Behavioral Cloning

Girish

Project 3 - Behavioral Cloning Write-up

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 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. Speed is set to 25.
- **model.h5** containing a trained convolution neural network
- **writeup_report.md** or writeup_report.pdf summarizing the results
- **video.mp4** video of the trained model running one complete lap on track 1.

2. Submission includes functional code

Using the Udacity provided simulator and my drive.py (only speed is changed to 25) 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 filter sizes and depths between 16 and 64 (model.py lines 159-202)

The model includes ELU layers to introduce nonlinearity (code lines 159-202), and the data is normalized in the model using a Keras lambda layer (code line 163). Since our output is a continous approximation instead of classification/logits, ELU activation function is used.

```
    # Pre-processing
    # Normalize pixel through Lambda layer. pixel range from -0.5 to +0.5
    model.add(Lambda( lambda x: x /255.0 - 0.5, input_shape=(160,320,3) ))

    # Crop the image as mentioned the course video by David Silver. top 75 and bottom 25 rows of pixel
    model.add(Cropping2D( cropping=((75, 25), (0, 0)) ))

    # Convolution layers
    model.add(Convolution2D( 16, 3, 3, subsample=(1,1), W_regularizer=l2(wt_reg)))
    model.add(MaxPooling2D( pool_size=(2, 2)))
    model.add(ELU())
    model.add(Dropout(do_ratio))
    model.add(Convolution2D( 32, 3, 3, subsample=(1,1), W_regularizer=l2(wt_reg)))
    model.add(MaxPooling2D( pool_size=(2, 2)))
    model.add(ELU())
    model.add(Dropout(do_ratio))
    model.add(Convolution2D( 64, 3, 3, subsample=(2,2), W_regularizer=l2(wt_reg)))
    model.add(MaxPooling2D( pool_size=(2, 2)))
    model.add(ELU())
    model.add(Dropout(do_ratio))

    model.add(Flatten())

    model.add(Dense( 100))
    model.add(ELU())
    model.add(Dropout(do_ratio))
    model.add(Dense( 50))
    model.add(ELU())
    model.add(Dropout(do_ratio))
    model.add(Dense( 10))
    model.add(ELU())
    model.add(Dropout(do_ratio))
    model.add(Dense( 1, activation='linear'))
```

2. Attempts to reduce overfitting in the model

The model contains dropout layers in order to reduce overfitting (model.py code lines 171-201).

The model was trained and validated on different data sets to ensure that the model was not overfitting (code line 280-303). The model was tested by running it through the simulator and ensuring that the vehicle could stay on the track.

Complete NVIDIA's end-2-end model for our case seems to overfitt even with just one epoch of training on approximately 40k of images which included augmented images. **Thus to avoid overfitting, extensive dropout layers and few MaxPool layers were added. Also two of five convolution layers were removed.**

3. Model parameter tuning

The model used an adam optimizer, so the learning rate was not tuned manually (model.py line 257).

```
model.compile( loss='mse', optimizer='adam', metrics=['mse'])
```

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. These additional training data included vehicle already on the edges of the road and started recovering correct steering angles.

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 start with one of the proven model NVIDIA's and tune for our use-case.

My first step was to use a convolution neural network model similar to the NVIDIA's as shown in the class by David Silver. I thought this model might be appropriate because **this model also trained for steering angle apart from other controls** of the vehicle.

In order to gauge how well the model was working, I split my image and steering angle data into a training and validation set **in 80:20 ratio.** 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 modified the model by adding dropout and MaxPool layers and removing two of the five Convolution layers, so that gap between training set MSE and validation set MSE are reduced.

Data is normalized using Keras Lambda layer in the beginning instead in the generator function.

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. These spots appeared to be where one side of the road's contour suddenly becomes smooth like road either due to plain ground without grass, shrubs or even uniform blue color lake patch. Spot where tree shade fell on the road also created problem.** To improve the driving behavior in these cases, **I cropped the images' top 75 and bottom 25 rows of the pixels** as shown in class video by David. This made model to ignore non-road scenery. Also **added randomly for 25% of the images shades and for another 25% images reduced the brightness to make model immune from sudden tree shades on the road or change in the brightness.**

