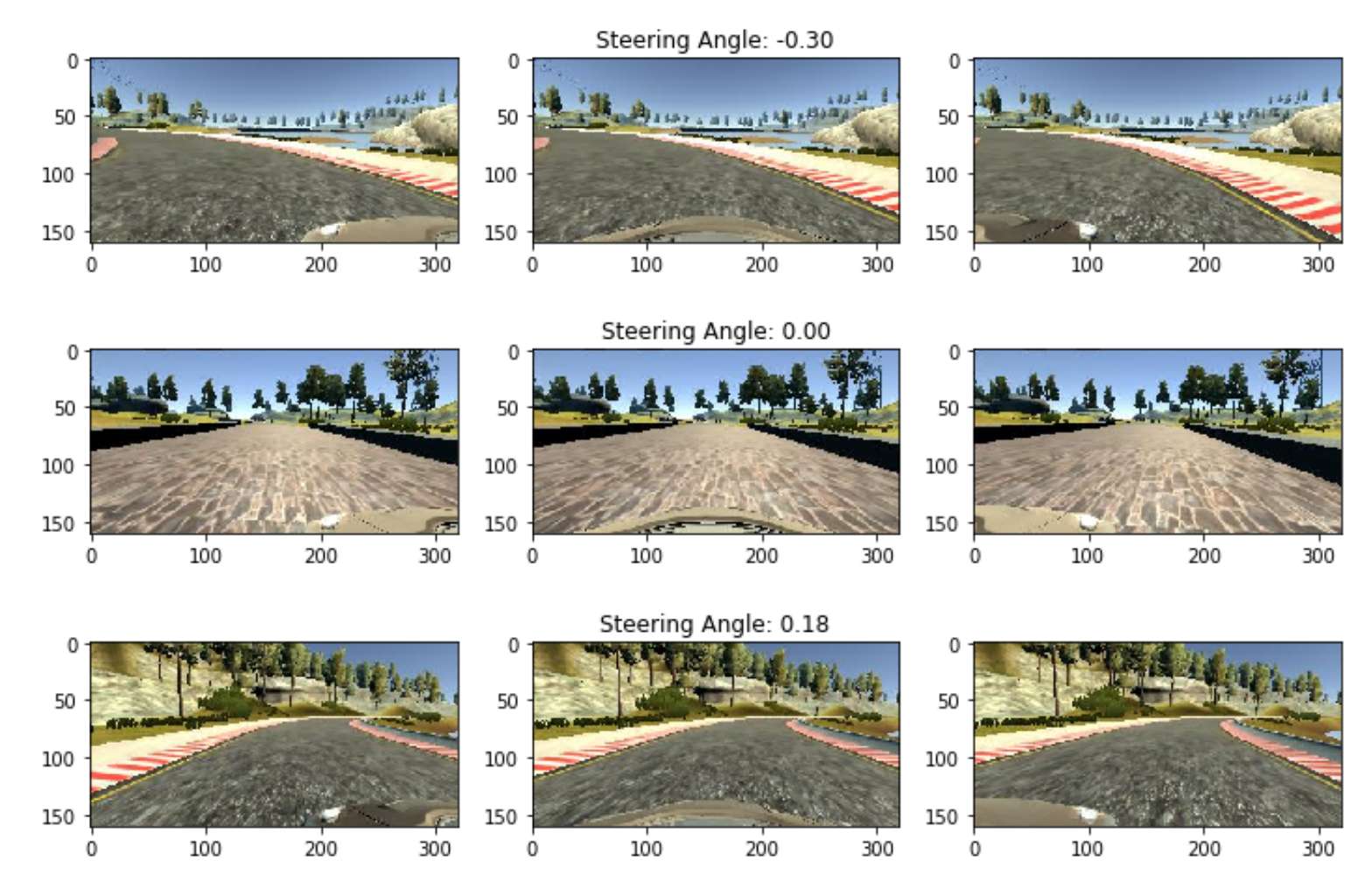


Figure. 3

METHODS

Data Acquisition and Augmentation

Behavioral Cloning in this project is to train neural networks with data that exhibits the very driving behavior we want to clone. Training and testing is conducted on the pre-built open source Udacity Simulator, which has two modes: Training Mode and Autonomous Mode. Training Mode is used to record frames from three cameras mounted on the simulated car's left, center, and right as well as driving statistics such as steering angle, throttle, brake and speed, recorded in a .csv file, as shown in figure 2, while autonomous Mode is used to test the model trained using this data. Udacity simulator also provides two tracks to train or test: Lake Track and Mountain track. We collected data only from Lake Track [5].

