Capstone Project

Machine Learning Engineer Nanodegree

Shanglin Yang

kudoysl@gail.com

December 25, 2016

Definition

1. Project Overview

The recent advancements in self-driving cars are built on decades of work by people around the world. There are many new technologies come out these years to solve problems in different aspects of self-driving.

One basic problem to deal with is to control car steer angle based on information describing the situation around the car. There are several ways to interpreting information, such as camera, radar and GPS. Considering the high volume of image information and low cost of camera devices, the camera images are priority usage in the current self-driving products.

Then, how to process image data and use these information to control car involves knowledge and algorithms from different area, including image processing, computer vision and machine learning. Meanwhile, deep learning, a recent popular technology for end-to-end machine learning implementation, has been widely applied on computer vision related problem.

Thus, in this project, I built a deep learning CNN model to predict steering angle of a car based on image from front direction camera. Specifically, I use car simulator[1] from Udacity to collect virtual images as training data and test the model performance.

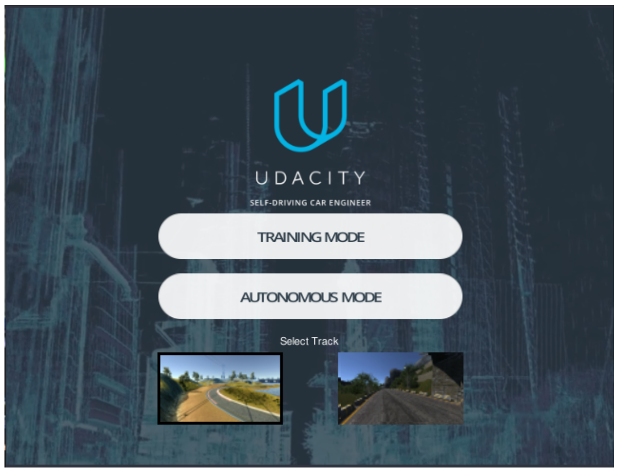
2. Problem Statement

The problem is basically a classic machine learning regression problem. The input are the images from front camera while the output is float value representing steering angle.

There are several ways to accomplish this work in traditional machine technologies, which are all high related with feature extraction. In fact, whether being able to extract good features decide the performance of learning results. However, the high complexity of image data makes it hard to get robust features to make prediction. Thus, I choose to use deep learning network to finish this work, which could extract corresponding features and do the prediction directly.

In conclusion, in the project, I at first collecting training data from simulator, then per-process training data. after the model structure defined, I train the model using processed data and fine-tune parameters. Finally, I test my model based its performance in the simulator.

Simulator UI



Train Data Collection

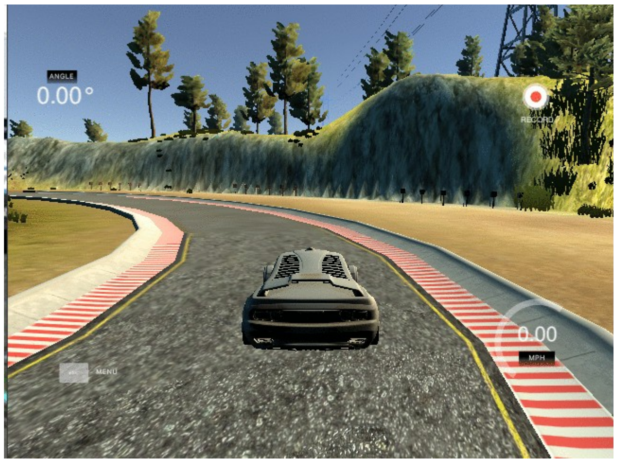


Figure.1 Simulator UI screen shot

3. Evaluation Metrics

In my implementation, rather than generate target value directly. I split it into two independent problem. The model will predict the direction and value separately.

The first one is classification problem and the second one is a regression problem. Thus, I choose to use classification accuracy to measure the performance to prediction and mean square error to measure the regression performance. Detail information introduced in **methodology** section.
Moreover, at last, I used the simulator to test our model. The simulator will load model structure and weight automatically and control the car to run on the road. If the model is trained well, the car should run within safe area without falling off.

