Behavioral Cloning

Udacity Self-Driving Car Nanodegree Project 3

Student: Florian Wolf

Behavrioal Cloning Project

The goals of this project were the following:

- Use the simulator to collect data of good driving behavior
- Build, a convolution neural network in Keras that predicts steering angles from images
- Train and validate the model with a training and validation set
- Test that the model successfully drives around track one without leaving the road
- Summarize the results with a written report

Rubric Points

Here I will describe how I addressed each point in my implementation.

Submitted files for the project

1. Files included

- model.py containing the script to create and train the model
- drive.py for driving the car in autonomous mode
- model.h5 containing a trained convolution neural network
- writeup_report.md summarizing the results

2. Functionality of each file

Using the Udacity provided simulator and my drive.py file, the car can be driven autonomously around the track by executing:

```
sh python drive.py model.h5
```

The file model.py creates and trains the used neuronal network with given training images. The code is commented with explanations and instructions.

Model Architecture and Training Strategy

1. Solution approach and final model architecture

To create a functional model that processes images and return a single float value the following approach was made. The architecture of my model has been inspired by the NVIDIA model describes in the scientific paper "End to End Learning for Self-Driving Cars". The structure of the model is the following:

- Cropping2D Layer
- Lambda Layer
- Convolution2D Layer: 5x5 filter
- 'Relu' Activation Layer
- Dropout of 0.2
- Convolution2D Layer: 5x5 filter
- 'Relu' Activation Layer

- MaxPooling2D Layer
- Convolution2D Layer: 3x3 filter
- 'Relu' Activation Layer
- MaxPooling2D Layer
- Convolution2D Layer: 3x3 filter
- 'Relu' Activation Layer
- MaxPooling2D Layer
- Flatten Layer
- Dropout Layer
- Dense Layer
- Activation Layer
- Dropout Layer
- Dense Layer
- Activation Layer
- Dense Layer
- Activation Layer
- Dense Layer

The first Layer crops 40 pixels off the top and 20 pixels off the bottom of the input image. The input image size is 160x320 with 3 colour channels. Next step is the normalization of the image array. Therefore a Lambda Layer has been used to set the mean to 0 and the standard derivation to 0.5. Then the data is passed on to a convolution layer. The model consist of 4 different convolution layers, all use the border mode "valid". First and second conv. layer use a filter of 5x5 and the 4th and 5th use a filter of 3x3. Each conv. layer is followed by an activation layer using the Relu function. After the first activation layer a dropout of 20 percent happens. Then a maximum pooling layer with a 2x2 filter is applied. The array with shape 10, 38, 3 is flattened and passed on to 4 fully connected layer with a dropout of 0.5 and a "Relu" activation. These fully connected layers have the outputs 1024, 512, 128, 32 and 1.

IMAGE OF MODEL LAYERS SUMMARY

2. Attempts to reduce overfitting in the model

In order to avoid overfitting the model uses 3 dropout layer. and a maximum pooling layer after each conv. layer. The validation set is a 20 % split from the training set. The model has been trained with several training sets, wich are explained later on. To test the model the autonomous mode of the Udacity simulator was used.

3. Model parameter tuning

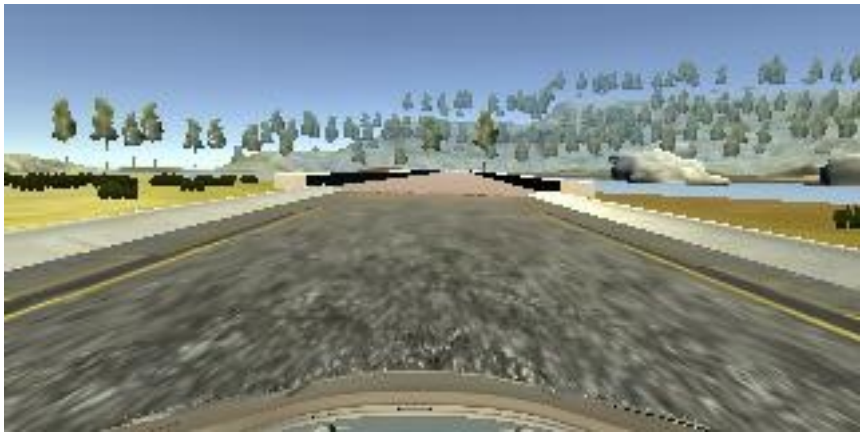
As optimizer the adam optimizer has been chosen so the learning rate has not been tuned manually. The model has 1,763,841 parameters of which none could not be trained.

