Behavioral Cloning

The Project Goals/Steps

- 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

Files Submitted & Code Quality

1. Submission includes all required files and can be used to run the simulator in autonomous mode

My project includes the following files:

- ***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.pdf*** summarizing the results
- ***final_video.mp4*** – result track video

2. Submission includes functional code

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

```
python drive.py model.h5 out_images
```

3. Submission code is usable and readable

The model.py file contains the code for training and saving the neural network. The file contains comments to explain how the code works.

The main methods inside scripts:

get_model_architecture – build the network used to train

read_the_data – image samples information loading from the driving_log.csv file

generator – generator method to load batch samples images

Model Architecture and Training Strategy

1. An appropriate model architecture has been employed

My model uses transfer learning concept and use VGG pretrained model inside. To tune the model for my data I've:

- added two convolution layers before VGG
- used pretrained VGG model
- added two fully connected layers for the correct model output

2. Attempts to reduce overfitting in the model

The model was trained and validated on different data sets to ensure that the model was not overfitting. I haven't seen real overfitting so didn't add pooling level explicitly, but I believe pooling implicitly used inside VGG model. Moreover, when I've tried to add additional pooling level explicitly, the result was little bit worse, so I've removed it.

The model was tested by running it through the simulator and ensuring that the vehicle could stay on the track.

3. Model parameter tuning

The model used an ***adam optimizer***, so the learning rate was not tuned manually. I use **mean squared error** to train the model. I've also investigated model accuracy, but in case current model isn't classification task, model accuracy doesn't provide real picture and not really identify quality of the model

4. Appropriate training data

Training data was chosen to keep the vehicle driving on the road, return to the center from the left and right side and surfaces and flipped data to have the enough right turn data.

Model Architecture and Training Documentation

1. Solution Design Approach

For my model I wanted to use Transfer Learning concept and in case the model contains image handling, I've tried to use different imagenet pretrained models. Firstly, I've tried to use Inception model inside (Goglenet network), but maybe because of specific image sizes input (additional unneeded scaling) current model usage haven't received acceptable results for me.

After several other attempts I'd chosen VGG model which shows nice results for my opinion.

To adapt input, I've added two convolutional layers before the pretrained model. I've used already pretrained parameters for VGG and didn't train it additionally. To receive the needed output, I've added two fully connected layers.

In order to understand how well the model was working, I split my image and steering angle data into a training and validation set (80% training set and 20% validation set)

I've used 5 epochs to train the data. But I've not seen much progress even after 3 epochs, so decided to not add the additional epochs to train:

```
Epoch 1/5
214/214 [==============================] - 108s - loss: 0.0650  val_loss: 0.0242
Epoch 2/5
214/214 [==============================] - 104s - loss: 0.0232  val_loss: 0.0224
Epoch 3/5
214/214 [==============================] - 104s - loss: 0.0217  val_loss: 0.0211
Epoch 4/5
214/214 [==============================] - 105s - loss: 0.0208  val_loss: 0.0329
Epoch 5/5
214/214 [==============================] - 105s - loss: 0.0214 val_loss: 0.0319
```

The final step was to run the simulator to see how well the car was driving around track one. It was the trickiest part of the project for me. As overall behavior was good but there were a few spots where the vehicle fell off the track. To avoid current issues, I've added additional training data for these places to train model to return to the track

At the end of the process, the vehicle can drive autonomously around the track without leaving the road. The result can be explored on result video (final_video.mp4)

2. Final Model Architecture

The final model architecture consists of such layers:

Layer (type)	Output Shape	Param #	Description
lambda_1 (Lambda)	(None, 160, 320, 3)	0	Image normalization
cropping2d_1 (Cropping2D)	(None, 80, 320, 3)	0	Crop unneeded data on the image
conv2d_1 (Conv2D)	(None, 80, 160, 3)	84	Convolution layer with (1,2) strides
conv2d_2 (Conv2D)	(None, 80, 80, 3)	84	Convolution layer with (1,2) strides
vgg16 (Model)	(None, 2, 2, 512)	14714688	Pretrained VGG model
dense_1 (Dense)	(None, 2, 2, 256)	131328	Fully connected layer
flatten_1 (Flatten)	(None, 1024)	0	Flat the output
dense_2 (Dense)	(None, 1)	1025	Provide needed output
Total params: 14,847,209 Trainable params: 132,521 Non-trainable params: 14,714,688			

Creation of the Training Set & Training Process

Training data was chosen to keep the vehicle driving on the road. Firstly, I've tried to gather all the data myself, but from my childhood I don't like racing games and my data wasn't liked as really good data to train 😊

Because of this I've gather data from two sources:

- used the sample data images for base training
- added own recorded data for edge cases where base model moved from the track.

The additional data was gathered near:

- shadows

- data recorded from road borders to return to road center

- data gathered near places where lane borders were disappeared

- data near bridge to correctly leave the blocks

I've used the images only from center cameras as didn't see real performance from left and right cameras additional images usage (maybe because of incorrect correction angle). But I've added the flipped images to the samples as most of the track contains only left turns and right turns are desired to have balanced data to train

I've used generator concept for image loading for the shuffled data set. I've put 20% of the data into a validation set.

To preprocess the images such layers were added to the model architecture:

- lambda layer to normalize the images
- cropping layer to remove unneeded data from the image (like sky, trees, etc)