Behavioral Cloning

Project No. 3

Isaac Kargar

Behavioral Cloning Project

The goals / steps of this project are the following:

- Use the simulator to collect data of good driving behavior or use provided data by UDACITY team
- 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

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.pdf summarizing the results

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
```

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. You can also find behavioral_cloning_model.ipynb file which is a notebook contain training procedure of the model.

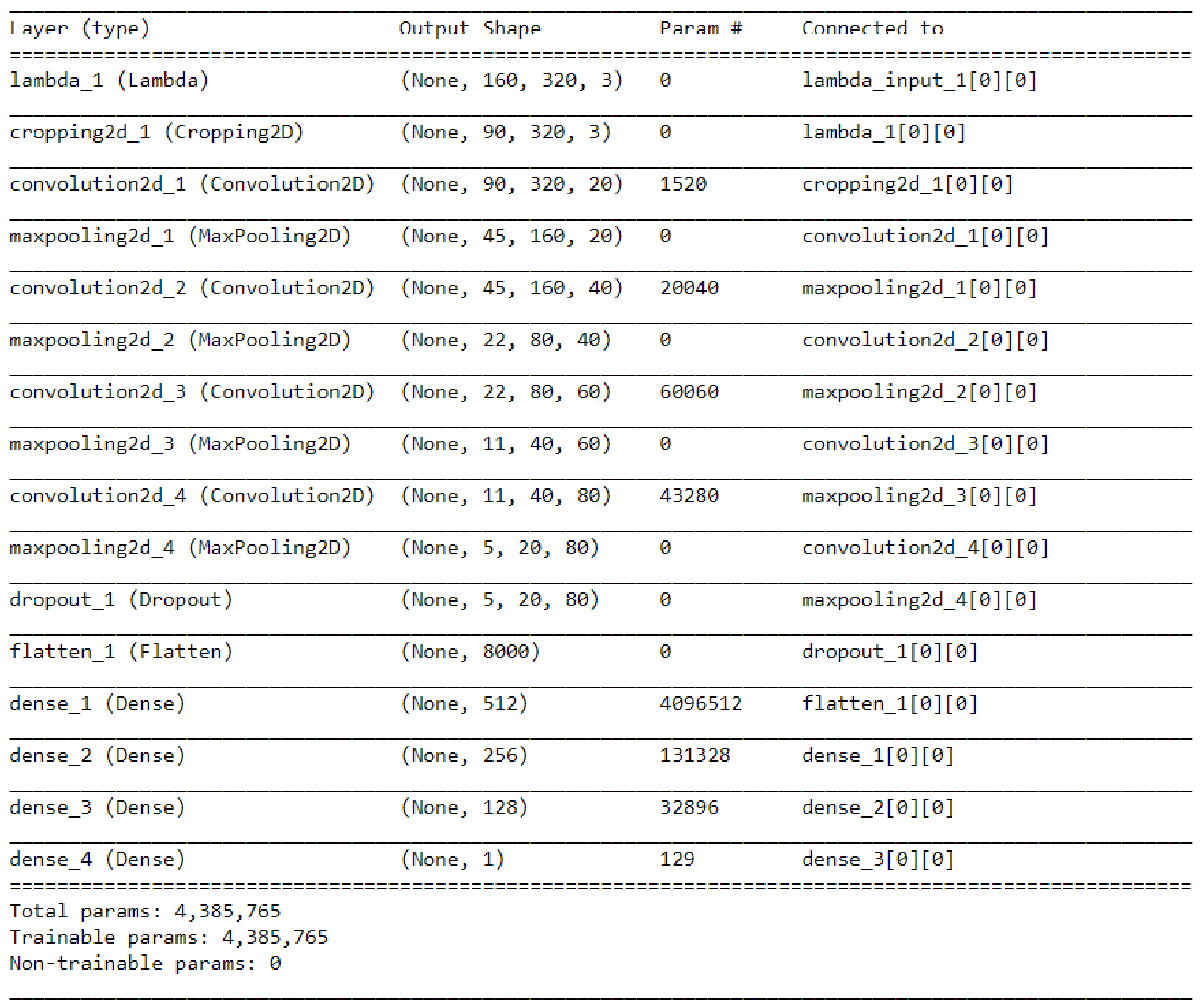
Model Architecture and Training Strategy

My model consists of a some convolution layers with 5x5 and 3x3 filter sizes and depths between 20 to 80 following with max pooling layers.

The model includes RELU and ELU layers to introduce nonlinearity, and the data is normalized in the model using a Keras lambda layer.

The model includes dropout to prevent overfitting, a flatten layer and four fully connected layers.

You can see the model summary below:



```
____________________________________________________________________________________________________
Layer (type)                     Output Shape          Param #     Connected to
====================================================================================================
lambda_1 (Lambda)                (None, 160, 320, 3)   0           lambda_input_1[0][0]
____________________________________________________________________________________________________
cropping2d_1 (Cropping2D)        (None, 90, 320, 3)    0           lambda_1[0][0]
____________________________________________________________________________________________________
convolution2d_1 (Convolution2D)  (None, 90, 320, 20)   1520        cropping2d_1[0][0]
____________________________________________________________________________________________________
maxpooling2d_1 (MaxPooling2D)    (None, 45, 160, 20)   0           convolution2d_1[0][0]
____________________________________________________________________________________________________
convolution2d_2 (Convolution2D)  (None, 45, 160, 40)   20040       maxpooling2d_1[0][0]
____________________________________________________________________________________________________
maxpooling2d_2 (MaxPooling2D)    (None, 22, 80, 40)    0           convolution2d_2[0][0]
____________________________________________________________________________________________________
convolution2d_3 (Convolution2D)  (None, 22, 80, 60)    60060       maxpooling2d_2[0][0]
____________________________________________________________________________________________________
maxpooling2d_3 (MaxPooling2D)    (None, 11, 40, 60)    0           convolution2d_3[0][0]
____________________________________________________________________________________________________
convolution2d_4 (Convolution2D)  (None, 11, 40, 80)    43280       maxpooling2d_3[0][0]
____________________________________________________________________________________________________
maxpooling2d_4 (MaxPooling2D)    (None, 5, 20, 80)     0           convolution2d_4[0][0]
____________________________________________________________________________________________________
dropout_1 (Dropout)              (None, 5, 20, 80)     0           maxpooling2d_4[0][0]
____________________________________________________________________________________________________
flatten_1 (Flatten)              (None, 8000)          0           dropout_1[0][0]
____________________________________________________________________________________________________
dense_1 (Dense)                  (None, 512)           4096512     flatten_1[0][0]
____________________________________________________________________________________________________
dense_2 (Dense)                  (None, 256)           131328      dense_1[0][0]
____________________________________________________________________________________________________
dense_3 (Dense)                  (None, 128)           32896       dense_2[0][0]
____________________________________________________________________________________________________
dense_4 (Dense)                  (None, 1)             129         dense_3[0][0]
====================================================================================================
Total params: 4,385,765
Trainable params: 4,385,765
Non-trainable params: 0