Image data and its corresponding steering angles are used as model input and expected to predict the steering angle in the range of [-1, 1] in real time driving. Image data from all three cameras are used as training data, because we want to handle the issue of recovering from being off-road driving and also to balance the dataset with non-zero steering angels. This is achieved by adjusting steering angle by applying correction value +0.27 for left image and -0.27 for right image. Figure 3 shows some examples of images and their corresponding steering angles [2].

To capture good driving behavior, we use center lane driving. As a result, the dataset is dominated by small steering angles, as shown in figure 4. After masking out small angled samples, the steering angles distribution is shown in figure 5.

We ended up with 8,036 samples after more than half an hour of driving around the track 1 in simulator, which is mostly not enough for the model to generalize well, so a couple of data augmentation techniques are applied to extend the dataset during training. These includes randomly flipping image to simulate the opposite direction, shifting image vertically and horizontally to simulate the effect of the car at different positions of a lane, altering image brightness to simulate day and night conditions, cropping image to remove the sky at the top and car front at the bottom. Figure 6 shows the effect of some augmentation [2].

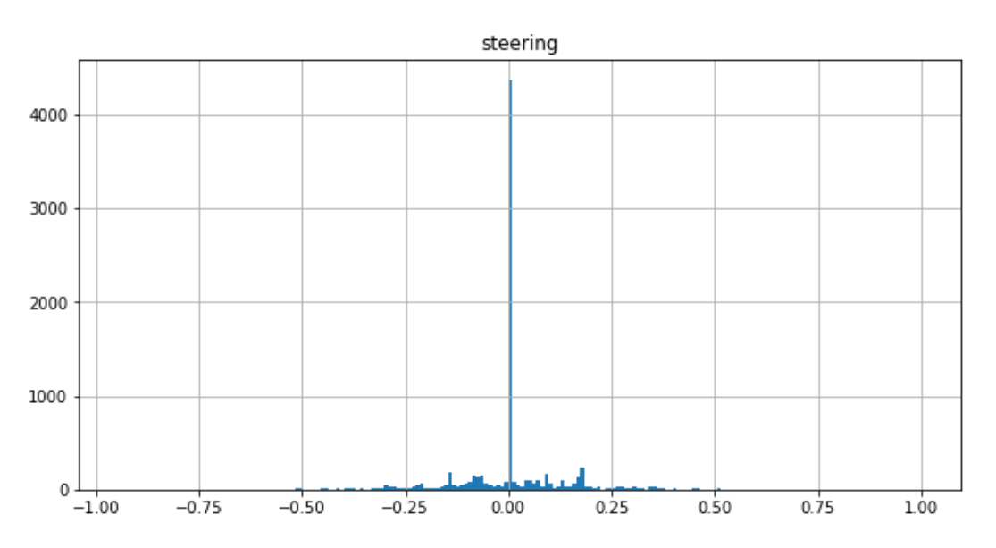


Figure. 4

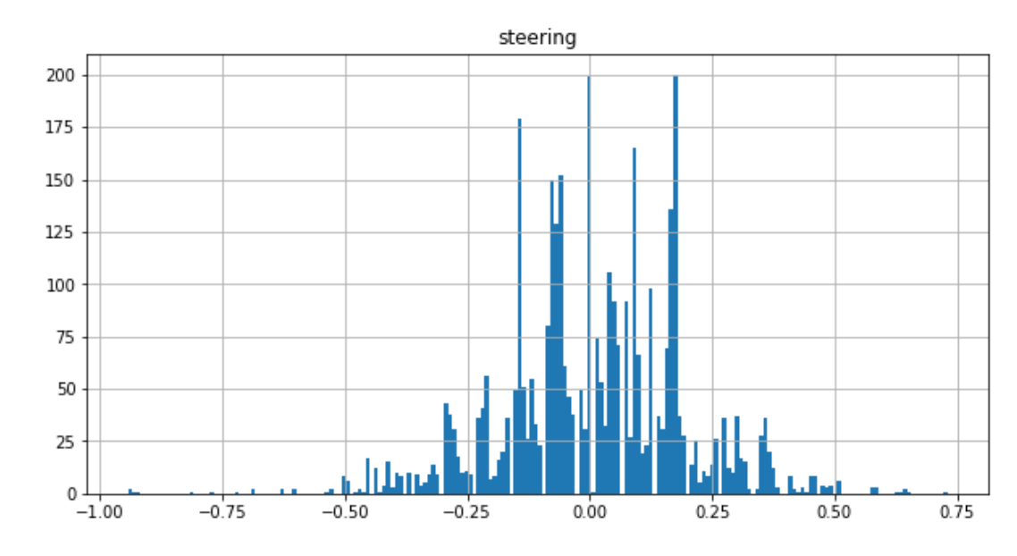


Figure. 5

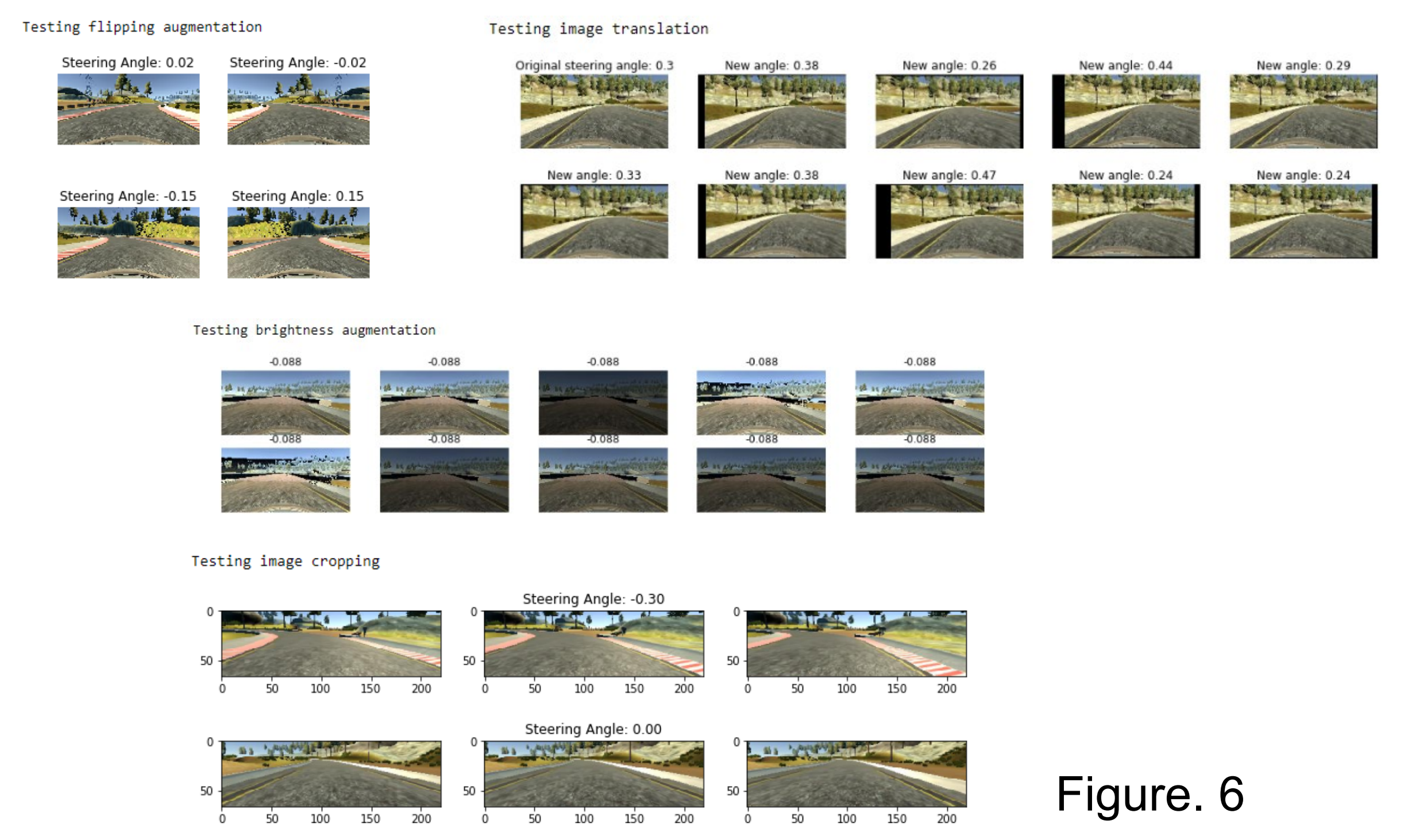


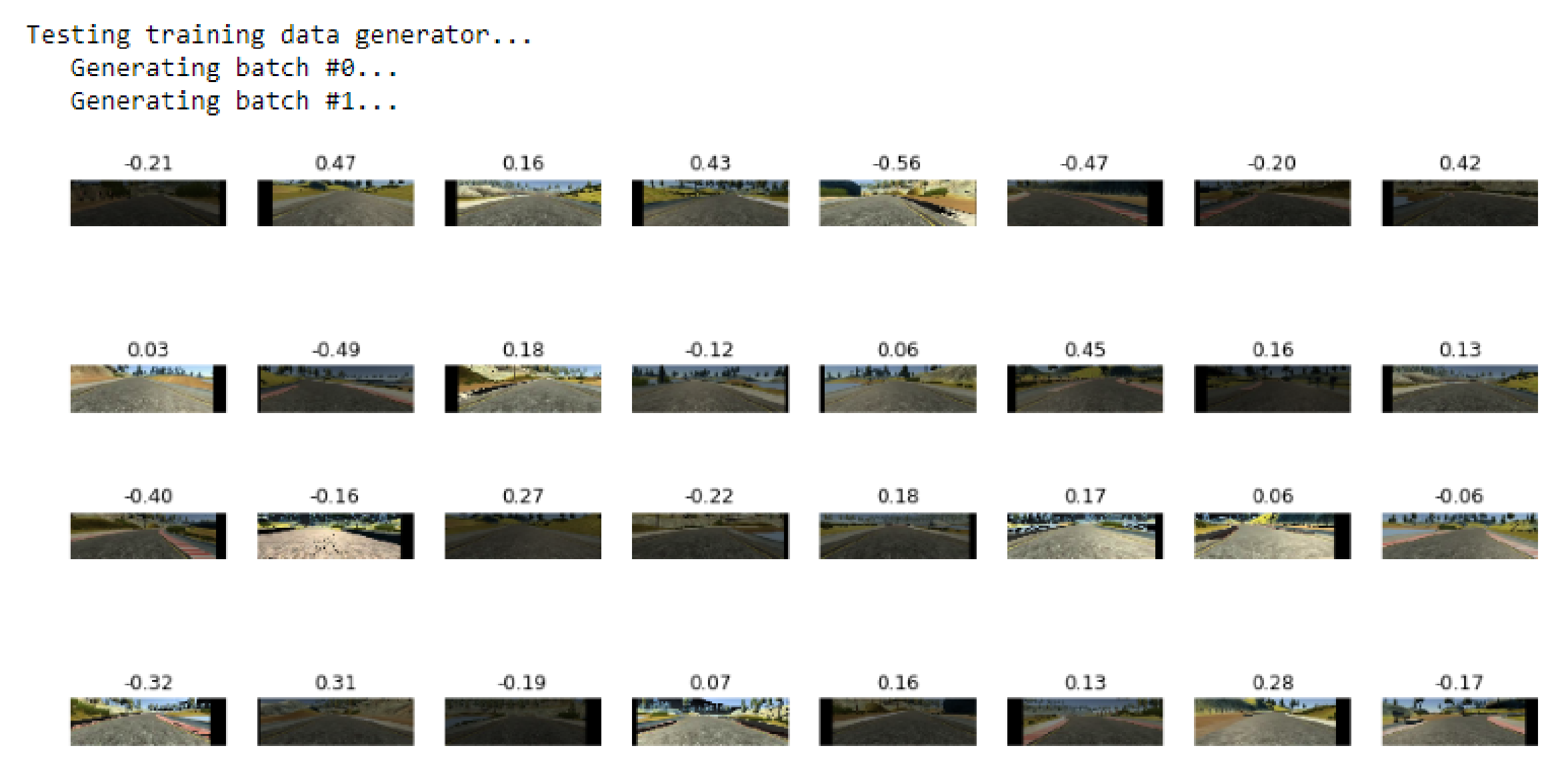
Figure. 6

Augmentation pipeline is applied using Keras batch-by-batch fit_generator in training process to generate data. There are two generators for this