Behavioral Cloning

Jean-Paul Wilson

1 Abstract

The following project is a demonstration of the ability to clone driving behavior by implementing deep learning techniques and tools. More specifically, a convolutional neural network was used to implement an agent that, provided with i.) steering angles and ii.) camera images of the road ahead as inputs, was able to output the correct steering angles given new camera images. The model was compiled, trained and validated using Keras on top of Tensorflow.

For safety and cost purposes, a simulator, built by Udacity (using Unity) is used for this project instead of a car with video of real world driving. However, the model should work just as well with a real car and real-world camera inputs, as done by NVIDIA, and described in their publication on end to end learning. The model architecture employed by NVIDIA in the above-linked paper formed the foundation for the model used in this project.

Following the completion of training and validation, the agent was required to successfully navigate a complete lap of the track from Udacity's simulator without contact with a curb (or display any behaviour that would be deemed unsafe if performed in a real world situation).

The project results were favourable in that the model was able to clone driving behavior and then output the appropriate steering angles required to navigate a lap of the track provided, with only camera images and corresponding steering angles as its input.

2 Submitted files

The following files are included in this repository make hyperlinks:

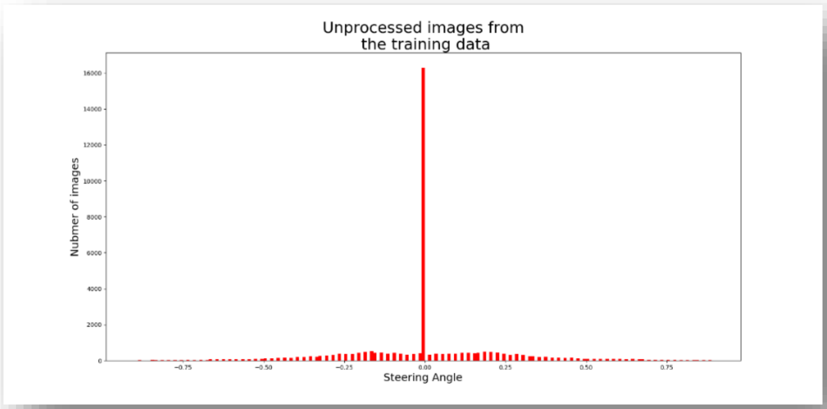
- model.py Contains all of the code required for the generation of the model
- drive.py Contains supporting code, including the methods required for driving the car around the simulator track in autonomous mode, making use of the training agent built in model.py. The only changes that were made to this file by myself were to accommodate the resized images in the 'telemetry' method:
    - Addition of the `resize` method (lines 54-55)
    - Modification of code to run `model.predict` on resized images (lines 70-71)
- **Recording.mp4** This is a recording of the vehicle navigating a lap of the track in autonomous mode.
- **model.h5** This file contains the compiled model after training and validation have been completed.
- writeup_report.pdf

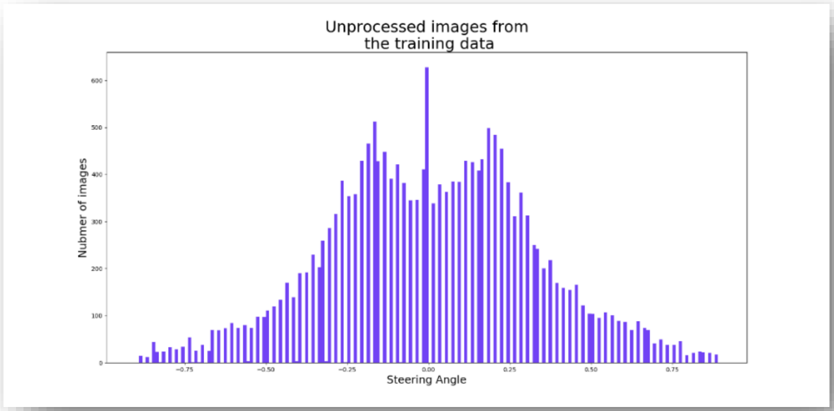
3 Methodology

3.1 Project Steps

The methodology for completing this project can be described as iterative. To begin, suggestions provided in the course material by the instructor (David).

