Behavioral Cloning

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

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.md summarizing the results

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
```

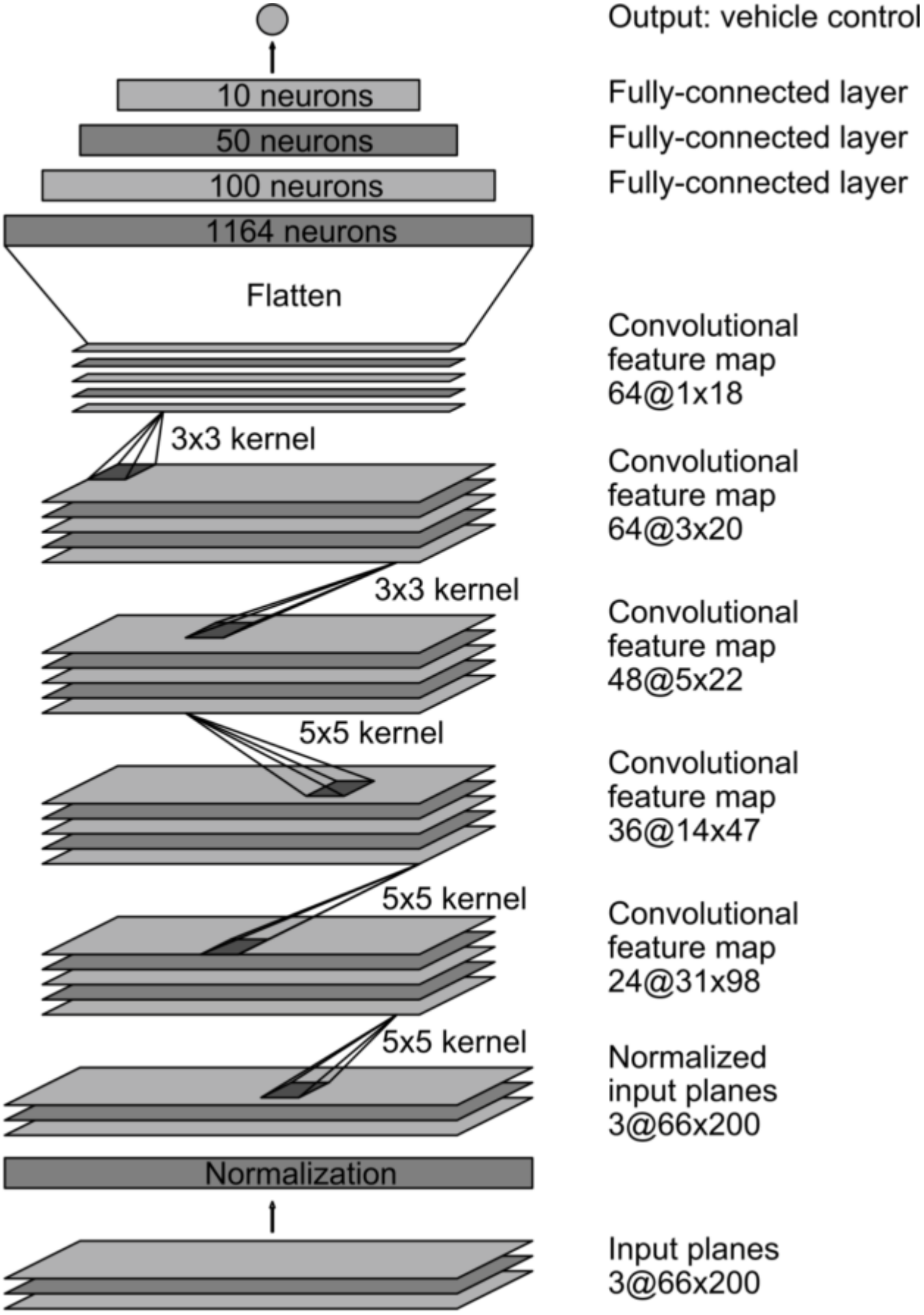
3. Submission code is usable and readable

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

After considering few initial simple models for training the data, I have chosen the Nvidia's CNN model that was employed in the end-to-end training approach using fron-facing camera images. Nvidia's CNN end-to-end approach



The model worked well but I had to modify the model for few changes like adding Cropping layer after the Normalization layer. The model consists of the following layers as shown below:

Layer	Description
Input	160x320x3 RGB image
Normalization layer	outputs Normalized Image, mean = 0.0
Cropping2D	outputs 80x320x3 Cropped Image
Convolution2D 5x5	2x2 subsample, VALID padding, out: 38x158x24
RELU	
Convolution2D 5x5	2x2 subsample, VALID padding, out: 17x77x36
RELU	
Convolution2D 5x5	2x2 subsample, VALID padding, out: 7x37x48
RELU	
Convolution2D 3x3	VALID padding, out: 5x35x64
RELU	
Convolution2D 3x3	VALID padding, out: 3x33x64
RELU	
Flatten Layer	out: 6336
Fully connected	6336x100
Fully connected	100x50
Fully connected	50x10
Output layer	10x1

The network consists of 10 layers, including Normalization layer, Cropping layer, 5 Convolutional layers, 3 Fully connected layers, and an output layer. The input image size is 160x320x3 and the input images are Normalized (to have a mean of about 0) using the keras.Lamda layer (model.ipynb line). And also the images are cropped using keras.Copping2D layer to remove the not so useful top and bottom portions of the images. The Normalization and Cropping layer are implemented using keras layers helps accelerated via GPU processing. The Convolution layers are implemented for feature extraction to extract the features of the path in the images. Convolutional layers are implemented with subsample/stride of 2x2 for 3 layers with 5x5 filter and 2 layers are implemented with 3x3 filter with stride of 1x1. The Convolutional layers are followed by RELU activation layer. The five convolutional layers are followed by 3 fully connected layers and final output layer evaluates a output steering angle value.The fully connected layers are designed to function as controller for steering.

2. Attempts to reduce overfitting in the model

The module showed overfitting during the initial runs when the image data was only from center images. To avoid model overfitting data was augnmented using the left and right images from the camera using slight correction of center image steering angle. Also the images are flipped with steering angle reversed to add additional data to help with overfitting. The model later did not show overfitting with augmented data.

3. Model parameter tuning

The model used an adam optimizer, so the learning rate was not tuned manually (model.py line 123).

4. Appropriate training data

The training data was added uding the image data from track one. Training data from center image data with steering angle data and also the images from left/right camera data is taken with slight correction of the steering angle. The training data is augmented with flipped images using the cv2.flip opencv function with steering angle reversed. Additional data is collected for the curves on the path to help with training.

Model Architecture and Training Strategy

1. Solution Design Approach

To arrive at the appropriate model for achieving the autonomous driving, initially started with the classic LeNet model that become popular with image classification. The network had (2) Convultional layers with 5x5 filterseach followed by a maxpooling layers. The input data was also Normalized using the keras.Lambda layer for better performance. The network was trained using the center image data using mean-squared-error(mse) loss and adam optimizer function. And the model was validated by using a split of the data (of 20%). The model showed high mse on both training data and validation data showing underfitting characteristic. To deal with that I tried to add additional full connected layers making the model more deeper but did not show much improvement. Also tried using cropped data but it did not help.

So, I resorted to Nvidia model that was used for end-to-end deep learning for autonoumous driving and the initial results on training data showed positive results showed decrease in mse for training data but the validation data showed high mse showing overfitting characteristic. To combat this I have collected additional data for driving around the curves and also the image data is augmented using left/right images with certain correction of steering angle. And also the image data if horizontally flipped (with steering angle reversed) to aid in the recovery from deviations off the road and also help driving during curves. This helped overcoming the overfitting problem. And the model was saved to HDF5 format (model.py line 129).

The final step was to run the simulator to see how well the car was driving around track one. There were a few spots where the vehicle fell off the track like turing around the curves, places where car was going from lane markings to road where lane markings were missing.. So I collected the data and the model was trained again with this additional data using the saved model.

At the end of the process, the vehicle is able to drive autonomously around the track without leaving the road.

2. Final Model Architecture

The final model architecture (model.py lines 108-120) consisted of the CNN with the following layers and layer sizes.

Here is a visualization of the architecture (note: visualizing the architecture is optional according to the project rubric)

