Behavioral Cloning

Writeup

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

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 summarizing the results
- video.mp4 containing recording of two laps of the car driving autonomously using the trained 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

My model consists of a convolution neural network with 3x3 and 5x5 filter sizes, depths of 6 and 36, along with a couple of max pooling of 2x2 followed by a dense layer of 120.

The model includes ELU layers to introduce nonlinearity. ELU is chosen as it works better than RELU in problems of regression.

The data is normalized in the model using a Keras lambda layer.

Cropping2D layer was also deployed to crop the image to the data that is needed and avoiding all the unnecessary data that is of no use to train the model.

2. Attempts to reduce overfitting in the model

The model was trained and validated on different data sets to ensure that the model was not overfitting.

20% of the data is classified as validation data set and 80% of the data is used as training data set. The mean squared error loss function is used to find the loss on the training and the validation data and to compare them to avoid over fitting.

The model was tested by running it through the simulator for two laps continuously and thus 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.

4. Appropriate training data

Training data was chosen to keep the vehicle driving on the road.

I was being a little ambitious that, I made up my mind on using only the sample data provided and not record any new data and make the vehicle to run autonomously and I have finally done it. But yeah, it took me lot of time.

I used images of center, left and right camera images from the sample data provided by Udacity.

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 find a model that can make the vehicle steady and almost stay in the middle of the road all the time.

My first step was to use a convolution neural network model not similar to any of the pre-defined architecture that was already given to me. I wanted to do my own research and find a model of my own.

In order to gauge how well the model was working, I split my image and steering angle data into a training and validation set as I said earlier.

I have tried a lot of models and tried running the simulator on the model created. I have tried with very least amount of data possible to make the model work. But if we give very little amount of data and train the model in a very good convolution network, the model tries to mimic the behavior. So I stick on to very simple model and give medium amount of data and not over fit the model.

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 like the right turn after the bridge and also in the bridge where the vehicle went towards an

end. That's when I started using left and right camera images too instead of using only the center camera images. Since I needed more data.

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 consisted of a convolution neural network with the following layers and layer sizes.

- Cropping2D(cropping=((55,20), (0,0))
- Lambda(lambda x: (x/255.0) - 0.5)
- Convolution2D(6, 3, 3) → MaxPooling2D((2,2)) → Activation('elu'
- Convolution2D(36, 5, 5) → MaxPooling2D((2,2)) → Activation('elu')
- Flatten()
- Dense(120) → Activation('elu')
- Dense(1)

As I have said earlier, I have used 2 Convolution layers with max pooling and a dense layer. I have used ELU as the activation layer.

I have also used lambda for normalization and used Cropping layer of Keras to crop the upper and lower part of the image which has unnecessary information.

I feel that two convolution layers are enough to find the track and is enough for the course of the problem. IF the tracks becomes more complex or the course of the problem itself is changing, then I feel this model won't fit in. So it depends on the problem we are solving.

Please find below the summary of the architecture.

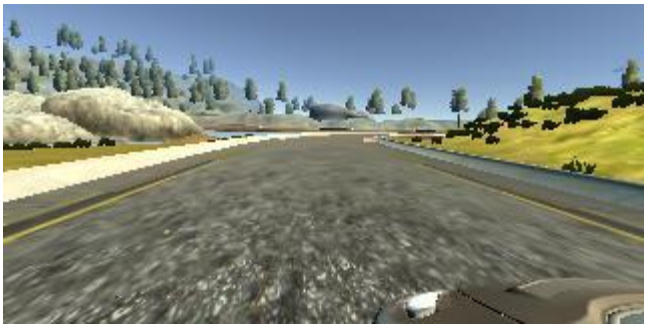
Layer (type)	Output Shape	Param #	Connected to
cropping2d_3 (Cropping2D)	(None, 85, 320, 3)	0	cropping2d_input_3[0][0]
lambda_3 (Lambda)	(None, 85, 320, 3)	0	cropping2d_3[0][0]
convolution2d_5 (Convolution2D)	(None, 83, 318, 6)	168	lambda_3[0][0]
maxpooling2d_5 (MaxPooling2D)	(None, 41, 159, 6)	0	convolution2d_5[0][0]
activation_7 (Activation)	(None, 41, 159, 6)	0	maxpooling2d_5[0][0]
convolution2d_6 (Convolution2D)	(None, 37, 155, 36)	5436	activation_7[0][0]
maxpooling2d_6 (MaxPooling2D)	(None, 18, 77, 36)	0	convolution2d_6[0][0]
activation_8 (Activation)	(None, 18, 77, 36)	0	maxpooling2d_6[0][0]
flatten_3 (Flatten)	(None, 49896)	0	activation_8[0][0]
dense_5 (Dense)	(None, 120)	5987640	flatten_3[0][0]
activation_9 (Activation)	(None, 120)	0	dense_5[0][0]
dense_6 (Dense)	(None, 1)	121	activation_9[0][0]

