Machine Learning Engineer Nanodegree
Capstone Project : DeepTesla

Guangwei Wang

March 15, 2017

1 Definition

Project Overview

This project is based on Course MIT 6.S094: Deep Learning for Self-Driving Cars [1].

There is an unprecedented interest, activity, and excitement in the field of intelligent vehicles [2]. The culmination of research efforts of the past decades in a broad range of disciplines, including vehicle control, robotics, sensing, machine perception, navigation, mapping, machine learning, embedded systems, human-machine interactivity, and human factors, has realized practical and affordable systems for various automated features in automobiles.

In vehicle control task, generally, the output of an obstacle avoidance system is a set of possible steering angles that direct the robot toward traversable regions. One straightforward idea is to view the entire problem of mapping input an image frame from a front camera to possible steering angles as a single indivisible task to be learned from End-to-End [3]. In this approach, the autonomous driving system is often learned from data rather than manually designed, mainly due to sheer complexity of manually developing a such system video data[4].

The End-to-End learning can be traced back to late 80s [5], in which ALVINN (Autonomous Land Vehicle In a Neural Network) with a single hidden layer neural network is designed for lane tracking. It takes images from a camera and a laser range finder as input and produces the steering wheel angle as output to follow the road. Later, a CNN (convolutional neural network) was trained to to emulate the obstacle avoidance behavior of a human driver for an off-road robots [3]. A fully integrated real-time system for autonomous off-road navigation that uses end-to-end learning from onboard sensors, operator input, and stereo cameras to avoid myopic behavior [6]. Recently, a deeper CNN [7] and CNN combined with LSTM [8] method are used for lane tracking based only on a front camera.

In all these cases, deep neural network has been found to be surprisingly effective at learning a complex mapping from a raw image to control. In this project, the End-to-End learning approach will be adopted to learn a model from Tesla dataset for steering wheel angle control.

Figure 1: Video frame snapshot

Table 1: The CSV data format

ts_micro	frame_index	wheel
1464305394391807	0	-0.5
1464305394425141	1	-0.5
1464305394458474	2	-0.5

Problem Statement

The goal of this project is to predict the steering wheel angel from Tesla dataset based on the video of the forward roadway.

Datasets and Inputs

This dataset can be found in this github [9].

There are 10 videos in this dataset, each video is encoded in H264/MKV with the resolution of 1280x720 and 30 fps. One corresponding csv file records the steering wheel angle along each frame of these videos. One video frame sample is shown in Fig. 1.

Wheel value was extracted from the in-vehicle CAN, and the corresponding steering value linking with each video frame is recorded in a csv file. The csv data format is expressed in Table. 1, in which ts_micro is time stamp, $frame\_index$ is frame number, $wheel$ is steering wheel angle (Based on the horizontal direction, + denotes clockwise, - denotes counterclockwise).

Problem Definition

Let x_t denotes the t-th frame of the dataset, $X_t = \{x_{t-n+1}, x_{t-n+2}, ..., x_t\}$ denotes a n frames long video. $Y_t = \{s_{y-n+1}, y_{t-n+2}, ..., y_t\}$ expresses steering wheel angles associated with these

Figure 2: Typical driving scenarios

frame. Training a model to drive the vehicle can be defined as estimating the function $F : \mathbb{R}^{1280x720x3xn} \times \mathbb{R}^n \rightarrow \mathbb{R}^{1280x720x3}$ that predicts $x_{t+1} = f(X_t, Y_t)$.

Note that function $f(X_t, Y_t)$ in this definition can be approximated in a straightforward way with the proposed End-to-End method.

Evaluation Metrics

This is a regression problem, MSE metrics is adopted here.

