Vatche Donikian

Project 3 Writeup

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

My code works by executing the following line in the miniconda prompt with the carnd-term1 environment activated:

```
python drive.py model.h5
```

Once this is entered, opening the simulator and clicking on autonomous mode will make the vehicle be controlled by my trained model.

3. Submission code is usable and readable

The model.py file contains the code for training and saving the convolution neural network as the file ‘model.h5’. The training file shows the pipeline I used building the model, and it contains comments to explain how the code works. I can retrain all the weights, or load the previously saved model and weights to fine-tune my own model, like in transfer learning.

Model Architecture and Training Strategy

1. An appropriate model architecture has been employed

My model consists of a convolution neural network constructed in Keras. It consists of 4 convolutional layers, with max pooling and activation, and 3 fully connected layers before the output layer. There are also two preprocessing steps: cropping and normalization, which were implemented in the model for the GPU to parallelize the computation. Normalization is done with the Keras Lambda Layers. The activation method used was RELU and was used in order for the model to act in nonlinear ways. There is also one dropout layer to prevent overfitting. The model structure can be found in model.py in lines 63-112.

2. Attempts to reduce overfitting in the model

As stated above, the model contains one dropout layers to reduce overfitting (model.py lines 75). The dropout rate used was 50%.

To prevent overfitting, the training data recorded was from different tests. If the training data was the same lap over and over, the data could easily be overfit. Thus, I recorded a couple laps regularly, then I turned the car around and recorded a couple laps going the other direction. I also recorded some extra curves for the model to understand how to best approach a sharp curve.

3. Model parameter tuning

The model used an adam optimizer, so the learning rate was not tuned manually. This can be found in the model compilation command on line 115 of model.py.

4. Appropriate training data

Training data was chosen to keep the vehicle driving on the road. As stated above I recorded a few laps in opposite directions and some additional curves. I used all three cameras instead of recording recovery laps.

For details about how I created the training data, see the next section.

Model Architecture and Training Strategy

1. Solution Design Approach

To create an accurate model, I needed to record data different ways, then train the neural network, then test the system in autonomous mode until I was satisfied.

First I recorded data. The first set of data I recorded included some times getting close to the edge of the road on the curves. After my model was not learning well enough for my standards, I actually scrapped the first recorded data set and started over. I used a couple laps regular, then a few the opposite direction, in order to take care of the left turn bias in Track 1. I also recorded sharp curves because my model was running off the road without this extra recording. I felt that recovery recordings were a little unrealistic, since you cannot perform that safely on an actual car. Thus, I used all three cameras on the vehicle. I fed in the left and right cameras with an adjustment of angle in order to help with drifting out of the lane (lines 31-43 model.py).

Next, I constructed my convolutional neural network. I started with simply one convoluted layer with max pooling, dropout, and activation, followed by one fully connected layer and its associated activation. However, this was not complex enough and my model was not performing well. Therefore I ended up trying the network described in the NVIDIA paper referenced in the project lesson. This ended up working well, I just tweaked it to add an extra activation layer to handle the nonlinearities.

After splitting the data into training and validation, I compiled and ran the network to train. The mean squared error was the cost parameter I was minimizing. The mean squared error was close in training and validation, so this meant I was not overfitting. However, I adjusted the number of epochs run to get a lower mean squared error and a more accurate model. I tested with 3, 5, 10, 15, and 20 epochs and settled on 10.

Once I was happy with the training and validation results, I ran the model through the simulator in autonomous mode, using the drive.py file provided. As I stated above, the first couple times the model did not respond well so I ended up creating a new dataset to train from, and this worked well. One thing I did was adjust drive.py so that the vehicle would drive faster, because autonomous mode was very slow. I adjusted the speed and throttle up (lines 37-44 in drive.py).

