Behavioral Cloning Project - Final Submission

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

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
- model.h5 containing a trained convolution neural network
- writeup_report.pdf for 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

My model consists of two convolutions neural network with 5x5 filter sizes each and depths of 24 and 32 respectively (model.py lines 103 and 107) I am also doing MaxPooling after both convolution layers.

The model includes RELU layers to introduce nonlinearity (code line 103 and 107), and the data is normalized in the model using a Keras lambda layer (code line 101).

The model flattens the output of last convolution layer and then passes the flattened output through the 3 Dense layers of size 32, 16, and 1 respectively (code line 112, 115 and 117).

I am also using mean square error (***MSE***) loss function and ***ADAM*** optimizer to train the network (code line 119).

2. Attempts to reduce overfitting in the model

The model contains 2 dropout layers in order to reduce overfitting (model.py lines 113).

The model was trained and validated on different data sets to ensure that the model was not overfitting (code line86). I used sklearn library to split the dataset into 2 parts, one for training and another one for validation with the test_size of 20%. 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 (model.py line 118).

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. While training the network, I considered all left, center and right camera images of each frame, by correcting steering angles for left and right camera images

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

Model Architecture and Training Strategy

1. Solution Design Approach

The overall strategy for deriving a model architecture was to keep the car running on center of the road.

My first step was to use a convolution neural network model similar to the LeNet Architecture. I thought this model might be appropriate because it is one of the powerful and first successful image recognition model.

In order to gauge how well the model was working, I split my image and steering angle data into a training and validation set. I found that my first model had a high mean squared error(***mse***) on the training set. Then I normalized the input image and the mse stared decreasing. After training the network for 5 epochs, mse was low but the results were not up the mark.

Then I tried NVEDIA architecture. In that architecture I cropped and downscaled my images so that the model can learn faster. But in this architecture I was getting a low mse on training set but high mse on the validation set. This implied that the model was overfitting.

To combat the overfitting, I modified the model by removing some Convolution Layers and Fully Connected Layers.

After changing the model architecture and training the model for 3 epochs, I finally got the training loss as 0.0146% and validation loss of 0.0117%

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. To improve the driving behavior in these cases, I augmented the data by flipping the images taken by center camera, adding more images in the training set taken from left and right cameras with steering angle correction.

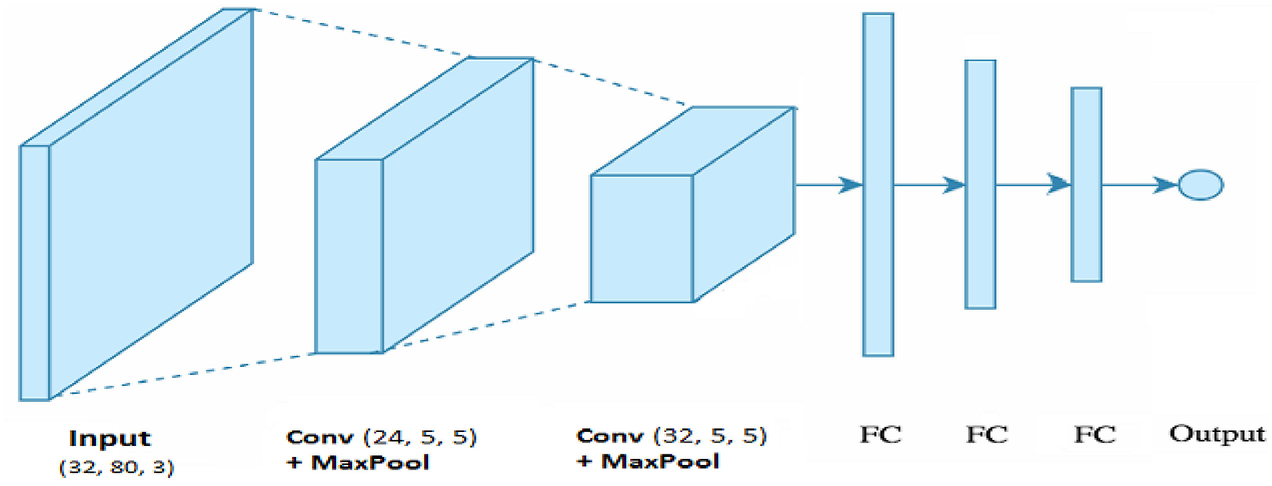
At the end of the process, the vehicle is able to drive autonomously around the track without leaving the road.