$$MSE = \frac{1}{N} \sum_{i=1}^{N} \sqrt{y_p - y} \tag{1}$$

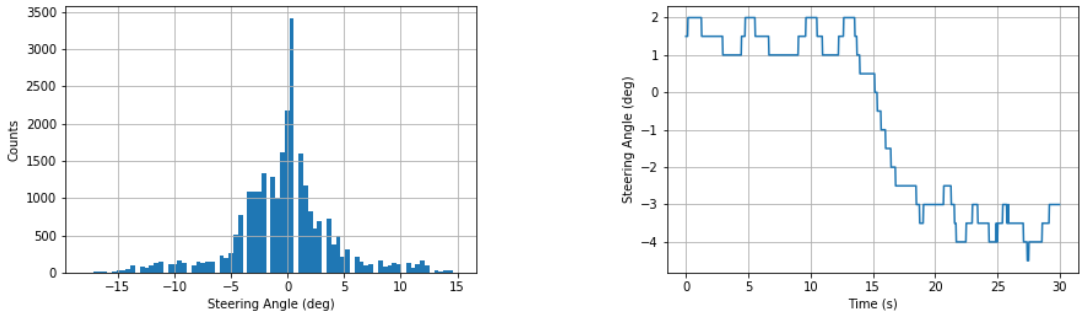
where, y_p is the predicted steering wheel angle, y denotes the reference steering wheel angle, and the sample number is N. The lower MSE is preferred.

Remark 1 *MSE is not reasonable in some real cases, such as self-driving car. The percentage of the vehicle's autonomy time in where by counting the simulated human interventions that occur when the simulated vehicle departs from the center line by more than one meter is introduced in [7]. However, MSE is still used to simplify this simulation in this project.*

2 Analysis

Data Exploration

These videos are captured by a centered front-facing camera. the ROI (region of interest) lies in the lower half except for the bottom part contains vehicle head, in where it contains lane marking of the road. The upper half is the sky states. In these videos, there are different road scenarios, i.e., straight, cure, upslope and downslope, in which there contains sunny and



(a) (b)

Figure 3: (a)Histogram of Steering Angle; (b) Driving operation snippet

cloudy weather. Great brightness caused by the sun glare andblurry lane lines are observed .Some typical driving scenarios are given in Fig. 2. With a glance of the steering data, there are 27,000 samples, and they are located in [-18, 15] deg.

Exploratory Visualization

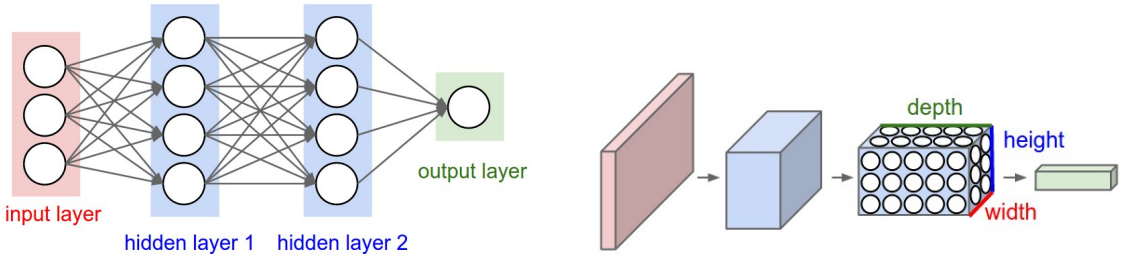
The histogram of steering wheel angle is expressed in Fig. 3(a), in which we can see, the steering angel distributed in the center more than other position, which means the model may less sensitive when come across some sharp corners. Moreover, a snippet of driving operation with 30 seconds long is shown in Fig. 3(b). The steering angle with human driving is discrete and noncontinuous, however, it cannot be considered with a classification problem due to the curse of dimension.

Algorithms and Techniques

To employ End-to-End method, we need to extract information from raw pixes firstly. Convolutional Neural Networks (CNN) can succeed in the role with the computational ability increasing. CNNs are very similar to conventional Neural Networks, they are made up of neurons that have updatable weights and biases. Each neuron receives some inputs, performs a dot product and optionally follows it with a non-linearity, i.e. activation function. The differences lie in that convnets arranges its neurons in three dimensions (width, height, depth), as visualized in one of the layers, see Fig. 4. Every layer of a ConvNet transforms the 3D input volume to a 3D output volume of neuron activations. In this project, the red input layer holds the image, so its width and height would be the dimensions of the image, and the depth would be 3 (Red, Green, Blue channels).

Generally, CNN consists of three main types of layers to build ConvNet architectures: Convolutional Layer, Pooling Layer, and Fully-Connected Layer (exactly as seen in regular Neural Networks):

- Convolutional Layer will compute the output of neurons that are connected to local regions in the input, each computing a dot product between their weights and a small region they are connected to in the input volume.



(a) (b)

Figure 4: (a) A regular 3-layer Neural Network; (b) A Convolutional Neural Network [10].

