####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
- This report summarising 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

Instructions:

To train the model -

python3 new_model.py --epoch 10

To test the outputs in the simulator:

```
docker run -it --rm -p 4567:4567 -v `pwd`:/src
udacity/carnd-term1-starter-kit python drive.py model.h5
```

####3. Submission code is usable and readable

My code is in the file model.py and is appropriately modularised and commented where required.

###Model Architecture and Training Strategy

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

I've adopted nVidia's self driving car model https://devblogs.nvidia.com/parallelforall/deep-learning-self-driving-cars/ consisting of :

- Initial convolution layers : 5 x 5 kernel size with L2 regularisation added
- RELU activations to achieve non-linearity
- Dense layers 80, 40 , 16, 10 and 1
- Dropouts between all dense layers

- Adam optimiser with a learning rate of 0.0001 and a Mean Squared Error loss

The images are normalised using the keras lambda functions before being fed to the network

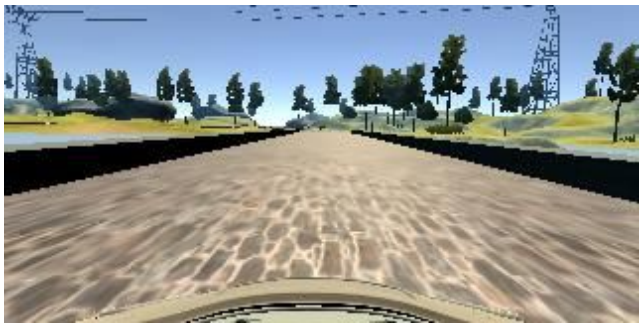
To achieve this model architecture, a TON of trial and error was involved. I started off with the basic nVidia neural net and about 5 laps of training data which included clockwise and anticlockwise driving. I tried several variants of training data with several laps from 8 - 15. None of these worked. I started fine-tuning the network step by step, first I used only centre camera images and added recovery laps to the training data as suggested in the forums and by my mentor. Once that didn't work, I tried adding dropouts to all the layers, as I was not sure where to add the dropouts to. That didn't work either. The car would always veer off after crossing the bridge.

