Behavioral Cloning Project

The goals / steps of this project are 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 consider the rubric points individually and describe how I addressed each point in my implementation.

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 (This is provided by Udacity, my only modification was to increase the car speed by setting variable set_speed = 20)
- model.h5 containing a trained convolution neural network
- writeup_report.pdf summarizing the results
- run1.mp4 video recording of the autonomous driving.

2. Submission includes functional code

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

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

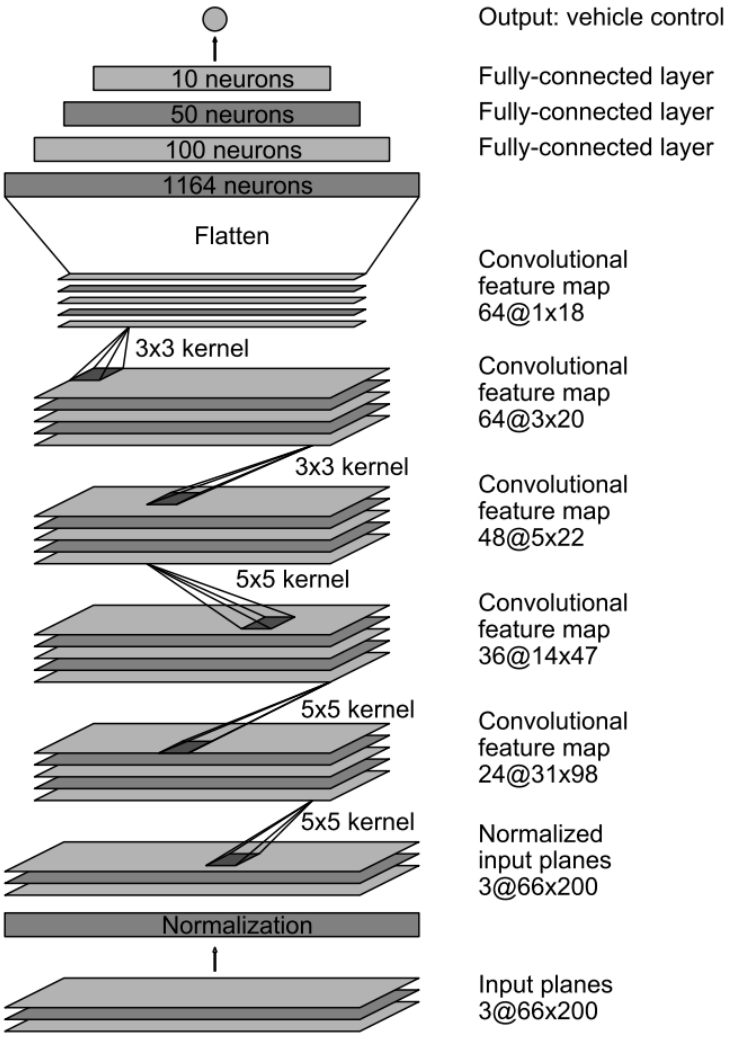
3. Submission code is usable and readable

The model.py file contains the code for training and saving the convolution neural network. The file shows the pipeline I used for training and validating the model, and it contains comments to explain how the code works.

Model Architecture and Training Strategy

1. An appropriate model architecture has been employed

The [nVidia Autonomous Car Group] model was choosen, and the car drove the complete first track after just 7 training epochs.



