Udacity SDCND

3. Behavior Cloning

Daniel Tobias

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

Dataset technique

I actually did a project similar to this with an actual scale robotic car. For that project I used Nvidia's deep convolution network from the paper *End to End Learning for Self-Driving Cars.* My experiences with training a scale car to run in a closed loop hallway helped me with my driving technique for this project.

To gather data I did roughly 7 laps around the track, I also selectively recorded extra data on major turns so that the net had ample turning data. Another technique I learned from my real life project was that to take the corners very slowly and "gun" the straightaways. The idea behind this was to generate significantly more data for the turns since this was the most critical part of staying inside the track There were also some corrective steering around the bridge area to help the car converge to the center of the lane before crossing. One of the biggest challenges was getting the car to correctly turn right after crossing the bridge. To overcome this I did multiple passes around this bend at a very slow speed. For the entire data collection run I ran the car at 8-11 mph, similar to that of autonomous driving speed. The final dataset was close to 100,000 images with its corresponding steering angle. It is also important to note that I did use an xbox controller attached to my computer to generate finer steering angles. I looked at the helper guide that was suggested for this project and saw that it was almost necessary to have, and I happened to have one from my project. Data was also collected using the lowest possible resolution and graphic settings on the simulator.

Udacity SDCND

3. Behavior Cloning

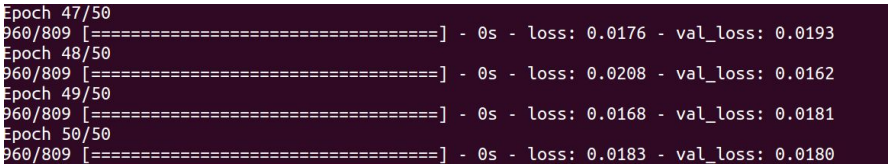
Daniel Tobias

Augmentation

To give the model more data, I used all the left and right images to help create artificial turning data. The idea was to take the current angle with one of the left or right images and subtract 0.20 or add 0.20 respectively. This plus my driving corrections and additional steering around corners was overkill. I think that if I could have gotten away with less laps and no additional segments of recording around corners. I also tried a black and white image and saw similar performance with it being able to accurately lap the course.

Architecture

Multiple architectures were tried, primarily based around a multi-layer convolutional neural network. The network can be described as the same convolutional layers of the nvidia End to End model but using the true size of the image at 320x160, instead of 66x200 that nvidia describes in their paper. At the end of the conv layers and flattening operation, the neurons are put through a dropout layer of value 0.8 so that only the strongest of correlations can get past. After dropout it was connected to 120 neuron fully connected layer,then 50 to 12 then finally 1. The fully connected layers looked similar to the fully connected layers of the sign classifier in the previous project. The reason I did this was from my experience deploying a similar net on my robotic rc car. I learned that you don't really need 1164 neurons on the as the first layer of fully connected neurons, to drive in a simple loop. Maybe amount is needed for a more diverse setting like in real life, but in my case and this project's case the simulator are rather simple environments. Also the fully connected layers are what really bulk up the final memory size of the trained weights, in our case the model.h5 file size. Since I am using the full size of the image, if I ran the output of the conv layers to the beginning of a 1164 neuron fc layer the resulting model size would be close to 75mbs roughly 10x the size of my current model. I tried both models the 7.6MB and 76MB model on the track and noticed no improvement of performance. Also a smaller learning rate of 0.0001 was used instead of the adam optimizer's default value of 0.001. I noticed that while it took longer to converge requiring more epoch using the smaller learning rate, often the model performed better than which I go over a bit more in the conclusion. The validation set was 20% of my total data and after 50 epoch both mse rates were identical at 0.0182



