Behavioral Cloning Project

Introduction

The objective of this project is to teach the computer to drive a car based on the data collected in the supplied simulator from Udacity. Here we apply the concepts of deep learning and convolutional neural networks to teach the computer to drive independently.

We feed the data collected from Simulator to our model, this data is fed in the form of images captured by 3 dashboard cams center, left and right. The output data contains a file data.csv which has the mappings of center, left and right images and the corresponding steering angle, throttle, brake and speed.

Using Keras Deep learning framework we can create a model.h5 file which we can test later on simulator with the command "python drive.py model.h5". This drive.py connects your model to simulator. The challenge in this project is to collect all sorts of training data so as to train the model to respond correctly in any type of situation.

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

1. Submission includes all required files and can be used to run the simulator in autonomous mode

My project includes the following files:

- new_model.py containing the script to create and train the model
- drive.py for driving the car in autonomous mode
- modelNvidia.h5 containing a trained convolution neural network
- writeup_report.md or writeup_report.pdf summarizing the results

2. Submission includes functional code

Using the simulator provided by Udacity, drive.py and my model.h5 file which is the model of the neural network, the car can be driven autonomously around the track by running.

```
$ python drive.py modelNvidia.h5
```

3. Submission code is usable and readable

The new_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 using the Nvidia model seen in progress I made test games with the Lenet model which did not give good results. This LeNet model whose architecture you can see by following the link below could not keep the car on the road and therefore did not meet the objectives of the project. This LeNet model is a convolutional neural network with filter sizes 5 * 5 and depths of 6 (lines new_model.py 107 - 121) and it also contains Relu layers for the nonlinear. The model data is primarily normalized in the model using a Lambda Keras layer (linen 111).

Sources:
-https://classroom.udacity.com/nanodegrees/nd013/parts/168c60f1-cc92-450a-a91b-e427c326e6a7/modules/6b6c37bc-13a5-47c7-88ed-eb1fce9789a0/lessons/fe16d91d-bc00-455b-bc7b-a61b5e55d473/concepts/badd25bd-e30d-4b07-a4ee-bca38280c067

-https://medium.com/analytics-vidhya/cnns-architectures-lenet-alexnet-vgg-googlenet-resnet-and-more-666091488df5

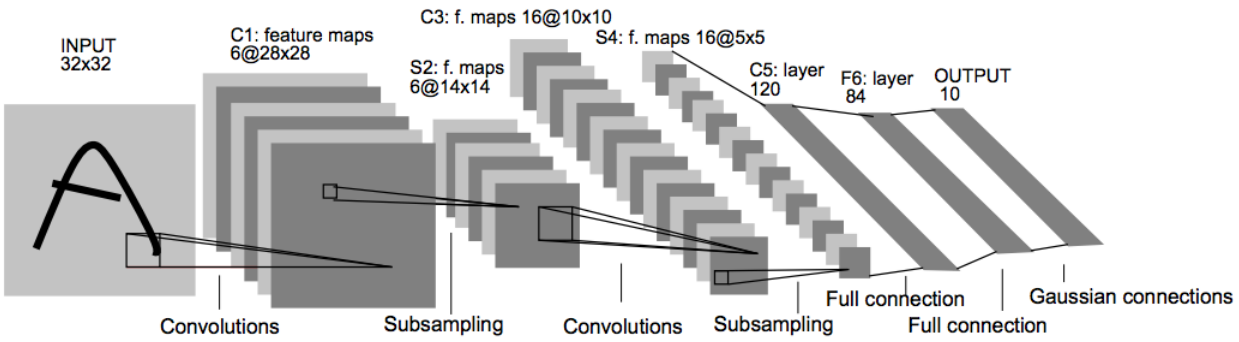
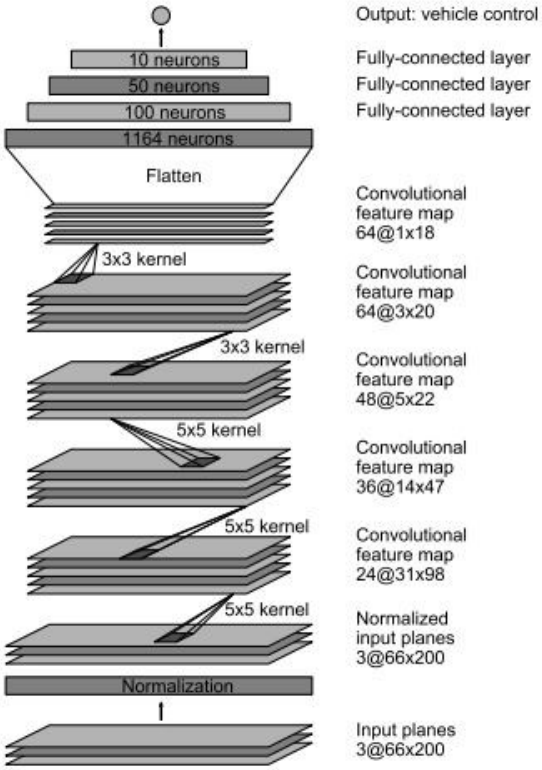


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

- The model used in a second step for this project is the one provided by NVIDIA. The architecture of the model is described by NVIDIA is described by the diagram below. My model consists of a convolutional neural network with 5 * 5 and 3x3 filter sizes and depths between 24 and 64 (lines new_model.py 91-95).
The model also includes RELU layers to introduce non-linearity and the data is normalized in the model using a Keras lambda layer (code line 89).

