Behavioral Cloning

Writeup Report

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

Rubric Points

Here I will consider the rubric points (https://review.udacity.com/#!/rubrics/432/view) 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.md and writeup_report.pdf summarizing the results
- weights.h5 from previous training
- run1.mp4 containing the video of Autonomous driving using my model.

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

The top layer is a Cropping2D layer which crops top and bottom parts of the input image.

My model consists of 5 Convolutional layers followed by one Maxpooling and 4 Fully Connected(Dense) layers. The convolutional layers are designed for feature extraction. First 3 convolutional layers are designed with a 5x5 kernel and 2x2 strides and depths 24,36,48 respectively.(code line 59-61). The other 2 convolutional layers are non-strided with a 3x3 kernel and with a depth of 64(code line 63-64). Then I have used a MaxPooling layers with a pool size of (2,2)(Code line 65). I flattened the network using keras Flatten() function(Code line 66). These layers are followed by 4 Dense layers with sizes 100,50,10 and 1 (code line 68-74). In the model I have used RELU activations to introduce nonlinearity, and the data is normalized in the model using a Keras lambda layer (code line 57).

2. Attempts to reduce overfitting in the model

The model contains dropout layers in order to reduce overfitting (model.py lines 71 & 73).

The model was trained and validated on different data sets to ensure that the model was not overfitting. 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 with a decaying learning rate from keras model callbacks. (model.py line 76).

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. I have collected the data by driving two laps on track 1 and two laps on track 2.

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

Model Architecture and Training Strategy

1. Solution Design Approach

My first step was to use a convolution neural network model similar to the NVIDIA autonomous car architecture. I thought this model might be appropriate because it has 5 convolutional layers for feature extraction followed by fully connected layers to estimate the steering angle.

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 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.

To combat the overfitting, I have added MaxPooling layer before Flatenning the data and added dropout layers with probability 0.5. It reduced the validation loss from 0.1496 to 0.0547.

Then I ran the model for 5 epochs and save the weights from it to use in next training. Next I loaded the weights and trained the model for another 10 epochs it drastically reduced the training loss and validation loss.

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 have added more data from left and right cameras with a corection of 0.15. I have also added more data by flipping the images horizontally to generalize the model better.

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 55-74) consisted of a convolution neural network with the following layers and layer sizes.

Layer	Description
Input	160x320x3 RGB image
Cropping2D	cropping=((50,20), (0,0))
Lambda	Normalization (X/255-0.5)
Convolution 1	5x5 kernel,24 Filters, 2x2 stride, relu activation
Convolution 2	5x5 kernel,36 Filters, 2x2 stride, relu activation
Convolution 3	5x5 kernel,48 Filters, 2x2 stride, relu activation
Convolution 4	3x3 kernel,64 Filters, relu activation
Convolution 5	3x3 kernel,64 Filters, relu activation
Max pooling	2x2 stride
Flatten	
Dense 1	Outputs 100, relu activation
dropout	prob=0.5
Dense 2	Outputs 50, relu activation
dropout	prob=0.5
Dense 3	Outputs 10, relu activation
dropout	prob=0.5
Dense 4	Outputs 1, relu activation

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 go back to center if it is going out of the track.

Then I repeated this process on track two in order to get more data points.

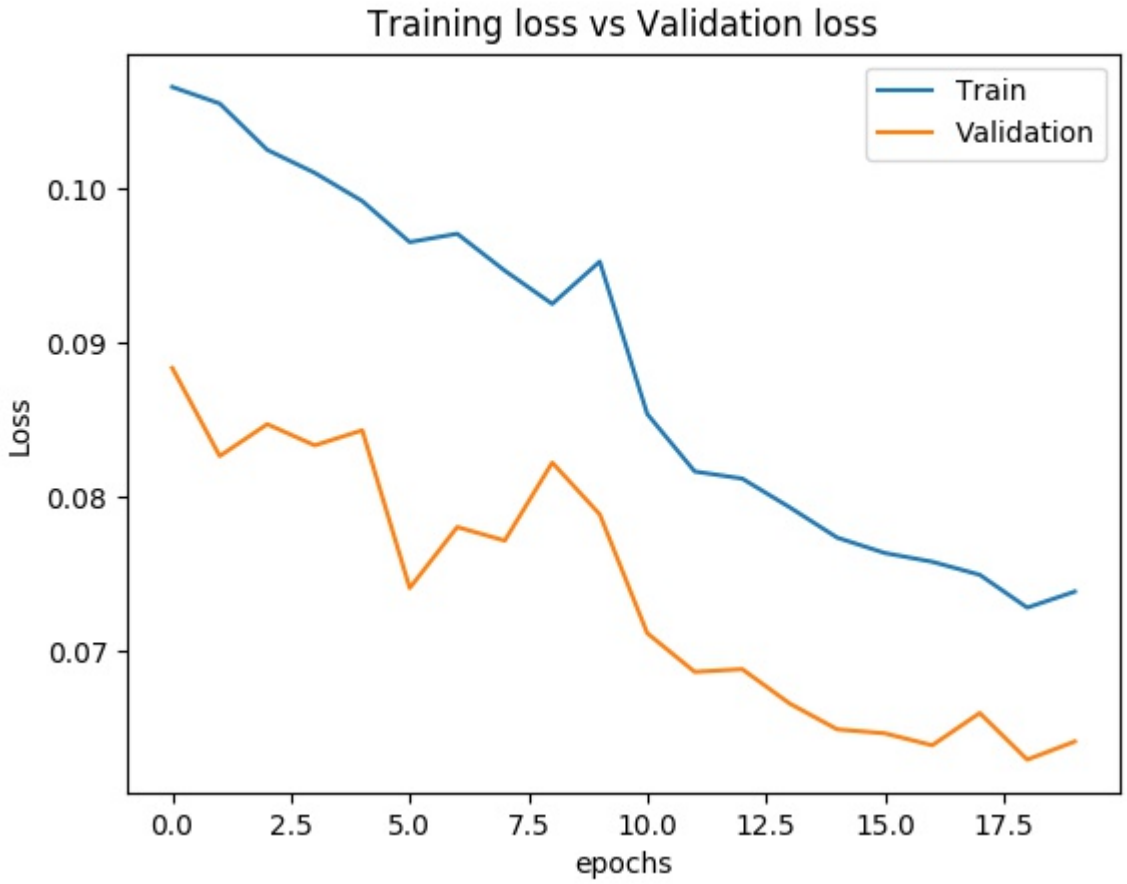
To augment the data set, I also flipped images horizontally(code lines 41-47) and angles thinking that this would help the model generalize better while taking turns. For example, here is an image that has then been flipped:

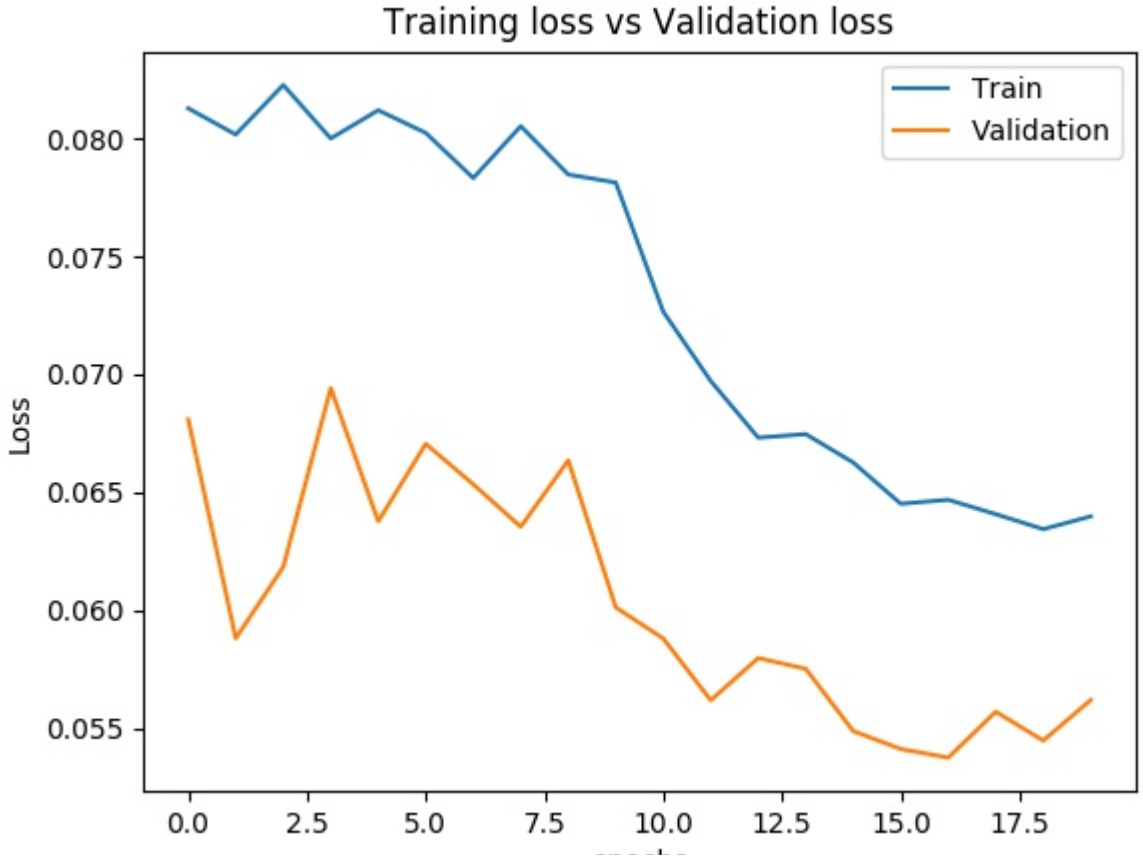
Also, while reading the images I have converted them to RGB space since cv2.imread() reads images in BGR space and drive.py uses RGB images.

After the collection process, I had 42000 number of data points. I then preprocessed this data by cropping(code line 56) 50 pixels on the top, 20 pixels on the bottom and 10 pixels on the right in order to remove unwanted data from the images. I have used keras Cropping2D function to crop the images while training. Then, I normalized the data using keras Lambda function(Code line 57). 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. First I have trained the model for 5 epochs and saw that loss is still decreasing so I saved the weights and used them again while re traing the model and now I have used 20 epochs and observed that training loss and validation loss settled. I used an adam optimizer and used a keras callback fucntion LearningRateScheduler to use decaying leraning rate while training(code line 10 & 77) .

Below I have provided the Learning curves of my model.


Training loss vs Validation loss
Train
Validation
Loss
0.10
0.09
0.08
0.07
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
epochs


Training loss vs Validation loss
Train
Validation
Loss
0.080
0.075
0.070
0.065
0.060
0.055
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5