project. Training generator is to generate samples per batch; at each batch, random samples are picked, applied augmentation and preprocessing, so training samples feeding into model are always different. Validation generator is also to generate samples per batch for validation, but unlike training generator, only central images are used and data augmentation is not applied.

Model Architecture

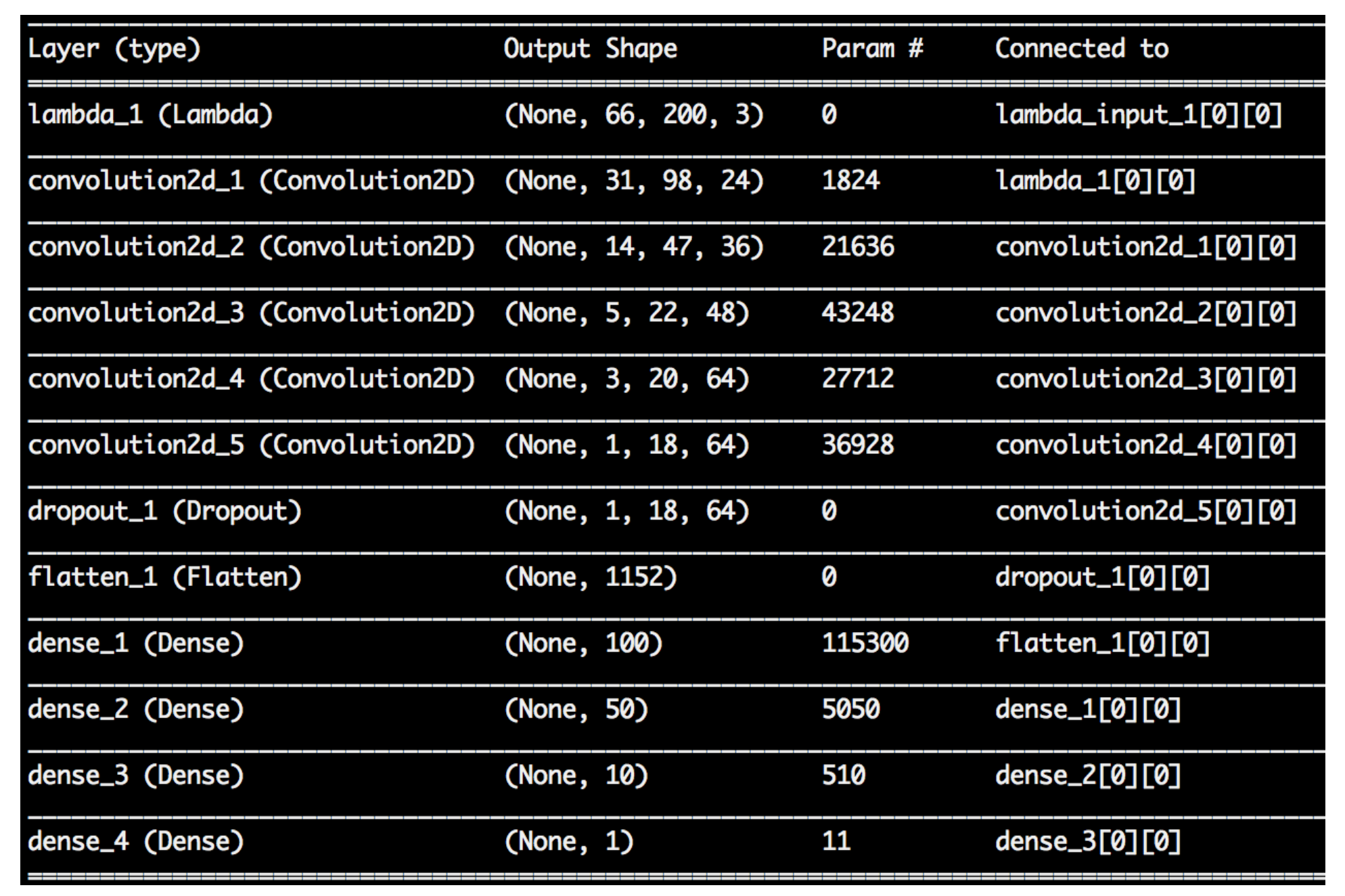
The model is developed using Keras with TensorFlow as backend, and trained in open source Conda environment built in a MacPro 15" with NVIDIA GeForce GT 750M GPU 2GB.