Sources:

- https://classroom.udacity.com/nanodegrees/nd013/parts/168c60f1-cc92-450a-a91b-e427c326e6a7/modules/6b6c37bc-13a5-47c7-88ed-eb1fce9789a0/lessons/3fc8dd70-23b3-4f49-86eb-a8707f71f8dd/concepts/7f68e171-cf87-40d2-adeb-61ae99fe56f5
- https://images.nvidia.com/content/tegra/automotive/images/2016/solutions/pdf/end-to-end-dl-using-px.pdf



2. Attempts to reduce overfitting in the model

The introduction of the suppression layers in the model allowed to reduce the overfitting and guaranteed us low squared errors in training and validation. After training the model, it was validated on different datasets to ensure the model was not overfitted (code lines 142-148). To make sure everything was good or looks correct I ran the model through the simulator and observed if the vehicle would stay on the track.

3. Model parameter tuning

The model used an adam optimizer, so the learning rate was not tuned manually (new_model.py line 147).

4. Appropriate training data

In order for the vehicle to stay on the road when driving, we needed a collect strategy in the simulator, this was given in one of the sections of the project (see source).

Source:

- https://classroom.udacity.com/nanodegrees/nd013/parts/168c60f1-cc92-450a-a91b-e427c326e6a7/modules/6b6c37bc-13a5-47c7-88ed-eb1fce9789a0/lessons/3fc8dd70-23b3-4f49-86eb-a8707f71f8dd/concepts/b0034fd8-66a9-42b0-bbeb-26604b948817

In order for the vehicle to stay on the road when driving, we needed a collect strategy in the simulator, this was given in one of the sections of the project (see source). For the collet of the data I used a driving combination focused on the center lane (2 laps), a recovery lap on the sides but I captured only the images on the curves (turns), another lap but in counterclockwise and finally a turn on track 2 so that the model is not specific to track 1 only.

Model Architecture and Training Strategy

1. Solution Design Approach

The overall strategy to derive a model architecture was to go through the transfer learning seen in the course. This consists of taking a pre-trained neural network (in my case I chose the one from Nvidia) and adapting it to a new data set that is different depending on the size and the simulitude of the new data set.

My first step as I said was to use a Convolutional Neural Network (Lenet) model. I wasn't expecting the model to work flawlessly but believed that with a little bit of modification (adding convolution layer, pooling layer, or dropout layers) I would get there to train a model . But the squared error of the training and validation set data were too high and in the simulator the verification of the model confirms it to us when exiting the track. Then I decided to use Nvidia's neural network architecture for autonomous cars.

I found that my first model had a low root mean square error on the training set, but a high mean square error on the validation set. This implied that the model was over-adjusted.

To combat overfitting, I modified the model by adding layers of abadons after the fully connected layers of the model.

The last step was to run the simulator to see how hard the car was going on the first track. There were a few spots where the vehicle fell off the track (particularly in the turns). To improve the driving behavior in these cases, I tried to increase the number of epochs (from 1 to 5) because at the beginning I had done it with only one epoch.

At the end of the process, the vehicle is able to drive autonomously on the track without leaving the road but in some places it gets too close to the end of the track. I wanted to add layers of max Pooling to overcome this problem but the GPU resources being exhausted I did not have the chance to do it (see code new_model.py).

Sources:

- https://developer.nvidia.com/blog/deep-learning-self-driving-cars/

- https://classroom.udacity.com/nanodegrees/nd013/parts/168c60f1-cc92-450a-a91b-e427c326e6a7/modules/6b6c37bc-13a5-47c7-88ed-eb1fce9789a0/lessons/818a5b8e-44b3-42f9-9921-e0e0e49f104e/concepts/10489223-72fa-4393-848b-f882ba3cf7f9

2. Architecture du modèle final

The final architecture of the model (lines new_model.py 87-108) consists of four convolutional layers followed by an activation function, five fully connected layers separated from each other by a stall. (I wanted to integrate layers of pooling between the convolutional layers but this could not be done "GPU resources exhausted").

