P3- Behavioral Cloning using a Convolutional Neural Networks

In this project, CNN is designed to generate steering angle of self-driving car. This CNN learns from training images and driving behavior and predicts the current steering angle as a real valued number. This is more like a regression task. Simulator provides the tracks to test drive the car and extract images, steering angles. The test car, equipped with 3 cameras (one at the center, one at the left and one at the right) provide images from 3 different viewpoints. It is driven on a training track which has sharp corners, exits, bridges, partially missing lane lines. CNN is designed in such a way that it generalizes to unseen data/images rather than memorizing to the test track.

In design phase, I have considering VGG and NVIDIA models for the solution. VGG model had a validation error of 7% whereas NVIDIA model had validation error of 3.4%. Also, NVIDIA model is well documented and well tested. I used NVIDIA model and retrained it with the test images obtained from the simulator runs.

The model starts with a preprocessing layer that takes in images of shape 64x64x3. Each image gets normalized to the range [-0.5,0.5]. Model consists of 5 convolutional layers and 3 fully connected layers. The model looks like as follows:

- Image normalization
- Convolution: 5x5, filter: 24, strides: 2x2, activation: ReLU
- Convolution: 5x5, filter: 36, strides: 2x2, activation: ReLU
- Convolution: 5x5, filter: 48, strides: 2x2, activation: ReLU
- Convolution: 3x3, filter: 64, strides: 1x1, activation: ReLU
- Convolution: 3x3, filter: 64, strides: 1x1, activation: ReLU
- Flatten Layer()
- Drop out (0.3)
- Fully connected: neurons: 100, activation: ReLU
- Fully connected: neurons: 50, activation: ReLU
- Fully connected: neurons: 10, activation: ReLU
- Drop out (0.3)
- Fully connected: neurons: 1 (output)

Following the convolutional layers, Dropout layer is used with a factor of 0.3. Then three fully connected layers with ReLU activations are used. The final layer is a single neuron that provides the predicted steering angle. I used mean squared error for the loss function to measure how close the model predicts to the given steering angle for each image. I used Adam optimizer for optimization in the CNN model.

I tried few experiments such as adding dropouts between convolution layers,using activation function as ELU, inserting one more fully connected error etc. But they didn't improve the validation accuracy much.

Data Preprocessing/Image augmentation

The raw training data was gathered by driving the car as smoothly as possible right in the middle of the road for 3-4 laps in one direction. This training data has two issues. (highly skewed and higly biased towards zero data). Since, most of the turns in training tracks required left angle steering, training data had left-right skew issue. This was resolved by flipping images and steering angles simultaneously for random images in training data. Secondly, most of the time, steering angle during normal driving is small or zero and this resulted in highly biased training data. To fix this, low steering data was randomly removed from the training set in the preprocessing phase. After fixing these issues, It turned out that the number of training samples were small compared to the parameters in the CNN model. Image augmentation was added into the pre-processing phase to generate additional data on the fly during model execution.

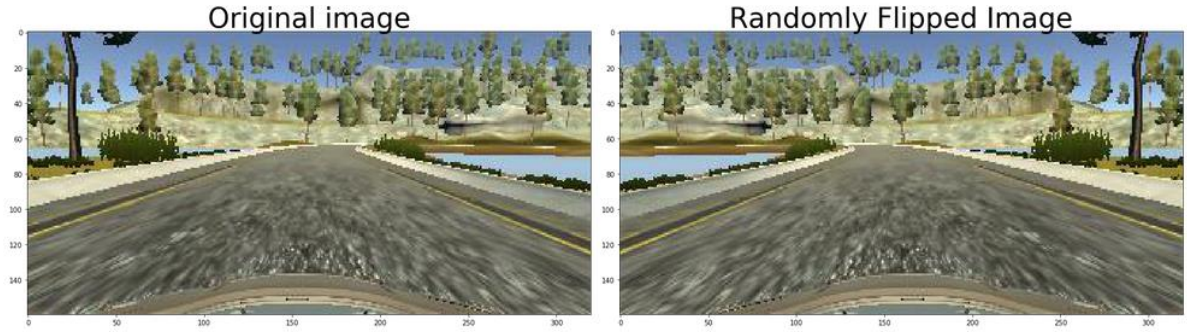
Python generators were used to pre-process and augment the training dataset. These generators process a batch of images at a time and feed them into CNN model for training. Using generators decrease the memory usage of the model in training phase. Using the left/right images is useful to train the recovery driving scenario. The horizontal translation is useful for difficult curve handling (i.e. the one after the bridge).

Here is the list of steps done in preprocessing phase.