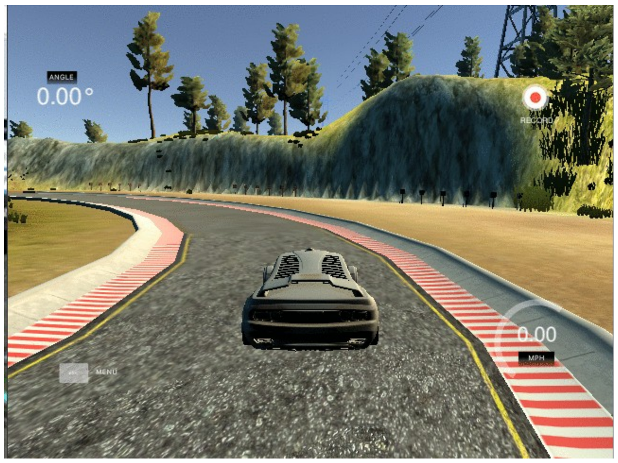
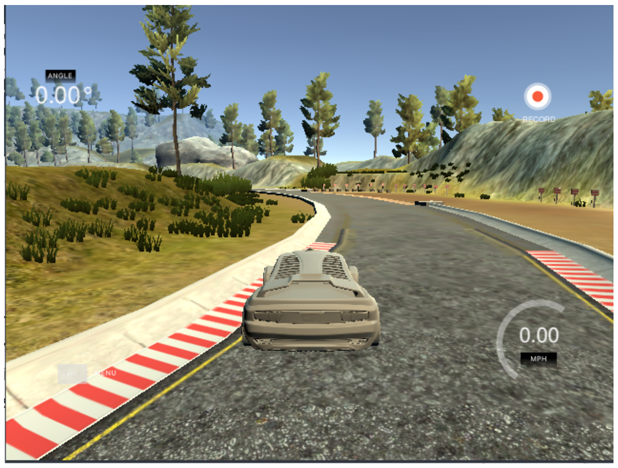
	
Satisfied	**Wrong Performance**

Figure.2 Simulator test Performance

Analysis

1. Data Exploration

I collect data using the training mode of the simulator by driving the car with your arrow keys The data would be saved with images from center camera and the corresponding steering angle. Continue driving for a few laps The image paths and angle data will be saved in csv file. There will be one IMG folder which contains all the frames of your driving and one driving_log.csv in which each row of the sheet correlates image with the steering angle, throttle, brake, and speed. I will mainly use the steering angle.

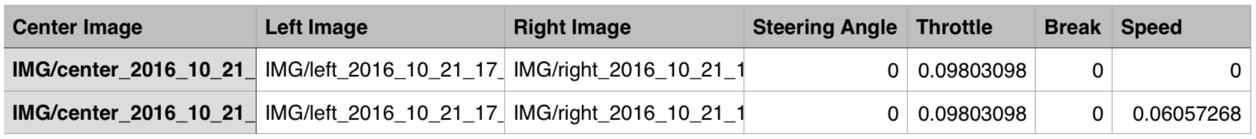
The key of the data collection is to cover all possible situation and keep sample number balance. There are three direction type - keep straight, turning left and turning right. They are the three classes I will use later. The balance is to make sure the image number of three sections should not vary much. Thus, I drive the car clock-wise and anti-clockwise of the lab several times. This will also lead to lots of zero-value samples, so I down-sample them to remove the redundancy.

Besides, I at first need to collect data for normal situation. However, if the training data is all focused on running in the middle of the road, the model won't ever learn what to do if it gets off to the side of the road. So I need to teach the car what to do when it's off on the side of the road. One approach is to constantly wander off to the side of the road and then wander back to the middle.

Moreover, these data may still not be able to cover all situations could happen in the driving process. Thus, I choose to collect data from where the model perform not good enough to reinforce our model. Another problem with the training data is the 'noisy'. Because of the sensitivity of the simulator, it is hard to generator smooth and accurate data. These noisy will low down the model performance and decay the speed of convergence. Data requiring smoothing. More detail will be introduced in the Methodology section.

2. Exploratory Visualization

Here are the visualization of the training csv file and the samples image data with corresponding steering angle.



