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

Files Submitted & Code Quality

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
- writeup_report.md or writeup_report.pdf summarizing the results

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
```

Submission code is usable and readable

The model.py file contains the code for training and saving the convolution neural network.

The first part of the code is the generator used to load the images and create the data augmentation when needed.
The second part of the code consists of the keras model.

Model Architecture and Training Strategy

An appropriate model architecture has been employed

I decided to use the convolutional neural network described in the Nvidia paper linked in this class. I chose this network because I thought that choosing a network that was known to work in real life should still work for this project.

My model consists of 5 convolutional layers with 5x5 filter sizes for the first three and 3x3 filter sizes for the last two, and depths between 32 and 64 (model.py lines 112-128)

The model includes RELU layers to introduce nonlinearity, and the data is normalized in the model using a Keras lambda layer (code line 18).

The data is also cropped to get rid of the background.

I tried an experiment resizing the images by 2. It didn't seem to change much my results. In order to have the model work, I had to reduce the filter size of the last 2 convolutional layers from 3 to 2.

Attempts to reduce overfitting in the model

I made the mistake of not using dropout layers for most of the project. I was surprised to see the validation loss stay roughly around 2% and have the car in the simulator perform a lot worst than when I had a thousands of training data.
At first I thought it was linked to the training set that I had recorded. After starting the data recording all over again and not managing to have the car drive in the center of the road, I realized that I had forgotten dropout layers. I started by adding 2 dropout layers between the first two fully connected layers but it was still not driving as I wanted so I added a third one.

The model was trained and validated on different data sets to ensure that the model was not overfitting (code line 36). The model was tested by running it through the simulator and ensuring that the vehicle could stay on the track.

Model parameter tuning

The model used an adam optimizer, so the learning rate was not tuned manually (model.py line 144). I tuned the batch size to choose something that seemed reasonable for the GPU I have on my computer (4Gbs memory).

I installed this GPU specifically for this project and I didn't regret it. For more than 50000 images, each epoch was taking ~2min.

I played a little bit with the epoch to see how much of a difference it would make to train the data with a small number of iterations (epoch = 3) vs a larger number (epoch = 10). Very quickly, after epoch #3 the validation loss wasn't improving at all so I decided to keep the number of epoch to 3.

Appropriate training data

Training data was chosen to keep the vehicle driving on the road. I mostly data from when the car was in the center of the road. I collected my data in an incremental fashion, trying to add to the dataset only the turns or the flat portion of the road that my current model needed. I also added some 'recovery' situations where the car was starting close to the shoulders or was starting at a weird angle.

I used the following as part of my training data:
- driving the track on the other direction
- using flipped images
- using right and left cameras

I started using a joystick to drive the car around. The major difference I saw between keyboard and joystick was the smoothness of the turns.

For details about how I created the training data, see the next section.

Model Architecture and Training Strategy

Solution Design Approach

My first step was to implement the Nvidia convolutional network. Thanks to keras, it is very easy to write the different layers and train the model. It took me less than a day to have the model up and running in a decent way.

I started using the data provided by Udacity to check the performance of the model. When I tried to run it, the car wasn't moving. I realized that I needed to convert the image to YUV space as it is what I used for my model. When I tried the experiment of resizing the training data, I also had to resize the image given by the simulator in order to run the model.

