Behavioral Cloning

Writeup Template

By ~Karanj

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

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 or writeup_report.pdf summarizing the results

You're reading it! and here is a link to my project code (ipynb) and model.py

- Important Note :
  - model.py and 'test6-copy4' uses the same code
  - 'test6-copy4' and 'test6-copy5' are successful, running and final project codes.
  - At few places here in documentation I have described 'Cell #' for better reference from 'test6-copy4'

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 overview

1. An appropriate model architecture has been employed.

I tried with 3 different model Architecture:

1. I derived my own small neural network for initial testing purpose
2. Comma.ai steering model
3. NVidia model

I ended up using Comma.ai's model with some modifications, because it was providing me with least training and validation loss.

This one contains 3 convolutional layers, 3 dropout layers, 1 maxpool, 3 Dense, and 5 relu(ELU) layers. Data is normalized in the model using a Keras lambda layer at start of the model.

2. Attempts to reduce overfitting in the model.

The model contains dropout and maxpool layers in order to reduce overfitting.

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.

First I tried to use Adam optimizer using manual parameters and tried to vary the parameters, but I was not able to tune parameters with good precision:

```
adam = Adam(lr=1e-4, beta_1=0.9, beta_2=0.999, epsilon=1e-08, decay=0.0)

optimizer=adam
```

So I switched back to automatic Adam optimizer with just learning rate set manually:

```
optimizer=Adam(lr=1e-4)
```

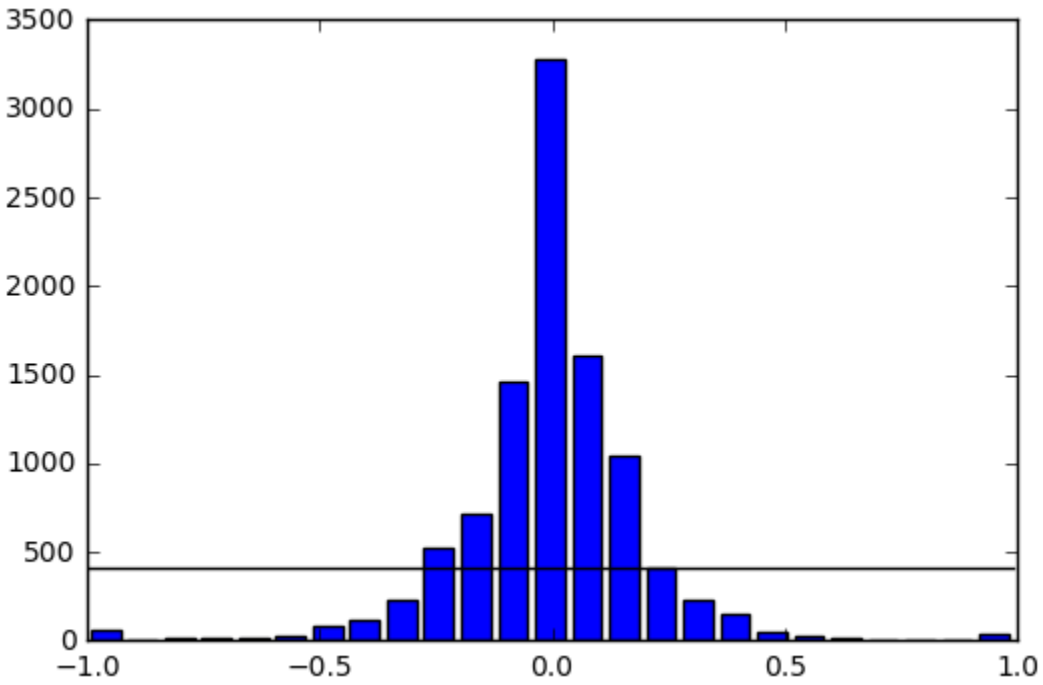
4. Appropriate training data.

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, Reverse track and more recovery data from two points on track where it was getting off the road.