Center Image	Left Image	Right Image	Steering Angle	Throttle	Break	Speed
IMG/center_2016_10_21_	IMG/left_2016_10_21_17_	IMG/right_2016_10_21_1	0	0.09803098	0	0
IMG/center_2016_10_21_	IMG/left_2016_10_21_17_	IMG/right_2016_10_21_1	0	0.09803098	0	0.06057268

Figure.3 CSV recording information

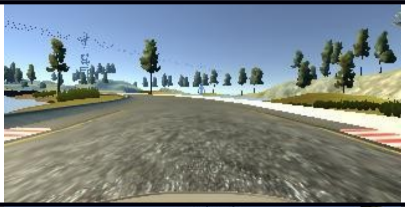
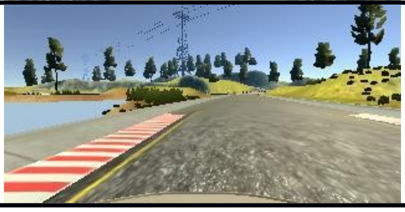
Description	Image	Steering Angle(float)
Go Straight		0
Turn Left		-0.2
Turn Right		0.2

Figure.4 Sample Train Image data

3. Algorithms and Techniques

This problem is basically an regression problem of machine learning. In this project, I decide to use to deep learning methods building and training CNN model to resolve this. The key of CNN model is to use convolution layer extracting useful information from image data. Besides, to improve performance, activation layer bring in non-linearity. Moreover, using drop_out layer and normalization layer make the model more robust.

Furthermore, rather than, use only one unit as the final output layer, which represents the prediction of the steering angle, considering the high noisy in training data, I separate the final results into two problems, one as classification, and another one as regression.

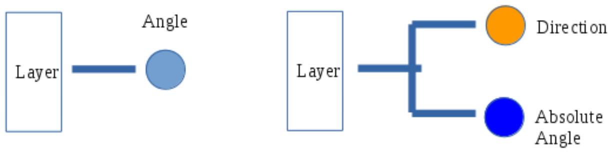


Figure.5 Previous and Current output structure

The classification is for the direction decision. I made it as a three-classes problem. The label 0 represents turning left, label 1 represents going straight, and label 2 represents turning right. The reason to do that is because the direction is more robust . For example, when we turn right, the angle varies much considering the simulator sensitivity or different situation, however, the direction is still same which has high confidence

The regression problem is for absolute steering angle.

Then, according the equation (1), the model calculates final angle controlling car.

$$Steering - Angle = \begin{cases} -Angle_{abs} & prediction = 0 \\ 0 & prediction = 1 \\ Angle_{abs} & prediction = 2 \end{cases} \tag{1}$$

4. Benchmark

The Benchmark is the random steer angle generation. Mose specific in my project, the basic requirement is to finish lap without falling off.

Methodology

1. Data Preprocessing

1) Normalization : Normalize input image data from 0~255 to -1 to 1.

$$Data_{norm} = (Data/255.0) * 2 - 1 \tag{2}$$

2) Smoothing

Considering the sensitivity of the simulator. The training data contains lots of noisy . Thus, it is better to smoothing the training data. I use pyasl.smooth[2] to smooth the training data. This would generate steering data vary continually. Below is the comparison of before and after smoothing .

3) **Augmentation and Cross_validation Split**

To weak the effect of overfitting and gather more valuable information, I generated augmented images and split data into training set and test set.
The augmentation operations include zoom scale, shift, brightness noisy, and random rotation.

The ratio for training to validation is 10 to 1.