The model summary is as follows:

```
Layer (type)                     Output Shape          Param #     Connected to
====================================================================================================
lambda_1 (Lambda)                (None, 160, 320, 3)   0
lambda_input_2[0][0]
____________________________________________________________________________________________________
cropping2d_1 (Cropping2D)        (None, 90, 320, 3)    0           lambda_1[0][0]
____________________________________________________________________________________________________
convolution2d_1 (Convolution2D)  (None, 43, 158, 24)   1824        cropping2d_1[0][0]
____________________________________________________________________________________________________
convolution2d_2 (Convolution2D)  (None, 20, 77, 36)    21636
convolution2d_1[0][0]
____________________________________________________________________________________________________
convolution2d_3 (Convolution2D)  (None, 8, 37, 48)     43248
convolution2d_2[0][0]
____________________________________________________________________________________________________
convolution2d_4 (Convolution2D)  (None, 6, 35, 64)     27712
convolution2d_3[0][0]
____________________________________________________________________________________________________
convolution2d_5 (Convolution2D)  (None, 4, 33, 64)     36928
convolution2d_4[0][0]
____________________________________________________________________________________________________
flatten_1 (Flatten)              (None, 8448)          0
convolution2d_5[0][0]
____________________________________________________________________________________________________
dense_1 (Dense)                  (None, 100)           844900      flatten_1[0][0]
____________________________________________________________________________________________________
dense_2 (Dense)                  (None, 50)            5050        dense_1[0][0]
____________________________________________________________________________________________________
dense_3 (Dense)                  (None, 10)            510         dense_2[0][0]
____________________________________________________________________________________________________
dense_4 (Dense)                  (None, 1)             11          dense_3[0][0]
====================================================================================================
Total Images: 24108
Train samples: 19286
Validation samples: 4822
===============
```

2. Attempts to reduce overfitting in the model

I decided not to modify the model by applying regularization techniques like Dropout or Max pooling. Instead, I decided to . use Keras **ModelCheckpoint & EarlyStopping** callback function (https://keras.io/callbacks/)

ModelCheckpoint

Model will save the weights whenever validation loss improves

- **period**: Interval (number of epochs) between checkpoints. (`default period is 1 so not passing that param)`

checkpoint = ModelCheckpoint(filepath = 'model.h5', verbose = 1, save_best_only=True, monitor='val_loss')

EarlyStopping

Discontinue training when validation loss fails to decrease.

- **patience**: number of epochs with no improvement after which training will be stopped.(set to 2)

callback = EarlyStopping(monitor='val_loss', patience=2, verbose=1)

In addition to that, I split my sample data into training and validation data. Using 80% as training and 20% as validation.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.

Adam optimizer.

Default parameters follow those provided in the original paper.

Arguments

lr: float >= 0. Learning rate.

References

[Adam - A Method for Stochastic Optimization] (https://arxiv.org/abs/1412.6980v8)

Code: adam optimizer

```
# Compile model with adam optimizer and learning rate of .0001

adam = Adam(lr=0.0001)

model.compile(loss='mse',optimizer=adam)
```

4. Appropriate training data

Training data was chosen to keep the vehicle driving on the road. I used a combination of center lane driving, recovering from the left and right sides of the road ...

For details about how I created the training data, see the next section.

5. Solution Design Approach

I have used the below preprocessing layers (refer function : createPreProcessingLayers)

- Lambda layer to normalized input images to avoid saturation and make gradients work better.
- Cropping2D to crop top portion of the image which not useful for the training (this reduces memory foot print & helps in avoiding unwanted training data)

The nVidia Autonomous Car Group model was chosen. The only modification was to add a new layer at the end to have a single output as it was required. This time the car did its first complete track, but there was a place in the track where it passes over the "dashed" line. More data was needed. Augmented the data by adding the same image flipped with a negative angle(code:94-95). In addition to that, the left and right camera images where introduced with a correction factor on the angle to help the car go back to the lane(function: mergeImageData Line:58 -71). After this process, the car continues to have the same problem with the same "dashed" line. I needed more data, but it was a good beginning.

6. Final Model Architecture

The final model architecture is shown in the following image:

7. Creation of the Training Set & Training Process

- I collected test data using the below approach
    - One track driving forward.
    - One track driving backward.

Data Stored in more_data folder. Due to lack of variations in the data set the trained model was not able to handle sharp turns. Then I used the dataset provided by Udacity. About 8000 images.

After this training, the car was driving down the road all the time on the first track. Please refer run1.mp4 for the output.