Model Architecture and Training Strategy

1. Creation of the Training Set & Training Process.

- In total after including various recorded dataset (images of track), I have had more than 50000 Images.
    - After filtering out straight images whose camera Angles were between -0.01 and 0.001 using keep_probablitiy of 15%, ended up with 10,000+ images.
    - Divided into 80%:20% as training set and validation set.
- Histogram of Angles(x-axis) vs No. of images(y-axis) looked like this:

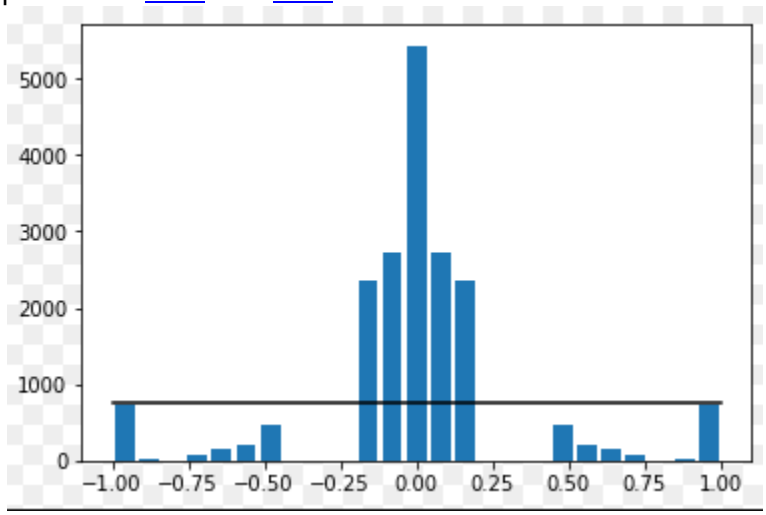


- Whenever Generator (Cell 11) is being called, it calls Augmentation function (Cell 9), which inturn calls respectively:
    - Read_image (Cell 4):
        - Here we randomly select using probability which camera (center/left/right) to use:

```
camera = np.random.choice(['center', 'left', 'right'], p=[0.3,0.35,0.35])
```

        - Set Camera angle correction of around 0.22
            - Add if left camera used, subtract if right camera used.
    - Transform_image (Cell 5):
        - Rotation
        - Translation
        - Shift shear (adjust steering angle a bit)

        - Justification: Rotation, Translation, Shift shear provides modification to images so that to gather images which seems to be taken from a moving object (a car), where viewing angle, visual part of image to camera and tilt may not be in ideal position.
    - Crop_image (cell 6):
        - image cropping top:1/4th of height, bottom: 25px, left:20px, right:20px
        - if "train=true", provide some randomization (adjust steering angle a bit)
        - Resize image to 64x64
    - Flip_image (cell 7):
        - Flip image to balance out number of left turning images with right turning images and also invert the steering angle for flipped images
    - Random_brightness (cell 8):
        - Provide some random brightness to accommodate for random natural environments and different luminous settings.
- Things which didn't worked:
    - I tried to normalize histogram using "def SubSample(samples, a_angles)" function here before throwing images to generator, but didn't work due to the fact that Augmentation of images on the fly (augmentation function) tries to balance out the number of images on each side of histogram (see below image):
    - I also tried to pre-process and augment all the images(not on the fly) and save it to arrays and then use this image array with generator to train model. Again the same problem as of above point. See here and here.



    - Also another failure in the rank, where I did pre-process and augment all the images (not on the fly) and save it to arrays, histogram balancing and then sub sampling. See 3 histograms for more understanding here.

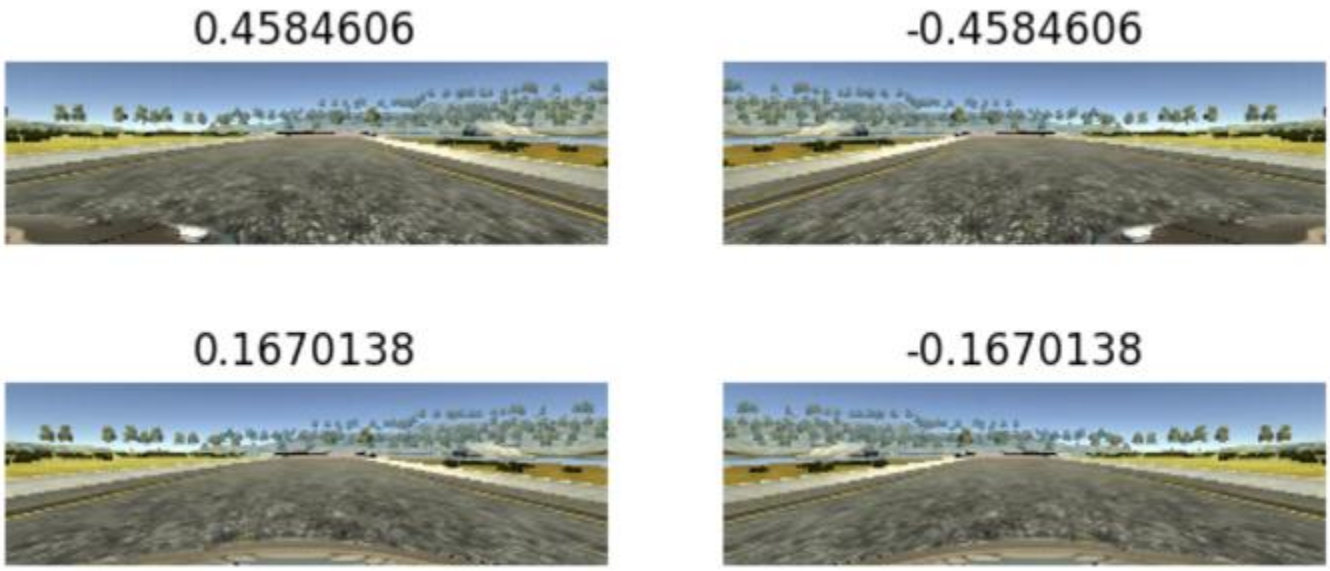
2. Final Model Architecture.

