Behavioral Cloning Project

Samuel Navarro

June 21, 2019

Contents

1 Model Architecture and Training Strategy

1.1 An appropriate model architecture has been employed

My model is similar to the one used in NVIDIA. It consists of a convolutional neural network with 5x5 kernel size for the 3 first layers. Here I used a depth between 24 and 48. After that, I used two more convolutional layers with 3x3 of kernel size and 64 number of filters (depth). Then I used 5 more fully connected layers. With Dropout of 0.5 after the second fully connected layer and 0.25 after the fourth fully connected layer. This is in `model.py` in lines 70-84.

The model include RELU activations to introduce nonlinearity and the data is normalized in the model using a Keras lambda layer. The data is also Cropped to reduce noise. I cropped 60 pixels from above the image and 25 from below. I didn't cropped pixels from the sides.

The reason why I used those values was to simply cut off the sky in the image.

1.2 Attempts to reduce overfitting the model

The model contains dropout layers with `p=0.5` after the second fully connected layer and `p=0.25` after the fourth fully connected layer. (lines 80 and 83 in `model.py`)

1.3 Model parameter tuning

The model used an Adam optimizer so the learning rate was not tuned manually. (line 87 in `model.py`)

1.4 Appropriate Training Data

Training data was chosen from the simulator. I repeatedly play the simulator in order to obtain data with good quality and from different angles. I play the simulator several times in order to be confortable with it so I can obtain data with the car in the center of the road as long as possible.

The data was collected in 3 ways:

1. Driving Clockwise on both tracks.
2. Driving Counter-Clockwise on both tracks.
3. Again in segments when the model find difficult to keep on the middle of the road.

To perform the third point, I played the simulator in Autonomous mode and when I found that the model was having trouble to stay on track, I played the simulator in Training Mode just in that particular segment.

Here's an example image of center lane driving:

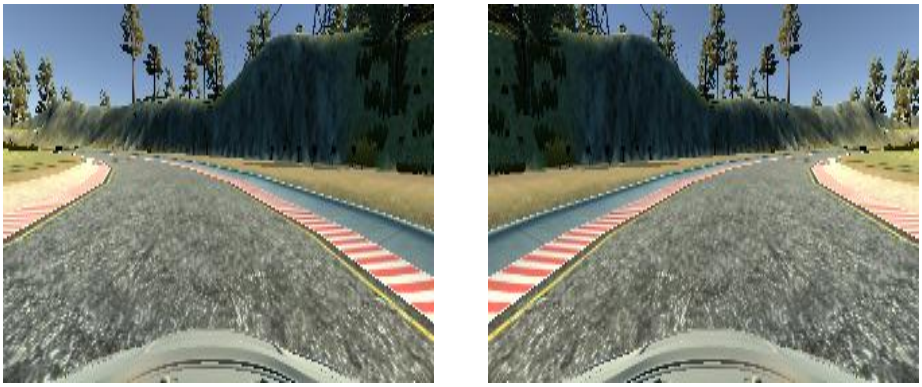
Figure 1: Center Example

Another thing I used to get more data was to flip the images.

The Figure 2 shows the original and the flipped image.

1.5 Final Model Architecture

I decided to start with the model used in NVIDIA, simply because that had been tested in the past. At the beginning, my model doing good in the training set but it had a poor performance with the validations samples so



(a) Original (b) Flipped

Figure 2: Data Augmentation

I added the two dropout layers. After that the model started to perform almost the same in the two samples.

The data was splited in three sets:

- Training set: 80 % of the data
- Validation set: 18 % of the data
- Test set: 2 % of the data

The number of epochs was 8. This seemed obvious after we take a look at the validation and training loss plots in the Figure 3

With this plot we can see that is likely to overfitt the training data if we continue to train.

The final model architecture (lines 71-85 of `model.py`) was mentioned in the Section 1.1

The visualization of the architecture is in Figure 4

1.6 Test set

The best model was taken from the checkpoint (line 90 `model.py`) and we picked the model with the lowest **val_loss**.

The result of this model was **val_loss: 0.0492**. The loss in the **test set was 0.0489.**

But, I believed that driving the car in **autonomous** mode was kind of cheating because the model had already seen the images. Because of that, I tried to drive the car in another Roadway apart from the jungle and the lake tracks that you guys provide us.

I took This one from the unity asset store.

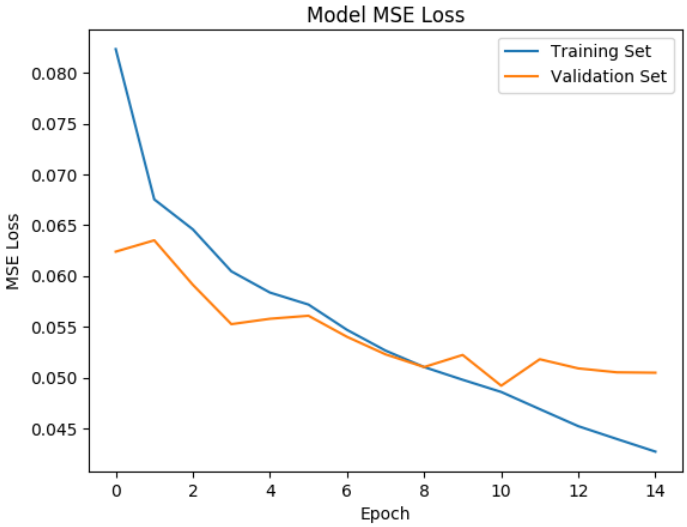


Figure 3: Loss

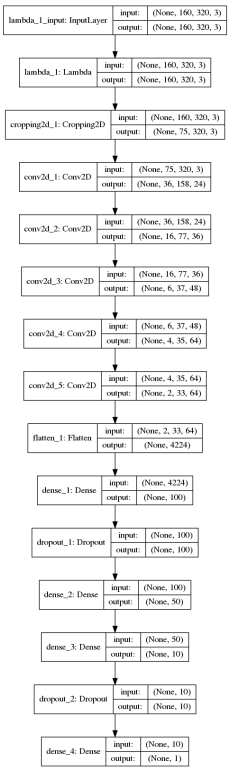


Figure 4: Model Plot

When I tested the model in this new environment I was impressed for good the model was performing.