Total params: 5,993,365
Trainable params: 5,993,365
Non-trainable params: 0

3. Creation of the Training Set & Training Process

As I have mentioned earlier, I wanted to use the data set provided as sample alone. So I have used the images from the data set itself. Please find below a sample image taken from each of the center, left and right cameras at the same moment.

Left Camera Image:



Center Camera Image:

Right Camera Image:

I have tried a variety of model and got successful in few of those.

I followed a variety of approach too. I have first read excel contents to find the path of the images to read them. I shuffled them before reading images so as to produce randomness and to avoid mimicking the existing route.

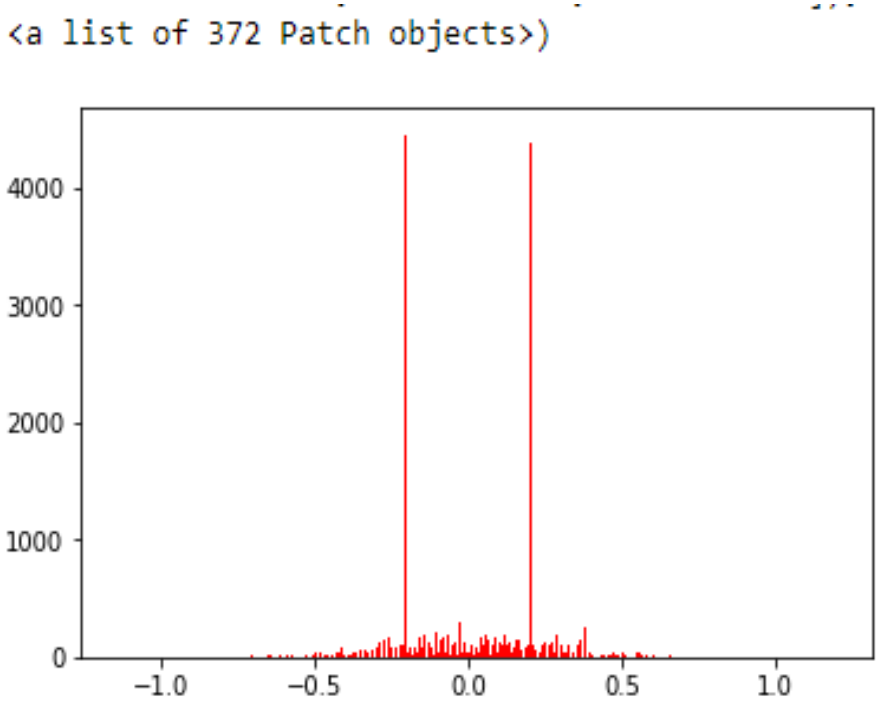
The process I have followed is, I have shuffled the images before reading. Then I read all the center, right and left camera images.

If the data has more images which has same steering angle, the neural network tends to go that angle all the time. So, I have a method Reduce_Extra_Images which will read the images and will take care of the data set such that, there are not more than 350 images in a particular angle. In this way I have also reduced the number of images that we use to train the model. And also tis method leaves the images for particular steering angle which has less that 10 images as it is not so important.

As I tried different methods, I have used augmentation before in certain models of mine, which I have tried. So I have left those methods in model.py file. But I didn't use it in my final model.