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

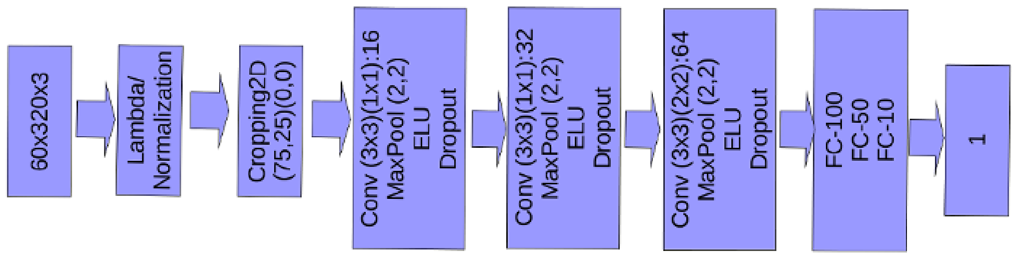
2. Final Model Architecture

The final model architecture (model.py lines 146-204) consisted of a convolution neural network with the following layers and layer sizes.

Layer	Input	Output	Convolutions	Activation
Lambda	160x320x3	160x320x3		
Cropping2D	160x320x3	60x32x3		
Convolution2D	60x32x3	16	(3,3) (1,1)	MaxPooling2D, ELU, Dropout

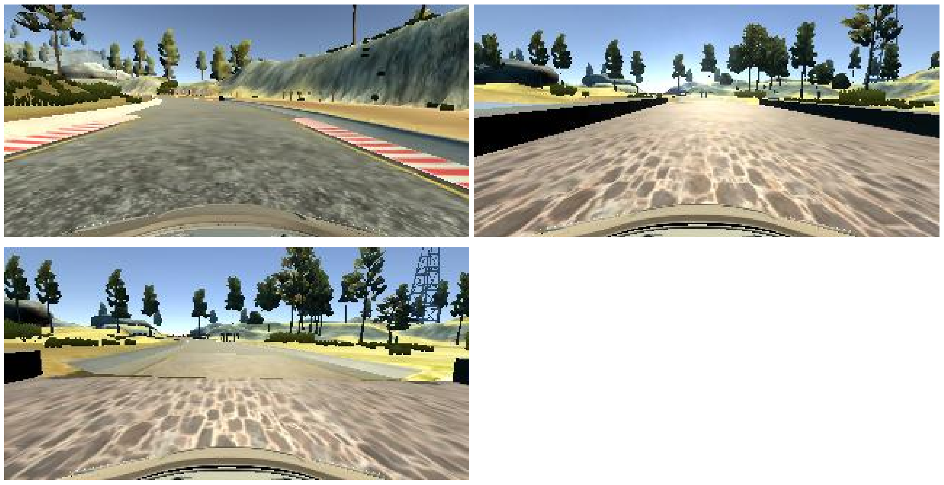
Layer	Input	Output	Convolutions	Activation
Convolution2D		32	(3,3) (1,1)	MaxPooling2D, ELU, Dropout
Convolution2D		64	(3,3) (2,2)	MaxPooling2D, ELU, Dropout
FC/Dense		100		ELU, Dropout
FC/Dense		50		ELU, Dropout
FC/Dense		10		ELU, Dropout
FC/Dense		1		ELU

Here is a visualization of the architecture (note: visualizing the architecture is optional according to the project rubric)


60x320x3
Lambda/ Normalization
Cropping2D (75,25)(0,0)
Conv (3x3)(1x1):16 MaxPool (2,2) ELU Dropout
Conv (3x3)(1x1):32 MaxPool (2,2) ELU Dropout
Conv (3x3)(2x2):64 MaxPool (2,2) ELU Dropout
FC-100 FC-50 FC-10
1

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:



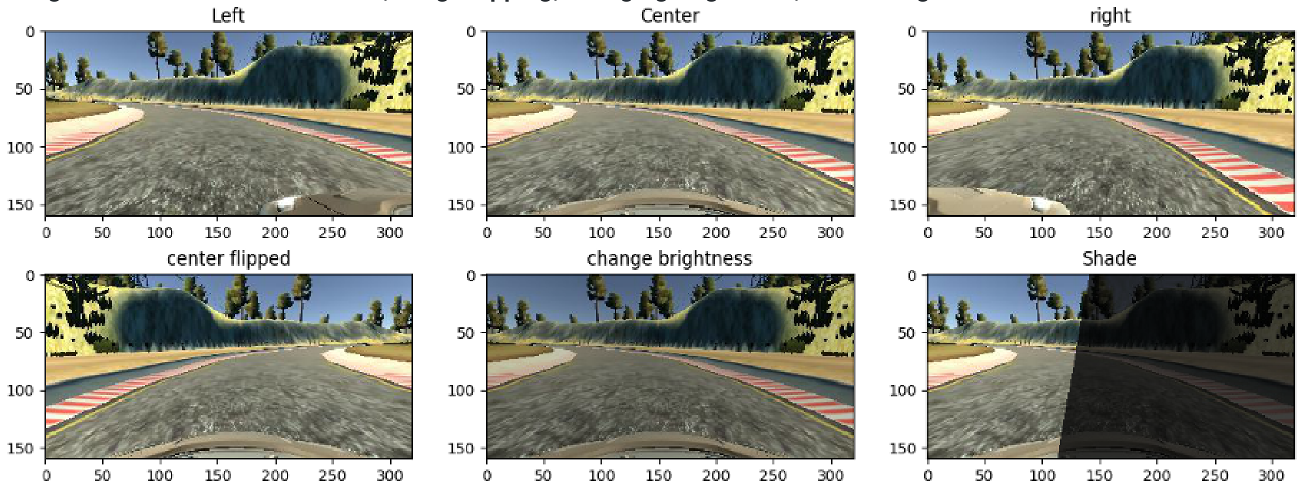
I then recorded the vehicle recovering from the left side and right sides of the road back to center so that the vehicle would learn to recover from edges of the road. These images show what a recovery looks like starting from right most edge of the road to recovering to the center of the road and preparing to turn left.:

Images from the recovery training set:

To augment the data sat, I also used images from left and right cameras apart from the center camera images. I also added additional images by flipping these images and angles thinking that this would help in training model as if additional data is collected by driving clock-wise around the track.

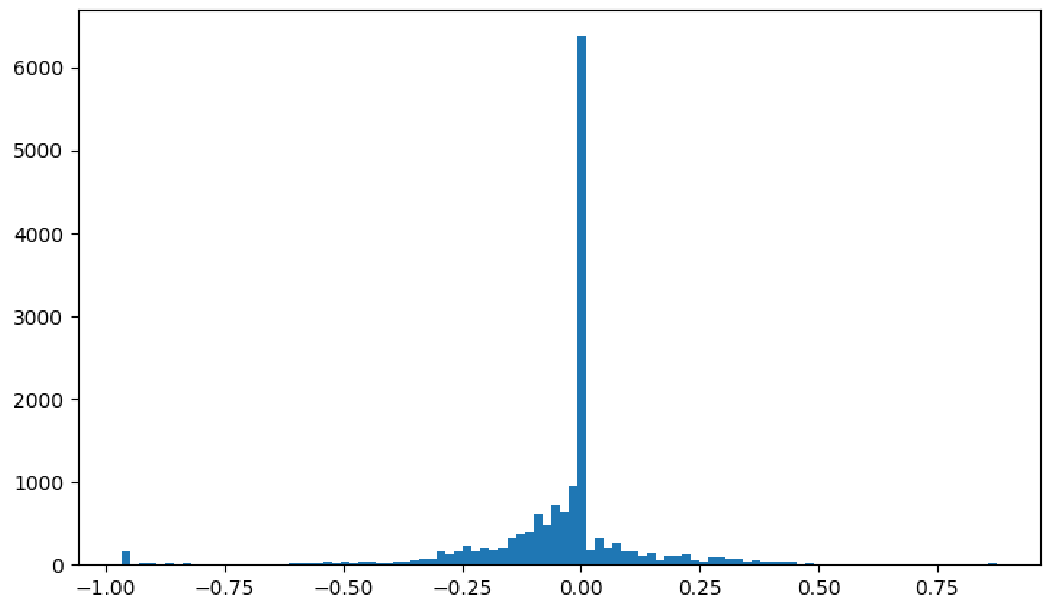
Below are examples of the images from all the three cameras and one of them being flipped, brightness reduced, shade added.

Images from all the three cameras, image flipping, changing brightness, and adding shades:



After the collection process, I had 8,309 (6877 + 1432) number of data points. I then preprocessed this data by reducing images with straight steering angles. Used steering angle distribution and defined any steering angle greater than -0.03 and less than +0.03 as straight steering angles. Almost half of the straight steering angles samples were filtered in ordered to keep even distribution of the samples. After augmentation, close to 40K samples were generated.

Distribution of steering angles in sample data:



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 3 as evidenced by no improvements in training loss and validation accuracy. I used an adam optimizer so that manually training the learning rate wasn't necessary.