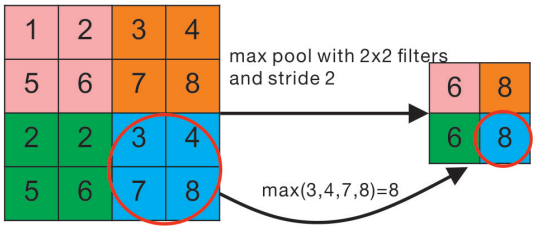


Figure 5: Max pooling

- Pool layer will perform a down sampling operation along the spatial dimensions (width, height). For example, We slide one 2 x 2 window by 2 cells (also called stride) and take the maximum value in each region. As shown in Fig. 5, this reduces the dimensionality of our feature map.

- Fully-connected layer will compute the high-level reasoning, steering angle in this case. Before this, the matrix has to be flattened, because the spatial information of rows and columns doesnt matter any longer. As with ordinary Neural Networks and as the name implies, each neuron in this layer will be connected to all the numbers in the previous volume.

In this way, ConvNets transform the original image layer by layer from the original pixel values to the final steering angle, which is compared to the angle of the human driver.

To increase performance of the training model, the following parameters will be tuned:

- **Preprocess parameters**: image size, ROI.

- **Hyper parameters**: number of epochs, batch size, optimizer type, learning rate.

- **Network architecture**: number of layers, layer types, layer parameters.

3 Methodology

Data Preprocessing

The data preprocessing code is written in preprocessing.py file and called in the DeepTesla.ipynb notebook, the following steps are included:

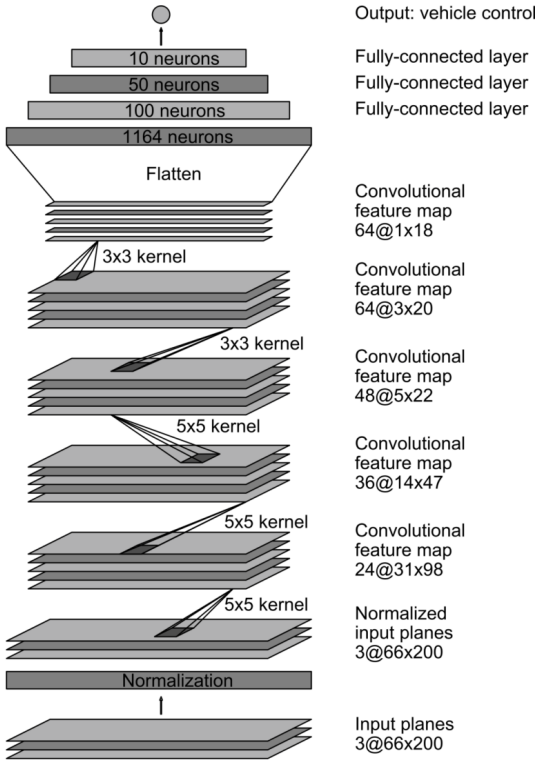


Figure 6: benchmark training results with MSE metirc

- Crop off 1/2 from the top and cut bottom 150px, which contains the head of car.
- Reshape images size into 66x200x3, keep their original height/width ratio, split the input image into YUV planes.
- Use epoch 1-9 as training data, select the remaining epoch 10 as test data. The input is each 66x200x3 pixes, and the corresponding steering angle is output.
- Flip the images to get double training data.
- Normalize the input training data divided by 255.
- Split training data into train data and valid data.

After data preprocessing, there are 24,300×2 (flip) training and 2,700 test samples with 66x200x3 pixes.

Two kinds of data split method, i.e. with or without shuffle, are used in this project to compare their effects on model performance. The vehicle control is continuous time process and the steering angel in the previous frame can be referable for the next. We should keep the order of input data definitely while LSTM is used, because LSTM is a recurrent neural network (RNN) architecture and excels at remembering values for either long or short durations of time. However, CNN is used only in this project, the impact of data order is investigated for the CNN algorithm to see the difference between these two kinds of data.

Benchmark

NVIDIA model is used as benchmark in this project [7], as shown in Fig. 6, which consists of 9 layers, including a normalization layer, 5 convolutional layers, and 3 fully connected layers. The input image is split into YUV planes and passed to the network.

