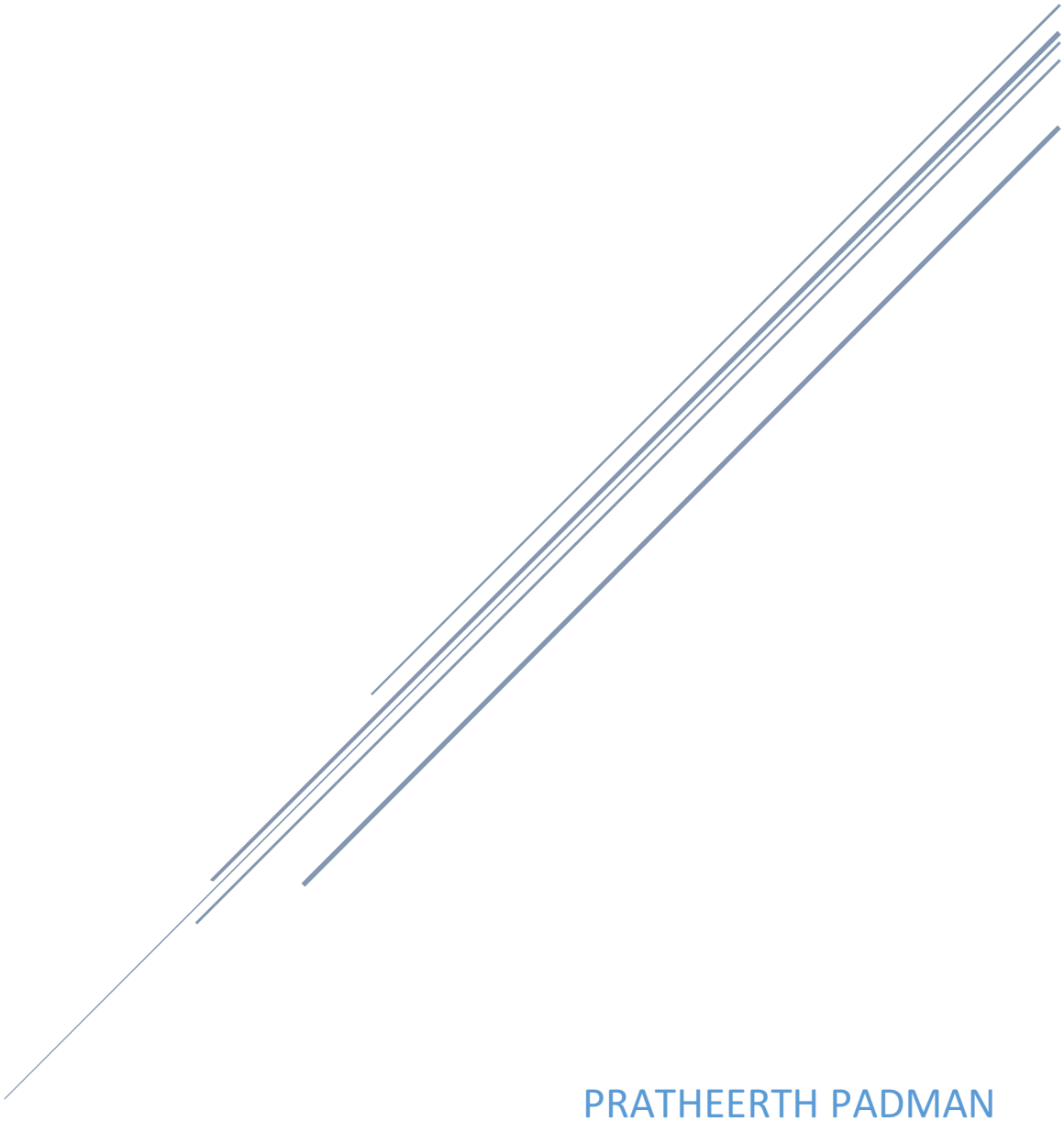
P3 – BEHAVIOURAL CLONING

SELF DRIVING CAR NANODEGREE



PRATHEERTH PADMAN

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

RUBRIC POINTS

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
- writeup_report.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.

