REPORT: PROJECT 3. BEHAVIORAL CLONING

Author: Markus Buchholz

1. Introduction

This report discusses the design of convolutional neural network(CNN) in Keras, which predicts steering angle of simulated car from images. In order to optimize and speed up the design, recommended Paul Heraty (https://slack-files.com/T2HQV035L-F50B85JSX-7d8737aeeb) suggestions were used.
Following the presented recommendations and having middle class simulation machine: CPU i7-3.4Ghz, 8GB RAM, GForce 1050i 4GB (and lack of joystick) forced to perform tuning of the final predictor model on provided by Udacity pictures (at the beginning the pictures were taken while controlling the car by keyboard arrows but without sufficient good results). Additionally, for the predictor design there were applied some (with author modifications) Udacity best practices e.g. choice of network architecture, picture processing and training methodology)

In this project, it was experienced that in order to elaborate best solution it would be necessary to verify many possible neural network setups, enlarge data set by additional augmented data set and adjust the hyperparameters. Due to lack of sufficiently fast simulation equipment setup, only some test scenarios of this investigation could be verified (the simulation with standard data set and 5 epoch took approximately 75minutes – for simplest CNN architecture).

Influence of PID controller was also investigated. Trained model, which fulfilled the project requirements (final version) was investigated by checking the Integral part of PID controller (in drive.py file). When the simulated car is driving (the steering angle is predicted by CNN) some overshoots of steering angle can be seen. It was thought that compensation of this overshoot could be done by changes in integral part of implemented PID controller. Unfortunately changes of regarding coefficient by factor 2, 4 or 10 increased only the time when the car brakes were used. Therefore, the idea of changes of PID was sacrificed. The author focused only on the controller AS IS.

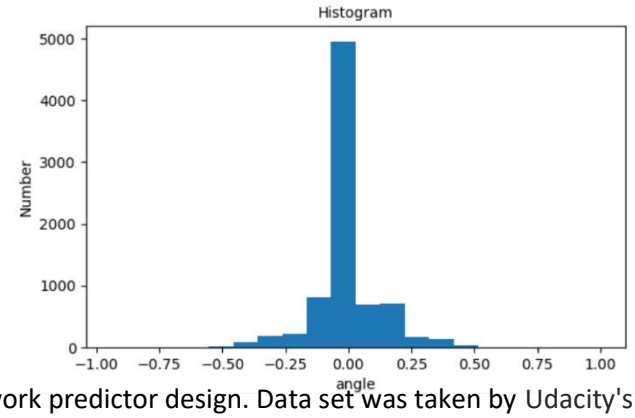
Project consists of following files:

Name	Description
model.py	The script used to create and train the model
drive.py	The script to drive the car.
model.h5	The saved model of designed predictor
Project3_SDCar_markus_buchholz.pdf	Report which presents results, explains the structure of the network and training approach.
video.mp4	A video recording of the vehicle driving autonomously.

2. Exploration of data

Exploration of data set was done by use of Panda, Numpy and Matplotlib. Research data set can be represented in statistical form as follows:

Name	Value
Number of training examples (pictures) 320 x 160 pixel image with 3 color channels	8036 * 3
Minimum steering angle	-0.9426954
Maximum steering angle	1.0
Mean steering angle	0.0040696
Median steering angle	0
Standard deviation steering angle	0.128832



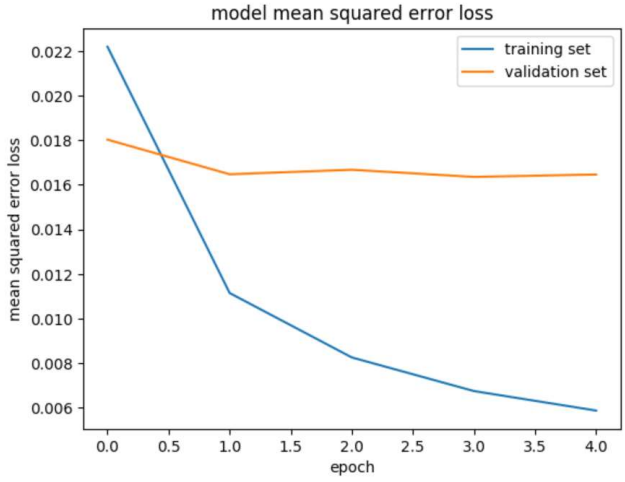
Data set provided by Udacity was taken to perform the neural network predictor design. Data set was taken by Udacity's car-driving simulator. Sampling time is 10Hz. Each time following data set features are taken:

- Image taken from the center camera on the car
- Image taken from the left camera on the car
- Image taken from the right camera on the car
- Value of steering angle (minimum value: -0.9426954, maximal value 1)
- Value of the throttle
- Value of the speed
- Value of the brake

Example images taken at the same timestamp:

3. Final model and Architecture approach

Following the Udacity recommendations NVIDIA architecture was taken first into consideration ("End to End Learning for Self-Driving Cars" paper: https://arxiv.org/pdf/1604.07316.pdf). Unfortunately, in this case lack of positive test results (simulated car could not keep the track well enough), forced that this architecture was abandoned. Architecture proposed by Vivek Yadav https://chatbotslife.com/using-augmentation-to-mimic-human-driving-496b569760a9 was also investigated. In this case also simulated car did not keep the right road track correctly (crash on the bridge). The next architecture which was also explored was the network presented during the lecture. Here, preliminary positive test result (the car could keep the track during the whole lap) encouraged to focus more attention on this approach. It order to improve the quality of the angle prediction, it was decided to enlarge the network.
Deployed architecture was mainly base on “commaai” research, presented in: https://github.com/commaai/research/blob/master/train_steering_model.py. For this architecture mean squared error (MSE) for training and validation set was depicted on below figure. Beside the MSE drops for training data set, the error rate for validation set slightly increases from 4 (5 epoch test). It was concluded that the model is overfitted. Therefore, bearing in mind Udacity recommendation for overfitting avoidance the pooling layer was added.



Applied in this project architecture can be presented as follows:

Layer	Cropping	Convolutional_1	Convolutional_2	Fully_Connected_1	Pooling	Flatten	Fully_Connected_1	Fully_Connected_1
Input	160x320x3	65x320x3	17x80x16	9x40x32	5x20x64	2x10x64	6400	512
Filters	-	16	32	64		-	-	-
Output	65x320x3	28x28x32	9x40x32	5x20x64	2x10x64	1280	512	50
Stride	-	4x4	2x2	2x2	2x2	-	-	-
Padding	-	Same	Same	Same		-	-	-
Activation	-	ELU	ELU	ELU		-	-	-
Dropout	-	-	-	-		0.2	0.5	-

```
Layer (type)                 Output Shape              Param #
=================================================================
lambda_1 (Lambda)            (None, 160, 320, 3)       0
_________________________________________________________________
cropping2d_1 (Cropping2D)    (None, 65, 320, 3)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 17, 80, 16)        3088
_________________________________________________________________
elu_1 (ELU)                  (None, 17, 80, 16)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 9, 40, 32)         12832
_________________________________________________________________
elu_2 (ELU)                  (None, 9, 40, 32)         0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 5, 20, 64)         51264
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 2, 10, 64)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 1280)              0
_________________________________________________________________
dropout_1 (Dropout)          (None, 1280)              0
_________________________________________________________________
elu_3 (ELU)                  (None, 1280)              0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               655872
_________________________________________________________________
dropout_2 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 50)                25650
_________________________________________________________________
elu_4 (ELU)                  (None, 50)                0
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 51
=================================================================
Total params: 748,757
Trainable params: 748,757
Non-trainable params: 0
_________________________________________________________________
Training model...
```

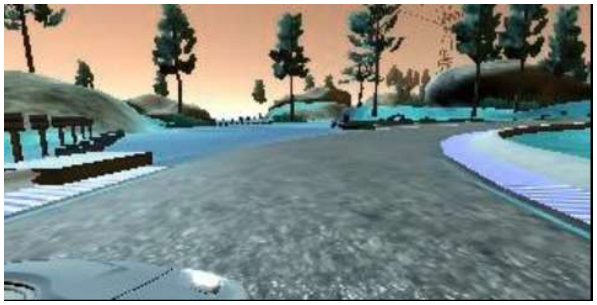
For the activation function it was chosen ELU which introduce non-linearity into the model. In order to prevent network from overfitting, two times dropout was used (https://se.mathworks.com/help/nnet/ug/improve-neural-network-generalization-and-avoid-overfitting.html).
As demonstrated during the lectures the generator was incorporated into the design. It can be assumed that asking on the fly the generator to give the sample (preprocessed image) improved considerably memory management (https://www.youtube.com/watch?v=bD05uGo_sVI).
Following the Udacity best practice and secure image normalization the the Lambda layer was used.

4. Image Processing Pipeline

Base on Udacity best practice recommendations and Vivek Yadavs channel some augmentation image techniques were used. It was decided to not increase the number of samples (data set) so it had an impact on final design of proposed pipeline.
Simulator takes at the same timestamp picture using beside the center also left and right camera. For simulating effect of the car wandering off to the side, and recovering, it was added 0.25º angle to the left camera and subtracted a 0.25º angle from the right camera. Pictures taken for final training are taken randomly (center, left or right).
The simulated car goes around the track against clockwise so more turns are to the left. In order to help with the left turn bias, it was decided (as recommended) to involve image flipping and taking the opposite sign of the steering measurement. Flipping process is performed also randomly.
Finally it was decided to perform randomly shift of the image (horizontally and vertically). In this case it was possible to simulate the effect of car being at different positions on the road, and add an offset corresponding to the shift to the steering angle (VY).