We've started with a well-known and often-used model in autonomous vehicle community from NVIDIA. We've kept adjusting and optimizing the NVIDIA model. Variants of the model have been tried: a simpler model with 3 convolutional layers with depth from 16 to 64 and 3 fully connected layers with 2 dropout out of 3 dense layer. This model achieved the lowest validation loss of all models tried; however, this model has difficulty in making sharp turns: the car ran into the lake at the first sharp turn. A slightly more complicated model with 5 convolutional layer and 5 fully connected layers with L2 weight regularization applied in every convolution and dense layer and 50% dropout for the first dense layers. This model also cannot finish the first lap and would run into the lake at sharp turns. After many trial and error with variants, the model so far turns out to work best is shown in figure 8.



```
Testing training data generator...
   Generating batch #0...
   Generating batch #1...
```

Figure. 7



```
Layer (type)                     Output Shape          Param #     Connected to
====================================================================================
lambda_1 (Lambda)                (None, 66, 200, 3)    0           lambda_input_1[0][0]
____________________________________________________________________________________
convolution2d_1 (Convolution2D)  (None, 31, 98, 24)    1824        lambda_1[0][0]
____________________________________________________________________________________
convolution2d_2 (Convolution2D)  (None, 14, 47, 36)    21636       convolution2d_1[0][0]
____________________________________________________________________________________
convolution2d_3 (Convolution2D)  (None, 5, 22, 48)     43248       convolution2d_2[0][0]
____________________________________________________________________________________
convolution2d_4 (Convolution2D)  (None, 3, 20, 64)     27712       convolution2d_3[0][0]
____________________________________________________________________________________
convolution2d_5 (Convolution2D)  (None, 1, 18, 64)     36928       convolution2d_4[0][0]
____________________________________________________________________________________
dropout_1 (Dropout)              (None, 1, 18, 64)     0           convolution2d_5[0][0]
____________________________________________________________________________________
flatten_1 (Flatten)              (None, 1152)          0           dropout_1[0][0]
____________________________________________________________________________________
dense_1 (Dense)                  (None, 100)           115300      flatten_1[0][0]
____________________________________________________________________________________
dense_2 (Dense)                  (None, 50)            5050        dense_1[0][0]
____________________________________________________________________________________
dense_3 (Dense)                  (None, 10)            510         dense_2[0][0]
____________________________________________________________________________________
dense_4 (Dense)                  (None, 1)             11          dense_3[0][0]
====================================================================================
```