The first layer of the network performs image normalization. The normalizer is hard-coded and is not adjusted in the learning process. Performing normalization in the network allows the normalization scheme to be altered with the network architecture, and to be accelerated via GPU processing. The input size is (200, 66, 3) representing a width of 200, a height of 66, and 3 channels (red-green-blue). The convolutional layers are designed to perform feature extraction, and then use strided convolutions in the first three convolutional layers with a 22 stride and a 55 kernel, and a non-strided convolution with a 33 kernel size in the final two convolutional layers. The fully connected layers are designed to function as a controller for steering. However, by training the system end-to-end, it is not possible to make a clean break between which parts of the network function primarily as feature extractor, and which serve as controller. The activation function is ReLU [12], optimizer is adam.

After 10 epochs training with batch size of 512, MSE value with 0.38 and 8.11 on valid data are achieved with the ordered and YUV data. The results are shown in Figs. 7. There is a huge variance between training and valid loss, see the blue solid and dash line, which indicates that the trained model has been over-fitting. Another reason might be the data split method with ordered. The steering angle showed a normal distribution and time associated in Fig. 3, and the dataset used in this project is quite small so that the training data might have no experience about valid data part, which leads to a high variance.

Remark 2 *The image flip does not work as expected in implementation and was abandoned during model training. The model doesn't learn and is stuck frequently. One reason I could catch is the camera center is not on the right ahead, the image calibration should be done further.*

Remark 3 *The image normalization is considered as the first layer in NVIDIA model to accelerate the compute speed with the advantages of GPU. However, The computer would be stuck once the batch size is less than or equal to 256. The reason might be the heavy compute burden of the image normalization when a smaller batch size used (the smaller batch size, the less time needed for each batch). It is better to find a balance between computer's computational capability and batch size.*

Implementation

The effects of color mode and data shuffle on training performance are investigated in this part, they are trained on the same model structure to have unity variables. The results are shown as Fig. 7, the final loss can be found in Table. 2. It is obvious that the valid losses with models trained by the ordered data are extremely large while their training losses have rapid convergence speeds. With the shuffled training data, both training and valid losses on RGB and YUV can converge to smaller values. However, the models trained with YUV

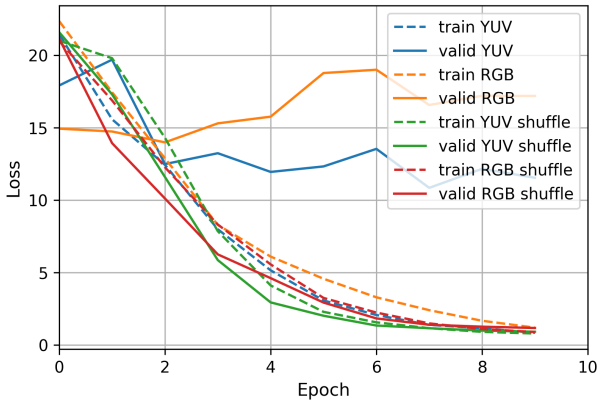


Figure 7: Results with different setting on benchmark model.

images have a slight advantage than that with RGB color mode. Hence, the YUV color mode is used in the following models.

Remark 4 *The performance improvement with shuffled training data is not a determined fact, see test loss in Table.* **??**. *It is noteworthy that test losses are still large in models trained by shuffled data while their train and valid loss are very small. The conditions with or without data shuffle are simultaneously considered in the following simulations.*

During the exploration of model performance, the convolutional filter size is set to 3x3 instead of 5x5 in the benchmark model, maxpooling with filter size of 2x2 is used to replace the down sampling method with 2x2 filter strides. Dropout [13] regularization method is used in the fully connected layer for reducing over fitting. Truncated Normal weights initialization is also applied. Twice epochs (20) are used to cope with slow adaptation induced by dropout. In addition, ELU is used to replace ReLU activation function, because it has smoother derivatives at zero, and hence are expected to be slightly better for predicted continuous values. The detailed architecture is described in Table. 3. This model is trained on a GPU Intensive workloads, which has 61G memory and 12G GPU memory. The code is implemented in python notebook with some local functions, which can be found in [14].

The performances of models with ELU and ReLU can be found in Fig. 8, in which the model structures are same except for the activation function. The model with ELU with a rapid converge time and smoother variance as expressed in Fig. 8(b), the effectively of ELU is confirmed for this project. The training and valid values, see Table. 2, have a deterioration with these revisions. But better test losses are achieved compared to benchmark model.