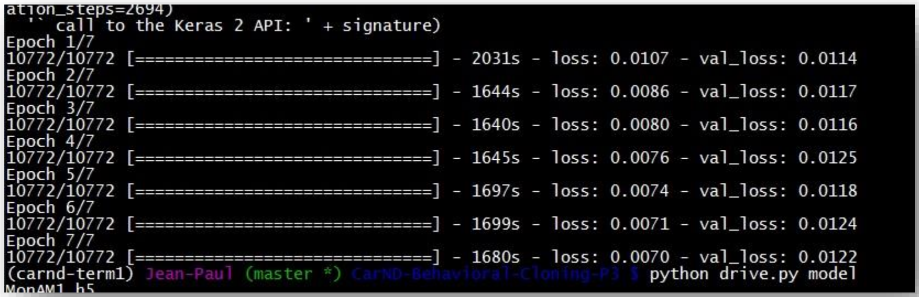
1. Initially a very basic model was generated, simply to ensure that the environment, image capturing and simulator, etc were operating as required.
2. Once this was achieved, and the vehicle was able to drive and steer (although not stay on track), a slightly more sophisticated model architecture was implemented; LeNet-5. At this point, the vehicle was able to navigate to some small extent. That is, it seemed that the steering outputs from the car in autonomous mode corresponded to the layout of the track in the video input. The car was not able to stay on track for long periods, however.
3. Data Collection: At this point, more data was collected.
   - The mouse was used for steering.
   - The fastest simulator graphics setting, and smallest (320 x 240) screen size was selected. This was to ensure a processing time of as fast as possible.
   - Recording was done as follows:
     - Two laps in each direction (four laps)
     - Three laps in each direction, recording only recovery
       1. With car parallel to road edge
       2. With car facing outward moving off of road.
     - Two laps in each direction recording only the corners.
     - Several instances where I read in the forums of issues (on, before and after bridge).
     - Over a lap in each direction on the second track.
   - The total number of images came to roughly 35 000 images.
4. Pre-processing and image augmentation techniques were employed as follows:
   - Removal of a large percentage of the images associated with straight line data. (See figures below)





- Image resizing: In order to speed up the processing, the pixel dimensions of the training images were reduced from 320 x 160px to 128 x 64px.
- Image Cropping: In order to remove irrelevant data from the images (such as the sky, and hood of the car), the images were all cropped. The Cropping2D layer of Keras' Sequential model was used for this, which meant that cropping would occur when making predictions without any modifications to the in drive.py.
- Image flipping: One quarter of the images in each batch of 32 images are flipped, and their steering angle multiplied by -1.

5. In order to speed up processing, the large data set collected was reduced to a smaller amount (generally about 5 000 images, and training was carried out (on the models described in the proceeding steps) with 3-4 epochs during the iteration (model improvement) process.
6. The NVIDIA model architecture was then put in place of LeNet-5.
7. The steps above provided improvements, but the car still kept going off of the track in certain places. At this point, for a period of about 3-4 weeks, I experimented with various changes to the model. The model's loss reduced substantially for the first four epochs, and then slightly or not at all in later epochs. The validation losses however did not improve at all, which was a major concern. This is seen in the figure below.

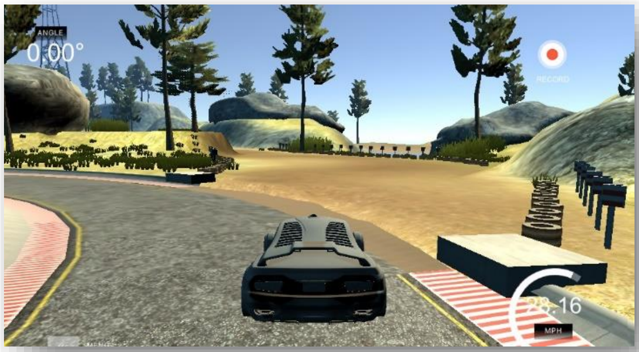


```
ation_steps=2694)
  `` call to the Keras 2 API: ' + signature)
Epoch 1/7
10772/10772 [==============================] - 2031s - loss: 0.0107 - val_loss: 0.0114
Epoch 2/7
10772/10772 [==============================] - 1644s - loss: 0.0086 - val_loss: 0.0117
Epoch 3/7
10772/10772 [==============================] - 1640s - loss: 0.0080 - val_loss: 0.0116
Epoch 4/7
10772/10772 [==============================] - 1645s - loss: 0.0076 - val_loss: 0.0125
Epoch 5/7
10772/10772 [==============================] - 1697s - loss: 0.0074 - val_loss: 0.0118
Epoch 6/7
10772/10772 [==============================] - 1699s - loss: 0.0071 - val_loss: 0.0124
Epoch 7/7
10772/10772 [==============================] - 1680s - loss: 0.0070 - val_loss: 0.0122
(carnd-term1) Jean-Paul (master *) CarND-Behavioral-Cloning-P3 $ python drive.py model
MonAM1.h5
```

