End-to-end Driving Controls Prediction from Images

Jan Felix Heyse, Maxime Bouton {heyse, boutonm}@stanford.edu

CS229 poster session

Motivation

In 2015, the DOT reported that more than 35,000 people died on US roadways due to crashes. *94% of these crashes happened because of human error* [1].
Using an end-to-end approach could lead to a *robust autonomous driving capability* while relaxing any modeling assumptions. The work described in [2] is a promising step in this direction using Convolutional Neural Networks (CNNs).

Setup

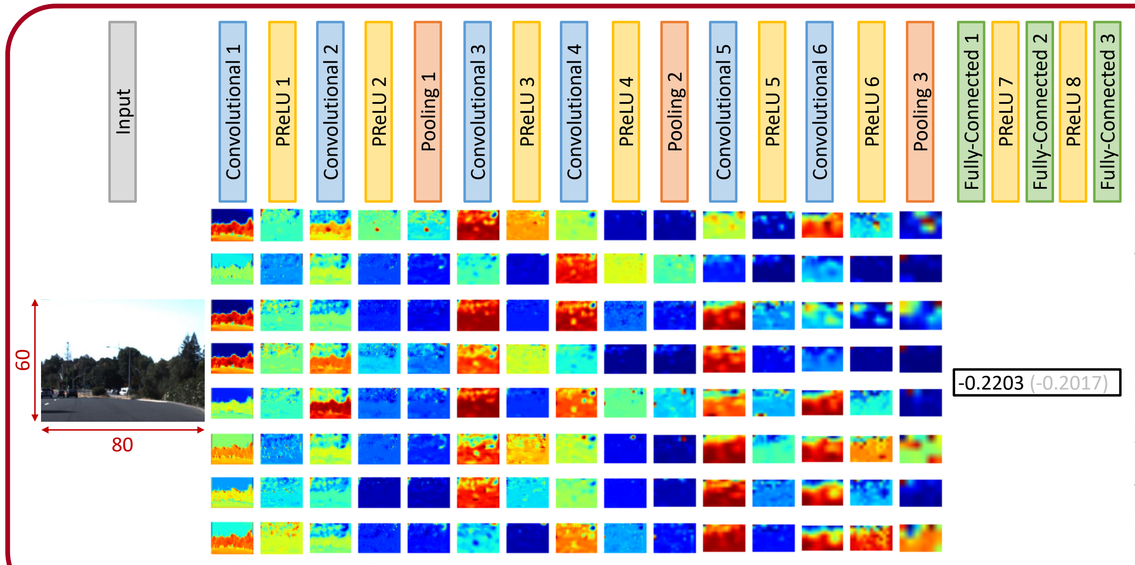
Dataset: We used center camera images at a frame rate of 20Hz and their associated steering angles for a 10 minutes of driving data after pre-processing the dataset.[3]
Pre-processing: For the pre-processing we removed situations where the driver intentionally made a turn at an intersection and where the car was standing at a red light.

Approaches: We used three different approaches with increasing model complexity. To simplify the problem, the first two approaches were classification problems where we used the three different steering directions *straight*, *left*, and *right*, while in the third approach we tried to predict the exact steering angle.

1. SVM algorithm: We divided the raw images into sub-images and reported for each of them the minimum, maximum, and mean gray scale value the features. The trained model was predicting straight all the time.

2. CNN classification: Convolutional Neural Networks (CNNs) allow an automated and powerful use of features and showed a significant improvement to SVM.

3. CNN regression: Again a CNN structure was used, this time to directly predict the steering angles from the raw images.



Visualization of the last regression task convolutional network architecture

Models and architectures

A common architecture [3] for convolutional neural networks is:

- **INPUT -> [[CONV -> RELU]*N -> POOL]*M -> [FC -> RELU]*K -> FC**

Here, *CONV* stands for Convolutional layer, *RELU* for Rectified Linear Unit layer, *POOL* for Pooling layer, and *FC* for Fully Connected layer, which are all standard layer types in CNN applications. The parameters *N*, *M*, and *K* are to be chosen specifically for the problem. We started off by orientating at a configuration used for the comparable cifar10 [5] case.