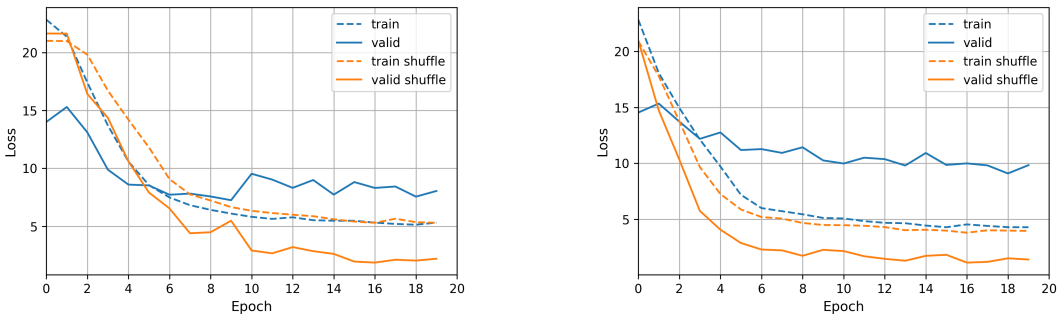
To improve the built model further, the following steps are done:

1. Implement He normal initializer [15]. It draws samples from a truncated normal distribution centered on 0 with stddev $= \sqrt{2/n_{in}}$, where n_{in} is the number of input units in the weight tensor.

2. Add batch normalization [16]. Normalize the activations of the previous layer at each batch, i.e. applies a transformation that maintains the mean activation close to 0 and the activation standard deviation close to 1.

Table 2: Model performances with different model structures and data types

Model name	Type	Train loss	Valid loss	Test loss
Benchmark model	Ordered YUV	0.82	11.53	4.43
	Shuffled YUV	0.78	0.90	5.47
	Ordered RGB	1.17	17.18	8.47
	Shuffled RGB	0.88	1.56	7.74
Revised model	Ordered ReLU	5.32	8.05	5.81
	Shuffled ReLU	5.30	2.20	4.50
	Ordered ELU	4.30	9.84	4.25
	Shuffled ELU	4.45	2.10	3.83
Final model	Ordered	4.64	7.07	3.59
	Shuffled	4.11	1.22	2.59



(a) (b)

Figure 8: (a) Training results based on the built model and ReLU; (b) Training results based on the built model and ELU.

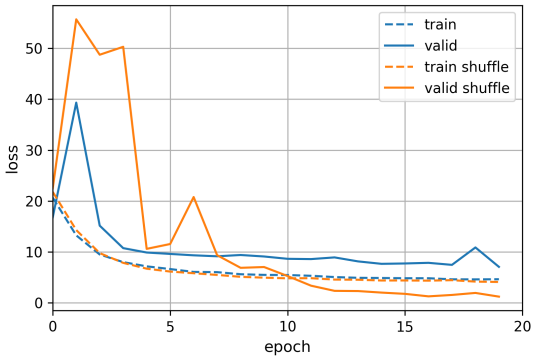


Figure 9: Final results with ordered and shuffled training data.

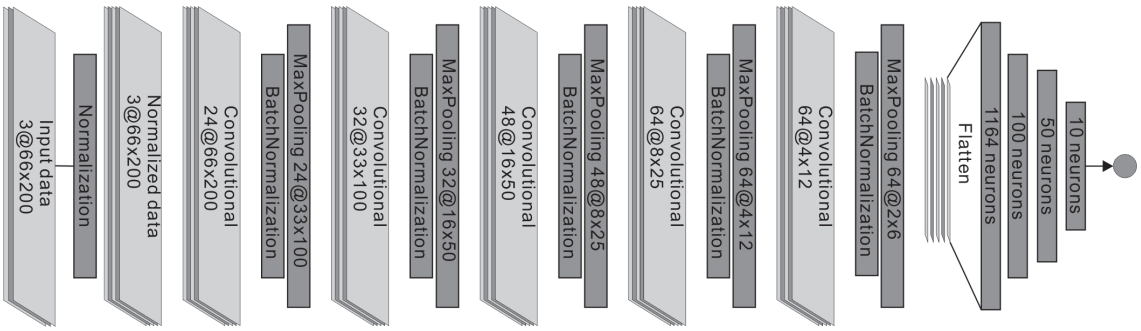


Figure 10: Final model architecture.

