Behaviour 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

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 and depths between 24 and 53 (model.py lines 18-24) The model includes RELU layers to introduce nonlinearity and the data is normalized in the model using a Keras lambda layer.

```
from keras.models import Sequential
from keras.layers import Flatten, Dense, Lambda, Conv2D, Dropout

def model(loss='mse', optimizer='adam'):
    model = Sequential()
    model.add(Lambda(lambda x:  (x / 127.5) - 1., input_shape=(70, 160, 3)))
    model.add(Conv2D(filters=24, kernel_size=5, strides=(2, 2),
activation='relu'))
    model.add(Conv2D(filters=36, kernel_size=5, strides=(2, 2),
activation='relu'))
    model.add(Conv2D(filters=48, kernel_size=5, strides=(2, 2),
activation='relu'))
    model.add(Conv2D(filters=64, kernel_size=3, strides=(1, 1),
activation='relu'))
    model.add(Conv2D(filters=64, kernel_size=3, strides=(1, 1),
activation='relu'))

    model.add(Flatten())
    model.add(Dense(100, activation='relu'))
    model.add(Dense(50, activation='relu'))
    model.add(Dense(10, activation='relu'))
    model.add(Dense(1))

    model.compile(loss=loss, optimizer=optimizer)

    return model
```

2. Attempts to reduce overfitting in the model

To prevent overfitting, I used several data augmentation techniques like flipping images horizontally as well as using left and right images to help the model generalize. 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, so the learning rate was not tuned manually.

4. Appropriate training data

Training data was generated using recorded footage of me driving the car in the simulator with 3 cameras recordings from centre, left and right.

Model Architecture and Training Strategy

1. Solution Design Approach

The overall strategy for deriving a good model was to use the Nvidia architecture since it has been proven to be very successful in self-driving car tasks. I would say that this was the easiest part since a lot of other students have found it successful, the architecture was recommended in the lessons and it's adapted for this use case.

In order to gauge how well the model was working, I split my image and steering angle data into a training and validation set. Since I was using data augmentation techniques, the mean squared error was low both on the training and validation steps.

The hardest part was getting the data augmentation to work. I had a working solution early on but because there were a lot of syntax errors and minor omissions, it took a while to piece everything together. One of the problems, for example, was that I was incorrectly applying the data augmentation techniques and until I created a visualization of what the augmented images looked like, I was not able to train a model that drives successfully around the track.

2. Final Model Architecture

The final model architecture code is shown above. Here is a visualization of the architecture:

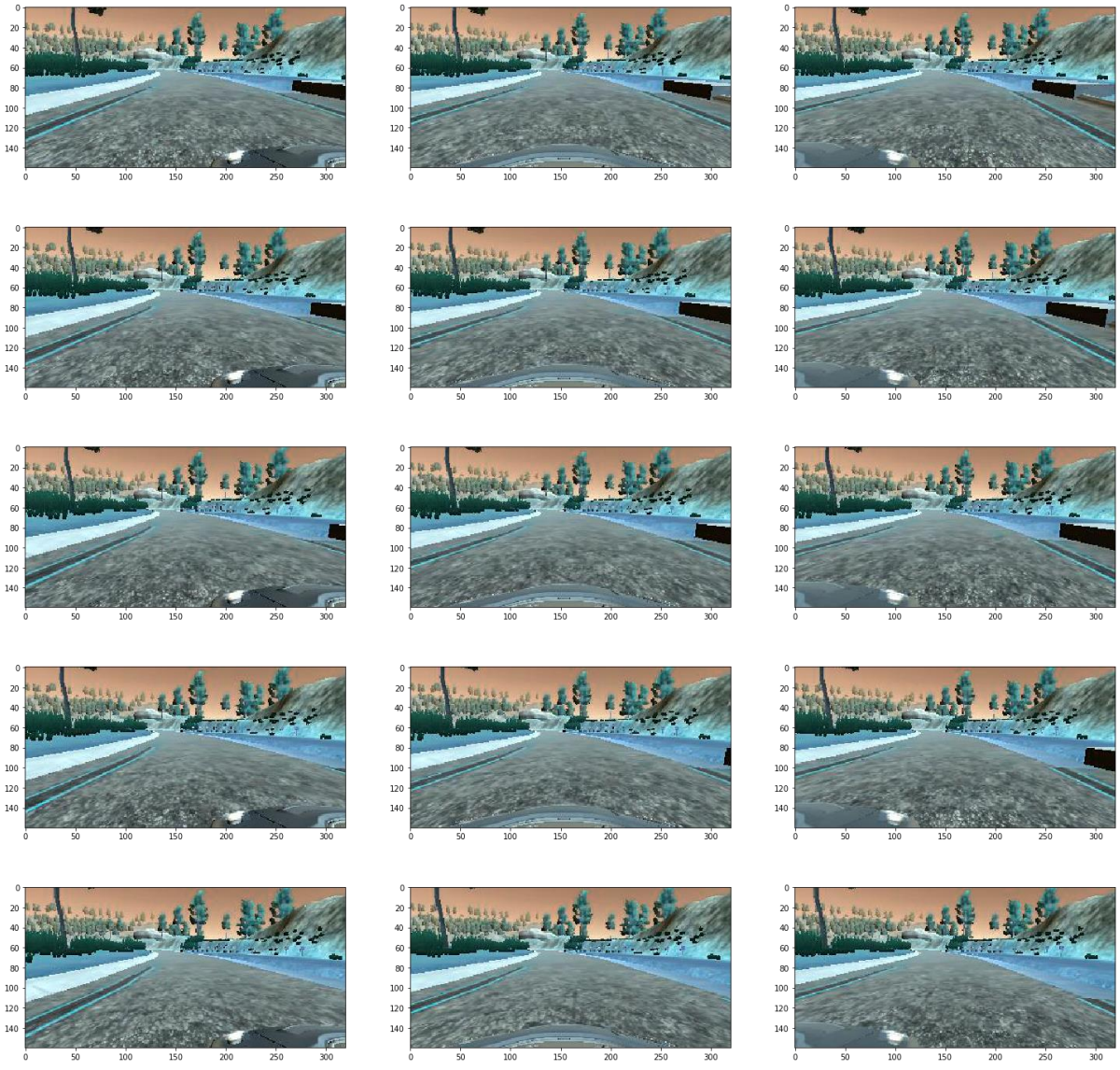
3. Creation of the Training Set & Training Process

