Behavioral Cloning Project

The project is the coursework of the Udacity CarND Behavioral Cloning Project, more information of the coursework and dependencies are in the https://github.com/udacity/CarND-Behavioral-Cloning-P3
And the link of Self-Driving Car Simulator is https://github.com/udacity/self-driving-car-sim

coursework video 1: https://www.youtube.com/watch?v=pM4m96XhH_0&feature=youtu.be
coursework video 2: https://www.youtube.com/watch?v=j0wOLHm7TrU&feature=youtu.be

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 convolution neural network.
- writeup_report.pdf summarizing the results.

Using simulator provided by Udacity and my drive.py file, the car can be driven autonomously around the track by executing: **python drive.py model.h5**

Steps of the project:
- Collect video data of driving behavior by using simulator
- Preprocess the data
- Choose a CNN model and create it by Keras
- Train and validate the model
- Verify if the model predicts correct angles during autonomous driving.

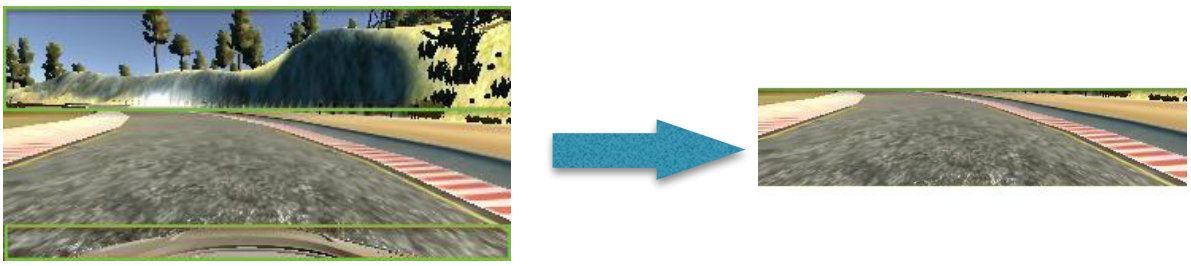
Data collection
The data is very important for the project, I followed the suggestion that general guidelines for data collection in Udacity course, so my strategy is:
- collect 3 laps of center lane driving, of which 2 laps is counterclockwise, of which 1 lap is clockwise, it is all, the number of images is 14313, it includes the images of left, center and right camera.

Data preprocessing
The first step:
For the more quickly convergent of model, normalized and mean centered the pixels, following the suggestion from Udacity course:

Lambda(lambda x: x/255.0 - 0.5, input_shape = (160, 320, 3))

In Keras, lambda layers can be used to create arbitrary functions that operate on each image as it passes through the layer.

The second step:
Reviewed the some of images, I found that the cameras in the simulator capture 160 pixel by 320 pixel images, and not all of these pixels contain useful information, however. In the image, the top portion of the image captures trees and hills and sky, and the bottom portion of the image captures the hood of the car, for example:

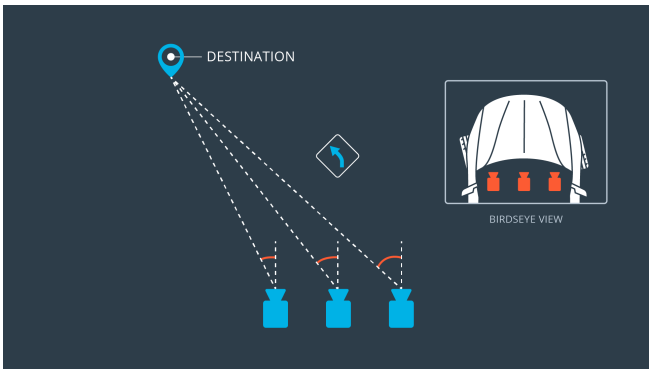


so I finally rescaled all images from dataset from 160x320 pixels to 60x320, use Cropping2D Layer to crop the images:

Cropping2D(cropping=((75, 25), (0,0)))

75 rows pixels from the top of the image
25 rows pixels from the bottom of the image
0 columns of pixels from the left of the image
0 columns of pixels from the right of the image

The third step:
Thinking how to handle the problem of multiple cameras work, the image below gives a sense for how multiple cameras are used to train a self-driving car. This image shows a bird's-eye perspective of the car. The driver is moving forward but wants to turn towards a destination on the left:



From the perspective of the left camera, the steering angle would be less than the steering angle from the center camera. From the right camera's perspective, the steering angle would be larger than the angle from the center camera, if I want to use the images of left and right camera, I have to handle the steering angle.