4. Appropriate training data

To train the model only center lane driving images were used. Therefore the Udacity training set was downloaded and only the center images extracted. To teach the model to recover from the edge of the road a new training set was used. In addition the model was trained with both training sets but with flipped images. Therefore the images were switched within the generator using the Numpy library and the steering angle was inverted.

3. Creation of the Training Set & Training Process

The first step was to train the model with the existing Udacity training set. The Udacity training set has 3 times 8,036 images. How already mentioned only the center images were used. Here are two examples of this image set:



The model was trained with the Udacity training set for 22 epochs. In the end the loss was 0.008 and the validation loss was 0.0091

```
316s - loss: 0.0083 - mean_squared_error: 0.0083 - val_loss: 0.0094 - val_mean_squared_error: 0.0094
Epoch 17/22
317s - loss: 0.0084 - mean_squared_error: 0.0084 - val_loss: 0.0095 - val_mean_squared_error: 0.0095
Epoch 18/22
315s - loss: 0.0084 - mean_squared_error: 0.0084 - val_loss: 0.0092 - val_mean_squared_error: 0.0092
Epoch 19/22
314s - loss: 0.0083 - mean_squared_error: 0.0083 - val_loss: 0.0091 - val_mean_squared_error: 0.0091
Epoch 20/22
315s - loss: 0.0081 - mean_squared_error: 0.0081 - val_loss: 0.0091 - val_mean_squared_error: 0.0091
Epoch 21/22
318s - loss: 0.0080 - mean_squared_error: 0.0080 - val_loss: 0.0092 - val_mean_squared_error: 0.0092
Epoch 22/22
314s - loss: 0.0080 - mean_squared_error: 0.0080 - val_loss: 0.0091 - val_mean_squared_error: 0.0091
```

To help the model recover to the center of the road once it got to the side of the track a separate training set was created. This recovery training set contains 4,584 images. In addition the images were flipped and the angles inverted so that the total amount of images to train was doubled. The model has been trained with this training set, fine-tuning the already tuned weights from the Udacity training set. To avoid changing the models behaviour too much, the model was trained the recovery set for only 3 epochs. For the recovery a loss of 0.087 and a validation loss of 0.063 has been reached. Here are two examples for a recovery from the left and from the right:

4. Implementing a generator

Generators should be used when dealing with a big amount of data. Instead of loading all the data into memory at once, the Generator only loads pieces of data when they are needed. This hepls avoiding memory errors and is more efficient. For this project a generator was written, which loads only the needed bach of images for the model to train. The generator also flips and shuffles the images. Once the image is loaded, it is flipped and the steering angle is inverted. before passing the data on to train the model, the data is shuffled. For the validation set also the same generator has been used.

The input of the generator is a list with the path of the images and the batch size. The list of image's path is created when recording training data with the Udacity simulator. Before loading the list into the generator, it is randomly shuffled. The batch size of the generator is 32 images.

5. Speed of the car

By default the speed of the car in the Udacity simulator is set to 20 mph. This is set in the drive.py file. The speed was set to 11 mph. The simulator regualtes this command to a speed 13 mph as shown in the folloing image.

Reducing the speed gives the model more time to react and recover from mistakes. This value was found out impericaly. Slower speed does not improve the models performance. Faster speed results in the car getting off track.

Summary

Using the the describes architecture, trained with both the Udacity data set and the recovery data set as well as adjusting the speed led to an working model. The car was able to perform two laps autonomously before getting stuck at the bridge, which seems to be a difficulty for the model. Using a generator and flipping the images to additionally train the model was a key feature for the success of this project. To achieve a better model the more training would be required. For example training it with images from another track would adapt the model to new sceneries. Another feature to improve the model would be to use the side cameras of the car with an offset added or subtracted to the steering angle.