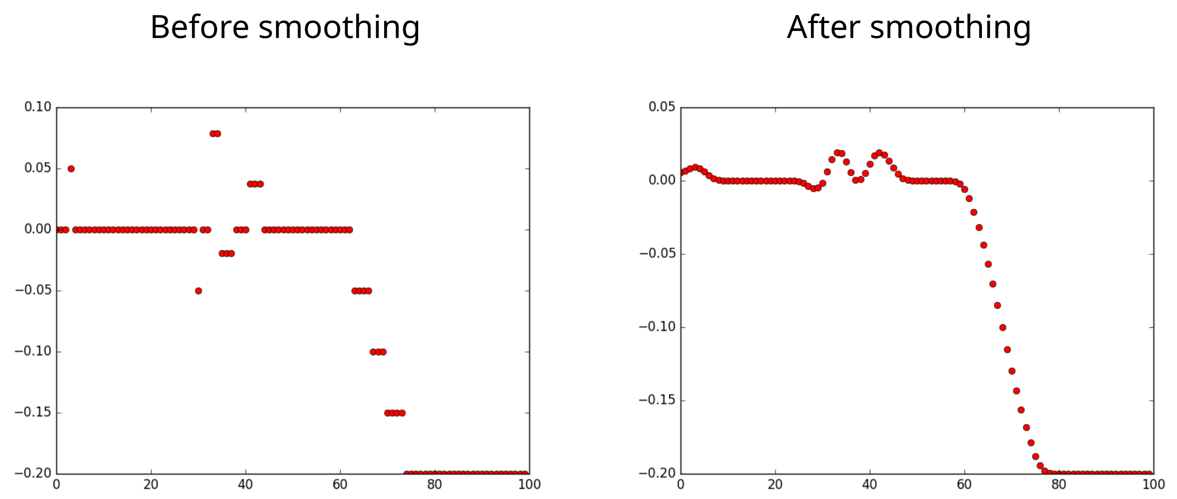


Figure.6 Comparison of non-smoothed and smoothed training data

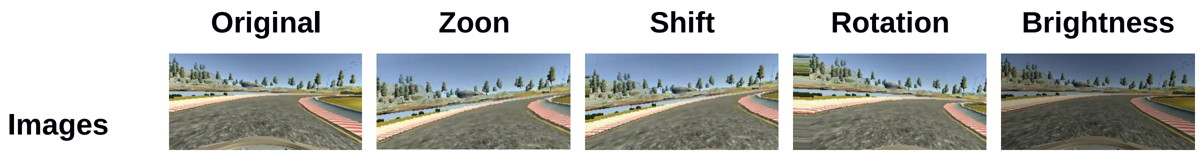


Figure.7 Augmentation Train samples

4) **Data Refining**

To ensure better performance and speed up the convergence. I decided to refine training data.

Thus, I at first remove half of zero value training data, then I built an Python-based API to refine the data by visualize the input image and target value.

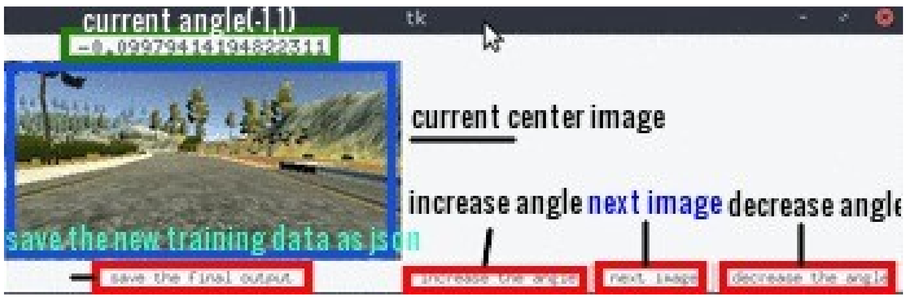


Figure.8 Train Refinement API

Within this API , I can take a look of the current image and prediction from our model. I then increase or decrease the target value based on the image, which is called data refinement. It on the one hand, help to remove the wrong training data, on the other hand, it decrease the noisy in training data. By the way, it also help to see what the car do wrong in the test.

5) Data conclusion

The conclusion of data per-processing shown in the table.

I at first split training and validation, then do Normalization on both train and validation set. After that, I only do Smooth and Augmentation on training data. The reason for that is I only need to generate more useful information for training but not want to disturb the accuracy when doing evaluation. Thus, I kept the original data for test.

Table 1. Data conclusion

	Train	**Validation**
Original	20000	4000
NOrmalization	Yes	Yes
Smooth	Yes	No
Augmented	10000	0
Refined data	10000	0
Total	40000	4000

2. Implementation

1) Tools

In this project, I used tensorflow[3] as the back-end and Keras[4] as front library for implementing Deep Learning method. The codes are written in Python.

2) Model Structure

The basic structure inspired by the idea from the reference paper [5]. The key differences include three parts:

- I use RGB value rather than Y value as input;
- Rather than take the steering angle as output. I separate the output into two part.
- Bring in activation,normalization and dropout layer.

The main graph includes convolution-activation-normalization called TRI_structure for feature extraction, full connected layer for combine global information, and softmax layer for probability generation.