MODEL ARCHITECTURE AND TRAINING STRATEGY

1. Model Architecture

My model architecture can be found in code-block – 24. It was inspired by the Nvidia Architecture, found here:
http://images.nvidia.com/content/tegra/automotive/images/2016/solutions/pdf/end-to-end-dl-using-px.pdf

The first layer uses a Keras Lambda layer to normalize the image data. The second layer crops the image so that elements apart from the track itself are not available for the network.

It is followed by three Convolutional layers of a 2x2 stride and 5x5 kernel size (24, 36 and 48 filters respectively), which is then followed by 2 strideless Conv layer of 3x3 kernel size (64 and 64 filters). The Conv layers include “ReLU” activations layers to introduce non-linearity.

These are followed by 3 fully connected layers interspersed with a 0.5 Dropout Layer to reduce over-fitting to the data. It was also trained only on 5 epochs to reduce over-fitting.

2. Model Parameter Tuning

An **Adam Optimizer** with a learning rate of 0.0001 was used. The model was trained for 5 epochs, each epoch having access to **15248 images for training** and **3813 images for validation**.
The validation loss hovered around 0.223

3. Training Data

Data provided by Udacity was used. It was down-sampled and augmented as discussed below.

Initially there were 24108 images provided by Udacity. To procure a balanced dataset, I randomly chose between the left, right and center camera images to add to the training set.

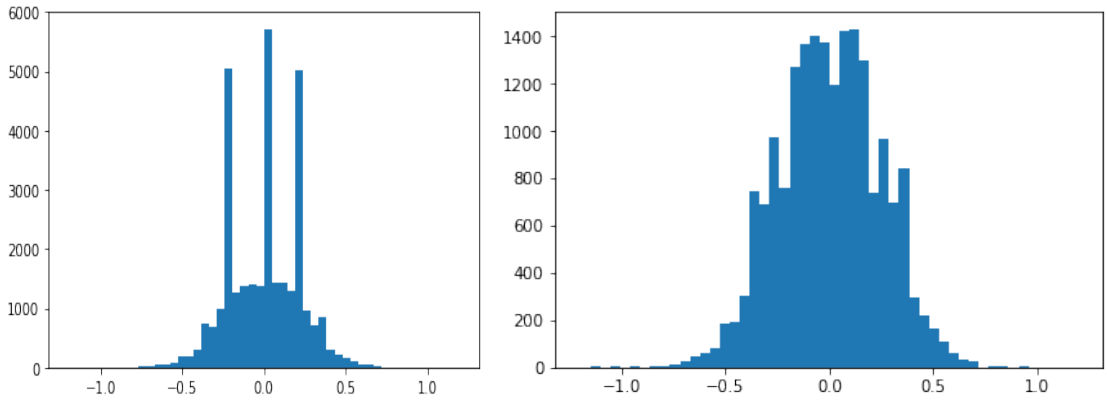
To the left camera image, I added 0.20 to the steering angle value and to the right camera image, I subtracted 0.20.
I ended up with 8036 images. The whole operation can be seen in codeblock-4
To help generalize the model, we have to augment the data. Three types of augmentations were chosen, flipping the image (inverting the steering angle), random translations and random brightness. The application of all three led to procuring 32,144 images. This can be seen in code-blocks 5, 6, 7 and 8.

Plotting the measuring angle revealed a heavy bias towards three values: 0, +0.25 and -0.25 which was understandable given how we have procured the images.

Feeding this data as it is would have resulted in terrible driving behavior. I decided to down-sample the three values which resulted in a fairly uniform graph. Down-sampling can be seen on code-blocks 12 and 18.

The result was **19061** images

The first picture represents the measurement values before down-sampling and the second one represents the values after.



drive.py uses images in the RGB formats and ours were all in the BGR format. The conversion was promptly done on code-block 21.