2. Final Model Architecture

The final model architecture (model.py lines 99-120) consisted of 2 Convolution Neural Networks with 3 Fully Connected layers.

- **Convolution Layer 1**: 5x5 with 24 filters
- **Convolution Layer 2**: 5x5 with 32 filters
- **Flatten Layer**
- **Dense Layer 1** - 32
- **Dense Layer 2** - 16
- **Output Layer** - 1

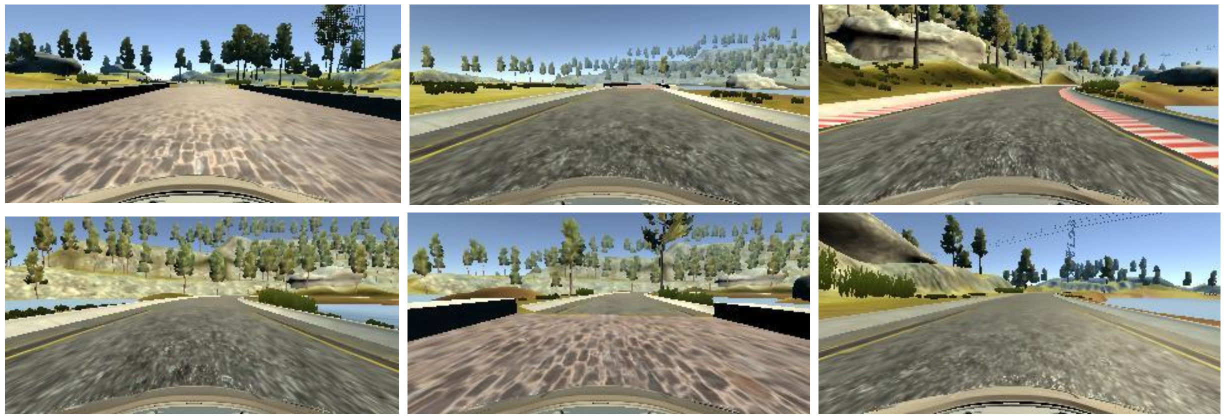
Here is a visualization of my architecture:



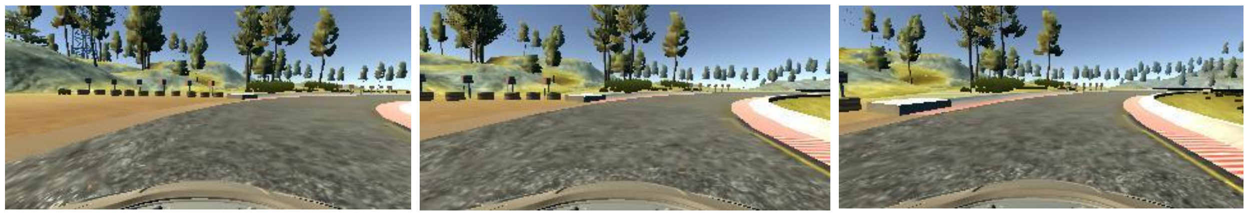
3. Creation of the Training Set & Training Process

To capture good driving behavior, I first recorded two laps on track one using center lane driving.

Here is an example image of center lane driving:



I then recorded the vehicle recovering from the left side and right sides of the road back to center so that the vehicle would learn to keep itself in middle of road on steep turns. These images show what a recovery looks like.



To augment the data sat, I also flipped images and angles thinking that this would help the car to stay in middle of the roads while turning on different curves. Also, I used left and right camera images to train the network by correcting the steering angle by +- 0.20 degree.

After the collection process, I had 25712 number of data points. I then preprocessed this data by converting the color scheme from BGR to RGB (since cv2 reads the image in BGR color scheme and simulator sends images in RGB format). Then I cropped and rescaled the image to get the final image shape as **32, 80, 3.** This helped in training the model faster.

I finally randomly shuffled the data set and put 20% of the data into a validation set.

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 3 as after 3rd epoch the model started oscillating around 0.014% training loss. Also, I used an Adam optimizer so that manually training the learning rate wasn't necessary.

3. Simulation

The output.mp4 video shows the working of my architecture. Here for every frame the simulator sends the center camera images to the model in backend to get the steering angle which is determined in real-time by processing the image and passing it through the network.