The input image at first go though five TRI_structure, then information are combined after two full connected layer, the final features feed into two branches. One is sent to another full connected layer to one unit and another one to full connected and softmax. Table 2 describe layer information and figure 9 describe model graph structure.

Table 2. Basic Layer information

Layer Name	**Layer Number**
Convolution2D	5
Full connected	5
BatchNormalization	7
Activation layer	9
Dropout layer	1
Total	27

3) Training Methods

Train the model using designed structure. The input data is processed as introduced above. Then I split it into training dataset and evaluation dataset. Feed the training image data into the model and train the model using specific loss function and optimizer method. Meantime, after each epoch, evaluate model performance using evaluation dataset. At last, fine-tune the model hyper parameter based on the evaluation performance.

Loss Function: For classification, I used categorical_crossentropy and for regression, I use 'MSE'.

Optimizer: I choose 'Adam', which not only stores an exponentially decaying average of past squared gradients like Adadelta and RMSprop, but it also keeps an exponentially decaying average of past gradients. RMSprop is an extension of Adagrad that deals with its radically diminishing learning rates. It is identical to Adadelta, except that Adadelta uses the RMS of parameter updates in the numinator update rule. Adam, finally, adds bias-correction and momentum to RMSprop.

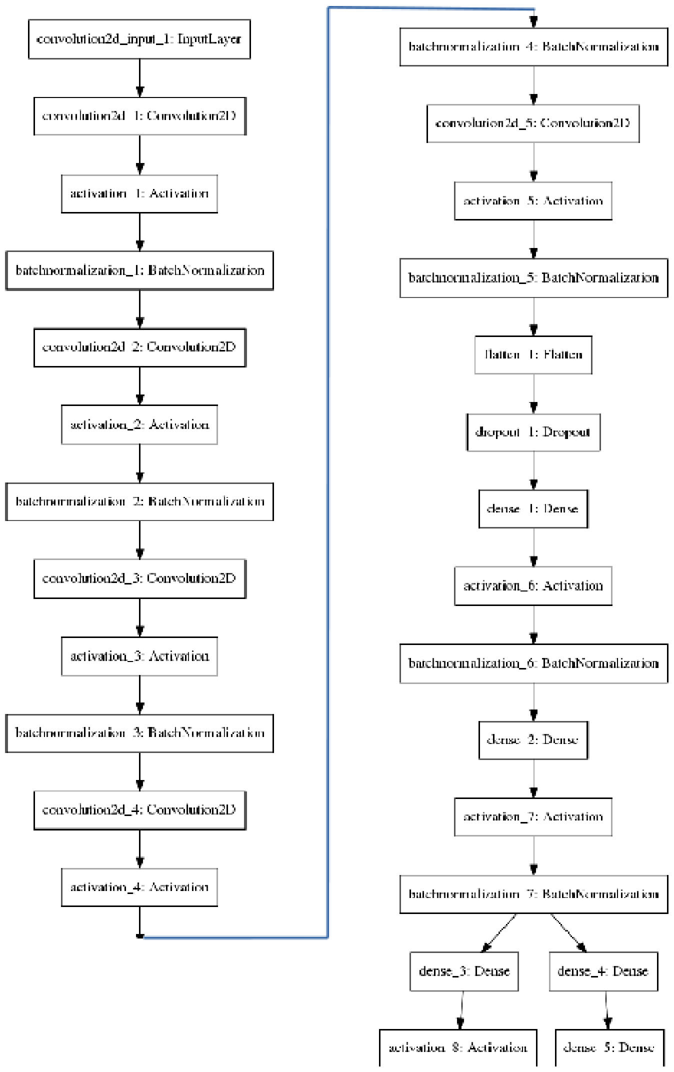
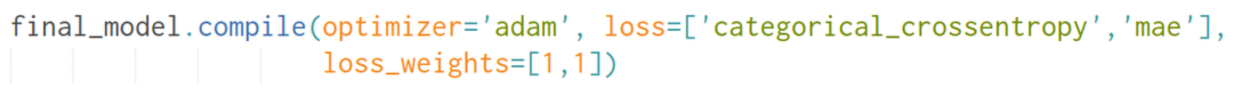