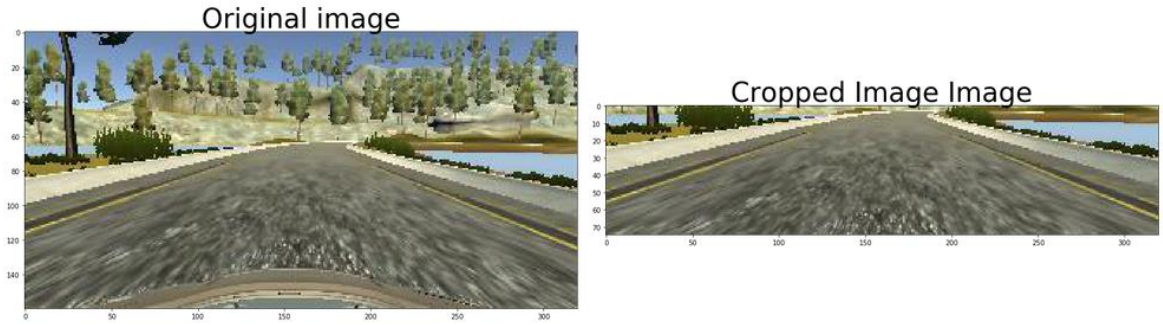
- Randomly choose right, left or center images.
- Crop the image to remove sky and car front parts
- For left image, steering angle is adjusted by +0.20
- For right image, steering angle is adjusted by -0.20
- Randomly flip image left/right
- Randomly translate image horizontally with steering angle adjustment
- Randomly translate image vertically

Sample output of random flipping an image

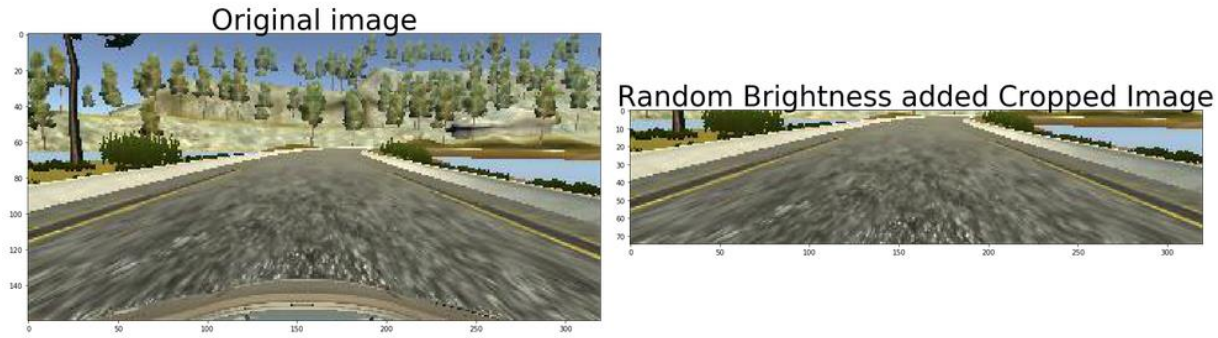
```
orig_img = plt.imread('testimage.jpg')
flipped_img =flip_image(orig_img)

plot_inline_images(orig_img,'Original image',flipped_img,'Randomly Flipped Image')
```



Sample output of image cropping operation to remove sky



Random brightness added to cropped image



Training

Input training data is split into training and validation data sets. (85% for training and 15% for validations). Below is the output of the model on in the training phase t of steps

Epoch 1/5

5184/5184 [==============================] - 161s - loss: 0.0352 - val_loss: 0.0154

Epoch 2/5

5184/5184 [==============================] - 139s - loss: 0.0287 - val_loss: 0.0135

Epoch 3/5

5184/5184 [==============================] - 142s - loss: 0.0258 - val_loss: 0.0125

Epoch 4/5

5184/5184 [==============================] - 145s - loss: 0.0249 - val_loss: 0.0123

Epoch 5/5

5184/5184 [==============================] - 136s - loss: 0.0234 - val_loss: 0.0136

Output

Initial model has fewer sample of zero steering angles and this resulted in car drifting quite a lot. After increasing the number of zero steering angles and reducing the steering shift offset for left/right images from 0.25 to 0.2 resulted in much smoother drive for the car.

With this updated model can drive the course without bumping into the side ways. Attached video in the github is the output of this model when it was tested in autonomous mode in the simulator

References

- NVIDIA model: https://devblogs.nvidia.com/parallelforall/deep-learning-self-driving-cars/
- Udacity Self-Driving Car Simulator: https://github.com/udacity/self-driving-car-sim