The final model structure can be found in Table. 2, and its pipeline is described as Fig 10. With 20 epochs iterating, for ordered data input, a loss value 4.64 on training and 7.07 on valid data are achieved; for shuffled data input, 4.11 and 1.22 are obtained for train and valid loss, respectively.

The interesting thing is the loss value on valid dataset increases dramatically on the 2rd epoch, the reason might be caused by dropout method, which stops too many neurons and the model has not adapt to the unknown variables. However, with the above methods implemented, the loss value decreases remarkable within 5 epochs iterating, see Fig. 9, and the MSE value keeps insensible decrease, but the valid metric is not rise. It illustrates that the model has been almost converged under this model structure and over fitting problem is prevented. The model with shuffled data has a better performance than that with ordered, in addition, a better test loss, i.e. 2.59, is also achieved. Hence, the model trained with shuffled data is finally adopted in this project. The ordered training data would be considered if time-related algorithms are used in future.

4 Results

Model Evaluation and Validation

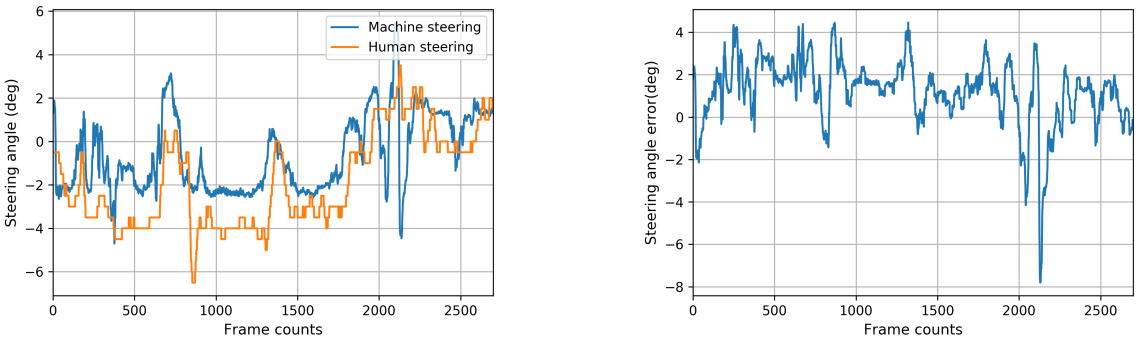
The epoch 10 of the dataset is used for evaluating the revised model. The steering angles calculated by final trained model are compared with the human driving performance, shown as Fig. 11, from which we can see the driving trends is similar, but there is a positive error among most frames. Moreover, it looks that the trained model cannot deal with sharp corners, which can be found the steering angle error is large and does not have corresponding trend as human driver during 2000-2300 frames. The reason is that the training data does not contain enough sharp corners for model learning. One thing can be confirmed is that the trained model do not make obvious mistakes at least. More work should be conducted further to have an acceptable performance.

Justification

Compared with the benchmark model, the improvement of MSE value can be found as the performances shown Table. 2. However, the final solution is not convincing enough to

Table 3: Model architecture

	Benchmark model		Revised model		Final model	
Layer	Output Shape	Params	Output Shape	Params	Output Shape	Params
Normalization	(None, 66, 200, 3)	0	(None, 66, 200, 3)	0	(None, 66, 200, 3)	0
Convolutional	(None, 31, 98, 24)	1824	(None, 64, 198, 24)	672	(None, 66, 200, 24)	672
					BatchNormalization	96
			MaxPooling 2x2			
Convolutional	(None, 14, 47, 36)	21636	(None, 30, 97, 36)	7812	(None, 33, 100, 32)	6944
					BatchNormalization	128
			MaxPooling 2x2			
Convolutional	(None, 5, 22, 48)	43248	(None, 13, 46, 48)	15600	(None, 16, 50, 48)	13872
					BatchNormalization	192
			MaxPooling 2x2			
Convolutional	(None, 3, 20, 64)	27712	(None, 4, 21, 64)	27712	(None, 8, 25, 64)	27712
					BatchNormalization	256
			MaxPooling 2x2			
Convolutional	(None, 1, 18, 64)	36928	(None, 2, 19, 64)	36928	(None, 4, 12, 64)	36928
					BatchNormalization	256
			MaxPooling 2x2			
Flatten	(None, 1152)	0	(None, 2432)	0	(None, 768)	0
Fully-connected	(None, 1164)	1342092	(None, 1164)	2832012	(None, 1164)	895116
			Dropout 0.2			
Fully-connected	(None, 100)	116500	(None, 100)	116500	(None, 100)	116500
			Dropout 0.2			
Fully-connected	(None, 50)	5050	(None, 50)	5050	(None, 50)	5050
			Dropout 0.5			
Fully-connected	(None, 10)	510	(None, 10)	510	(None, 10)	510
			Dropout 0.5			
Output	(None, 1)	11	(None, 1)	11	(None, 1)	11
Total params:		1,104,243		1,595,511		1,104,243