convolution2d_input_1: InputLayer
convolution2d_1: Convolution2D
activation_1: Activation
batchnormalization_1: BatchNormalization
convolution2d_2: Convolution2D
activation_2: Activation
batchnormalization_2: BatchNormalization
convolution2d_3: Convolution2D
activation_3: Activation
batchnormalization_3: BatchNormalization
convolution2d_4: Convolution2D
activation_4: Activation
batchnormalization_4: BatchNormalization
convolution2d_5: Convolution2D
activation_5: Activation
batchnormalization_5: BatchNormalization
flatten_1: Flatten
dropout_1: Dropout
dense_1: Dense
activation_6: Activation
batchnormalization_6: BatchNormalization
dense_2: Dense
activation_7: Activation
batchnormalization_7: BatchNormalization
dense_3: Dense
dense_4: Dense
activation_8: Activation
dense_5: Dense

Figure.9 Model Graph Structure

Insofar,methods RMSprop, Adadelta and Adam are very similar algorithms that do well in similar circumstances. Kingma et al. [6] show that its bias-correction helps Adam slightly outperform RMSprop towards the end of optimization as gradients become sparser. Insofar, Adam might be the best overall choice.



```
final_model.compile(optimizer='adam', loss=['categorical_crossentropy','mae'],
                    loss_weights=[1,1])
```

Figure.10 Keras Implementation of training model

Hyperparameter: For Learning rate, cause I use 'Adam', it would decay automatically.

The other parameter need to Fine-tune is Drop_out Percentage. Thus, I will change the percentage to 0, 0.15, 0.25,0.5 and 0.75 to see the model performance.

Evaluation: Using evaluation dataset. The performance based on loss function and accuracy metric.

3. Refinement

My refinement including splitting output result into two problem rather than basic regression problem. And also fine_tuning the parameter of drop_out percentage.

1). **Split output.**

Table 3 . Performance of Splitting output (150 epochs. 6400 samples/epoch)

Performance Description		**Regression solution**	**Regression + Classification**
Training	**MSE**	0.256	0.051
Test	**MSE**	0.233	0.032
	Finish Running	NO	YES

From table above, without splitting , the car can not finish the loop, while it successfully finish running after splitting. Besides, the MSE errors decrease for both train and test dataset.

2). **Fine_tune Parameter**

The target of fine-tunning is to find best drop_out percentage. Dropout is a regularization technique for reducing overfitting in neural networks by preventing complex co-adaptations on training data.

From the table below. We can see the Training MSE not vary much for after enough training epochs , the training error should be minimized. But the performance for validation improved along with the percentage.

Table 4 . Performance of Drop_out Percentage varies (150 epochs. 6400 samples/epoch)

Performance Description		**Drop_out Percentage**				
		0	0.15	0.25	0.5	0.75
Training	**MSE**	0.051	0.057	0.053	0.052	0.051
Test	**MSE**	0.047	0.046	0.038	0.032	0.035
	Finish Running	YES	YES	YES	YES	YES

Results

1. Model Evaluation and Validation

1) Model Evaluation performance

Table 5. Training Performance along with epochs (6400 samples per epoch)

Performance Description		**Epochs**				
		10	**20**	**50**	**100**	**150**
Training	**MSE**	0.697	0.355	0.138	0.062	0.052
Test	**MSE**	0.472	0.324	0.108	0.068	0.032
	Finish Running	No	No	No	YES	YES

From table above, we can see the MSE decrease along with the epochs increase. It finally become stable around 100 to 150 epochs.

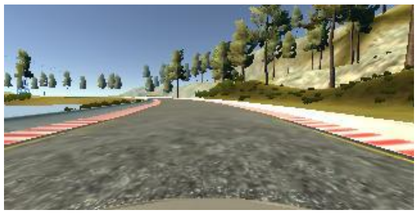
For my final model, I use the model from 150 epochs.

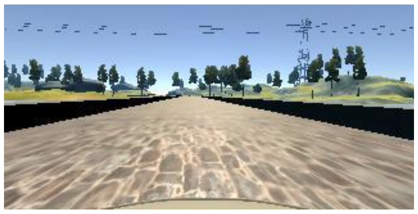
2) Samples of output

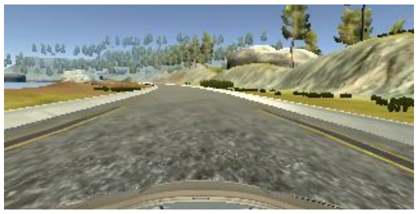
Here I show some samples of model prediction results.

