# Behavioral Cloning

---

The goal of this project is to build a neural network based learning model that is able to mimic the human driving behaviour and drive a car on a known track using behavioral cloning technique.

The project includes the following main steps:

- 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:

- 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
- Behavioral_Cloning_P3_HIzadi.pdf summarizing the results

### 2. 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
```

### 3. Submission code is usable and readable

The 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 (Alternatively, the file model_generator.py uses a generator to train the model).

## Model Architecture and Training Strategy

### 1. An appropriate model architecture has been employed

My model is inspired by the NVIDIA convolutional neural network published here: *https://images.nvidia.com/content/tegra/automotive/.../end-to-end-dl-using-px.pdf*, with the following modifications:

1) One convolutional and one fully connected layer are removed to reduce the number of parameters, avoid overfitting and speed up training.
2) A dropout layer is used between convolutional layers and fully connected layers to increase robustness (model.py line 110).
3) Number of neurons in fully connected layers is halved to avoid over-fitting

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

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

The model contains dropout layers in order to reduce overfitting (model.py lines 110).

The model was trained and validated on different data sets to ensure that the model was not overfitting (model.py line 88). Also, the cropping layer is added (model.py code 89) to remove less useful part of the images. 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 with the learning rate of 0.0001 (model.py line 136).

**4. Appropriate training data**

The provided training data was chosen to keep the vehicle driving on the road. I used a combination of center lane driving and recovering from the right side of the road. Also, the augmentation technique is used by flipping the images to make sure the model output is not biased and handles both left and right turns (model.py lines 68-75).

## Model Architecture and Training Strategy

**1. Solution Design Approach**

The overall strategy for deriving the model architecture was to use a combination of convolutional and fully connected layers.

My first step was to use a convolution neural network model similar to the one used by NVIDIA team. I thought this model might be appropriate because it has shown great performance for this specific application.

In order to gauge how well the model was working, I split my image and steering angle data into a training and validation set by the portion of %80 to %20. I found that my first model had a low mean squared error on the training set but a high mean squared error on the validation set. This implied that the model was overfitting. And then to avoid over-fitting, number of tuning parameters are reduced by removing two layers, reducing the number of neurons in fully connected layers and also through using a dropout.

The final step was to run the simulator to see how well the car was driving around track one. There were a few spots where the vehicle fell off the track; for example, the first left turn after the bridge where there's a gravel side road was a really challenge to handle; I tackled that mainly by using the right camera images with a correction factor of 0.2.

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

**2. Final Model Architecture**

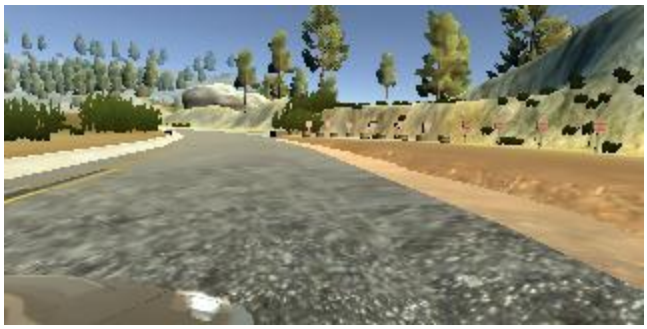
The final model architecture (model.py lines 92-131) consisted of the following layers:

- A normalization layer
- A cropping layer
- 4 convolution neural network layers with stride of 2, 'valid' padding, maxpooling and relu for each layer.
- A dropout layer
- 4 fully connected layers with 582, 50, 5 and 2 neurons.

**3. Creation of the Training Set & Training Process**

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

I then used recorded data with the vehicle recovering from the left side and right sides of the road back to center so that the vehicle would learn to steer rapidly back to the road whenever it is about to leave the road. I also used the right camera (model.py code line 42) to generate more images and help recovery Here's an example of right camera used for the gravel part of the road:



To augment the data sat, I also flipped images and angles thinking that this would reduce any bias in the steering angles because of always turning one direction. This helped car not drive on the road sides.

After the collection process, I had 5367 number of data points.

I finally randomly shuffled the data set and put 20% of the data into a validation set.

I used this training data for training the model. The validation set helped determine if the model was over or under fitting. The ideal number of epochs was 3 as evidenced by monitoring the training and validation loss which would get their minimum within 3 to 5 epochs.

## Summary:

A CNN model with fully connected and dropout layers is developed to mimic the behaviour of a human driver on the specified track; the new model uses a dropout technique to improve learning performance. The developed model successfully learned from the provided data and was able to autonomously and smoothly drive the car in the track without leaving the road.