Derek Wu 03/07/2017
Udacity Self Driving Car Nanodegree Project 3 – Submission 2

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:

1) model.py containing the script to create and train the model
2) drive.py for driving the car in autonomous mode
3) model.h5 containing a trained convolution neural network
4) P3_writeup.PDF 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 a simplified convolution neural network of Nvidia's Dave-2 system as mentioned in class. The main difference that I changed is first fully connected layer. By adding the full 1164 size layer, the model tends to over fit the data.

The model includes RELU layers to introduce nonlinearity using the model.add API in keras (code line 24 to 42). The model also has ELU layers in the fully connected layers. The data is normalized in the model using a Keras lambda layer (code line 24).

2. Attempts to reduce overfitting in the model

As shown in the comments, I have tried various methods to prevent over fitting. I have tried using dropout layers in the fully connected layers. However, every time I added the dropout layers, the model tends to run into the vanishing gradient problem where no matter what input data I provide, the output is a constant small angle. This is mediated with the help of ELU activation functions on the fully connected layers.

The model was trained and validated on different data sets to ensure that the model was not over fitting (code line 164). Also, the program has early stop implemented as well to prevent over fitting. However, since I only ran the model with 3 epochs, this method is not as effective. 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 (model.py line 47). Therefore there was no need to adjust the learning rate.

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, the left and right camera drive data, and random actions that flip the left and right camera images. The output data for the training was only the steering angle. The steering angle for the left and right camera were compensated with a calibration offset. This offset was manually tuned until the model performed well on the track.

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 use trial and error and using some of the intuition that this class has provided.

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 worked pretty well with the previous project. However, during the turns where one side is not marked, the car tends to drift completely off-road. In the end, I closely followed Nvidia's Dave-2. Also, multiple location of dropout layers was placed to prevent overfitting. In addition, max pooling layers were attempted but the model performed worse as well. The final model was extremely closely to the original model with the first fully connected layer removed since it over fit the data, 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 relative low mean squared error on the training set but a high mean squared error on the validation set. This implied that the model was overfitting. I adjusted the LeNet architecture with additional dropout layers at the fully connected layer section. However, the model was not adequate to keep the car on track. After, the Nvidia Dave-2 architecture was explored. The overall training and validation error was significantly lower than LeNet (about 100 times better).

To combat the over fitting, I modified the model so that it had dropout layers at different strategic locations. However, when I added additional drop out layers, the model would tend to run into the vanishing gradient problem and outputs a constant single steering angle. The final method to prevent over fitting was to add drop out layers with ELU activation functions for the fully connected layers.

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, particularly at the area without the lane markings. To improve the driving behavior in these cases, I removed the 1164 fully connected layer as shown in Dave-2. Also, I created artificial augmented data by flipping the left and right camera images horizontally. By creating this data, my final model performed outstandingly well and was able to correct its course where it couldn't in the previous models.

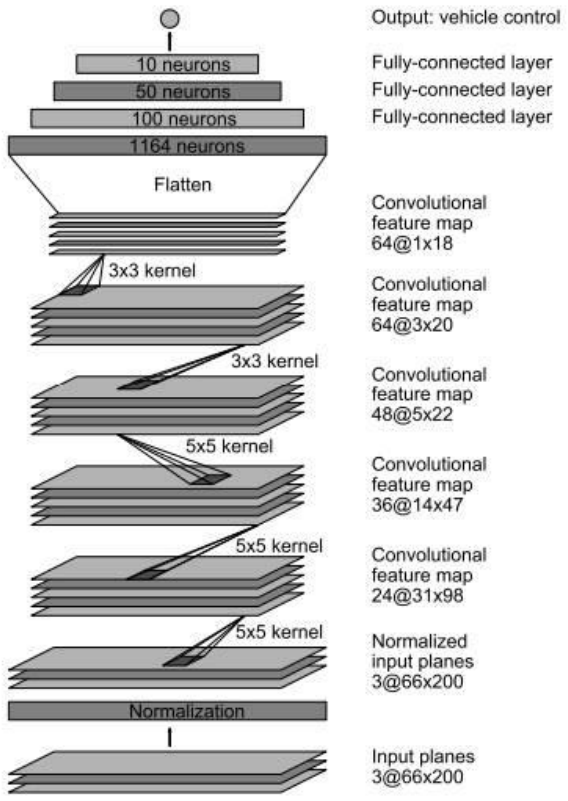
At the end of the process, the vehicle is able to drive autonomously around the track without leaving the road.