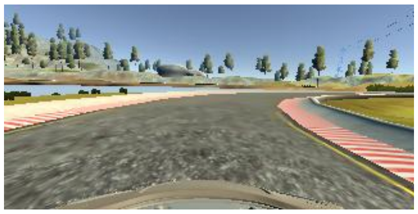
Table 6. Samples of model prediction results.

Images	Angle/ Prediction/ |Angle|

	Angle = -0.21
	prediction = 0 (turn left)
	|Angle| = 0.21

	Angle = 0
	prediction = 1 (go straight)
	|Angle| = 0

	Angle = -0.05
	prediction = 0 (turn left)
	|Angle| = 0.05

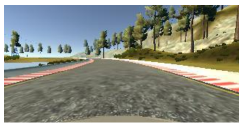
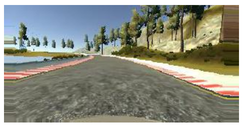
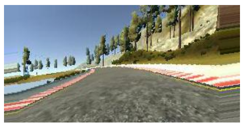
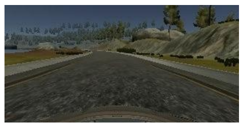
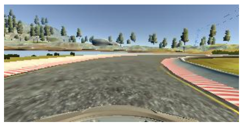
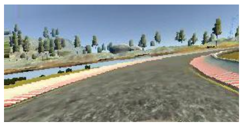
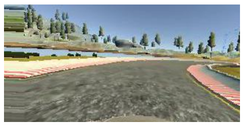
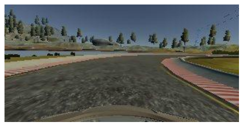
	Angle = 0.25
	prediction = 2 (turn right)
	|Angle| = 0.25

3) Various inputs Performance

Here I test whether the model robust to little variation of input data. Here I test mode by zooming , rotating and Adjusting Brightness to see performance.

From the table below, the angle not vary much for different input data. Thus, the model can be taken robust enough.

Table 7. Performance for various input data

Operations	Original	Zoom	Rotate	Brightness
Images				
Steering Agrees	- 0.212	- 0.201	-0. 221	-0.213
Images				
Steering Agrees	0.255	0.277	0.269	0.253

2. Justification

After training, the car in the simulator could run through the lap with right prediction angles for input images. Compared to benchmark, the performance has been high improved. Thus, the model has satisfied it target. For full running video. Take the sample video[7].

Table 7. Performance comparison to benchmark

	MSE	**Finish Running**
Benchmark	3.015	No
My_model	0.032	Yes

Moreover, we can see the model results do not vary much according to small change of input data. The model is robust to various situation image data.

Conclusion

1. Free-Form Visualization

Here, we explore the quality of model by visualize difference between the exploration angle and prediction angles.

From figure we can find two point:

The first one is that the prediction angle is high similar to evaluation angle. In other words, the difference between them is small enough. Moreover, the direction of angle are almost same. Thus, the model’s performance is guaranteed.

Besides, from the plot, we can see the prediction results are mush smoother than evaluation samples. In more detail, when car need to turn to one direction, the absolute predicted angle value increase earlier than evaluation and during the process, the prediction seems to inclined to be smaller than the evaluation.

This property represents the robustness of model for it can prevent car from ‘bad’ position by exploring deep the image information. Thus, there are less required steering angle along the turning process. It is good for driving in practical application for people won’t take risk of making sudden big steering angle in driving and prefer smoother process which is safe and comfortable.