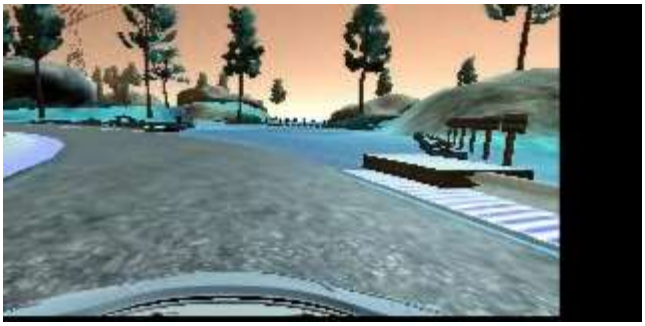
Original

Flipped



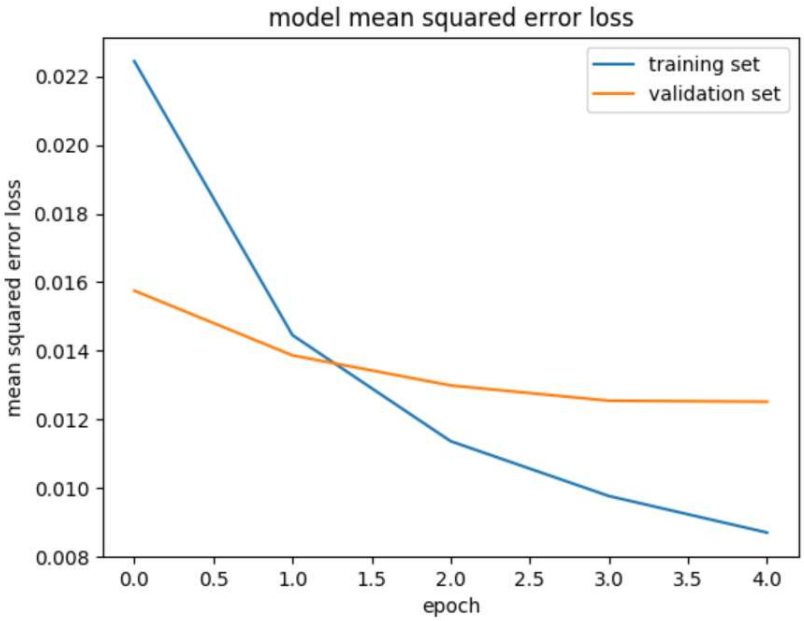
Original. Steering angle 0º

Shifted. Steering angle: -0.329587782147 º



5. Training and testing

Proposed and describe above CNN was taken to training. It was decided to apply batch size equals 32 and following the Paul Hearty's recommendation applied 5 epochs which were enough for this training. Learning rate for ADAM optimizer was adjusted to 0.001. Below figure depicts mean squared error MSE for training and validation set. The error is small and decrease while number of epoch increase.




```
Epoch 1/5
1022s - loss: 0.0228 - acc: 0.0905 - val_loss: 0.0184 - val_acc: 0.0913
Epoch 2/5
919s - loss: 0.0146 - acc: 0.0897 - val_loss: 0.0158 - val_acc: 0.0914
Epoch 3/5
916s - loss: 0.0112 - acc: 0.0898 - val_loss: 0.0147 - val_acc: 0.0917
Epoch 4/5
921s - loss: 0.0096 - acc: 0.0905 - val_loss: 0.0144 - val_acc: 0.0946
Epoch 5/5
923s - loss: 0.0085 - acc: 0.0899 - val_loss: 0.0143 - val_acc: 0.0896
dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
##################
##################
CCN model saved to file.
```

Low variance (difference between training MSE and validation MSE) indicated that the model in not overfitted and could generalize enough well to predict angle while driving a simulated car. Test track confirmed these assumptions. Car keeps the track sufficiently well (car stays in the center of the road).
Included video includes two full car laps around the track. Project meets the rubic main requirements (no tire may leave the drivable portion of the track surface).

6. References:

Deep learning:

http://deeplearning.net/tutorial/

https://se.mathworks.com/campaigns/products/artifacts/deep-learning.html

http://www.deeplearningbook.org/

http://cs231n.stanford.edu/2016/syllabus

http://cs231n.stanford.edu/syllabus.html

https://www.youtube.com/watch?v=ILsA4nyG7I0&t=4s

https://www.youtube.com/playlist?list=PL6Xpj9I5qXYEcOhn7TqghAJ6NAPrNmUBH (neural network)

https://www.youtube.com/watch?v=s716QXfApa0

Keras and Python:

https://github.com/fchollet/keras-resources

https://www.youtube.com/watch?v=L0IVu_sKOOY&t=1366s

http://cs231n.github.io/python-numpy-tutorial/

http://www.scipy-lectures.org/advanced/image_processing/

http://www.lfd.uci.edu/~gohlke/pythonlibs/