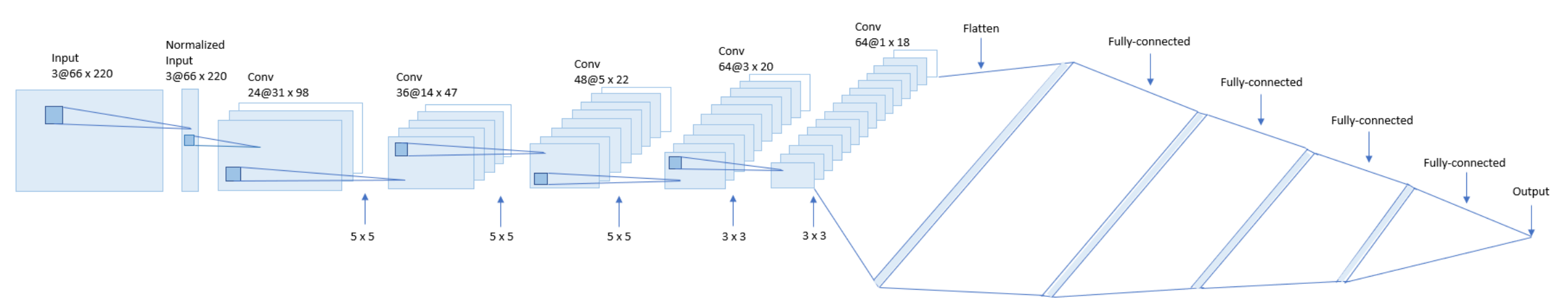


Figure. 8

The network consists of 9 layers, including a normalization layer, 5 convolutional layers and 3 fully connected layers. The convolutional layers are meant to handle feature engineering: strided convolutions are used in the first 3 convolutional layer with 2 x 2 stride and 5 x 5 filter size and non-strided convolutions with 3 x 3 filter size are used in the last two convolutional layers with depth varying from 24 to 64. To avoid overfitting, Relu activation is applied after each convolution layer. Followed by convolutional layers are fully connected layers for predicting the steering angle [3]. The model is trained using Adam optimizer with a learning rate = 1e-4 and mean squared error as loss function. 20% of the training data is split out for validation, and validation loss is as low as 0.0098 after 40 epochs, as shown in figure 9. Contrary to common sense that training loss is always lower than validation loss, our model's validation loss is much lower than training loss. We assume that excessive data augmentation applied in training set makes validation set somehow simpler for the model. Also, we found that low validation loss is not always a great indication of how well car performs on the track. So we tried to save every model at each epoch and see which one drives best on the simulator track.

Model Testing is conducted in the simulator and only the center camera image input is fed to the neural network. The network outputs steering angle value, which is fed to the virtual car. The Udacity simulator servers as the server and the neural network or python program serves as client. The simulator outputs the images, the python program analyses it and outputs the steering angle. The simulator receives this steering angle value and turns the car accordingly. The whole process goes on cyclically [1].