The changes to the model included:

- Changing the number of convolutional model layers (alternating between 3 and 9).
- Adding dropout layers (with percentages from 20% up to 60%).

    - Adding MaxPooling layers
8. Additional training data was added when the vehicle was unable to stay on track in certain places (for example, after the bridge, where there is a dirt road and no paving edge (left turn before dirt, shown below).



9. Once the model was able to successfully navigate a lap of the track as required, the model architecture was recorded and the navigation was recorded and a video made using video.py

3.2 Observations

Following the methodology in the notes, in order to train an agent to steer the car and navigate with a success rate of about 90% of course navigation is relatively straight forward. However, in order to construct a model that can navigate the track completely requires substantial insight and understanding of the complexities of the training library (Keras), Deep learning concepts in general, and convolutional neural networks in particular, along with experience building and testing models, in order to get an 'intuitive feel' for what strategies are effective or not.

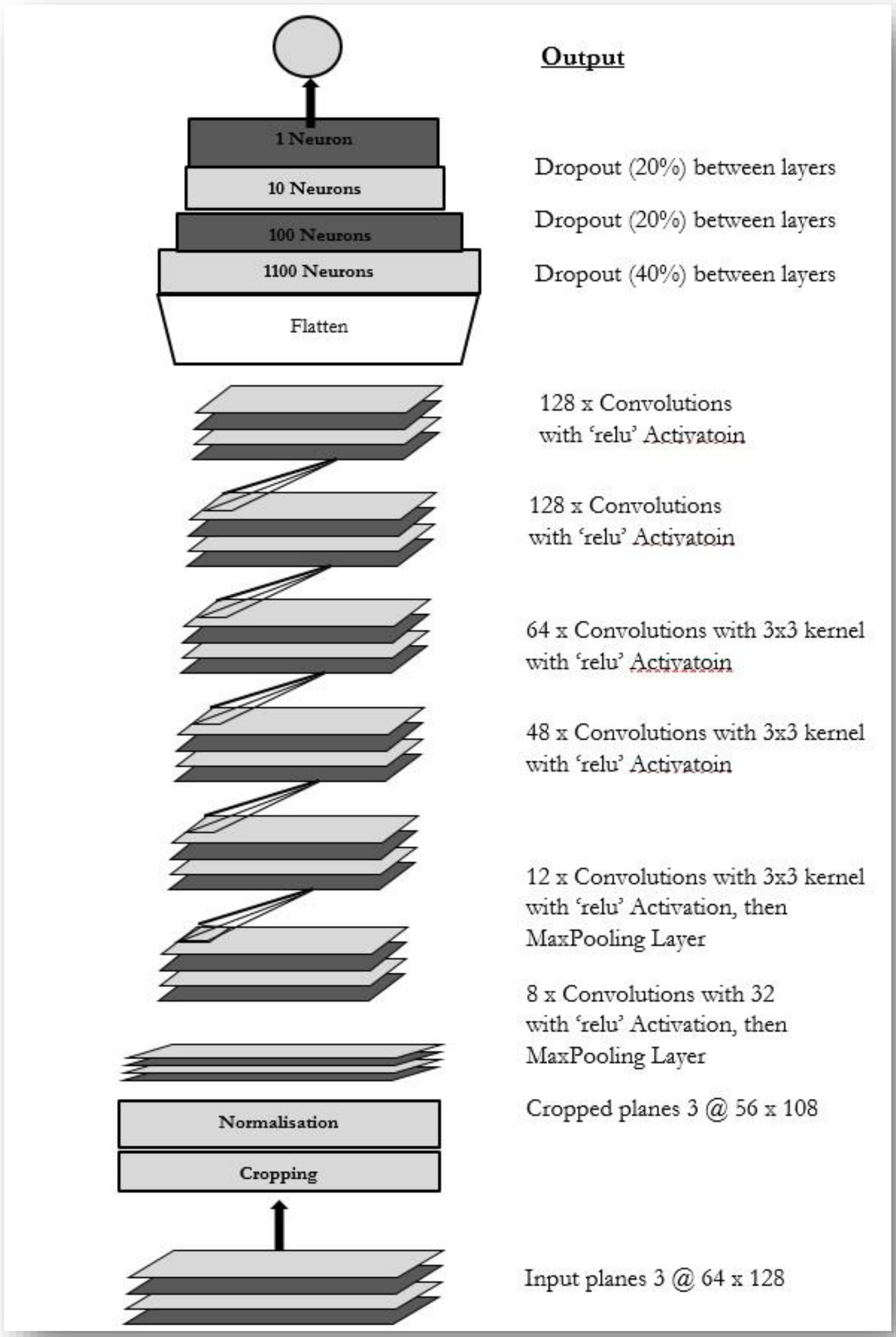
4 Training Agent Architecture

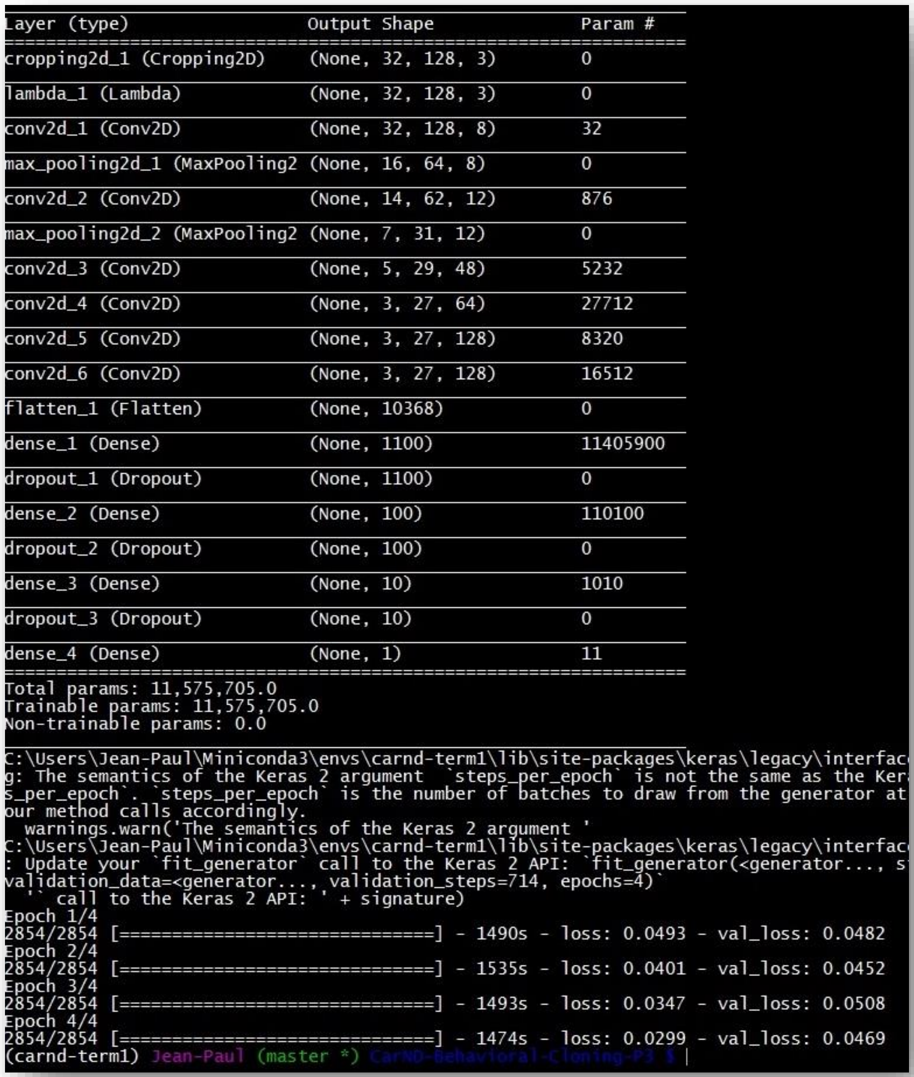
The final architecture includes the following features:

- 11 575 705 parameters
- 6 Convolutional layers (some with 3x3 kernels, others 1x1 -single pixel (feature maps))
- Dropout layers between the fully connected layers
- Maxpooling layers between some of the convolutions

The final losses were:

- Training loss: **0.0299**
- Validation loss: **0.0469**


Output
1 Neuron
10 Neurons
100 Neurons
1100 Neurons
Flatten
Dropout (20%) between layers
Dropout (20%) between layers
Dropout (40%) between layers
128 x Convolutions with 'relu' Activatoin
128 x Convolutions with 'relu' Activatoin
64 x Convolutions with 3x3 kernel with 'relu' Activatoin
48 x Convolutions with 3x3 kernel with 'relu' Activatoin
12 x Convolutions with 3x3 kernel with 'relu' Activation, then MaxPooling Layer
8 x Convolutions with 32 with 'relu' Activation, then MaxPooling Layer
Cropped planes 3 @ 56 x 108
Normalisation
Cropping
Input planes 3 @ 64 x 128



```
Layer (type)                 Output Shape              Param #
=================================================================
cropping2d_1 (Cropping2D)    (None, 32, 128, 3)        0
_________________________________________________________________
lambda_1 (Lambda)            (None, 32, 128, 3)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 32, 128, 8)        32
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 16, 64, 8)         0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 14, 62, 12)        876
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 7, 31, 12)         0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 5, 29, 48)         5232
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 3, 27, 64)         27712
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 3, 27, 128)        8320
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 3, 27, 128)        16512
_________________________________________________________________
flatten_1 (Flatten)          (None, 10368)             0
_________________________________________________________________
dense_1 (Dense)              (None, 1100)              11405900
_________________________________________________________________
dropout_1 (Dropout)          (None, 1100)              0
_________________________________________________________________
dense_2 (Dense)              (None, 100)               110100
_________________________________________________________________
dropout_2 (Dropout)          (None, 100)               0
_________________________________________________________________
dense_3 (Dense)              (None, 10)                1010
_________________________________________________________________
dropout_3 (Dropout)          (None, 10)                0
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 11
=================================================================
Total params: 11,575,705.0
Trainable params: 11,575,705.0
Non-trainable params: 0.0
_________________________________________________________________
C:\Users\Jean-Paul\Miniconda3\envs\carnd-term1\lib\site-packages\keras\legacy\interfac
g: The semantics of the Keras 2 argument  `steps_per_epoch` is not the same as the Ker
s_per_epoch`. `steps_per_epoch` is the number of batches to draw from the generator at
our method calls accordingly.
  warnings.warn('The semantics of the Keras 2 argument '
C:\Users\Jean-Paul\Miniconda3\envs\carnd-term1\lib\site-packages\keras\legacy\interfac
: Update your `fit_generator` call to the Keras 2 API: `fit_generator(<generator..., s
validation_data=<generator..., validation_steps=714, epochs=4)`
  '` call to the Keras 2 API: ' + signature)
Epoch 1/4
2854/2854 [==============================] - 1490s - loss: 0.0493 - val_loss: 0.0482
Epoch 2/4
2854/2854 [==============================] - 1535s - loss: 0.0401 - val_loss: 0.0452
Epoch 3/4
2854/2854 [==============================] - 1493s - loss: 0.0347 - val_loss: 0.0508
Epoch 4/4
2854/2854 [==============================] - 1474s - loss: 0.0299 - val_loss: 0.0469
(carnd-term1) Jean-Paul (master *) CarND-Behavioral-Cloning-P3 $ |
```