Udacity SDCND

3. Behavior Cloning

Daniel Tobias

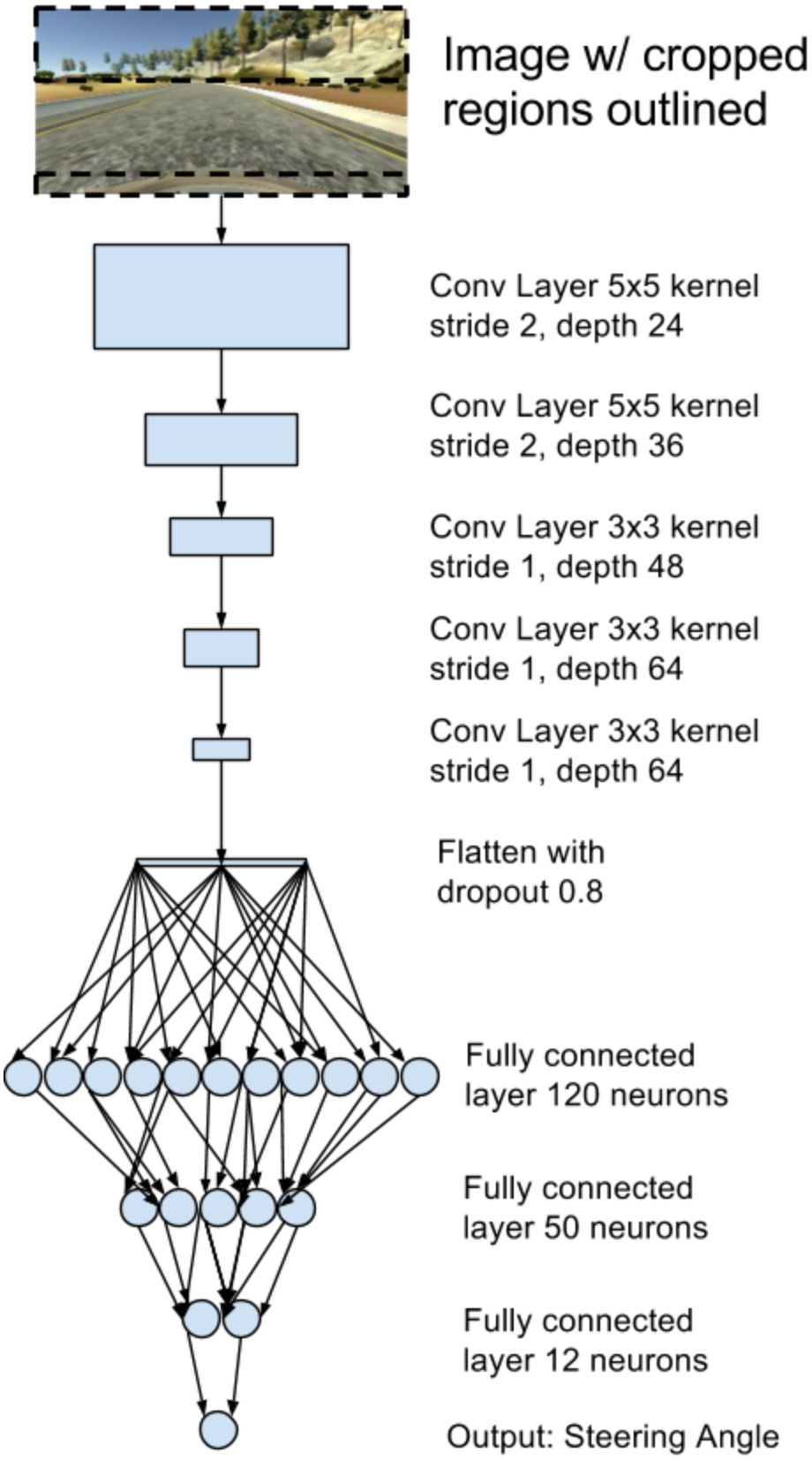


Figure 1. Model architecture

Udacity SDCND

3. Behavior Cloning

Daniel Tobias

Conclusion and remarks.

This project took significant time to master, it was also hard to pinpoint what was going wrong. For example if the problem was because the relevant features were not learned with the conv weights, or if I was unlucky with the original random starting point of the weights and it couldn't properly create a model that could drive. I messed around with the amount of epochs and even ran through all 100,000 images which took almost half an hour on a GTX 1080. But I saw no difference between using the entire dataset (1000 epochs of roughly 1000 images) and running it for only 50 epoch. Even 50 epochs was over kill I originally defaulted with 12 but I noticed after retraining on some the model couldn't drive correctly at certain turns, and realized that 50, while excess, was a safe bet. My main problem with the simulator is when you suddenly accelerate, decelerate/brake It leaves behind smoke which lingers for a few almost a minute. This smoke effect does get captured on the camera as you can see in this image

Figure 2. Burnout effect

This could be viewed as free data augmentation, but in my case I found it pretty annoying and the cloud could potentially throw off the training process of a model, if it correlated the cloud with a rare high steering angle. It is also arguable that you could complete this project by overfitting to your dataset, by doing no dropout operations. Overall the project was challenging and displays that simulations are great tools for testing and validating deep neural networks.