____________________________________________________________________________________________________
```

The model was trained and validated on different data sets to ensure that the model was not overfitting and was tested by running it through the simulator and ensuring that the vehicle could stay on the track. The video file has been saved and you can find it in my github repository.

Model parameter tuning

The model used an adam optimizer, so the learning rate was not tuned manually. Because the problem is a regression problem, I used MSE as loss function.

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 and used augmentation techniques to provide more images for different situations.

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

Model Architecture and Training Strategy

Solution Design Approach

I used the data set that UDACITY team provided.

My first step was to use a fully connected layer firstly and evaluate the model performance on the simulator. Then I added convolutional layers to model step by step and evaluated the model performance in the simulator. I tried to build a network with a structure like NVIDIA 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 used left and right camera images and added 0.25 correction factor to steering angle for left camera images and subtracted 0.25 correction factor from right camera images. I also used brightness augmentation to simulate different conditions such as night and day.

To combat the overfitting, I adde dropout layer with 0.5 keeping ratio after convolutional 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. To solve this problem I used explained augmentation techniques above and added convolutional layers to the model.

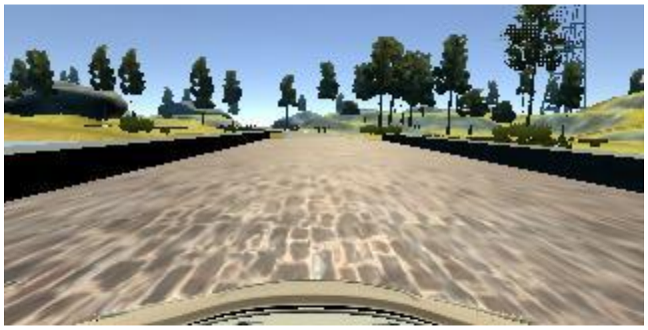
At the end of the process, the vehicle is able to drive autonomously around the track without leaving the road.

Final Model Architecture

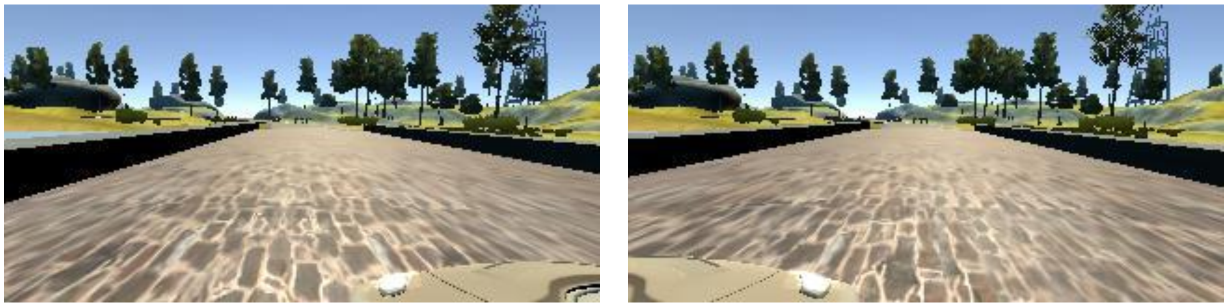
The final model architecture described in previous sections.

Creation of the Training Set & Training Process

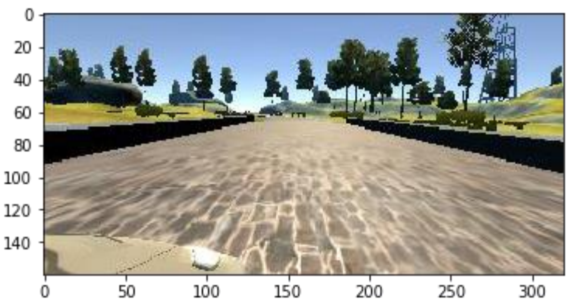
To capture good driving behavior, I first used center lane images. Here is an example image of center lane driving:

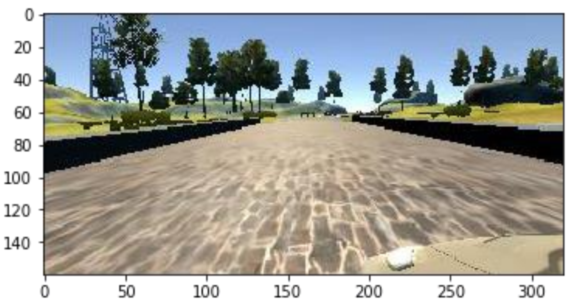


I then used left side and right side images of the road back to center so that the vehicle would learn to recover to center when it goes off the road. These images show what a recovery looks like.



To augment the data sat, I also flipped images and angles thinking that this would help to generate more data and the model to be more general and prevent overfitting. For example, here is an image that has then been flipped:





I also used adding and subtracting correction factor to steering angle of left and right camera images and used brightness augmentation to simulate different conditions such as day and night.

After the collection process, I had `72324` number of data points. I then preprocessed this data by normalizing and cropping in the model using keras Lambda and Cropping2D layers. I normalized image pixels by dividing to 255 and subtracting 0.5 and then cropped 50 rows of pixels from top of images and 20 rows of pixels from buttom of images.
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 15 as evidenced by performance of model in the simulator. I used an adam optimizer so that manually training the learning rate wasn't necessary.