Behavioural Cloning Project

Goals of this Project

- Use the simulator to collect data of good driving behaviour
- Build a convolutional neural network in Keras that predicts the steering angles from images.
- Train and validate the model using a training and validation set.
- Test that the model successfully drives around the tract without trailing off the road.

Files Submitted

My project includes the following files:

- model.py containing the script to create and train the model
- drive.py for driving the car autonomously
- model.h5 containing a trained CNN
- writeup_report.pdf summarising the results

Usage

Using Udacity's simulator and drive.py file, the car can be driven autonomously around the track by executing the following command:

- *python drive.py model.h5*

The model.py file contains the code for training and the saving the CNN. The file shows the pipelines I used for training and validating the model.

Model Architecture

I have used NVIDIA's end-to-end autonomous driving network. This model consists 9 layers: 5 convolutional layers followed by 3 fully connected layers followed by one output layer with 1 neuron. Before the convolutional layer is a lambda layer which does the normalisation (line 83). The input images to this model has the shape 160x320x3.

The first 3 convolutional layers use strided convolutions with 2x2 stride and 5x5 kernel. The remaining 2 convolutional layers are non-strided with a 3x3 kernel. Here are the corresponding depths of each convolutional layer from layer 1 through layer 5: 24,36,48,64,64. I have added a dropout layer with 50% probability after the first fully-connected layer to control overfitting. Each of the 8 layers are deployed with 'RELU' activation to introduce nonlinearity. The output layer utilises the 'tanh' activation because predicting steering-angles is a regression problem and tanh values vary between -1 and 1 which is suitable for the angle prediction.

Model Parameter Tuning

This model uses an adam optimizer and 'mean square loss' objective function and a default learning rate of 0.001.

Training Data

To train the model, I have used the driving data images provided by Udacity. However this data was insufficient for training the NVIDIA model and therefore I had to augment the training data. For augmentation purpose, I have introduced random brightness to all the images and also a random rotation to generalise the model. I haven't flipped the images around the vertical axis because the dataset consisted of flipped images and re-flipping the images might lead to overfitting. I have also added images depicting lane recovery where the car started off from the corner and recovered back to the centre of the road.

Solution Design Approach

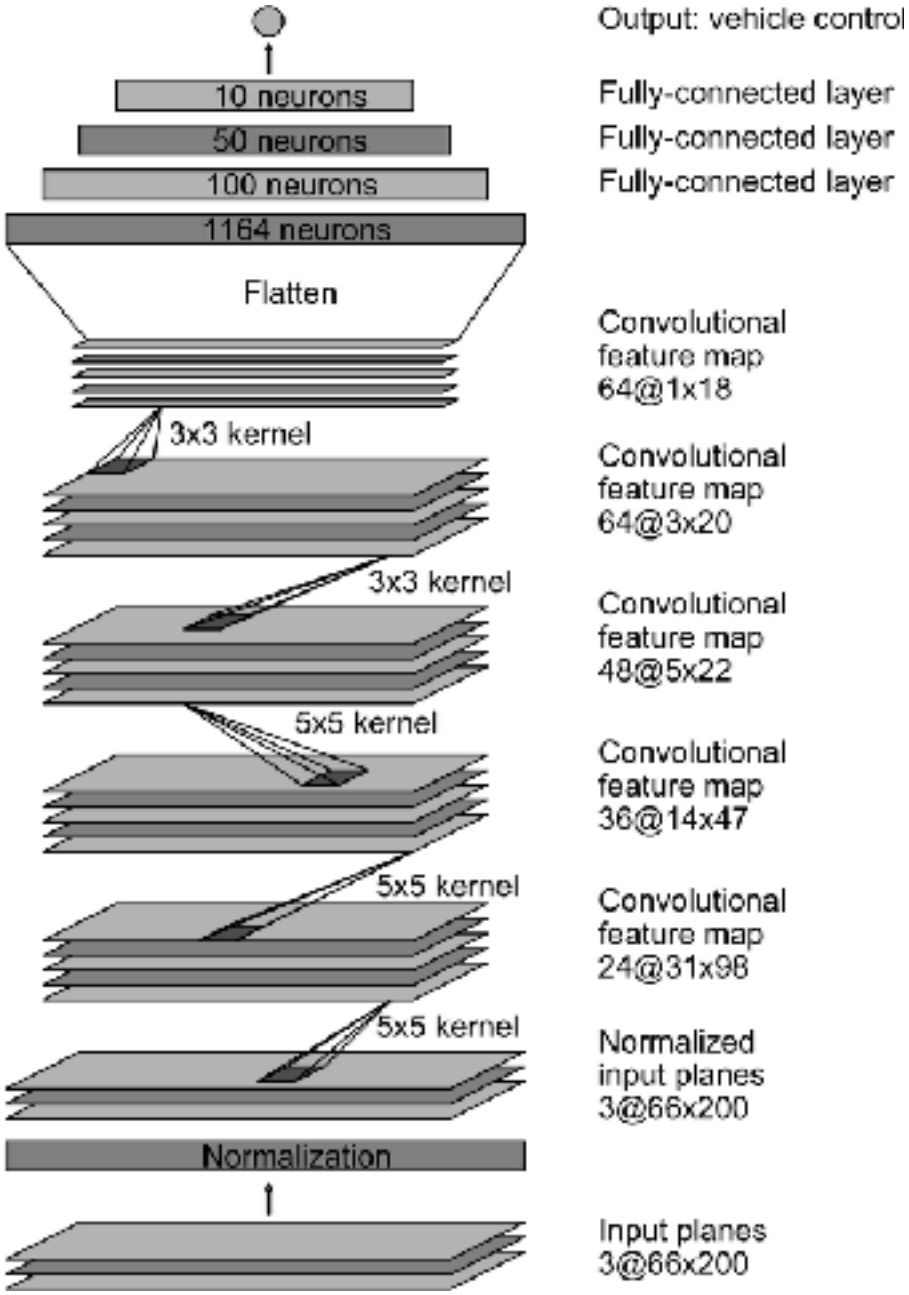
I have used NVIDIA's model for this problem because, NVIDIA's end-to-end driving model has shown great results in the real world autonomous car driving scenario. I have also tried using alexnet model and comma ai's autonomous driving model. Although, both these models worked, I found that NVIDIA's model was better in terms of generalisation. I have also tried adding MaxPooling2D layers and AveragePooling2D layers and using LeakyReLU activation function, but the original NVIDIA model worked better. I have used the 70:30 split ratio of the data for training-validation purpose. This model showed improved performance when more data was inputted for training.

To combat overfitting, I have used data augmentation as described above.

When I first ran my autonomous simulation, the car always fell in the water and performed worser on Track 2. I could solve this problem by augmenting the dataset and adding a dropout layer, and setting the correction (46, 57, 68) to 0.05.

Final Model Architecture

Below is the depiction of the NVIDIA model I have used for this project. source: https://devblogs.nvidia.com/parallelforall/deep-learning-self-driving-cars/



Training

For training purpose, I have used the following parameters.

samples_per_epoch = len(train_sample)*3 (x3 because there are three variation for each image: centre image, left image and right image)

nb_epoch = 10

nb_val_samples = len(validation_samples)

I have shuffled the data to avoid data skewing and used 70% of data for training and 30% of the data for validation. After collecting the data, I had around ~50,000 images in my dataset. I used adam optimiser for optimisation.