# Behavioral Cloning Project

## Submission 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

### 1.Reading files and separating them

The function `findImages ()` Reading images from their locations, dividing them to 3 arrays of Left Camera Images, Right Camera Images and Center Camera Images and one array of angle measurements.

The function `combineImages` combine images from center, left and right using the correction factor (0.2). With that, we are telling the trainer that when the center camera reads an image similar to one of the training data of left camera so it should correct the steering and add 0.2 to the right direction.

### 2.Preprocessing

First, I did normalize the data from 0~255 to -0.5~0.5 using Lambda.
Then I cropped the image to use only the region of interest to be an input to the training process.

### 3.Model architecture

The overall strategy for driving a model architecture was to use NVidia model.

- *Convolutional layers (5)*

| Conv. Layers | Kernel / Filters no. | Activation | Output shape |
|---|---|---|---|
| 2D-Convolutional 1 | 5*5 / 24 | Relu | (None,43,158,24) |
| 2D-Convolutional 2 | 5*5 / 36 | Relu | (None,20,77,36) |
| 2D-Convolutional 3 | 5*5 / 48 | Relu | (None,8,37,48) |
| 2D-Convolutional 4 | 3*3 / 64 | Relu | (None,6,35,64) |
| 2D-Convolutional 5 | 3*3 / 64 | Relu | (None,4,33,64) |

- *Flatten (convert from 3D to 1D array)*
- *Fully connected layers (4)*

| Fully Connected layers | Output shape |
|---|---|
| Fully Connected 1 | (None, 8448) |
| Fully Connected 2 | (None, 100) |
| Fully Connected 3 | (None, 10) |
| Fully Connected 4 | (None, 1) |

-The model includes RELU layers to introduce nonlinearity.

-In order to gauge how well the model was working, I split my image and steering angle data into a training and validation set. 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.

-To combat the overfitting, I modified the model so that decrease no. of epochs and used more training data from both tracks.

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... to improve the driving behavior in these cases, I changed the color space from BGR to RGB. Without this step, the trainer gets confused about the colors and run on the colored surroundings

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

## 4. Creation of the Training Set & Training Process

To capture good driving behavior, I first recorded two laps on track one using center lane driving. I then recorded the vehicle recovering from the left side and right sides of the road back to center so that the vehicle would learn to give the appropriate steering angle in this case. Then I repeated this process on track two in order to get more data points.
To augment the data sat, I also flipped images and angles thinking that this would generalize the training.

Here is an example of my data:
The next 3 images are taken using image_generator.py file:

- A sample from my training data.
- The sample after converting color space to RGB.
- The sample after flipping the image.

After the collection process, I had 33417 number of data points. I randomly shuffled the data set and put 20% of the data into a validation set using `train_test_split` function from `sklearn.model_selection` . So, the validation data points are 6684.

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. I used an adam optimizer so that manually training the learning rate wasn't necessary.

## 5. Attempts to reduce overfitting in the model

- The model was trained and validated on different data sets to ensure that the model was not overfitting.
- Choosing the best batch_size :32, I tried 64 and the validation loss was larger.
- Choosing the best no. Of epochs :3, I tried 5 and there was overfitting.
- The model was tested by running it through the simulator and ensuring that the vehicle could stay on the track.

## 6. Model parameter tuning

The model used an adam optimizer, so the learning rate was not tuned manually.

## 7. Visualizing validation /training loss

I tried more than a value for both no. of epochs and for batch_size and realized that the best set of parameters that produce least mean square error is (batch_size=32, no. of epochs=3)

Here is a visualization of model mean square error loss vs epochs when using this set of parameters.