To create the training data, I used the Udacity sample data as a base. For each image, normalization would be applied before the image was fed into the network. In my case, a training sample consisted of four images:

1. Center camera image
2. Center camera image flipped horizontally
3. Left camera image
4. Right camera image

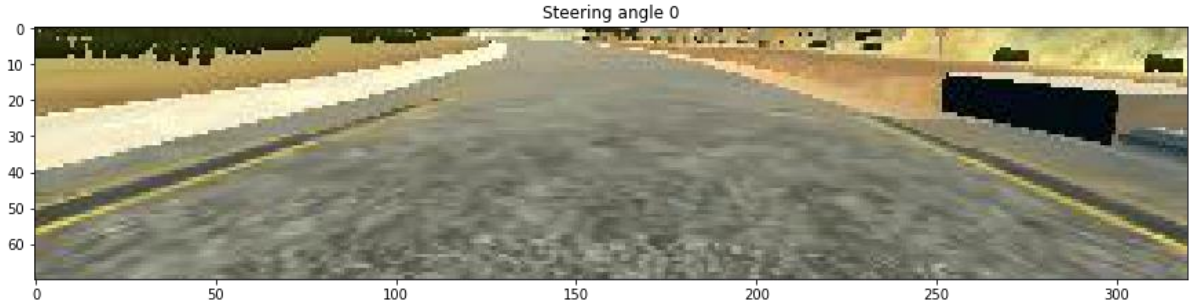
Here are some sample images of raw and preprocessed images:

Raw



Converted to RGB

Cropped



Resized to 160x70

I found that this was sufficient to meet the project requirements for the first track. To make the car run on the same track, additional data augmenation techniques like adding random brightness, shearing and horizontal shifting could be applied.

The provided sample data set (minus the 20% validation set) has 27696 images when the data is augemented. The network was then trained for 5 epochs an AWS `g3.4xlarge` instance for approximately 80 minutes.

```
Epoch 1/5
13848/13848 [==============================] - 988s 71ms/step - loss: 0.0048 -
val_loss: 0.0246
Epoch 2/5
13848/13848 [==============================] - 984s 71ms/step - loss: 0.0049 -
val_loss: 0.0248
Epoch 3/5
13848/13848 [==============================] - 979s 71ms/step - loss: 0.0060 -
val_loss: 0.0184
Epoch 4/5
13848/13848 [==============================] - 940s 68ms/step - loss: 0.0032 -
val_loss: 0.0159
Epoch 5/5
13848/13848 [==============================] - 979s 71ms/step - loss: 0.0028 -
val_loss: 0.0162
```

The model was then tested on the track to ensure that the model was performing as expected.