```
#*********************************************************************
model = Sequential()
model.add(Lambda(lambda x: (x/255.0) - 0.5, input_shape = (160, 320, 3)))
model.add(Cropping2D(cropping=((60,20),(0,0))))
model.add(Convolution2D(24,5,5, subsample=(2,2), activation="relu"))
model.add(Convolution2D(36,5,5, subsample=(2,2), activation="relu"))
model.add(Convolution2D(48,5,5, subsample=(2,2), activation="relu"))
model.add(Convolution2D(64,3,3, activation="relu"))
model.add(Convolution2D(64,3,3, activation="relu"))
model.add(Flatten())
model.add(Dense(100))
model.add(Dense(50))
model.add(Dense(10))
model.add(Dense(1))
#*********************************************************************
```

3. Creation of the Training Set & Training Process

To capture the driving behaviour, I have first recorded the data for one lap trying to stay in the center of the road. This data worked well with autonomous driving but the car was going off track at curves and where there are no lane markings. I also used the images from left/right cameras with slight adjustment of the steering angle correction to help with recovery.

Images (left,center,right) from drivingin the middle of the lane

Then I recorded the data for driving around the curves with reduced speed and maintaining in the center. This helped with the training but the autonomous driving was still failing. Then I recorded the data for going in anti-clockwise direction for one lap to aid with turning around the corners and getting view of the surroundings for proper steering angle.

Images (left,center,right) from driving in anti-clockwise direction

The data was also augmented with left/right images along with flipping of the images horizontally with adjusting of the steering angle. This finally helped the car to drive autonoumously.

Data-Augmentation: Images (normal, flipped-horizontally) are as shown below:


After the collection process, I had 8400 images and with data augmentation the number of data points went up 50400 images. This data has been pre-processed with Normalization/Cropping before feeding into the model. The images are cropped to accelerate the training process and also to remove unwanted features from the training model.

Images (Input, Cropped-Vertically)


I finally randomly shuffled the data set and put 20% of the data into a validation set. To accelerate the training process the data was provided using the generator function (model.py line 23) inorder to avoid such huge data in memory and this helped with speeding up the training the model.

I used this training data for training the model. The validation set helped determine if the model was over or under fitting. The ideal number of epochs was 5 as evidenced by plotting the mean-square-error loss vs. the epoch and after cpochs the training did not show much improvement. The data was plotted by capturing the history object (model.py line 71).

I used an adam optimizer so that manually training the learning rate wasn't necessary.

Summary

It was good experience doing this project relaizing the importance of data collection for autonomous driving and how the network model characteristics were changes (overfitting/underfitting) with the data augmentation. During training using the keras generator function helped with the memory and accerlation issues. The Network performed well for the giving test conditions and training data and can be further improved using different image space (like YUV) to help with feature map extraction. Overall the final network model was successful in achieving the autonomous driving task.