The final architecture for the CNN classification problem is the following:

- **INPUT -> [[CONV -> RELU]*2 -> POOL]*2 -> [FC -> RELU]*2 -> FC**

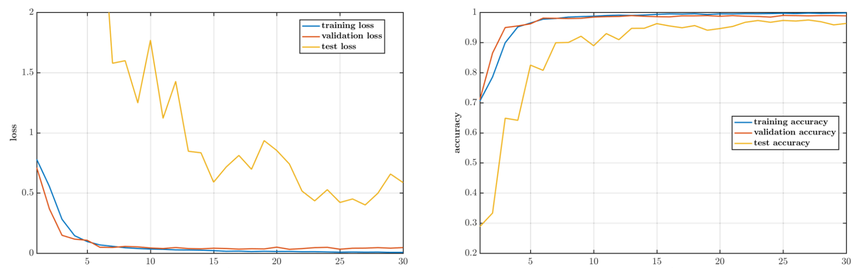
For the CNN regression, we have been working with four different architectures:

1. **INPUT -> [[CONV -> RELU]*2 -> POOL]*2 -> [FC -> RELU]*2 -> FC (CONF filter size 3)**
2. **INPUT -> [[CONV -> P-RELU]*2 -> POOL]*2 -> [FC -> P-RELU]*2 -> FC (CONF filter size 3)**
3. **INPUT -> [[CONV -> P-RELU]*2 -> POOL]*2 -> [FC -> P-RELU]*2 -> FC (CONF filter size 7)**
4. **INPUT -> [[CONV -> P-RELU]*2 -> POOL]*3 -> [FC -> P-RELU]*2 -> FC (CONF filter size 3)**

The ReLU activation layers have a zero gradient zone that can lead to neurons dying off during the optimization process. That is why for the CNN regression we switched to PReLU layers. These address that issue by adding a very small slope at the otherwise zero-slope zones. The value of the slope is learned during the optimization.

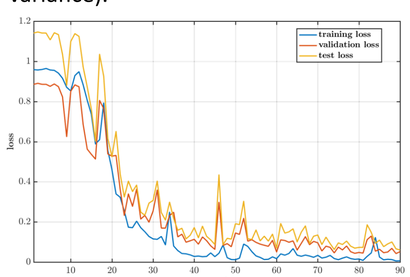
Results

The architecture used for the CNN classification showed generally good results. After only 30 epochs we obtained 97% accuracy on the test set.



Learning history of the deep CNN architecture for a 3 classes classification

The regression model with *ReLU activation layers* converged fast towards a insufficient solution. The change to *PReLU activation layers* improved the results significantly. Of the three *PreLU* architectures, the one with *six convolutional layers* (architecture 4) yielded the best results: We achieved less than 0.1 MSE on the test set (zero-mean, unit variance).

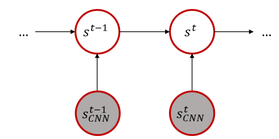


*Regression learning history for [[Conv->P-Relu]*2->Pool]]*3 (right)*

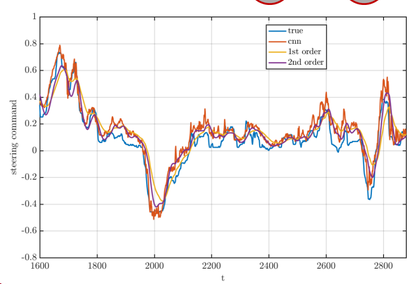
Model	Training Loss (MSE)	Validation Loss (MSE)	Test Loss (MSE)	Number of epochs
1	0.9572	0.8855	1.1410	6
2	0.0416	0.1242	0.1239	60
3	0.0304	0.1409	0.1542	60
4	0.0066	0.0519	0.0601	90

Exploiting the temporal structure

From the input image, the CNN gives a noisy measurement of the steering. The underlying structure of the system can be modeled as a Hidden Markov Model. We compared two different models, a first order one taking only the previous command and a second order model that performed better.



$$s^t = \underbrace{w_1 s^t_{CNN}}_{\text{CNN Prediction}} + \underbrace{w_2 s^{t-1}}_{\text{Command at } t-1} + \underbrace{w_3 s^{t-2}}_{\text{Command at } t-2 \text{ ("velocity" information)}}$$



Exploiting the temporal structure smoothed our predictions. The new predictions are dominated by the preceding time step and therefore lag behind the actual steering command, while the pure CNN predictions are noisier but follow changes in the true steering commands more promptly.

RMS of CNN predictions:	0.203156
RMS of first order smoothing:	0.488451
RMS of second order smoothing:	0.284186

Future work

There are several possible follow-ups for this project. First, we could test the current setup under *different driving conditions* (street types, daylight conditions, …). Second, we could use a simulator (e.g. realistic video game) to attempt *real-time prediction and control* using the pixels on the screen as input. Finally, using *more data or data augmentation techniques* could help building a more robust system.

References

[1] Daniel V McGehee, Mark Brewer, Chris Schwarz, and Bryant Walker Smith. Review of automated vehicle technology: Policy and implementation implications, 2016
[2] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. End to end learning for self-driving cars. 2016
[3] https://github.com/udacity/self-driving-car/
[4] Fei-Fei Li, Andrej Karpathy, and J Johnson. CS231n: Convolutional Neural Networks for Visual Recognition http://cs231n.github.io/convolutional-networks/, 2015
[5] F.Chollet. Train a simple deep cnn on the cifar10 small images dataset.https://github.com/fchollet/keras/blob/master/examples/cifar10_cnn.py, 2015