# Behavioral Cloning

---

## Model Architecture and Training Strategy

### 1. Solution Design Approach

First I used two Convolution2D 5x5 layers with MaxPooling2D after each of this layer, a flatten layer and 3 Dense layers.

With this architecture I was able to get drive the bridge, but my trained model failed to drive the bend directly afterwards.

Afterwards, I used tried out the nvidia architecture, which was recommendend in the course and forum. This is also my final architecture.

### 2. Final Model Architecture

I used the nvidia model as my final architecture (model.py lines 60-87). It consists of the following layers:

| Layer | Description |
|---|---|
| Input | 160x320x3 RGB image |
| Normalization | Normalization and cropping of images, output 160x220x3 RGB image |
| Convolution2D 5x5 | 2x2 stride, VALID padding |
| RELU | Activation |
| Convolution2D 5x5 | 2x2 stride, VALID padding |
| RELU | Activation |
| Convolution2D 5x5 | 1x1 stride, VALID padding |
| RELU | Activation |
| Convolution2D 3x3 | 1x1 stride, VALID padding |
| RELU | Activation |
| Convolution2D 3x3 | 1x1 stride, VALID padding |
| Flatten | - |
| Dropout | Rate: 0.25 |
| RELU | Activation |
| Dense | Output = 100 |
| Dropout | Rate: 0.25 |
| RELU | Activation |
| Dense | Output = 50 |
| Dropout | Rate: 0.25 |
| RELU | Activation |
| Dense | Output = 10 |
| Dropout | Rate: 0.25 |

| Layer | Description |
|---|---|
| RELU | Activation |
| Dense | Output = 1 |

The model includes RELU layers for all Convolution2D to introduce nonlinearity (model.py line 68-72) and after each dropout function (model.py lines 76, 79, 82, 85). Furthermore the data is normalized in the model using a Keras lambda layer (model.py line 55) and the images are cropped (75px from the top and 25px from the bottom) with the Keras Cropping2D function.

## 3. Attempts to reduce overfitting in the model

The model contains dropout layers in order to reduce overfitting (model.py lines 75, 78, 81, 84) with dropout probability of 25%. Additionally I used the keras fit function for training with a validation_split of 20% to ensure that the model was not overfitting (model.py line 96). The model was tested by running it through the simulator and ensuring that the vehicle could stay on the track.

## 4. Model parameter tuning

The model used an adam optimizer, so the learning rate was not tuned manually (model.py line 91). I used an offset of 0.25 to compute the steering for the images of the left and right of the camera. I tried different values for the offset (e.g.: 0.043, 0.2, 0.002, a calculated one, ...), but this one was the best.

## 5. Appropriate training data

Training data was chosen to keep the vehicle driving on the road. I only used the training data provided by Udacity as my own records where not good enough for training. I used images of all three cameras.

To augment the data sat, I flipped images and angles thinking that this would generalize the data (model.py lines 46 and 47). I used the keras fit function for training with 20% of the data as 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 5. I used an adam optimizer so that manually training the learning rate wasn't necessary. You can see the mse for each training epoch in the image below.

## 6. Video for track1

You can see that my model was able to drive the car in the udacity simulator around track in the "video.mp4" file.