Project 3 Writeup - Behavioral Cloning

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
- model.h5 containing a trained convolutional neural network that drives the car
- output_video.mp4 which contains the video of the car driving in autonomous mode
- writeup_report.md summarizing the results

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

Before training the model, I normalized the data using a Keras lambda layer. Originally, I implemented the NVIDIA architecture. My model consisted of 5 convolutional layers, followed by 5 fully connected layers. In between the layers I incorporated max pooling and dropout. The model also includes RELU layers to introduce nonlinearity.

After countless hours of tweaking the parameters, and not being able to get great results, I resorted to a much simpler architecture. My final architecture consisted of one convolutional layer, one max pooling layer, a dropout layer, a flatten layer and a final dense layer. I found this architecture to be significantly easier to work with due to the reduced number of parameters. I also recognized that ‘ELU’ was much more efficient than “RELU” as an activation function for this project.

2. Attempts to reduce overfitting in the model

In my model, I added a dropout layer to reduce overfitting. I also made sure to train and validate my model on different data sets to ensure there was no overfitting. I also used the model to drive the car, and based on it’s maneuvers, I was able to determine if the model overfit.

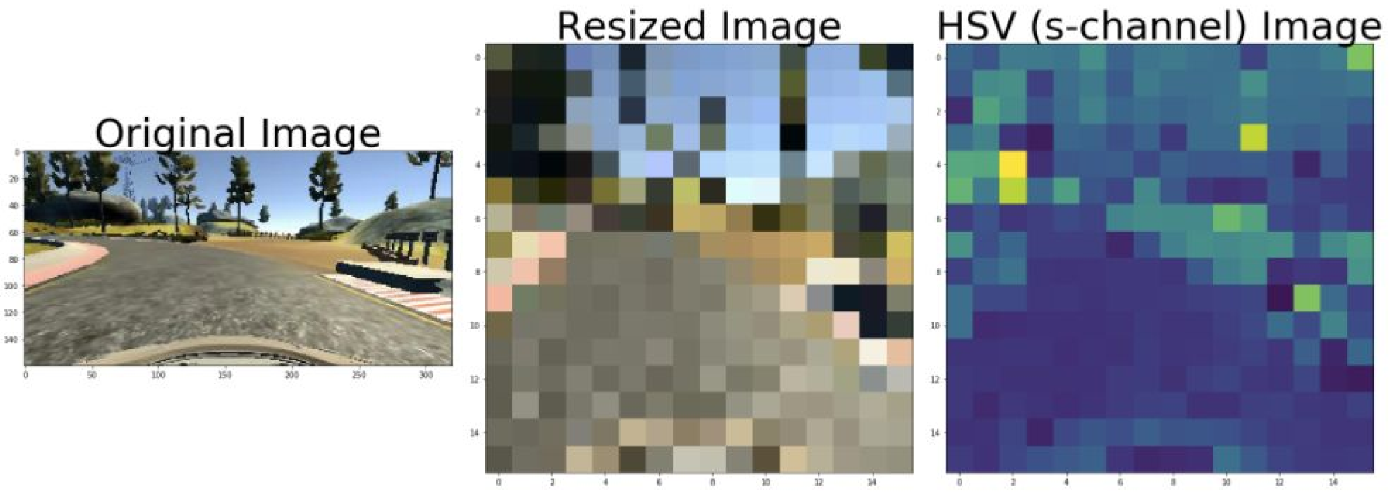
3. Model parameter tuning

The model used an adam optimizer, so the learning rate was not tuned manually. I resized each of my images to 16x16 to significantly reduce training time. I also incorporated a correction factor of .2 to create more data points in the steering data that weren’t zero. I trained on 9 epochs with a batch size of 128. I also changed the color space of the image to HSV, and solely focussed on the s_channel.

4. Appropriate training data

Originally I gathered all the data on my own using the simulator. But after a tremendous amount of struggle, I reverted to the Udacity training data. When I did gather my own data, I used a combination of center lane driving, recovering from the left and right sides of the road, driving both directions on the track, and driving back to the center starting from the sides.

When using the Udacity training data, I used several strategies for augmenting the data. The first thing I did was resize each image to 16x16 pixels. This step was taken mostly to reduce training time. With over 40,000+ images, it took a long time to train the model when each image was 160x320. The next step I took was to change the color space to HSV using the cv2.cvtColor() function. I then focussed solely on the s-channel. See below for an example of both resizing and color space transform.



<u>Model Architecture and Training Strategy</u>

1. Solution Design Approach

I found this project to be overwhelmingly difficult. I tried very hard to implement the NVIDIA architecture with my own gather data, but was incapable of succeeding with this route. I spent time reading through the forum to get a better idea of how my fellow classmates were accomplishing the goal.

The first change I made was deciding to use the Udacity data. It seemed people were getting more consistent results with that data, so I switched. The next thing I did was focus on data augmentation. This seemed to have a big impact on the simulator's ability to drive. I transformed each image into HSV space and focussed solely on the s-channel.

The next change I made was on the image size. I had over 40,000 images, and when I was training my model, it would sometimes take almost 30 minutes just to train the model. My testing process took way too long on the 160x320 images. I made every image in the dataset 16x16, which exponentially reduced training time.

After preprocessing, I split up my data into a test set and a validation set. I then input the data into my model. Overall it was a fairly simple architecture consisting of a Lambda layer for normalization, a convolutional layer, a max pooling layer, a dropout layer, a flatten layer and a fully-connected layer. See below for exact specifications for the model.

To compile the model I used an Adam optimizer and mean squared error for my loss function. After much time spent tweaking all the parameters, my car was successfully able to drive autonomously!

2. Final Model Architecture

The final model architecture was fairly simple. It consisted of a convolution neural network with the following layers:

1. Normalization with Lamba Layer

2. Convolution Layer (3x3x24) + ELU activation
3. Max Pooling Layer (pool size of 4x4 and strides of 4x4)
4. Dropout Layer w/ 0.25 keep
5. Flatten Layer
6. Dense Layer (1)

3. Creation of the Training Set & Training Process

On my first attempt, I captured all of my own training data. To capture good driving behavior, I first recorded two laps on track one using center lane driving going counter clockwise. Then I drove two laps counter clockwise. Next I recorder 1 lap recovering from the sides of the road going clockwise. I did the same thing going counter clockwise. I had 6 laps of recorder data.

After the collection process, I had 15,000+ data points. I then pre processed this data by adding a lambda layer to shuffled the data. I also randomly shuffled the data set and put 20% of the data into a validation set.

After many hours spent working with this data set, and not seeing great results, I reverted back to the Udacity test data. Thankfully, my results became significantly better with the Udacity data, and I was able to use that data to train my model and autonomously drive my car a full lap around the track!

I would like to point out that I did a significant amount of data augmentation before inputting it into my model. I resized each image to 16x16. I changed the color space of each image from RGB to HSV and I focussed on the s-channel. I flipped each image and added it to the total array. This not only allowed the car to steer both right and left, but it also doubled my total dataset.

Thank you for checking out my project. Hope you learned something about behavioral cloning! To see the final video, click here.