2. Final Model Architecture

The final model architecture (model.py lines 20-49) consisted of a convolution neural network with the following layers and layer sizes:

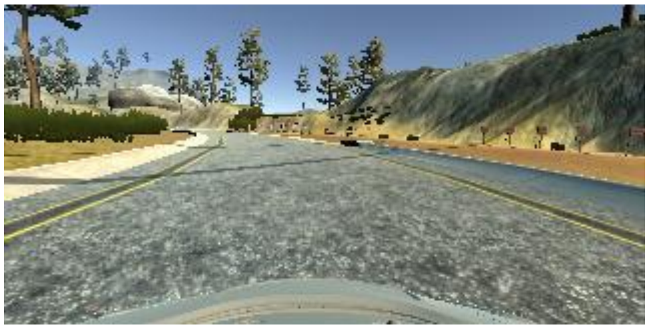
Type	**Description**
Convolution Layer	5x5 kernel with a depth of 24 and stride of (2,2)
Convolution Layer	5x5 kernel with a depth of 36 and stride of (2,2)
Convolution Layer	5x5 kernel with a depth of 48 and stride of (2,2)
Convolution Layer	3x3 kernel with a depth of 48 and stride of (1,1)
Convolution Layer	3x3 kernel with a depth of 48 and stride of (1,1)
Flatten Layer	
Fully Connected Layer	Size of 100
Fully Connected Layer	Size of 50
Fully Connected Layer	Size of 10
Fully Connected Layer	Output Layer (Size of 1)

Here is a visualization of the architecture

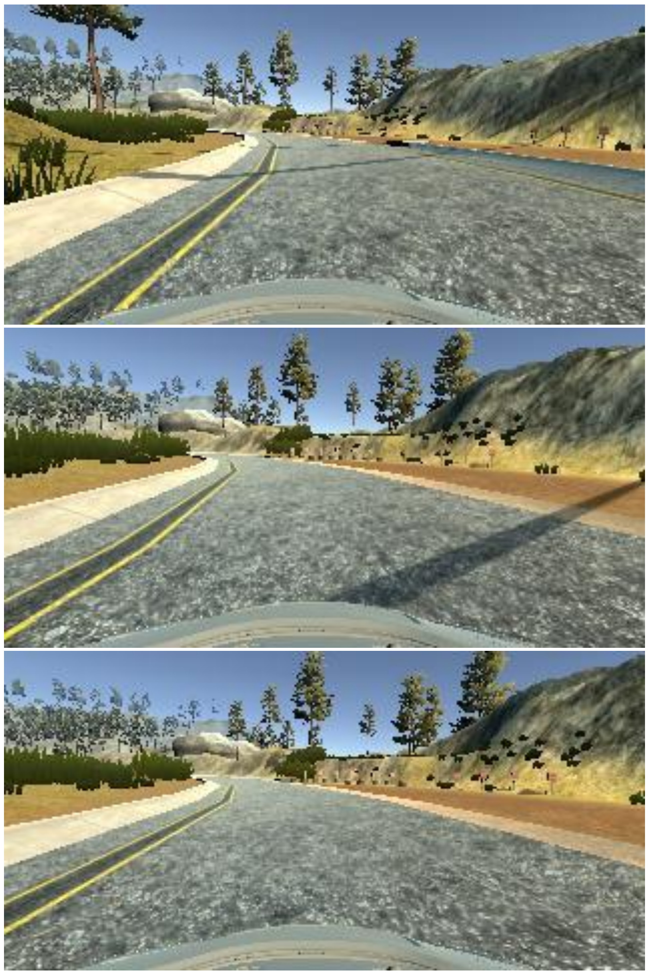


3. Creation of the Training Set & Training Process

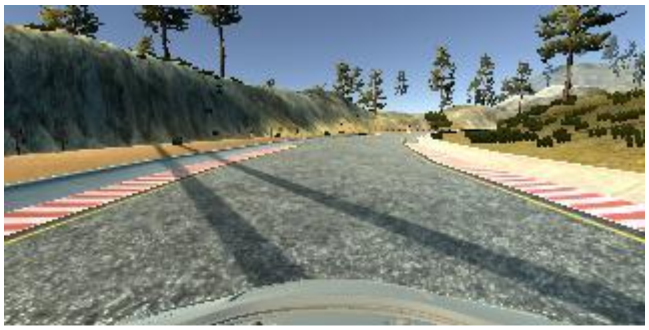
To capture good driving behavior, I first recorded two laps on track one using center lane driving. I then recorded two laps on the track in reverse direction. Here is an example image of center lane driving:



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 adjust itself when it gets too close to the sides. These images show what a recovery looks like starting from the left and corrected to the center:



To augment the data sat, I also flipped images and angles thinking that this would ... For example, here is an image that has then been flipped:



After the collection process, I had 26081 number of data points. I then preprocessed this data by normalizing the data and cropping away the scenery and the car.

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 evidenced by the fact that either the model did not perform well on the track or the validation data error will start increasing after too many epochs.