Here is a visualization of the architecture:

Layer (type)	Output Shape	Param #
lambda_1 (Lambda)	(None, 160, 320, 3)	0
cropping2d_1 (Cropping2D)	(None, 65, 320, 3)	0
conv2d_1 (Conv2D)	(None, 31, 158, 24)	1824
conv2d_2 (Conv2D)	(None, 14, 77, 36)	21636
conv2d_3 (Conv2D)	(None, 5, 37, 48)	43248
conv2d_4 (Conv2D)	(None, 3, 35, 64)	27712
flatten_1 (Flatten)	(None, 6720)	0
dense_1 (Dense)	(None, 1164)	7823244
dropout_1 (Dropout)	(None, 1164)	0
dense_2 (Dense)	(None, 100)	116500
dropout_2 (Dropout)	(None, 100)	0
dense_3 (Dense)	(None, 50)	5050
dropout_3 (Dropout)	(None, 50)	0
dense_4 (Dense)	(None, 10)	510
dropout_4 (Dropout)	(None, 10)	0
dense_5 (Dense)	(None, 1)	11

Total params: 8,039,735
Trainable params: 8,039,735
Non-trainable params: 0

3. Creation of the Training Set & Training Process

To capture good driving behavior, I first recorded two laps on track one using center lane driving. Here is an example image of center lane driving:

I then recorded the vehicle recovering from the left and right side of the road to the center so the vehicle would learn to steer when it drifts to the left or right side. For this, the images of the left and right camera are supplied to the model as if they came from the central camera. This is done in a function that is in the notebock new_model.py between line 39-67.

One technique suggested in the course to help with left turn bias is to reverse the frames and take the opposite sign of the direction measurement. this party is in a function on line 26 in the new_model.py file.

After the collection process, I had 11208 number of data points (the data collected for this project is in the /opt/ directory). I then preprocessed this data en utilisant la bibliothèque python csv pour lire et stocker les lignes du fichier "driving log.csv", puis pour chaque ligne extraire le chemin vers l'image de la caméra.

In order to assess how the model worked, I divided my image and steering angle data into a training and validation package using the Sklearn library.

- 75% of the information is used for the training set

- 25% of the data is kept for the validation set

I use the generator to generate the data to avoid loading all the images into memory and generating them instead in batches of 32

I used this training data to train the model. The validation set helped determine if the model was over or underfit. I trained the model with 5 epochs and the result is still to be desired, I tried to train the model with 10 or 15 epochs but the time was too much seen that each epoch was executed on an average time of 45 minutes.

When finished I launched the simulator to see how good the car was going and if it stayed on the track.

Output Video

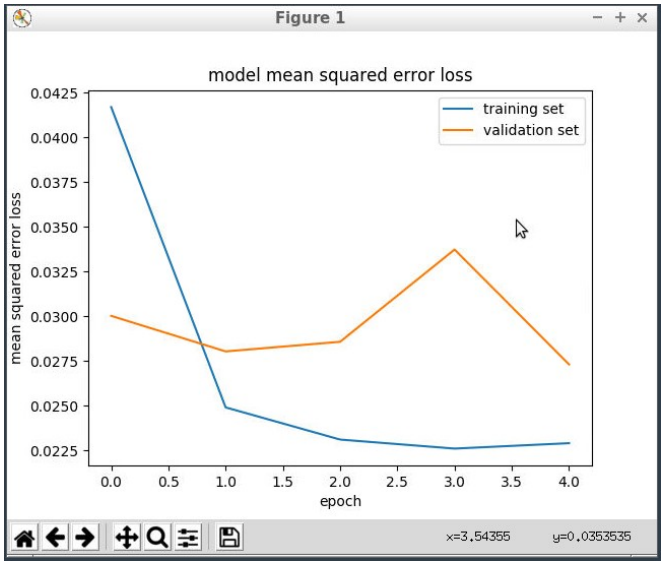
In the workspace you will find a video named "video.mp4" made during the model tests in the simulator thanks to a script provided by Udacity.

Discussions & difficulty:

1. I also wanted to try the transfer learning of some architecture (googleNet and VGG16) to find during some research but the time of the GPU was exhausted and I had to make a request near the support to have hours in addition to be able to complete this project. One of the problem was that every time I wanted to train the model with 5 or 10 epochs the execution was interrupted each time.

2. The sources of the codes used come from the course, you can follow the links which are in the report or in the code for more information (they will refer you to the pages of the courses of udacity).

3. I had somewhat off-standard results plotting the quadration errors of the training and validation dataset, as seen in the figure below the squared error of the validation dataset is assumed to be a decreasing function and not of this form. I would like to know what is the cause of these given that the model works.



4. I did not put enough data from track two reason why the model does not work on this one, in the data collection I collected more data from track 1 than from track 2. For Due to time optimization on the GPU I could not harvest the same data on both tracks. But if there was more data from track two in forming the model, it will be generalized.