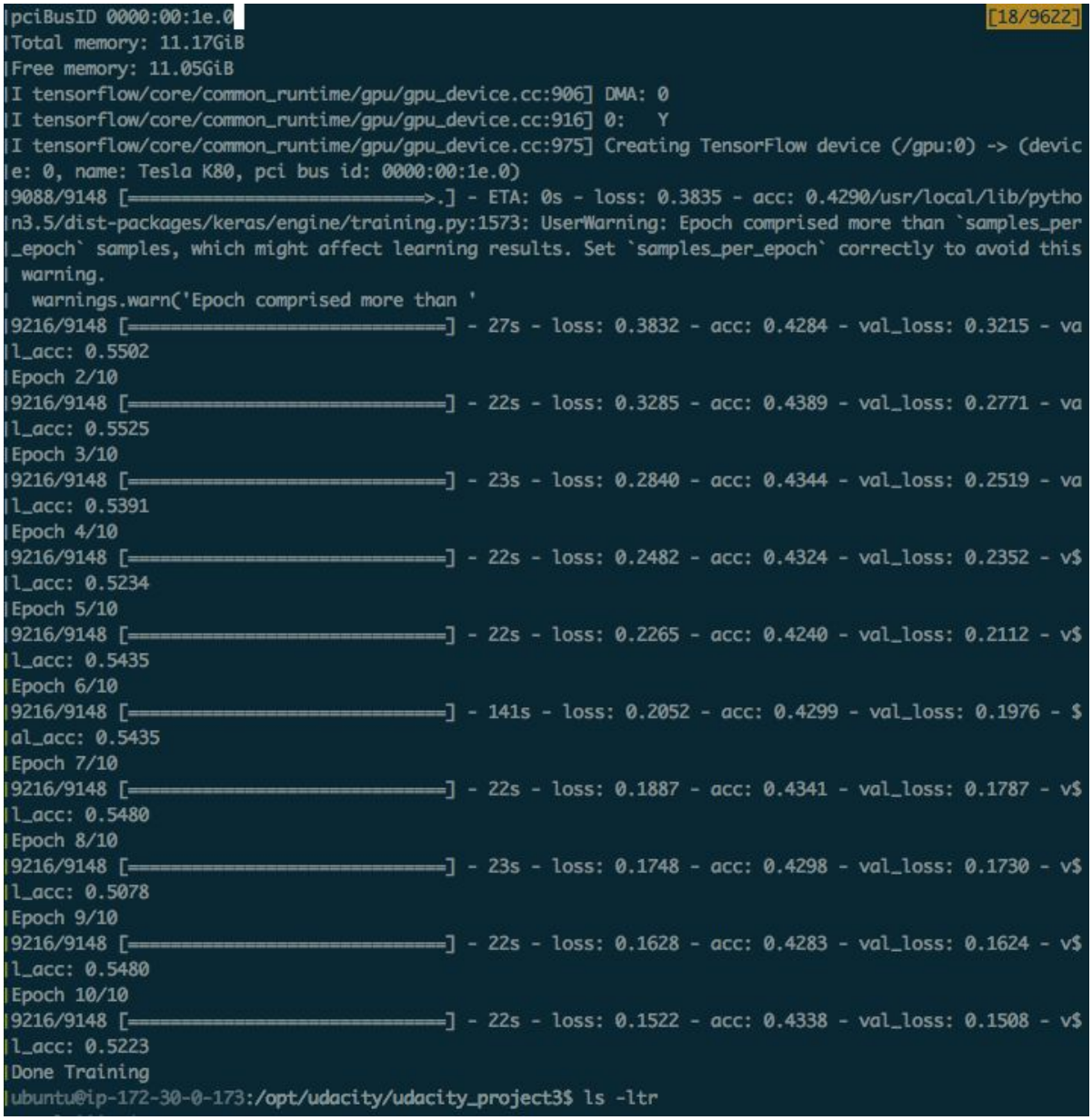
I then used only the images provided by Udacity sample data with some recovery laps thrown in and I was able to get pass the left turn after the bridge. However, I got stuck again at the immediate right turn.

I then came across the concept of random sampling in the forums. That was the key find. Once I added randomness into sampling and flipping, combined that with random brightness I was able to get the car to complete a loop.

Key training strategies:

- Pre-processing the images to remove the sky etc and crop the image to 64 x 64
- Data augmentation:
    - Left and right camera images by applying the appropriate correction. (0.25 - 0.27). Finally the model worked with 0.27
    - Random brightness generation
    - Random image flipping on the vertical axis

- Random sampling of images so that only the selected random set is trained.
- Dropouts in the dense layers
- L2 weight regularisation in all layers
- Adam optmiser with a learning rate of 0.0001
- Training and validation generators to save memory
    - Validation generator uses only centre camera images
- Data trial with several laps of driving both in clockwise and anti-clockwise direction.
- Final dataset used - the Udacity provided training data.
    - Example center lane image
    - 
- Random shuffling with a 90 to 10 train test split.
- Tried several epochs using trial and error and finally settled for 10. I asked my mentor about the warning shown below as the car was veering off randomly in the test run. I figured I had messed up with my drive.py script and once I got a new script from the repo, the car drove fine.



Testing and final run:

I ran the output model in the simulator and once the lap was complete I re-ran and recorded the output. The video output of my final testing is available in the repository under the output directory.

Issues and shortcomings and next steps

Deep neural networks are a black box when it comes to debugging. It's really hard to precisely debug like you do in traditional programming. For instance, it was impossible for me to find out why the car was veering away after the bridge. I

could only make a few educated guesses and then augment the training data. It’s really time consuming and frustrating to do this.

Also we cannot drive a car in whatever way we choose on the road to obtain training data. That’s another limitation.

I would like to try building new neural networks for self driving cars based on Imagenet winning networks such as VGGNet or Resnet.

Oh, and this will work only on a plain road, need to figure out a decision making process when traffic signs, pedestrians, obstacles and other practicalities are involved.