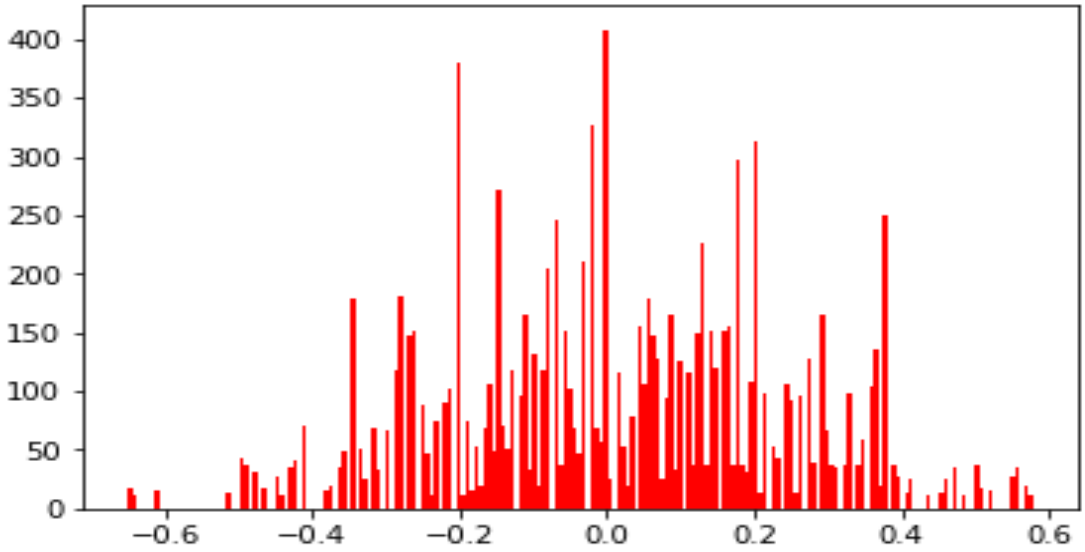
I have flipped the images using CV2 and also used brightness adjustment technique to augment the data. I have seen it in a blog that it will train the model well to make it more generic.

Before running my Reduce_Extra_Images method I had the data as plotted below against various steering angles.



And after calling the method, I had the data plotted as below against various angles.

```
<a list of 204 Patch objects>)
```

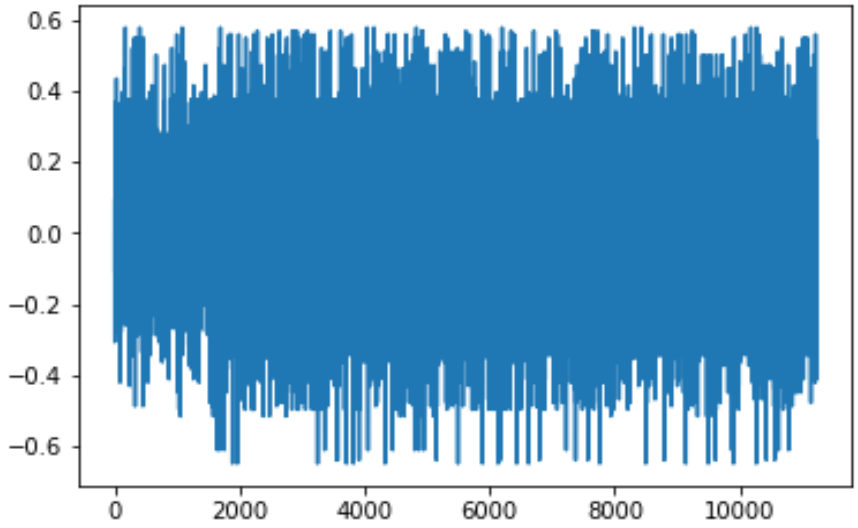


And you can see below, how the images are spread across the data set for various values of steering angles.

```
plt.plot(y_train)
```

```
See below to find how the steering angles are spread across the data set
```

```
[<matplotlib.lines.Line2D at 0x7fbf9c52b208>]
```



I preprocessed the data using Lambda layer and cropped top 55 pixel rows and bottom 20 pixel rows to make the model see what's important to focus upon.

I also converted the BGR image to RGB as drive.py operates on RGB image and recorded images were in BGR format. So, this was taken care by me.

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 5 for my model.

As I tried different model, I came to know that the No of epochs we choose depends upon the data set size and also the model we use and it depends on the purpose of the problem.

To the medium size data set that I have used, I had stick with 5 Epochs as more than didn't help me optimize the model.

For lower number of data set, higher number of Epochs seems to work well and vize versa.

I used an adam optimizer so that manually training the learning rate wasn't necessary.