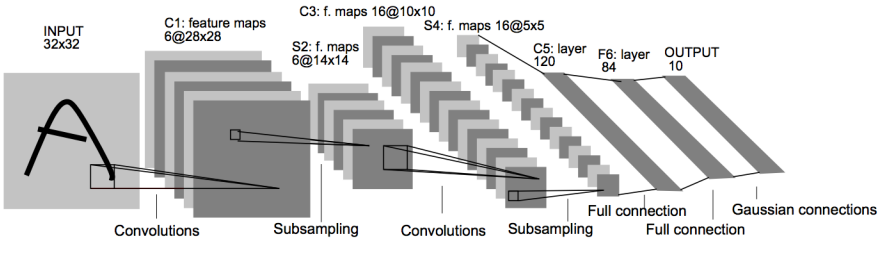
When recording, the simulator will simultaneously save an image for the left, center and right cameras. Each row of driving_log.csv file, contains the file path for each camera as well as information about the steering measurement, throttle, brake and speed of the vehicle.

For the problem, I used the correction parameter recommended by Udacity to solve it, the below:

```
 # create adjusted steering measurements for the side camera images
correction = 0.2 # this is a parameter to tune
steering_left = steering_center + correction
steering_right = steering_center - correction
```

Model Architecture

For the Behavioral Cloning project, I tested 2 models: LeNet-5 model and model reported in Nvidia paper for end-to-end learning for self-driving cars.
After testing models, adding/removing some layers I observed that the LeNet-5 model works better in general, and the more quickly and less capacity than Nvidia model.



Model Tuning

To prevent the overfitting, I chose the simpler model that LeNet-5, and captured the more images(3 laps), but the model still hit it. The most visually, the result is that the vehicle crash into the base after passed the bridge, the below:

Based on the above situation, I think that I should add one dropout layer in the LeNet-5 model, and where is added it? what number is the rate?

Followed the experience, I add the dropout before the Flatten layer.
After some times tested, with binary search algorithm, I am sure that the range [0.3, 0.5] of rate is good, the final I picked 0.45.

Final Model Architecture

LeNet-5 Model for the Project

image normalization - using Keras Lambda layer
image cropping -using Keras Cropping2D layer
convolution: 5 * 5, filters: 6, activation: RELU
max pooling operation, pool_size=(2, 2), padding='valid'
convolution: 5 * 5, filters: 6, activation: RELU
max pooling operation, pool_size=(2, 2), padding='valid'
Dropout(0.45)
Full connected, Dense(120)
Full connected, Dense(84)
Full connected, Dense(1)

Testing and final result

Finally, I used the LeNet-5 architecture and Dropout(rate=0.45) to get the model, the result of training and validation(0.2):
Epoch 1/5

loss: 0.0983 - val_loss: 0.0467
Epoch 2/5
loss: 0.0449 - val_loss: 0.0437
Epoch 3/5
loss: 0.0419 - val_loss: 0.0410
Epoch 4/5
loss: 0.0399 - val_loss: 0.0478
Epoch 5/5
loss: 0.0389 - val_loss: 0.0436

From the above information, we can found that there are still the overfitting, but from run the Udacity simulator in autonomous mode, it's good, the vehicle can run two circles in the first track between 9 speed and 20 speed.

Possible improvements

After ran the Udacity simulator in autonomous mode, the model(model.h5) can support the vehicle to run two laps and more, but there is a problem disturbing me, the stability of model that is generated by the LeNet-5 architecture every time is different, it is not always steady, which use the same architecture and same hyper-parameter to generate the model, but the model is not always well.
The issue is troubling me several days, so I do some experiments that tune the hyper-parameter or add/remove some layer, even use the different architecture from Nvidia paper, but it is still in the here.
So, I guess, the floating training dataset is the cause of the issue, in every training the model, 20% fraction of the training data to be used as validation data, in which maybe exist the some key data that maybe curve for training model, meanwhile the model is missing them, so it is not always steady, need more time to verify it.