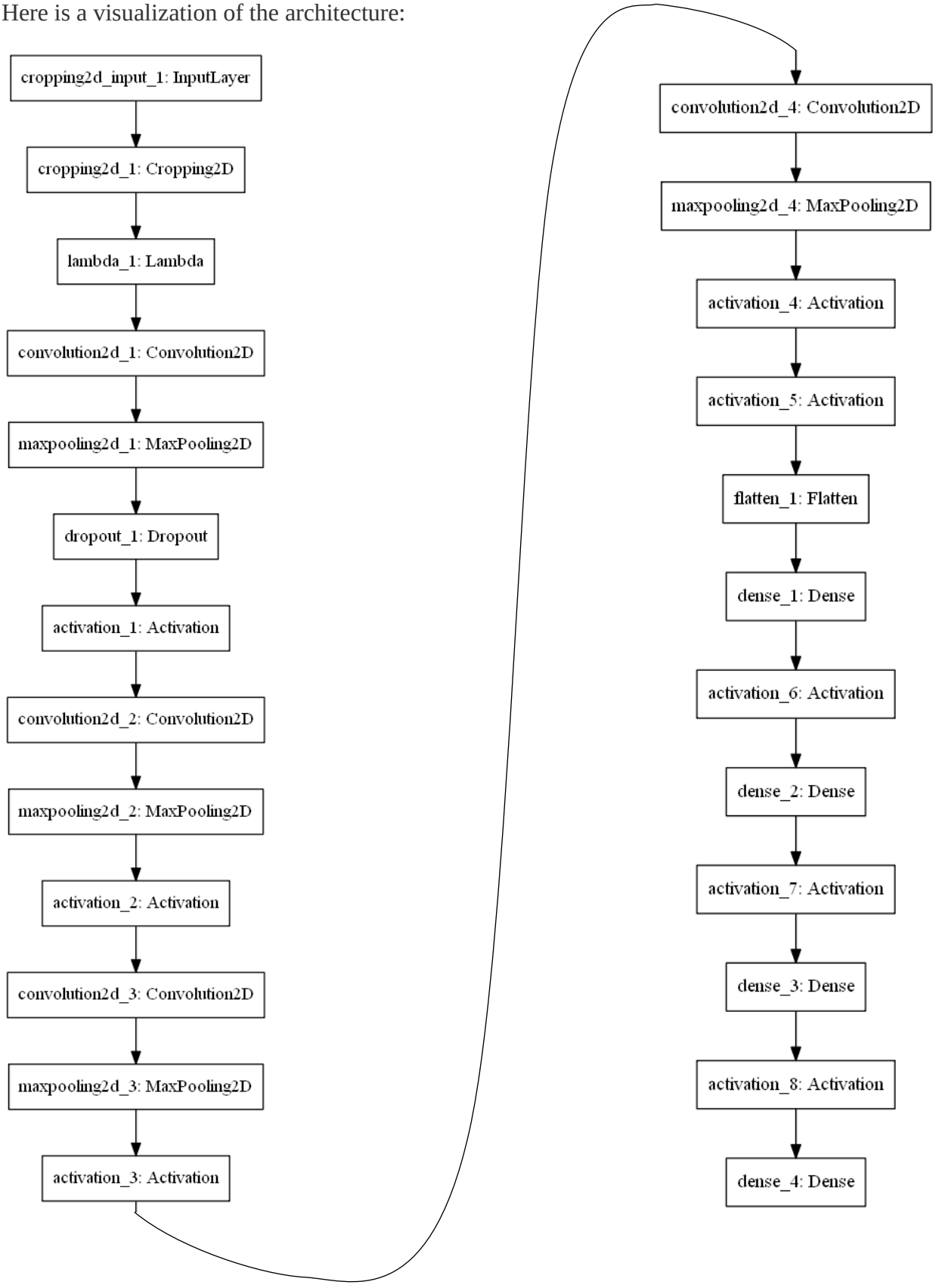
At the end of the process, the I was able to get the vehicle to drive autonomously around the track without leaving the road. It gets close on some turns but corrects itself, which I actually think is great and shows a strong network and that the three camera approach did well for recovery.

2. Final Model Architecture

The final model architecture consisted of a convolution neural network with the following layers and layer sizes:

- Image cropping
- Image pixel Normalization
- Convoluted layer 1: filter depth of 24, 5x5 kernel size
    - Max pooling with size 2x2
    - Dropout layer, 50%
    - Activation layer, with RELU
- Convoluted layer 2: filter depth of 36, 5x5 kernel
    - Max pooling with size 2x2
    - Activation layer, with RELU
- Convoluted layer 3: filter depth of 48, 5x5 kernel
    - Max pooling with size 2x2
    - Activation layer, with RELU
- Convoluted layer 4: filter depth of 64, 3x3 kernel
    - Max pooling with size 2x2
    - Activation layer, with RELU
- Extra activation layer, with RELU
- Flatten Network
- Fully connected layer 1: depth of 100
    - Activation layer, with RELU
- Fully connected layer 2: depth of 50
    - Activation layer, with RELU
- Fully connected layer 3: depth of 10
    - Activation layer, with RELU
- Output layer, final output of 1 (steering angle)

Here is a visualization of the architecture:



This visualization was created with the keras command ‘plot’.

3. Creation of the Training Set & Training Process

To capture good driving behavior, I first recorded two laps on track one using center lane driving. Here is an example image of center lane driving:

In order to get the model to deal with the going off the road, I used the left and right cameras as well. Here is an example of an instance with left, center, and right cameras:

I ended up using an adjustment of 0.3 for the steering angle for the images. This can be described with the equations below:

$$\theta_{left} = \theta_{\text{center}} + 0.3$$

$$\theta_{right} = \theta_{center} - 0.3$$

I then captured images from driving the vehicle the opposite way on the road, in order to get a good balance of left and right curves, and to prevent track memorization by the network. For a couple more laps, I just recorded the sharp curves, to provide more data to the network to train with. An example of these curves is below:

After the collection process, I had 5,793 instances, which due to the use of all three cameras, resulted in 17,379 data points. The preprocessing of this data was done in the keras model and described in the previous sections.

I finally randomly shuffled the data set and put 20% of the data into a 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 10. Anything less, my model was still learning and did not complete its accuracy. Any more epochs and I was just wasting computation as the model was not getting any better. I used an adam optimizer so that manually training the learning rate wasn't necessary.

In conclusion, I used a deep neural network model to successfully drive the simulator vehicle autonomously around the track without leaving the road. The network consisted of multiple convoluted layers and fully connected layers. Different types of training data were recorded to get a good sample and prevent overfitting. Overall I though this project was very useful and fun. A video of my simulator is included in the submission.