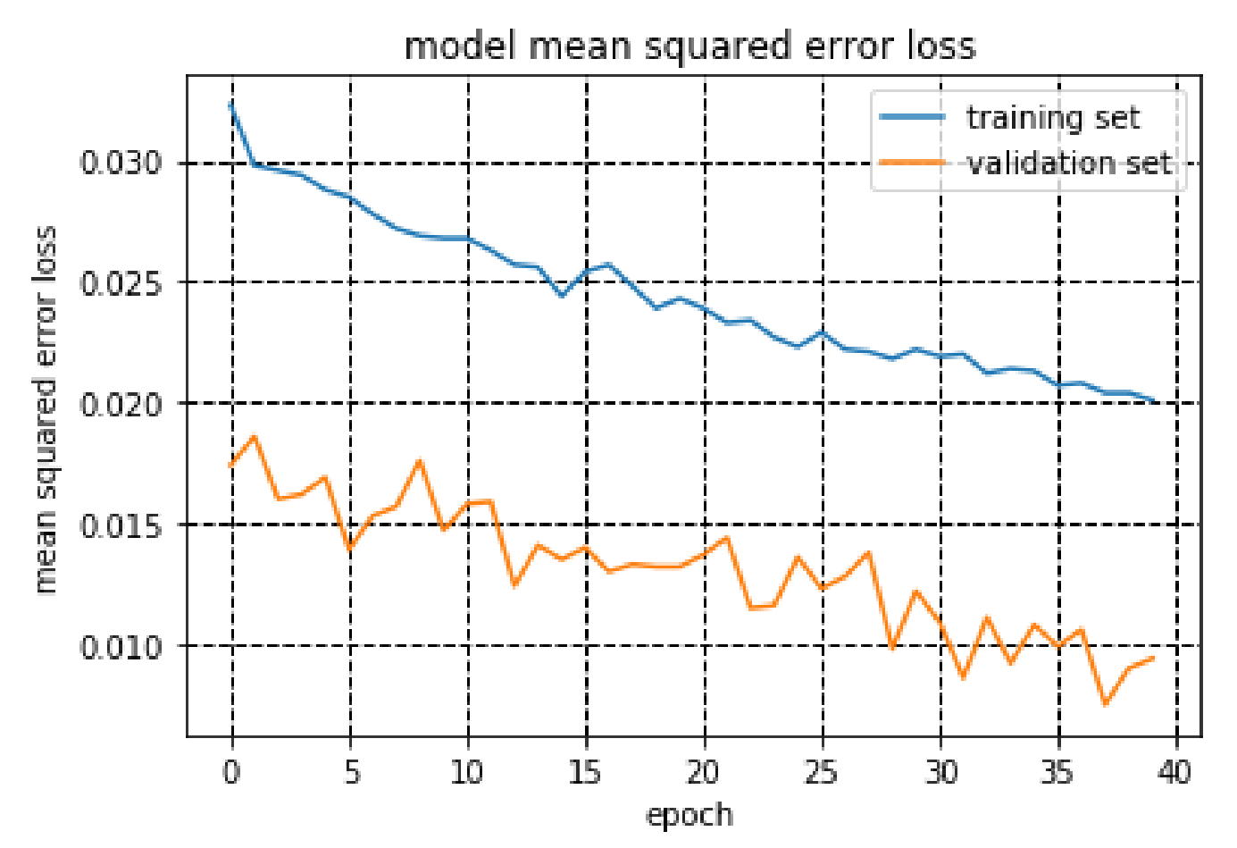


Figure. 9

RESULT & SUMMARY

The car managers to follow the road on the first Lake track for most of the time, which proves that CNN is able to learn meaningful road features from very sparse training signal, steering angle alone as shown on a YouTube video at: https://youtu.be/8PgDVEBlR9g. However, sometimes the car will go off the center of the road after a few laps on Lake Track and will ran into cliff in Mountain track. We think collecting more data from Mountain track and adding recovery scenarios to handle tricky curves and slopes will help reduce the error of the model. Also we plan to try out VGG16 and comma.ai model or to use of recurrent neural network to improve the model.

Reference

[1] Akarsh Zingade.(2018) Autonomous Driving using Deep Learning and Behavioural Cloing
[2] Alex Staravoitau. End-to-end Learning for Self-driving Cars
[3] Bojarski, M., Testa, D. D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., Jackel, L. D., Monfort, M., Muller, U., Zhang, J., Zhang, X., Zhao, J. & Zieba, K. (2016). End to End Learning for Self-Driving Cars.. *CoRR*, abs/1604.07316.
[4] Sammut C. (2011) Behavioral Cloning. In: Sammut C., Webb G.I. (eds) Encyclopedia of Machine Learning. Springer, Boston, MA
[5] Udacity Self-Driving Car Simulator: https://github.com/udacity/self-driving-car-sim