# Behavioral Cloning

**Behavioral Cloning Project**

The target is here to train the weights of our network to minimize the mean squarred error between the steering command output by the network and the command of either the human driver.

To make that, we collect firstly the data with a normal drive with a human driver. We will compare these recording data with the data of the cameras of our autonomous drive. The MSE will compare for every situation the desired steering command with the network computed steering command (from our CNN). One Error measurement will be measured and we will adapt the weight in the back propagation step to reduce in maximum this error.

---

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

My project includes the following files:

input for the application "behavior cloning" :

- model.py containing the script to create and train the model
- drive.py for driving the car in autonomous mode
- folder data with the csv file with the "training mode" data recording (features and labels)

---

output/results from the application "behavior cloning" :

- model.h5 containing a trained convolution neural network
- folder named run1 with the image after the "autonomous mode" testing
- run1.mp4 with the results of the SW pipeline for behavior cloning
- writeup_report.md to summarize the results

---

You cand find the differents file in the folder "CarND-Behavioral-CLoning-P3"

You can find in model.py these functionnalities :

- open and read the images implemented in the csv file with the "input data" (from the training set)
- increase the number of data (images) and the angle measurement with the strategies "flipped" (image translation (features: from the 3 cameras) and angle steering (labels)) to decrease the overfitting
- Angle correction implementation for the left and the right cameras (not theoritical calculated, more empirical methodic)
- Format conversion of the image because the format RGB was useful for the routine drive.py
- Slip the training set (80%) of the images and the validation set (20%) of the images
- generator implementation to resolve the problem of data size
- Mini-Batch strategie (32 images) for every step, it's more quick as a standard batch descend gradient or a stochastic gradient descent
- SW pipeline implementation (detailled architecture bellow in the next point)

- Initialization Running and Optimization (HE Initialzation, Mean squarred Error Algo between the steering command output by the network and the command of the human driver)

- I have taken the default values for the Adam optimizer (the learning rate, the beta1, beta2, epsilon are default values)

- Overfitting analyze (dropout implemented in the 2 full connected layers)

- Vizualization for every layers (convolution, polling, activation.....) and parameters used for every layers

You can see a lot of commentar to understand the differentes functionalities implemented in the code

---

## Model Architecture and Training Strategy

### 1. Pipeline

I don't have start from scratch : I used the NVIDIA pipeline. It's more a transfert learning application.

Only the end of the pipeline was changed to decrease the overfitting but no modification in the first layers.

The startegy is always the same, the w and the h must decrease and the depth (d) must incrase step by step.

---

Firstly, 2 preliminary teps are useful :

- the normalization of the images (mean and small variance implementation)

- a cropping step to delete the part of the images not useful four our treatement

---

Secondly,the model consists of a convolution neural network of 9 layers :

- 1 Normalization of all the images to start with the same initial conditions

- 5 convolutional layers (to perform fetaure extraction). The first 3 convolutional layers are a stride 2*2 and a 5*5 kernel and the 2 last convolutional layers are a non-strided (1*1) convolution with a 3*3 kernel

- 3 full connected layers with at the end the vehicle controls as output.

- for every layer, a standard ReLu activation Unit will be used.

---

**2. Attempts to reduce overfitting in the model**

The model contains dropout layers in order to reduce overfitting.

You can find 2 Dropout layer between the FC1 and the FC2 and between the FC2 and the FC3.

20% of the layers FC1 and FC2 ar deleted.

Firstly the L2 regularization was selected to delete the overfitting with a very small lambda (hyperparameters).

But the dropout solution gives better solution as the L2 Regularization.

The model was trained and validated on different data sets to ensure that the model was not overfitting.

The model was tested by running it through the simulator and ensuring that the vehicle could stay on the track.

---

**3. Model parameter tuning**

The model used an adam optimizer.

The learning rate was not tuned manually.

But it was possible to change the default values from the Adam optimizer (the learning rate, the beta1/RMSprop, beta2, epsilon are default values and can be change), but the tuning of these parameters has not increase the accuracy.

That why, I have worked with the adam optimizer with default hyperparameters.

---

#### 4. Appropriate training data

Training data was chosen to keep the vehicle driving on the road. For that, it was very important to increase the number of "input images" from the the training set.

The "flipped" solution was choosen to increase the number of input data : news features informations from the 3 cameras and the new labels (angle steering).

The second step was to split the input data in 2 parts : training set (80% of the images) and validation set (20% of the images), to be sure that we don't have problems of overfitting (big accuracy difference between the training and the validation set).

---

#### 5. Solution Design Approach

As said before, it was not really an "End to End" Learning. It was more a Transfert learning application.

Only two parts was adapted :

- the cropping parameters to delete the informations not interesting in the image for the CNN pipeline (to reduce the numbers of parameters and the time execution)

- the FC1 and FC2 has been adapted (overfitting problem)

To combat the overfitting, I modified the model with 2 doput layers, between the FC1 and FC2 et FC2 and FC3.

After experimentation, 20% deleted was the perfect solution.

After 5 epochs, the difference between the training set result and the validation set results was very small (no problem of overfitting after 5 epochs).

At the end of the process, the vehicle is able to drive autonomously around the track without leaving the road.

---

## Conclusion

For the model architecture vizualization, please generate the code model.py, and you can see :

- the SW pipeline with the differents layers

- The numbers of parameters used for every layer, and for all the model
- The result of the training set and the validation set for every epoch
- 981919 Parameters for all the models
- after 2 epochs : training set : 0,0025 and validation set : 0,0113
- Same results after 5 epochs, the gap between the training set and the validation set is always small (np overfitting problem)