All of the training data I had was split into a training and a validation set (80-20%). I also used the flipped image to add to the training data set.

I started to incrementally add training data until I reached 20000. The output had improved but it wasn’t working well enough.

I decided to add the right and left cameras to my training set only. I added the portion of code in the generator to change my X_train vector.
I chose the bias by assuming that ~10deg of difference in the steering angle from the left and right cameras would be sufficient to make an impact and improve the model. I converted those 10 deg in radians, and that’s how I got my bias of 0.17.
For fun I also derived the actual steering angle for the right camera (the calculation being the seme for the left one). The calculation is provided in the appendix.

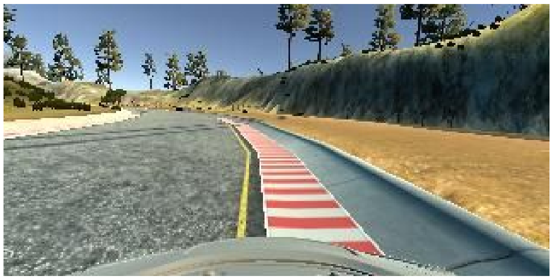
I trained the neural network multiple times and started to get discouraged as the car was not staying in the middle of the road when I realized I had missed some dropout layers.

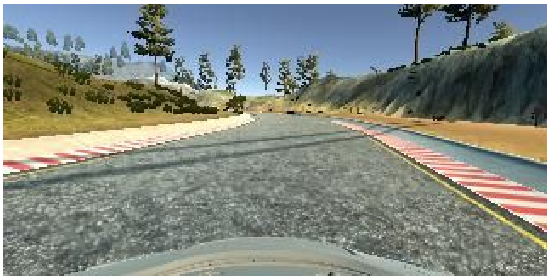
After adding the dropout layers, the vehicle is able to drive autonomously around the track without leaving the road. I ended up using 70784 images for my training set (actual size of the data collected was 17696, the factor 4 comes from the data augmentation).

Creation of the Training Set & Training Process

To capture good driving behavior, I started by recording my data incrementally, adding images to the data set when the car was going off road in some specific portion of the track.

To make the model robust, 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 center itself if veering too much on either sides. These images show what a recovery looks like starting from the right side:

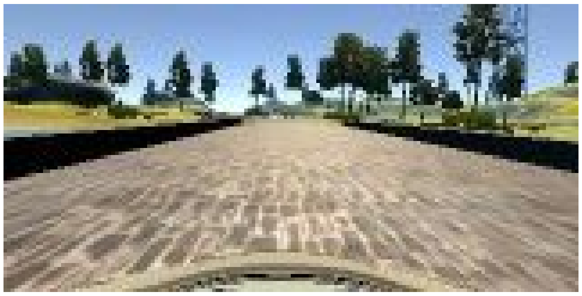


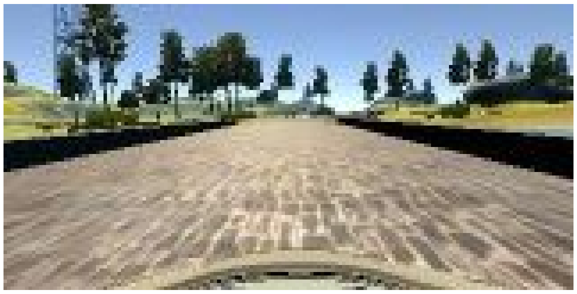


I made the choice to not use track 2 as for most of the project I didn’t have a joystick and it was really hard to drive on track 2 with a keyboard input.

To augment the data sat, I also flipped images and angles thinking that this would account for the fact that we are mostly steering left on this track.
For example, here is an image that has then been flipped:

Original image	Flipped image





After the collection process, I had 70784 images for my training set (actual size of the data collected was 17696, the factor 4 comes from the data augmentation).

The preprocessing of this data consisted of:

- Transforming the images from the GBR color space to YUV
- Applying a lambda layer on the keras model to normalize the pixel values

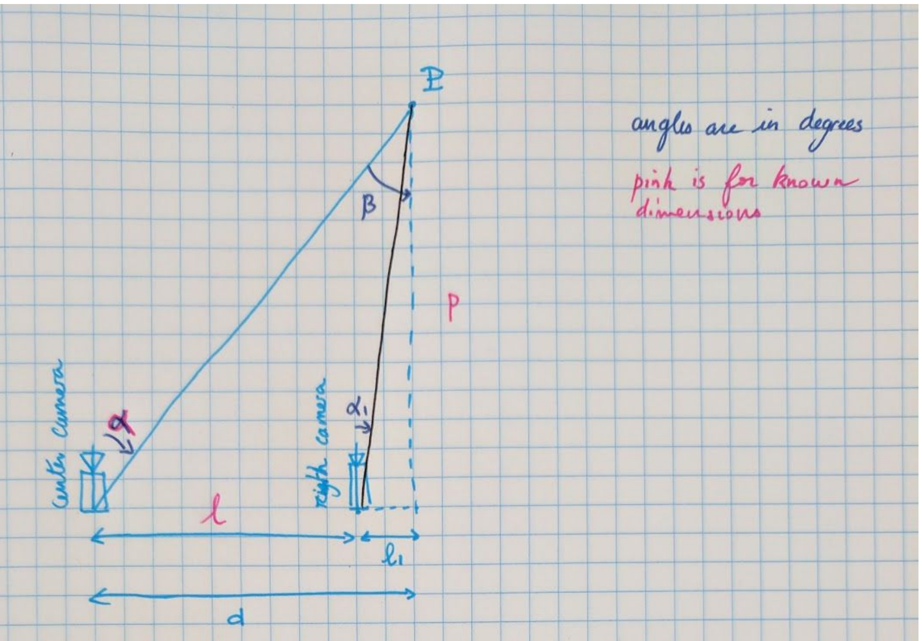
When preparing the training set, the first step is to separate a chunk of data for the training set and the other chunk for the validation (20% of the total set available).
The generator is then creating an array 4 times the size of the input training data (that is loaded by batches). When yielding the data, as the X_train array is built in such a way that all the flipped images and all the right and left cameras images are grouped together, I shuffled the data to mix everything and not bias the model.

I used an adam optimizer so that manually training the learning rate wasn't necessary.

Ideas to go further

- I would have liked to train a network to output both steering angle and car speed. It could have helped the car negotiate the sharp turns a better by learning to slow down.

Appendix: Mathematical calculation of the right camera steering angle knowing the center camera and the distance between the two cameras.



angles are in degrees

pink is for known dimensions

(1) $\tan(90-\alpha) = \frac{p}{d} \Rightarrow d = \frac{p}{\tan(90-\alpha)}$

(2) $l_1 = d - l$

(3) $\tan(90-\alpha_1) = \frac{p}{l_1} \Rightarrow \arctan\left(\frac{p}{l_1}\right) = 90-\alpha_1$

$\Rightarrow \alpha_1 = 90 - \arctan\left(\frac{p}{l_1}\right)$

$\Rightarrow \alpha_1 = 90 - \arctan\left(\frac{p}{\frac{p}{\tan(90-\alpha)} - l}\right)$

for $l = 30$ cm
$p = 10$ m
$\alpha = 10°$

$\Rightarrow \alpha_1 = 8°$

$\| \quad \alpha_1 = 90 - \arctan\left(\frac{p}{\frac{p}{\cot(\alpha)} - l}\right)$