Layer (type)	Output Shape	Param #	Connected to
lambda_1 (Lambda)	(None, 64, 64, 3)	0	lambda_input_1[0][0]
convolution2d_1 (Convolution2D)	(None, 32, 32, 16)	1216	lambda_1[0][0]
elu_1 (ELU)	(None, 32, 32, 16)	0	convolution2d_1[0][0]
convolution2d_2 (Convolution2D)	(None, 15, 15, 32)	4640	elu_1[0][0]
elu_2 (ELU)	(None, 15, 15, 32)	0	convolution2d_2[0][0]
dropout_1 (Dropout)	(None, 15, 15, 32)	0	elu_2[0][0]
maxpooling2d_1 (MaxPooling2D)	(None, 7, 7, 32)	0	dropout_1[0][0]
convolution2d_3 (Convolution2D)	(None, 5, 5, 64)	18496	maxpooling2d_1[0][0]
elu_3 (ELU)	(None, 5, 5, 64)	0	convolution2d_3[0][0]
dropout_2 (Dropout)	(None, 5, 5, 64)	0	elu_3[0][0]
flatten_1 (Flatten)	(None, 1600)	0	dropout_2[0][0]
dense_1 (Dense)	(None, 1024)	1639424	flatten_1[0][0]
dropout_3 (Dropout)	(None, 1024)	0	dense_1[0][0]
elu_4 (ELU)	(None, 1024)	0	dropout_3[0][0]
dense_2 (Dense)	(None, 512)	524800	elu_4[0][0]
elu_5 (ELU)	(None, 512)	0	dense_2[0][0]
dense_3 (Dense)	(None, 1)	513	elu_5[0][0]

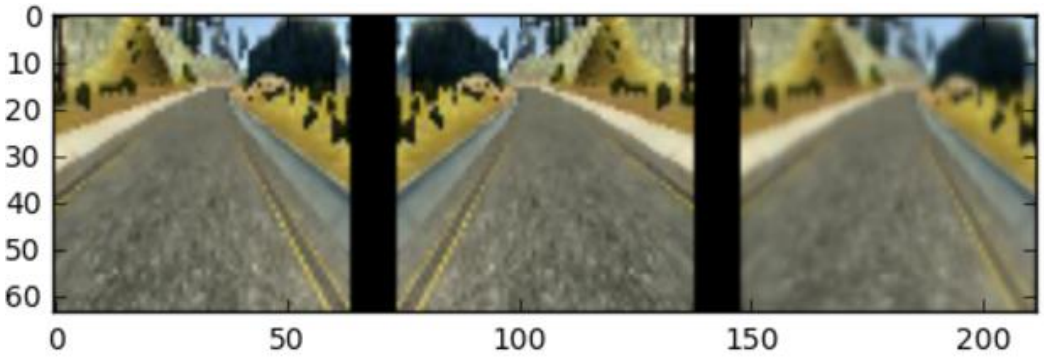
Total params: 2,189,089
Trainable params: 2,189,089
Non-trainable params: 0

3. Creation of the Training Set & Training Process.

- Most part I did cover above in "1. Creation of the Training Set & Training Process."
- When I used pre-process and augment all the images (not on the fly) and save it to arrays, after flipping (random 2 images) from 'test4-Copy1.ipynb':



- Images after all processing and augmentation (original, flipped, transformed) from 'test6-Copy4.ipynb':



- From 'test6-Copy4.ipynb' (model.py uses same code):