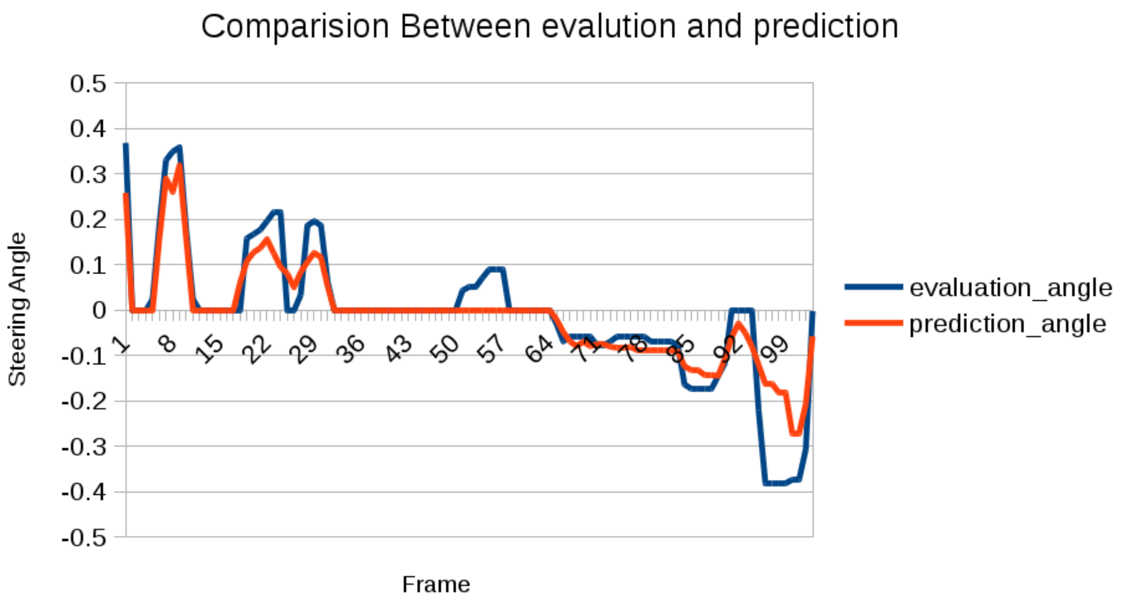


Figure.11 Difference between the exploration angle and prediction angles

2. Reflection

This project is the practical application of deep learning model for special target- self-driving. The end-to-end model takes image data as input and steering angle as the final output. The project is valuable and interesting. Because it can be applied to real usage in human life. It is interesting to detect useful information from the massive number of image data automatically.

Meanwhile, the difficulties of this model also with its super power. The hard points include two aspects, one is for training data generation and another one is model construction.

The difficulties of data generation is caused by the sensitivity of simulator and high variety of image data. To gather and generate valuable input data, I used several methods. At first, I smooth the training data to remove noisy information, then, I balance the positive, negative and zero value input data. Finally, I re-generate training data from where the model performs incorrectly. These methods help to improve the valuable information within the training data and remove misleading ones.

For model construction, the key is how to applied basic CNN structure to fit our model, for example, to speed up training, I need to down-sample image-size. Besides, to weaken overfitting, using drop-out layer and fine-tunning its parameter.

Finally, the model finally fit requirement giving good prediction for input data. Besides, the mode is robust input variation. Moreover, the model provides a smoother and safe control of the car.

3. Improvement

The model generate good performance on this graph. But it may required training on limited specific situations. Sometimes, the front information may be not enough,. Thus, one thing I can improve is to use left and right camera to enhance performance.

Another thing, currently, I do not take the speed into account for controlling. In the future, I will use speed information for training and try to control both steering angel and speed. It will help to increase the stability and overall speed of the car running. Moreover, this part could be done using reinforcement learning. Take image data and speed as status, the car itself as agent. Try to guide the car in an continually changing environment.

Reference

[1] Udacity selft_driving non_degree: https://medium.com/udacity/challenge-2-using-deep-learning-to-predict-steering-angles-f42004a36ff3#.odmez3qj7

[2] PyAstronomy 0.10.0 documentation: http://www.hs.uni-hamburg.de/DE/Ins/Per/Czesla/PyA/PyA/pyaslDoc/aslDoc/smooth.html

[3] Tensorflow Homepage: https://www.tensorflow.org/

[4] Keras Homepage: https://keras.io/

[5] Bojarski, Mariusz, et al. "End to End Learning for Self-Driving Cars." arXiv preprint arXiv:1604.07316 (2016).

[6] Kingma, D. P., & Ba, J. L. (2015). Adam: a Method for Stochastic Optimization. International Conference on Learning Representations, 1–13

[7] Final mode performance sample video: https://www.youtube.com/watch?v=SNHYIC_zBR4&feature=youtu.be