(a) (b)

Figure 11: Test results.(a) machine steering performance compared with human driver; (b) steering angle error in each frame of the test epoch.

implement this model in real world.

5 Conclusion

An autonomous vehicle steering control with End-to-End method is implemented in this project. CNN algorithm is adopted to extract features from raw pixes, and then the steering angle is generated with a fully connected layer. To improve performance of the trained model, several methods are implemented. i.e.ELU, batch normalization, weight initialization, dropout. The final test results have a reasonable performance compared with human driving behavior. The LSTM mehod will be adopted to improve the proposed results in the future.

Improvement

There are still a lot to be improved further for a better steering angle tracking performance. However, they have to be presented as future work due to the time problem.

1. More filters in Convnets, more epochs iterating with higher dropout value. These could extract more features from raw pixes to cope with various driving condition. for instance, line mark missing, light variance, etc.

2. The parameters in optimizer. Such as learning rate, decay rate. Higher converge speed and stable performance can be expected with these fine tuned parameters.

3. Long short-term memory (LSTM) [17]. A recurrent neural network (RNN) architecture might work on autonomous vehicle due to driving behaviors always are smooth expect for unexpected situation, the previous condition can be referable for the next moment.

4. Image preprocessing. We can revise the brightness and contrast of input images to simulate various driving conditions.

5. generative adversarial networks (GAN). This method can be used for training driving simulator and has been obtained a beautiful results [11], in which the MSE in the order of 10^{-2} is achieved.

References

[1] L. Fridman, "6.s094: Deep learning for self-driving cars," http://selfdrivingcars.mit.edu/, 2016.

[2] E. Ohn-Bar and M. M. Trivedi, "Looking at humans in the age of self-driving and highly automated vehicles," *IEEE Transactions on Intelligent Vehicles*, vol. 1, no. 1, pp. 90–104, 2016.

[3] Y. LeCun, U. Muller, J. Ben, E. Cosatto, and B. Flepp, "Off-road obstacle avoidance through end-to-end learning," in *NIPS*, 2005, pp. 739–746.

[4] J. Zhang and K. Cho, "Query-efficient imitation learning for end-to-end autonomous driving," *arXiv preprint arXiv:1605.06450*, 2016.

[5] D. A. Pomerleau, "Alvinn, an autonomous land vehicle in a neural network," Carnegie Mellon University, Computer Science Department, Tech. Rep., 1989.

[6] M. Bajracharya, A. Howard, L. H. Matthies, B. Tang, and M. Turmon, "Autonomous off-road navigation with end-to-end learning for the lagr program," *Journal of Field Robotics*, vol. 26, no. 1, pp. 3–25, 2009.

[7] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang *et al.*, "End to end learning for self-driving cars," *arXiv preprint arXiv:1604.07316*, 2016.

[8] H. Xu, Y. Gao, F. Yu, and T. Darrell, "End-to-end learning of driving models from large-scale video datasets," *arXiv preprint arXiv:1612.01079*, 2016.

[9] L. Fridman, "End-to-end learning from tesla autopilot driving data," https://github.com/lexfridman/deeptesla, 2016.

[10] A. Karpathy, "Cs231n convolutional neural networks for visual recognition," http://cs231n.github.io/convolutional-networks/, 2016.

[11] E. Santana and G. Hotz, "Learning a driving simulator," *arXiv preprint arXiv:1608.01230*, 2016.

[12] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th international conference on machine learning (ICML-10)*, 2010, pp. 807–814.

[13] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting." *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[14] G. Wang, "Capstone-deeptesla," https://github.com/gwwang16/Machine-Learning/tree/master/Projects/Capstone-DeepTesla, 2017.

[15] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1026–1034.

[16] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[17] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.